Prénom :
Nom :
Groupe TD :
Date :

Principes des systèmes d'exploitation

François Merciol, Matthieu Le Lain

UBS - IUT

DUT Informatique – 2nd année

Vannes – 2021/2022

Support de cours **étudiant**

http://m3101.merciol.fr/

(page gauche vide)

Table des matières

Table des figures

Liste des tableaux

Listings

Ce support de cours se compose des parties suivantes :

- **Introduction** : qui traite de la structuration du module enseigné et d'un survol des fonctions principales d'un système d'exploitation,
- **Ressources** : sur le partage des ressources et particulièrement l'ordonnanceur des processus,
- **Fichiers** : sur les systèmes de gestion de fichiers avec le cas particulier d'Unix,
- **Mémoire** : sur différentes organisations de mémoire pour une gestion efficace des ressources,
- **Tâches** : sur la gestion de tâches au sein d'un processus (également appelé processus légers),
- **Entrée/Sortie** : sur la gestion de cache et le formatage des données,
- **Réseau** : sur la présentation de l'interconnexion des systèmes d'exploitation et la présentation d'un outil : les "*sockets*"

Première partie

Introduction

Avant de commencer ce cours, nous devons mettre en place un certain nombre de règles. L'objet n'est pas de prendre les étudiants en traite en leur posant des questions inattendues sur une phrase perdue au milieu du cours. Pour que chacun soit évalué de façon équitable, il ne doit pas subsister de quiproquo. Cette section aborde l'organisation du cours et son évaluation.

1 Structuration du cours

Les cours en amphi demandent beaucoup d'attention. L'attention décroît après 40 minutes. Pour éviter la perte en concentration et rompre la monotonie, les cours seront organisés en 40 minutes de théorie / 10 minutes de réflexion / 40 minutes de pratique (suivant la table 1).

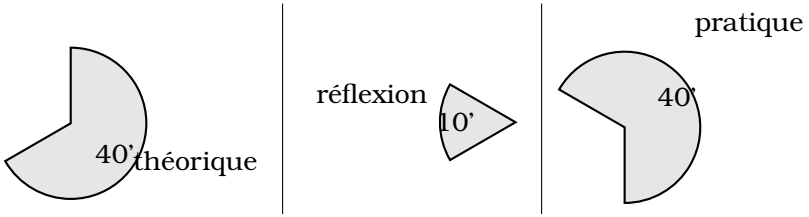


Figure 1 – Découpage des cours d'1h30

Concernant vos facultés d'apprentissage, les travaux de Michel Desmurget, chercheur en neurosciences cognitives, sont incontournables. Il a écrit "TV Lobotomie" (2011), où il collecte les résultats sans appel de toutes les études scientifiques qui montrent que les écrans : diminue les facultés mentales (10%), diminue l'espérance de vie, pousse au tabagisme, provoque des grossesses non désirées...

En particulier, les écrans nuisent à l'enseignement (à l'école et ailleurs). Pour mémoriser une leçon il faut que cela coût un effort. Cela ne veut pas dire que l'on doit être maso. Jouer à un jeu de société ou de plein air demande de la concentration et de l'effort. Il y a au finale une récompense agréable.

Ces cours vous paraîtront difficiles. Ils demanderont de la concentration. Mais vous apporteront un savoir en contrepartie.

Vous pouvez consulter des conférences en ligne sur "TV Lobotomie" (site http://fls56.org/) ou sur "Atelier Éducation 2016-2017 - 1 - L'école et le numérique".

Le contenu des cours du **D**iplôme **U**niversitaire de **T**echnologie **Info**rmatique est décrit dans le **P**rogramme **P**édagogique **N**ational (http://www.iut-informatique.fr/docs/ppn/fr.pdf).

Pour ce module les textes prévoient : 15 h CM / 14 h TD / 16 h TP soit 45 h. Nous ne disposons que de 3 créneaux de 1h30 pendant 6 semaines. Nous ne pourrons donc faire que : 9 h CM / 9 h TD / 9 h TP soit 27 h. Vous devrez combler ce manque de **18 h** par un travail personnel (préparation de vos TP en particulier).

Le découpage se fera en 6 thèmes suivant la table 1.

CM	TD	TP
Ressource	fork/exec	shell
Fichiers	SF Unix	Python
Mémoire	allocation	Java
Tâches	synchronisation	Java
Entrée/Sortie	formatage	Java
Réseaux	jeu du "Romancier Scilof"	Java

Table 1 – Découpage en chapitres

2 Évaluation du module

Au delà de la notation, l'évaluation consiste à mesurer les nouvelles connaissances acquises au cours de ce module. Vous trouverez dans ce document un Questionnaire à Choix Multiple (QCM) qui ne sera pas noté. Vous remplirez ce QCM avant le premier cours, puis à nouveau après le dernier. Vous serez à même de juger de votre progression. Vous restituerez ce QCM au dernier cours pour réaliser une évaluation de groupe. En revanche d'autres QCM (un par chapitre) seront notés.

À noter que l'on vous distribuera les QCM en deux exemplaires :

- le 1er dans ce support à conserver (en annexe)
- le 2nd à remettre à l'enseignant en début de chaque TD (fourni en amphi la semaine précédente)

Pensez à recopier les réponses sur votre exemplaire de ce support avant de le rendre.

Il y aura également des évaluations chiffrées en contrôle continu :

- coef 1 : QCM en début de TD
- coef 1 : Assiduité en TD et TP (1 point si présent et à l'heure par créneau / 12)
- coef 1 : Régularité du travail (1 point par TP rendu / 6)
- coef 4 : 2 Comptes-rendus de TP
- Pénalité de pollution : -2 points par TP rendu via l'Internet (courriel, moodle . . .)
- Bonus écologique en cas de rendu papier :
  - +1 point pour les TP rendus en binôme.
  - +1 point pour les TP rendus en 2 colonnes et recto-verso (4 pages par feuille)
- Pénalité de non-respect des règles :
  - identité : -1 point par élément manquant (nom, prénom, groupe TD, identité du TP)
  - retards : -1 point si non rendu le jour même et -1 par semaine supplémentaire

Et un contrôle terminal :

- 4 pts : QCM sur le cours
- 8 pts : théorique (résolution de problème)
- 8 pts : pratique (écriture/correction de code)

La note du module est 1/3 contrôle continu et 2/3 contrôle terminal.

2.1 Calcul de QCM

Pour ceux qui apprécient la carotte et le bâton, disons que les bonnes réponses donnent des points positifs (ex 7 pts) et les mauvaises réponses des points négatifs (ex -3pt). Comme il serait aberrant de ne pas faire la différence entre quelqu'un qui s'abstient et quelqu'un qui fait exprès de mal répondre à toutes les questions, les notes sont ramenées à des valeurs positives. Voici la grille de calcul pour 20 questions :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
points	0	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
0	0	6,0	6,7	7,4	8,1	8,8	9,5	10,2	10,9	11,6	12,3	13,0	13,7	14,4	15,1	15,8	16,5	17,2	17,9	18,6	19,3	20,0
1	-3	5,7	6,4	7,1	7,8	8,5	9,2	9,9	10,6	11,3	12,0	12,7	13,4	14,1	14,8	15,5	16,2	16,9	17,6	18,3	19,0	
2	-6	5,4	6,1	6,8	7,5	8,2	8,9	9,6	10,3	11,0	11,7	12,4	13,1	13,8	14,5	15,2	15,9	16,6	17,3	18,0		
3	-9	5,1	5,8	6,5	7,2	7,9	8,6	9,3	10,0	10,7	11,4	12,1	12,8	13,5	14,2	14,9	15,6	16,3	17,0			
4	-12	4,8	5,5	6,2	6,9	7,6	8,3	9,0	9,7	10,4	11,1	11,8	12,5	13,2	13,9	14,6	15,3	16,0				
5	-15	4,5	5,2	5,9	6,6	7,3	8,0	8,7	9,4	10,1	10,8	11,5	12,2	12,9	13,6	14,3	15,0					
6	-18	4,2	4,9	5,6	6,3	7,0	7,7	8,4	9,1	9,8	10,5	11,2	11,9	12,6	13,3	14,0						
7	-21	3,9	4,6	5,3	6,0	6,7	7,4	8,1	8,8	9,5	10,2	10,9	11,6	12,3	13,0							
8	-24	3,6	4,3	5,0	5,7	6,4	7,1	7,8	8,5	9,2	9,9	10,6	11,3	12,0								
9	-27	3,3	4,0	4,7	5,4	6,1	6,8	7,5	8,2	8,9	9,6	10,3	11,0									
10	-30	3,0	3,7	4,4	5,1	5,8	6,5	7,2	7,9	8,6	9,3	10,0										
11	-33	2,7	3,4	4,1	4,8	5,5	6,2	6,9	7,6	8,3	9,0											
12	-36	2,4	3,1	3,8	4,5	5,2	5,9	6,6	7,3	8,0												
13	-39	2,1	2,8	3,5	4,2	4,9	5,6	6,3	7,0													
14	-42	1,8	2,5	3,2	3,9	4,6	5,3	6,0														
15	-45	1,5	2,2	2,9	3,6	4,3	5,0															
16	-48	1,2	1,9	2,6	3,3	4,0																
17	-51	0,9	1,6	2,3	3,0																	
18	-54	0,6	1,3	2,0																		
19	-57	0,3	1,0																			
20	-60	0,0																				

TABLE 2 – Grille d'évaluation de QCM

Dans le cas d'un QCM à 4 réponses, si vous répondez aléatoirement sur 4 réponses, vous aurez 1 bonne et 3 mauvaises. Ou sur 20, 5 bonnes et 15 mauvaises. Notez les valeurs particulières :

réponses	en points	note sur 20
20 mauvaises	-60	0
20 aléatoires	-10	5
pas de réponse	0	6
20 bonnes	140	20

TABLE 3 – Valeurs caractéristiques de QCM

Pour ceux qui veulent voir ce barème de façon positive, il suffit de dire que vous avez 1 point par bonne réponse et 0,3 point par abstention (les mauvaises réponses ne sont pas comptées). Le tableau est strictement le même.

Une réponse au hasard ne vous donne que 0,25 points (en moyenne 1 chance sur 4). Vous n'avez donc aucun intérêt à répondre au hasard. Car, à chaque question, vous avez plus de point à vous abstenir que de répondre au hasard. Le mieux est évidemment de donner la bonne réponse.

2.2 Calcul de pollution

La question revenant chaque année, je donne quelques éléments sur le surcoût de la pollution induite par l'utilisation de l'Internet quand vous rendez vos TP de cette manière.

Sachant qu'un groupe de 2nd année et constitué de 4 groupes TD de 24 étudiants. Sachant qu'il y a 6 TP à rendre, de 2 feuilles de papier (8 pages). Sachant que les TP sont conservés 10 ans.

Calculez le nombre de feuilles stockées.

Calculez la consommation énergétique consommée par ces feuilles pendant 10 ans. Il faut tenir compte des années bissextiles (un indice : 0 × 365 0 × 36 666 = 0).

Pour le rendu électronique, faites une estimation du poids en Ko des comptes-rendus (8 pages avec graphiques). Faites le même calcul de stockage sur 10 ans des rendus électroniques.

Calculez la quantité de disques durs nécessaire à acheter. Calculez la consommation électrique correspondante. Ajoutez les systèmes de sauvegarde et la redondance matérielle (n'oubliez pas que le papier est plus fiable et est insensible aux virus).

Donnez une estimation pour le passage à un enseignement "tout numérique" (on prendra 20% de la population française comme étant scolarisée). Donnez la consommation en nombre de réacteur**s** nucléaire**s** de 1 300 megas watt.

Vous n'aviez pas conscience de la pollution de l'Internet ? Vous n'aviez pas non plus conscience que le papier ne consommait pas d'énergie une fois imprimé ? Alors, vous allez apprendre bien d'autres choses dans ce cours.

Vous voici fin prêts à commencer ce cours...

3 Système d'exploitation

Dans une représentation d'un ordinateur en couches de niveau d'abstraction (physique, logique ...), le système d'exploitation pilote le matériel suivant les demandes applicatives.

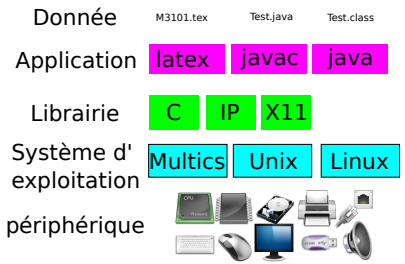


FIGURE 2 – Architecture mémoire

Définition 1 (système d'exploitation)**.** Un système d'exploitation (en anglais *Operating System*) permet d'exploiter (de tirer parti) des ressources d'une machine. Il est composé d'un ensemble d'automatismes que l'on peut déclencher pour faire fonctionner tous les organes de la machine.

Définition 2 (périphérique)**.** Un périphérique (en anglais *device*) est un composant matériel dédié à une fonction.

Définition 3 (pilote)**.** Un pilote (en anglais *driver*) est un composant logiciel communicant avec un périphérique suivant l'interface du constructeur (signaux électriques) et offrant toutes les fonctions disponibles pour les applications.

Définition 4 (gestion des droits)**.** Les systèmes d'exploitation multi-utilisateurs garantissent une gestion des droits. Ils mettent en place un moyen d'authentification (mot de passe sous Unix). Pour chaque ressource, il est spécifié : un propriétaire et quels utilisateurs peuvent y avoir accès.

Définition 5 (groupe d'utilisateurs)**.** Pour faciliter la gestion des droits, des profils génériques sont définis : des groupes (*/etc/group* sous Unix). Un mécanisme indique l'appartenance d'un utilisateur à un groupe (définitivement */etc/passwd* ou temporairement *newgrp* sous Unix).

Définition 6 (administrateur)**.** Les groupes définissent des niveaux d'accès (ou privilèges) : disque, réseaux, messagerie... Un administrateur dispose de tous les droits (*root* sous Unix).

Pour réaliser ses propres fonctions, le système d'exploitation s'utilise lui-même. Les instructions définissant le programme du pilote de la mémoire sont enregistrés dans cette même mémoire. Le temps de calcul du programme d'ordonnancement des tâches est alloué par ce même ordonnanceur de tâches.

Il existe un nombre limité de fonctionnalités élémentaires.

Définition 7 (noyau)**.** Le noyau (en anglais *kernel*) regroupe les fonctions élémentaires du système. C'est là que se réalise le contrôle d'accès et la répartition des ressources. Les processeurs peuvent fournir des instructions particulières pour offrir un mode spécifique d'exécution. Cela signifie que même avec les instructions assembleurs appropriées, vous ne pouvez pas entrer dans le noyau une fois le système lancé.

Voici une liste d'équivalence pour les termes anglophones.

- *device* : périphérique
- *disk* : disque
- *driver* : pilote
- *kernel* : noyau
- *keyboard* : clavier

- *login* : identifiant de connexion
- *mouse* : souris
- *password* : mot de passe
- *root* : administrateur
- *screen* : écran
- *user* : utilisateur

Quelques exemples historiques :

- 1960 :
  Premier ordinateur à lecteurs de cartes perforées (on y passe plus de temps à attendre qu'un périphérique termine son travail, qu'à exécuter des instructions).
- 1965 :
  Le MIT crée Multics (MULTiplexed Information and Computing Service). Système multi-tâches et multi-utilisateurs préemptif (durée maximum d'éxécution sans attendre de blocage d'un programme).
- 1969 :
  Dennis Ritchie des laboratoires Bell crée Unix pour lequel tous les périphériques sont considérés comme des fichiers. Les dates du système Unix commencent au 1/1/1970. Une généalogie des systèmes Unix se trouve sur https://fr.wikipedia.org/wiki/Berkeley_software_distribution.

  Il invente également un langage pour le développer : le langage C.

À noter que les systèmes d'exploitation sont souvent accompagnés d'un langage de commande propre. Pour Unix, le langage d'origine est le "Borne Shell". La variante la plus souvent utilisée aujourd'hui sous linux est "bash".

Parmi tous les systèmes lequel choisir ?

Chacun a sa particularité. Il est possible aujourd'hui de ne pas choisir.

Définition 8 (virtualisation)**.** La virtualisation permet de faire fonctionner plusieurs systèmes d'exploitation avec ses qualités sans nuire aux performances de la machine.

En effet, les processeurs actuels disposent d'un jeu d'instructions permettant de gérer leurs cohabitations (voir https://fr.wikipedia.org/wiki/Virtualisation).

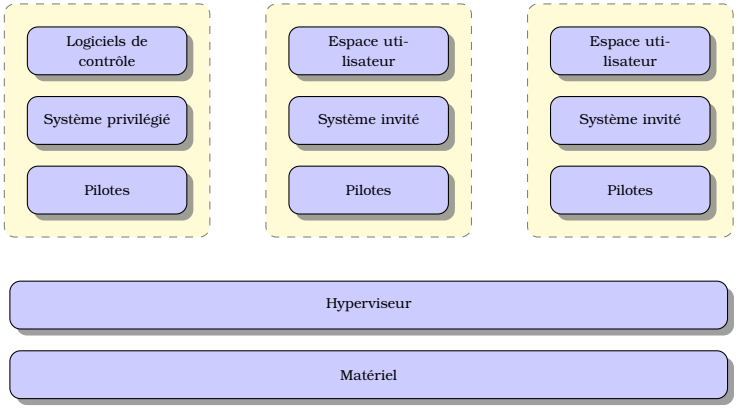


FIGURE 3 – Virtualisation

Un exemple particulier notable est le logiciel libre Docker. Il permet de lancer des applications dans des conteneurs qui inclu un système d'exploitation. Il est particulièrement efficace, car il s'appuie sur `LXC`, `cgroups` et le noyau Linux de la machine hôte. Les puristes diront qu'il ne s'agit pas d'une virtualisation puisqu'il ne permet pas l'excution de système autre que Linux.

FIGURE 4 – Logo docker

L'instatllation se fait de la manière suivante :

```
$ sudo bash

# add-apt-repository "deb [arch=amd64] https://download.
    docker.com/linux/debian bullseye stable"

# apt-get update

# apt-get install docker-ce

# systemctl status docker

# usermod -a -G docker myLogin
```

Listing 1 – Installation de Docker

Voici un exemple d'utilisation.

```
$ docker search debian

$ docker pull debian

$ docker images
07:45:10$ docker images
REPOSITORY                                          TAG                 IMAGE
    ID        CREATED         SIZE
```

```
debian                                                  latest
    fe3c5de03486   2 weeks ago    124MB

$ docker run -i -t debian /bin/bash
root@197238dbca93:/#
root@197238dbca93:/# apt-get update
Get:1 http://security.debian.org/debian-security bullseye-security InRelease [44.1
     kB]
...

root@197238dbca93:/# apt-get install -y procps
...

root@197238dbca93:/# ps -aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root            1  0.0  0.0   4092  3504 pts/0    Ss   17:46   0:00 /bin/bash
root          379  0.0  0.0   6700  2952 pts/0    R+   17:48   0:00 ps -aux

CTRL + P + Q

$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS         PORTS
    NAMES
197238dbca93   debian    "/bin/bash"   3 minutes ago   Up 3 minutes
    romantic_poincare

$
```

Listing 2 – Installation de Docker

Ce cours ne fera qu'effleurer la présentation des systèmes d'exploitation. Il y a tant de choses à dire ...

Deuxième partie

Partage des ressources

4 Ressources

4.1 Caractéristiques

Dans ce document, le terme *ressources* recouvre : soit des composants matériels, soit des sous éléments matériels que l'on manipule comme une abstraction identifiée par le système (fichier dans un disque, espace mémoire dans des composants électroniques, connexion via une carte réseau ...).

Concrètement des ressources sont des :

- processeurs (unité de calcul et de traitement)
- périphériques (connectés en interne à la carte mère ou en externe à l'unité centrale)
- fichiers (dans des périphériques de mémoire de masse)
- mémoires (gérées par des composants électroniques)
- canaux de communication (via un périphérique réseau)

Pour les systèmes d'exploitation de la famille Unix tout est fichier

- /proc/... : accès aux données des processus
- /dev/tty... : les écrans et claviers des terminaux
- /dev/sd... : les disques durs
- /dev/sr... : les lecteurs de disques optiques
- /dev/mem : la mémoire
- /dev/eth... : les cartes réseaux Ethernet
- y compris les...
  répertoires et fichiers.

Donc, dans notre cas, il s'agit principalement d'un partage de fichiers, même si le partage de fichiers posera des contraintes différentes suivant leur nature.

Dès que l'on utilise un système d'exploitation multi-tâches (voire multi-utilisateurs), se pose des questions d'accès concurrents aux ressources. Ils sont de différentes natures :

- la cohérence des données
- l'inter-blocage
- la vérification des droits
- le contrôle des limites

4.2 Cohérence des données

Pour illustrer le problème de cohérence des données, prenons l'exemple suivant. Le client d'une banque a un compte courant (CC) et un livret d'épargne (LE). Chaque mois, il met de côté 10% de son compte (tâche 2). Disons qu'aujourd'hui, chacun des deux comptes a un solde de 50 €. Exceptionnellement ce mois ci, il sait qu'il devra consommer 10 € de plus et choisit de faire un virement supplémentaire (tâche 1).

- tâche 1
  ① LE = LE - 10
  ② CC = CC + 10
- tâche 2
  ❶ tmp = 10%CC
  ❷ LE = LE + tmp
  ❸ tmp = 10%CC
  ❹ CC = CC - tmp

Intuitivement, nous imaginons que l'argent ne s'évapore pas lors de transfert de compte. Il y a donc un invariant :

- CC+LE = constante (100 € dans notre cas)

Une exécution séquentielle des deux virements peut donner le résultat suivant :

Tâche 1	LE	CC	CC+LE	tmp	Tâche 2
Solde	50	50	100		
① LE = LE-10	40				
② CC = CC+10		60			
fin	40	60	100		
				6	❶ tmp = 10%CC
	46				❷ LE = LE+tmp
				6	❸ tmp = 10%CC
		54			❹ CC = CC-tmp
	46	54	100		fin

L'invariant est respecté.

Cependant, lorsque le traitement est effectué de façon parallèle, l'imbrication des instructions pourrait donner le résultat suivant :

Tâche 1	LE	CC	CC+LE	tmp	Tâche 2
Solde	50	50	100		
				5	❶ tmp = 10%CC
① LE = LE-10	40				
② CC = CC+10		60			
	45				❷ LE = LE+tmp
				6	❸ tmp = 10%CC
		54			❹ CC = CC-tmp
	45	54	99		fin
fin	45	54	99		

Définition 9 (intégrité des données)**.** Un système d'exploitation doit garantir l'intégrité et la cohérence des données quel que soit l'ordre d'exécution des instructions.

Ce problème d'accès concurrent peut survenir avec tous les types de ressources. Par chance, cette situation n'arrive pas dans le cas de la mémoire vive, car le système d'exploitation isole l'espace d'adressage entre les processus au moment de leur exécution. Cependant, le cas est fréquent lorsque deux processus manipulent un même fichier. Par exemple, dans le cas de la modification simultanée de mots de passe (`/etc/passwd`) par plusieurs utilisateurs.

Un moyen de se prémunir de l'accès simultané à une même ressource est d'identifier les sections critiques et d'effectuer une réservation pour un usage exclusif en écriture.

Définition 10 (section critique)**.** En programmation parallèle, une section critique est une portion de code qui accède à une ressource qui ne devrait pas être accédé par plus d'une activité à la fois. Elle doit être protégée par un mécanisme de synchronisation : MUTEX, sémaphore, . . .

4.3 Inter-blocage

La réservation de ressources peut induire des inter-blocages. Pour l'illustrer, nous présentons l'exemple des 5 philosophes (voir https://fr.wikipedia.org/wiki/D%C3%AEner_des_philosophes).

Cinq philosophes sont assis autour d'une table ronde sur laquelle se trouvent cinq plats de spaghetti et cinq fourchettes. Chaque philosophe a devant lui un plat de spaghetti et il lui faut deux fourchettes pour le manger. Un philosophe passe son temps à manger et à penser. Quand il a faim, il tente de prendre deux fourchettes, l'une à sa gauche et l'autre à sa droite. S'il obtient les deux fourchettes, il mange pendant un temps, puis repose les fourchettes et ensuite se remet à penser. Comment permettre à chaque philosophe de se donner à ses activités (manger et penser) sans jamais être bloqué ou privé ?

FIGURE 5 – Dîner de philosophes

Nous comprenons vite que, si l'algorithme consiste à prendre la fourchette de gauche puis la fourchette de droite et que deux philosophes côte à côte décident de manger en même temps, l'un va être bloqué. Ce blocage peut assez vite se propager. Le cas extrême étant que les 5 philosophes aient leur fourchette gauche dans la main et attendent indéfiniment que celle de droite se libère.

Les phénomènes d'inter-blocage concernent toutes les ressources. Ce peut être le cas de la mémoire si des processus demandent plus de mémoire en cours d'exécution, ou cherchent à accéder à un fichier en cours d'utilisation par un autre.

Définition 11 (inter-blocage)**.** Un système d'exploitation doit garantir que les fonctions dont il a la charge (tel l'ordonnancement de processus) ne provoque pas d'inter-blocage.

4.4 Gestion des droits

Dans un système multi-utilisateur, les utilisateurs fictifs (*login*) sont :

- soit l'avatar d'un humain dans le mode réel,
- soit le représentant d'une fonction (*mail* : l'administrateur des courriels, *www-data* : l'administrateur de pages web. . . et *root* : l'administrateur suprême).

Les données ont un propriétaire : un utilisateur fictif responsable de leur gestion.

Les données peuvent être :

- communes (lisibles et modifiables par tous)
- publiques (lisibles par tous mais non modifiables)
- privées (uniquement accessibles par le propriétaire)

Dans tous les cas, un administrateur suprême a accès à vos données, sur votre machine ou sur des serveurs distants.

Il serait fastidieux de décrire toutes les combinaisons de droits entre tous les utilisateurs. Si le système compte n utilisateurs, il faudrait compter $n \times n$ descriptions de droits par ressource ! En conséquence, on définit des profils génériques : les groupes.

À titre d'exemple, Unix fixe la description des droits aux ressources à 3 domaines :

- le propriétaire
- le groupe propriétaire
- les autres

Les droits sont :

- **r** : lecture
- **w** : écriture
- **x** : exécution (pour les fichiers) / traverser (pour les répertoires)
- **s** : *suid* ou *sgid* (pour les fichiers) / nouveaux fichiers au nom du propriétaire de répertoire (pour les répertoires)
- **t** : utilisation fréquente (pour les fichiers) / suppression de fichier réservé au propriétaire (pour les répertoires)

Rappelons que **S** et **T** correspondent aux droits **s** et **t** sans la présence du droit **x**.

Droits

Un système d'exploitation multi-utilisateur doit fournir un mécanisme de gestion de droits.

4.5 Limitation d'accès

Une erreur de programmation ou une anomalie conjoncturelle peut engendrer une demande incessante de ressources. Par exemple, le programme suivant va provoquer une infinité de création de processus.

```
1 $0&
```

Listing 3 – Le shell de la mort

Le système va passer son temps à créer des processus et à les détruire. C'est une fonction qu'il sait faire rapidement mais qui doit être assurée par le noyau. Il ne pourra donc rien faire d'autre et ne sera pas interruptible.

Responsabilité

N'oubliez jamais que de grandes connaissances impliquent une grande sagesse.
(Pensez à sauvegarder votre travail avant de lancer une telle commande).

Vous constatez qu'un programme de 3 caractères seulement peut paralyser tout votre système d'exploitation. C'est gênant quand c'est un autre qui le lance pour vous nuire. C'est autant gênant quand c'est vous qui le faîte par erreur. Vous souhaitez légitimement que le système vous offre la pleine puissance de ses ressources. Dans le même temps, vous souhaitez qu'il soit vigilant et contrôle l'accaparement des ressources (y compris par vous). Il est donc possible de fixer des limites y compris pour soi-même.

La commande *quota* permet de fixer par utilisateur (ou par groupe) des limites d'utilisation de disque (limites molles et dures) ainsi qu'un délai de grâce.

De la même façon, il est possible de limiter d'autres ressources (nombre de processus, temps d'utilisation du processeur, nombre de fichiers ouverts, taille des programmes . . .) avec limits.conf (voir le manuel).

Limitation

Un système d'exploitation multi-tâches doit fournir un mécanisme permettant d'éviter l'accaparement des ressources par un processus.

5 Ordonnanceur

Il est fréquent en français de confondre le contenu avec le contenant (dire boire un verre au lieu de boire l'eau qu'il contient). Parce que leurs manipu-

lations seront différentes, il faut savoir distinguer les :

- programmes
- processus
- processeurs

Définition 12 (programme)**.** Un programme est une suite d'instructions imaginées pour réaliser un algorithme. Un programme a un nom (`/bin/bash`).

C'est un composant logiciel qui existe indépendamment de l'alimentation de la machine qui l'héberge. Un seul exemplaire d'un programme est nécessaire. Un programme n'a pas une date de début, de fin, ni de durée. Peut-être ne sera-t-il jamais exécuté.

Les programmes existent sous forme : de scripts qui sont interprétés par un interpréteur (bash, python, ...) qui le traduira en instructions d'un processeur ; ou de binaires (programmes compilés) pour être directement interprétés par le processeur.

Définition 13 (processus)**.** Un processus est l'exécution (une réalisation dans le temps) d'un programme. Un processus a un identifiant (exemple 1231).

Il peut y avoir simultanément plusieurs processus d'un même programme. Un processus débute à un instant et se termine (de préférence). Il comprend un environnement d'exécution, un état des mémoires, des canaux d'entrées/sorties et les positions d'avancement du programme.

Définition 14 (processeur)**.** Un processeur est un composant matériel qui interprète les instructions d'un programme suivant la progression gérée par le processus. En programmation classique, un processeur n'est normalement jamais désigné ni par l'utilisateur ni par le développeur. Ce peut être le cas en programmation d'algorithmes parallèles pour répartir les lieux d'exécution.

Dans un système multi-tâches, un processeur traitera les processus à tour de rôle donnant l'illusion qu'ils progressent en parallèle. Si un ordinateur possède n processeurs, il ne pourra y avoir au plus que n processus actifs en même temps (les autres seront en sommeil).

Dans une première approximation nous dirons que les cœurs sont comparables à des processeurs. Dans les faits, contrairement aux cœurs, 2 processeurs ne partageront pas nécessairement les mêmes horloges et bus de données.

Définition 15 (ordonnanceur)**.** L'ordonnanceur (en anglais *scheduler*) est un programme du système d'exploitation (exécuté par un processus), qui contrôle le déroulement des autres processus.

Dans le cas de systèmes d'exploitation à plusieurs processeurs (ou plusieurs cœurs), il peut y avoir physiquement un accès simultané à une même ressource.

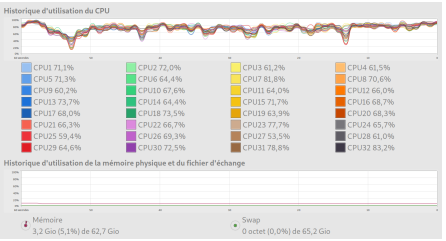


FIGURE 6 – Exemple multi-cœurs

6 Les processus Unix

Nous usons généralement de métaphores anthropomorphiques pour décrire les processus : père, fils, vie, mort, testament, ... Nous continuons donc dans ce style en ayant bien conscience que ce monde virtuel est sans pitié, sans féminin et où les processus passent leur temps à tuer leur progéniture.

Définition 16 (PID)**.** Les processus ont un identifiant, un entier strictement positif : le PID.

Au démarrage du système d'exploitation (après allumage de l'ordinateur), il y a un premier processus "**init**" de PID 1. Autrefois, fils de "**swap**" aujourd'hui fils de lui-même. La métaphore anthropomorphique a ses limites.

Définition 17 (fork)**.** Les processus sont créés par clônage. L'activité se sépare en 2 (comme une fourche en forme de **Y**) sous l'action d'une fonction du noyau : "**fork**". Seul le PID diffère entre le père et le fils.

En théorie, le processus est recopié en mémoire. En pratique, une optimisation évite cette opération, car la plupart du temps le programme associé au processus est rejeté (remplacé) immédiatement après la création du clone.

Un processus père peut avoir autant de processus fils que nécessaire (dans la limite de la capacité du système).

Une tâche (ou opération au sens commun) peut nécessiter la réalisation de sous-tâches (ou opérations intermédiaires). Il faut attendre la fin des sous-tâches (ou sous opérations) pour reprendre la tâche initiale. De la même façon, un processus créera des processus fils pour la réalisation de son but. Il attendra la fin des sous-buts (processus fils) pour poursuivre son but principal.

Dans ce contexte, nous comprenons la logique des ancêtres qui vivent plus longtemps que leurs descendances. Dans le monde réel, l'évolution des générations futures n'est possible que par la mort des générations précédentes.

Lorsqu'une tâche est abandonnée, ses sous-tâches n'ont plus de raison d'être. De même, lorsqu'un processus est tué, tous les processus fils reçoivent le signal de se terminer.

Un processus se termine parce qu'il :

- a terminé son programme
- met un terme à son exécution
- a reçu un signal lui demandant de se "suicider"

La demande de terminaison d'un processus s'effectue par l'envoi d'un signal qui précise la motivation de cette fin. En voici quelques exemples :

- `QUIT` (^C) : arrêt de l'utilisateur
- `HUP` : rupture de connexion
- `TERM` : kill standard
- `STOP` (^Z) : fige le processus et `CONT` réactive le processus

Définition 18 (kill)**.** En shell, c'est la commande "`kill`" qui permet d'envoyer un signal de demande de fin de processus. Exemple : `kill -TERM $(pidof emacs)`

Un programme peut prévoir un détournement du signal pour soit l'ignorer, soit réaliser une action. C'est utile pour demander confirmation à l'utilisateur avant de terminer réellement un processus. Ce moyen est contourné pour communiquer entre processus.

Définition 19 (trap)**.** En shell, c'est la commande "`trap`" qui permet d'ignorer ou de détourner un envoi de signal. Exemple : `trap "rm /tmp/resa-$$"0 1 2 3 15`.

Cependant, il existe un motif qui ne peut être détourné : le signal "`KILL`" ou "`9`".

Il est également possible de détacher un processus de sa lignée pour ne pas recevoir une demande de terminaison à la mort de son père. Il suffit d'ignorer le signal "`HUP`". À la mort de son père, le processus est adopté par le père de tous les processus : "`init`". Il n'est plus attaché à un terminal et devient un démon ("*deamond*").

L'utilisation des signaux est en réalité généralisée comme un moyen de communication entre processus. Un processus peut ainsi signaler la présence de données ou tout autre information de synchronisation simple.

Il existe un autre moyen de communication spécifique au moment de la mort d'un processus pour délivrer une sorte de "testament" vers son créateur. Évidemment, sous Unix seuls les fils délivrent un testament vers leurs ancêtres. L'information est limitée à la valeur d'un entier. Cette valeur est en général le motif de terminaison du fils, 0 indique que tout s'est bien passé, sinon la valeur correspond à un code d'erreur. Attention : la convention est inverse de celle du langage C.

- en C : 0 = false / autres valeurs = vrai
- en shell : 0 = ok / positif = nombre de résultats / négatif = code d'erreur

Définition 20 (exit)**.** En shell, c'est la commande "`exit`" qui permet de se suicider et d'envoyer un testament. Exemple : `exit -1`.

Un père n'est pas toujours dans un état compatible avec la lecture d'un testament. Un processus fils mort restera dans la table des processus tant que son père n'aura pas lu son testament. Il est à la fois vivant (dans la table des processus) et mort (exécution terminée). On l'appelle dans ce cas : défunt (en anglais *defunct*) ou zombie. Comme "`init`" est toujours à l'écoute des testaments de ses descendants (même s'il ne les lit pas), un démon (rattaché à "`init`") ne peut jamais devenir zombie.

Définition 21 (wait)**.** En shell, c'est la commande "`wait`" qui permet d'attendre la mort d'un fils et retourne son testament. Exemple : `wait $!`.

Le mécanisme de "fork" permet de créer une multitude de processus. Le seul problème est qu'ils seront tous identiques (des clones). Il est nécessaire de recourir à un autre moyen pour les différencier.

Définition 22 (exec)**.** Le programme d'un processus peut être changé au cours de l'exécution par substitution avec un autre programme. L'activité précédente est alors arrêtée, toutes les données sont oubliées (sauf les variables d'environnement et les canaux de communication) sous l'action d'une fonction du noyau : "**exec**". La commande échoue s'il n'existe pas de programme exécutable indiqué en paramètre (ou que certaines limites sont atteintes).

Si l'appel réussit, l'instruction suivant la commande exec ne sera jamais exécutée.

Il ne faut pas confondre, dans des langages comme *C* ou *python*, les fonctions "`exec`" et "`system`". La fonction "`system`" n'est que la composition des fonctions "`fork`", "`exec`" et "`wait`". Elle permet de cloner un processus, de différencier le père et le fils pour qu'ils réalisent des actions différentes :

- le fils va substituer son programme avec le nom fourni en argument.
- le père va attendre la fin du fils.

L'échange d'informations entre processus peut donc se faire par héritage des variables d'environnement (au moment du clônage et avant une substitution de programme), envoi de signaux ou envoi de testament.

Il est possible de communiquer les entrées/sorties standards. Au moment de la création de ses fils, un processus peut modifier après clônage les entrées/-sorties (avant une éventuelle substitution). En *shell*, voici quelques exemples de modification :

- `commande > file` redirige la sortie standard sur un fichier
- `commande 2> file`
- `commande < file`
- `commande1 | commande2`

Il existe d'autres redirections possibles que nous ne développerons pas dans ce document.

Il est également possible d'utiliser des fichiers en mode file (en anglais *fifo* pour *first in first out*). Ils sont aussi appelés pipe nommé (pipe pour tuyau, en anglais *named pipe*).

Définition 23 (tuyau ou en anglais *pipe*)**.** Un "*pipe*" est un canal de communication en mode file (en anglais *fifo*)

Pour créer une file on peut utiliser indifféremment l'une ou l'autre des commandes suivantes :

- `mkfifo maFile`
- `mknod maFile p`

Nous utiliserons enfin des sémaphores nommés avec les fonctions suivantes :

- initialisation :
```
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
```
- suppression :
```
int sem_unlink(const char *name);
```
- Puis-je ? :
```
int sem_post(sem_t *sem);
```
- Vas-y ! :
```
int sem_post(sem_t *sem);
```

Pour vérifier l'existence d'un sémaphore nommé, il faut regarder le contenu du répertoire : `/dev/shm/`.

Nous donnons les programmes utilisables en shell en annexe à la fin de ce chapitre.

En plus de ce que nous venons de décrire (fichiers, signaux, pipes nommés), Le noyau offre d'autres moyens de communication entre processus : les IPC pour *Inter-Process Communication*. L'avantage de passer par le noyau est que le système a la parfaite connaissance de la dépendance entre processus. Il sait donc quel processus est en attente de l'évènement d'un autre et pourra le laisser dormir dans la table des processus jusqu'à ce que les conditions soient réunies pour l'activité. Les processus ne sont donc pas en attente active.

7 Communication inter processus

La commande "ipcs" fournit la liste des ressources exclusivement réservées pour la communication entre processus :

- sémaphore
- mémoire partagée
- file de message

Toutes ces ressources sont sous le contrôle de droits d'accès des fichiers. Cela signifie qu'elles ont un propriétaire et un groupe. Il est possible de préciser les droits de lecture et d'écriture de chacun. Pour pouvoir être désignées sans ambiguïté entre des processus qui n'ont pas de lien de parenté (via "fork"), elles sont associées à des clefs.

7.1 Les sémaphores

Les sémaphores sont indispensables pour éviter des attentes actives.

- semget crée ou récupère un bloc de sémaphore.
- semop applique les opérations P et V sur des sémaphores.
- semctl réalise des opérations de contrôle (statistiques, droits).

7.2 Les mémoires partagées

La mémoire partagée est sous le contrôle d'un utilisateur et peut être superposée à celle utilisée dans un processus pour des variables. Ainsi, la modification d'un entier dans un processus écrit en C peut être instantanément liée à une autre variable entière d'un processus écrit en python.

- shmget crée ou récupère une mémoire partagée
- shmat attache une mémoire partagée à un processus
- shmdt détache une mémoire partagée d'un processus
- shmctl réalise des opérations de contrôle.

7.3 Les files de messages

Les files de messages fonctionnent en fifo (premier arrivé, premier servi). Il y a la garantie que les messages n'interfèrent pas. Cela peut être utilisé pour envoyer des données au fil de l'eau (provenant de capteur) ou des commandes (d'un système graphique).

- msgget crée ou récupère une file de messages.
- msgsnd ajoute un message dans la file.
- msgrcv retire un message de la file.
- msgctl réalise des opérations de contrôle.

8 Les commandes shell

Voici une liste restreinte de commandes shell dans le contexte de ce chapitre que nous pourrons être amenés à utiliser en TD et TP. Il est vivement conseillé de compléter ces explications avec la commande "man" ou de les lancer avec l'argument "–help".

8.1 Liste des processus

Commandes concernant les processus :

- "ps" permet de voir l'ensemble de ses propres processus ou de ceux des autres. Les liens de parenté sont décrits et permettent de recréer la généalogie des processus. Il est également précisé le temps passé depuis sa création en distinguant celui passé dans le noyau. On peut également voir les raisons d'une mise en sommeil (attente d'entrée/sortie).
- "pstree" Une version simplifiée, mais plus visuelle.

8.2 Les signaux

Voici un rappel des commandes citées dans ce chapitre :

- "kill" envoie un signal.
- "nohup" neutralise la réception du signal "hangup" pour réaliser un "démon".
- "trap" détourne ou ignore la réception de signaux
- "exit" écrit un testament pour son père avant un suicide.
- "wait" attend le testament d'un fils.

8.3 L'arrière-plan

Il est possible de jouer à faire passer les processus d'un terminal en avant-plan ou en arrière-plan. Les commandes n'ont de sens que pour les shells en mode interactif :

- & lance une commande en arrière-plan
- "jobs" liste tous les processus en arrière-plan ou interrompu.
- ^Z pause du processus en avant plan (correspond au signal "STOP")
- "bg" mise en arrière-plan (correspond au signal "CONT")
- "fg" mise en avant plan (donc avec perte du prompt) (correspond au signal "CONT" et wait)

8.4 Les variables

Pour visualiser les différents processus et leurs testaments, nous utiliserons les variables suivantes :

- $$ *(PID)* du shell initial
- $BASHPID PID réel de la partie de shell concerné
- $PPID PID du père
- $! PID du dernier fils lancé en arrière-plan
- $? dernier testament reçu à la dernière terminaison d'un fils lancé en arrière-plan
- $(cmd arg) substitut le texte par le résultat de la commande

8.5 La syntaxe

Voici enfin quelques rappels sur la syntaxe pour regrouper des commandes ou les ordonner séquentiellement.

- {} regroupe des commandes shell en tentant de ne pas créer de fils.
- () crée explicitement un shell fils pour exécuter une liste de commandes
- ; séparateur séquentiel de commandes (toutes les commandes sont censées être exécutées et une erreur interrompt le shell)
- || permet de lancer séquentiellement des commandes. La première est exécutée systématiquement. Les suivantes ne seront lancées que si une précédente échoue.
- && permet également de lancer séquentiellement des commandes. Toutes les commandes sont exécutées tant qu'elles réussissent.

9 Annexe : code source

Il n'existe pas de commande shell permettant la manipulation directe de sémaphores. En revanche, il existe une librairie C qui donne accès aux fonctions du noyau. Voici quelques programmes qui comblent ce manque.

Les programmes sont à compiler de la façon suivante :

```
#!/bin/bash
# (c) F. Merciol
# Plus d'informations sur http://m3101.merciol.fr
for file in mkNamedSem rmNamedSem namedSemP namedSemV
do
    gcc  ${file}.c  -lpthread -o ${file}
done
```

Listing 4 – Compilation des outils sémaphores

9.1 mkNamedSem

```
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>

void usage (char *prog) {
  fprintf (stderr, "Usage: %s semName initialValue\n", prog);
  exit (-1);
}

int main (int argc, char** argv, char** envp) {

  if (argc != 3)
    usage (argv [0]);

  int value = atoi (argv [2]);
  sem_t *sem;
  if ((sem = sem_open (argv[1], O_CREAT, S_IRUSR|S_IWUSR, value)) == SEM_FAILED) {
    perror ("can't create semaphore");
    exit (1);
  }
}
```

Listing 5 – Crée un sémaphore nommé

9.2 rmNamedSem

```
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#include <semaphore.h>

void usage (char *prog) {
  fprintf (stderr, "Usage: %s semName\n", prog);
  exit (-1);
}

int main (int argc, char** argv, char** envp) {

  if (argc != 2)
    usage (argv [0]);

  if (sem_unlink (argv[1])) {
    perror ("can't remove semaphore");
    exit(1);
  }
}
```

Listing 6 – Supprime un sémaphore nommé

9.3 namedSemP

```c
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#include <semaphore.h>

void usage (char *prog) {
  fprintf (stderr, "Usage: %s semName\n", prog);
  exit (-1);
}

int main (int argc, char** argv, char** envp) {

  if (argc != 2)
    usage (argv [0]);

  sem_t *sem;
  if ((sem = sem_open (argv[1], 0)) == SEM_FAILED) {
    perror ("can't find semaphore");
    exit (1);
  }

  if (sem_wait (sem)) {
    perror ("can't P");
    exit (1);
  }
}
```

Listing 7 – Puis-je ? sur un sémaphore nommé

9.4 namedSemV

```c
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#include <semaphore.h>

void usage (char *prog) {
  fprintf (stderr, "Usage: %s semName\n", prog);
  exit (-1);
}

int main (int argc, char** argv, char** envp) {

  if (argc != 2)
    usage (argv [0]);

  sem_t *sem;
  if ((sem = sem_open (argv[1], 0)) == SEM_FAILED) {
    perror ("can't find semaphore");
    exit (1);
  }

  if (sem_post (sem)) {
    perror ("can't V");
    exit (1);
  }
}
```

Listing 8 – Vas-y ! sur un sémaphore nommé

10 TD1 : Ressources

Dès votre arrivée en TD, pensez à déposer votre QCM rempli à votre encadrant. La partie à rendre est la dernière page de votre document (à découpe suivant les pointillés).

Après l'appel, c'est l'occasion de poser les questions concernant des éléments non compris du cours. Pensez à recopier le corrigé du QCM sur votre document (partie à concerver à la page précédante). Vous pourrez l'utiliser le jour de l'examen. Une page de notes personnelles est à votre disposition en fin de TD.

10.1 Compréhension shell

Les questions shell du TD et TP se font en *bash* (pour "*Bourne Again SHell*"). L'objet de cette partie est de rappeler la syntaxe du shell. Donnez l'effet des commandes et expliquez les excercices suivants.

Question :

```
echo $$ ${BASHPID} ; (echo $$ ${BASHPID})
```

Listing – PID

Réponse : ✍

Question :

```
echo $$ A-${BASHPID}
{ echo B-${BASHPID}; }
( echo C-${BASHPID} )
{ echo D-${BASHPID}; } | { read x ; echo ${x} D2-${BASHPID}; }
( echo E-${BASHPID} ) | ( read x ; echo ${x} E2-${BASHPID} )
```

Listing – pipe

Réponse : ✍

Question :

```
ls -d . && ls fichierInconnu ;  ls -d . || ls fichierInconnu
```

Listing – opérateurs

Réponse : ✍

Question :

```
echo A-${BASHPID} &
echo B-${BASHPID}
```

Listing – à table

Réponse : ✍

Question :

```
(echo A-${BASHPID}; exit 0; echo B-${BASHPID})
echo C-${BASHPID}; exit 0; echo D-${BASHPID}
```

Listing – la fin

Réponse : ✍

Question :

```
mkfifo pipe
(nohup sleep 30 & echo $! > pipe)&
read pid < pipe
rm pipe

wait
echo "Maintenant le pere de $pid est init"
pstree -sp $pid
ps -lp $pid

rm nohup.out
```

Listing – détaché

Réponse : ✍

Question :

```
mkfifo pipe
(sleep 1 & echo $! > pipe; exec sleep 30) &
read pid < pipe
rm pipe

ps -lp $pid
sleep 1
echo "$pid restera zombie pendant encore 29 seconde"
pstree -sp $pid
ps -lp $pid
wait
```

Listing – mort ou vivant ?

Réponse : ✍

10.2 Reproduction shell

Dans cette partie, il faut écrire un script shell qui correspond à la quetion.

Question : Ecrivez un programme qui crée une cascade de 4 processus (un petit-petit-fils), le dernier affiche la chaîne de processus.

Résultat attendu :
init(1)—konsole(2236)—bash(2255)—td1-4inc.sh(3462)
—td1-4inc.sh(3463)—td1-4inc.sh(3464)—td1-4inc.sh(3465)—td1-4inc.sh(3466)—
pstree(3467)

Réponse : ✍

Question : Ecrivez un programme qui crée 2 fils créant 2 petit-fils.

Réponse : ✍

Question : Créez un fils F1 qui crée un fils F2 et F3. F1 et F3 attendent un certain temps et retournent un code différent. F1 attend les 2 fils et affiche les codes de retour.

Réponse : ✍

10.3 Compréhension de synchronisation

Nous utiliserons des sémaphores avec les fonctions suivantes (programmes fournis) :

- mkNamedSem nomSemaphore valeurInitiale : création du sémaphore
- namedSemP nomSemaphore : puis-je ?
- namedSemV nomSemaphore : vas-y !
- rmNamedSem nomSemaphore : suppression du sémaphore

En fonction de la valeur initiale, nous les utiliserons dans des cas différents :

- 1 sémaphore initialisé à 1 : exclusion mutuelle
- 1 sémaphore initialisé à N : limitation à N ressources simultanées
- 2 sémaphores initialisés à 0 : rendez-vous
- 2 sémaphores initialisés 1 à 0 l'autre à N : producteurs/consommateurs avec tampon de taille N
- 2 sémaphores initialisés à 0 et un compteur : lecteurs / écrivains
- N sémaphores initialisés à 0 sauf 1 : synchronisation en carrousel

Pour aider à comprendre et à mémoriser ces situations, nous vous proposons un jeu de rôles : vous êtes un processus, les boîtes vides matérialisent des sémaphores et les crayons matérialisent la valeur des compteurs.

La distribution des rôles se fait par des petits papiers. En tant que processus, vous comptez dans votre tête jusqu'à dix. Si à la fin du décompte le sémaphore est vide vous recommencez à compter. S'il y a un crayon, vous prenez le crayon, vous réalisez l'action (dire mot, voir sur le papier), vous remettez le crayon dans le sémaphore convenu.

Question : Un rendez-vous entre deux processus est défini comme

- les codes des 2 processus avant le rendez-vous doivent être complètement exécutés avant le rendez-vous
- les codes des 2 processus après le rendez-vous ne peuvent commencer avant que les 2^{nd} aient atteint le rendez-vous.

A vous d'imaginer comment réaliser un "rendez-vous".

Réponse : ✍

Question : Dans la situation de lecture et d'écriture, il peut y avoir plusieurs lecteurs simultanés sans que cela n'altère le texte. En revanche, toutes les écritures doivent se faire en mode exclusif. Il faut donc imaginer la succession de groupe en exclusion mutuelle :

- un groupe de lecteurs (la réservation du texte se fait dès la 1^{re} lecture et sa libération après la dernière lecture).
- un groupe d'écriture (un seul écrivain).

A vous d'imaginer comment réaliser un modèle "lecteurs/écrivains".

Réponse : ✍

10.4 Situation Mutex

Nous allons créer un fichier de mots de passe fictifs et des processus qui écrivent des lignes pour simuler le changement de mot de passe (/etc/-passwd).

Nous partons du source :

```
> passwd.txt
mkfifo pipe
for (( i=0 ; i - 20 ; i++ ))
do
    {
  echo -n "${BASHPID} " > pipe
  for m in mot de passe de ${i}
  do
      echo -n "${m} "
  done
  echo
    } >> passwd.txt&
done
sleep 1
read pids < pipe
rm pipe
wait ${pids}
cat passwd.txt
rm passwd.txt
```

Listing 9 – Question Mutex

Sans synchronisation, les textes des différentes modifications s'entrelacent comme cet exemple de résultat :

```
mot de passe de 14
mot de mot mot passe de de de passe passe 1
de de 12 9

mot de mot passe mot de de passe 10 de
de passe 7 de
11
mot de passe de 17
mot mot de passe de de passe mot 15 de
de 8
passe de 3
mot de passe de 19
mot de mot passe de de 6 passe
de 5
mot de passe de 16
```

Listing – Trace

Question : Ajoutez le code nécessaire en utilisant les fonctions mkNamedSem, namedSemV, namedSemP et rmNamedSem pour rendre l'accès en exclusion mutuelle.

Réponse : ✍

10.5 Situation limitation

Nous allons simuler l'accès à N processus simultanément à la même ressource. Imaginons que 4 serveurs Apache sont simultanément à l'écoute de requête sur une même carte réseau pour fluidifier le service.

Nous partons du source :

```
maxBG=10
maxTime=2
mkfifo pipe
for (( i=0 ; i - ${maxBG} ; i++ ))
do
    (
  echo -n "${BASHPID} " > pipe
  sleep 1
  for  (( j=0 ; j - ${maxTime} ; j++ ))
  do
      echo $(date +"%H:%M:%S") " ($i)"
      sleep 1
  done
    )&
done
sleep 1
read pids < pipe
rm pipe

wait ${pids}
```

Listing 10 – Question limit

Sans synchronisation, tous les processus créés accèdent à la ressource en même temps.

```
21:06:16  (6)
21:06:16  (4)
21:06:16  (3)
21:06:16  (5)
21:06:16  (9)
21:06:16  (1)
21:06:16  (8)
21:06:16  (7)
21:06:16  (2)
21:06:16  (0)
21:06:17  (3)
21:06:17  (9)
21:06:17  (6)
21:06:17  (2)
21:06:17  (1)
21:06:17  (4)
21:06:17  (8)
21:06:17  (7)
21:06:17  (0)
21:06:17  (5)
```

Listing – Trace

Question : Ajoutez le code nécessaire en utilisant les fonctions mkNamedSem, namedSemV, namedSemP et rmNamedSem pour limiter à 4 les serveurs actifs.

Réponse : ✍

10.6 Situation RDV

Nous rappelons ici le principe déjà donné de la définition d'un rendez-vous :

- les codes des 2 processus avant le rendez-vous doivent être complétement exécutés avant le rendez-vous
- les codes des 2 processus après le rendez-vous ne peuvent commencer avant que les 2^{nd} aient atteint le rendez-vous.

Nous partons du source :

```
mkfifo pipe
(
    echo -n "${BASHPID} " > pipe
    sleep 1
    echo "avant RDV (A)"
    #RDV
    echo "apres RDV (A)"
)&
(
    echo -n "${BASHPID} " > pipe
    sleep 1
    echo "avant RDV (B)"
    #RDV
    echo "apres RDV (B)"
)&

sleep 1
read pids < pipe
rm pipe

wait ${pids}
```

Listing 11 – Question RDV

Sans synchronisation, un processus peut exécuter une instruction après le rendez-vous alors que l'autre processus ne l'a pas atteint.

```
avant RDV (B)
apres RDV (B)
avant RDV (A)
apres RDV (A)
```

Listing – Trace

Question : Ajoutez le code nécessaire en utilisant les fonctions mkNamedSem, namedSemV, namedSemP et rmNamedSem pour faire respecter le rendez-vous.

Réponse : ✍

10.7 Situation producteurs/consommateurs

Dans ce cas, nous supposons un tampon limité à un maximum de cases. Les producteurs remplissent les cases, les consommateurs les vident. La difficulté est de contrôler qu'on ne remplit pas une case déjà occupée ou de vider une case qui ne contient rien.

Nous partons du source :

```
maxP=2
maxC=2
maxTime=2
mkfifo pipe
for (( i=0 ; i - ${maxP} ; i++ ))
do
    (
  echo -n "${BASHPID} " > pipe
  sleep 1
  for  (( j=0 ; j - ${maxTime} ; j++ ))
  do
      echo $(date +"%H:%M:%S") " je produis (p$i)"
      sleep 1
  done
    )&
done
for (( i=0 ; i - ${maxC} ; i++ ))
do
    (
  echo -n "${BASHPID} " > pipe
  sleep 1
  for  (( j=0 ; j - ${maxTime} ; j++ ))
  do
      echo $(date +"%H:%M:%S") " je consomme (c$i)"
      sleep 1
  done
    )&
done
sleep 1
read pids < pipe
rm pipe

wait ${pids}
```

Listing 12 – Question producteurs

Sans synchronisation, un processus peut consommer alors que rien n'est produit ou produire alors que les stocks débordent.

```
11:38:04  je produis (p1)
11:38:04  je consomme (c1)
11:38:04  je produis (p0)
11:38:04  je consomme (c0)
11:38:05  je produis (p1)
11:38:05  je consomme (c0)
11:38:05  je produis (p0)
11:38:05  je consomme (c1)
```

Listing – Trace

Question : Ajoutez le code nécessaire en utilisant les fonctions mkNamedSem, namedSemV, namedSemP et rmNamedSem pour gérer le problème des piles vides et des piles pleines.

Réponse : ✍

10.8 Situation Lecteurs/Ecrivains

Dans la situation de lecture et d'écriture, il peut y avoir plusieurs lectures simultanées sans que cela n'altère le texte. En revanche, toutes les écritures doivent se faire en mode exclusif.

Nous partons du source :

```
maxR=2
maxW=2
maxTime=2
rCountFileName="nbL.txt"
trap "rm ${rCountFileName}" 0 1 3 15

echo 0 > "${rCountFileName}"
incrL () {
    read nbL < "${rCountFileName}"
    nbL=$(expr "${nbL}" \+ 1)
    echo "${nbL}" > "${rCountFileName}"
}
decrL () {
    read nbL < "${rCountFileName}"
    nbL=$(expr "${nbL}" \- 1)
    echo "${nbL}" > "${rCountFileName}"
}

mkfifo pipe
for (( i=0 ; i - ${maxR} ; i++ ))
do
    (
  echo -n "${BASHPID} " > pipe
  sleep 1
  incrL
  for  (( j=0 ; j - ${maxTime} ; j++ ))
  do
      echo $(date +"%H:%M:%S") " je lis (r$i)"
      sleep 1
  done
  decrL
    )&
done
for (( i=0 ; i - ${maxW} ; i++ ))
do
    (
  echo -n "${BASHPID} " > pipe
  sleep 1
  for  (( j=0 ; j - ${maxTime} ; j++ ))
  do
      echo $(date +"%H:%M:%S") " j'ecris (w$i)"
      sleep 1
  done
    )&
done
sleep 1
read pids < pipe
rm pipe

wait ${pids}
exit # nettoyage dans le trap
rm "${rCountFileName}"
```

Listing 13 – Question lecteurs/écrivains

Sans synchronisation, un processus peut lire ou écrire pendant qu'un autre est déjà en train d'écrire.

```
09:52:56  j'ecris (w0)
09:52:56  j'ecris (w1)
09:52:56  je lis (c0)
09:52:56  je lis (c1)
09:52:57  j'ecris (w0)
09:52:57  j'ecris (w1)
09:52:57  je lis (c0)
09:52:57  je lis (c1)
```

Listing – Trace

Question : Ajoutez le code nécessaire en utilisant les fonctions mkNamedSem, namedSemV, namedSemP et rmNamedSem pour exclussion mutuelle des écritures, mais permettre la lecture simultanée.

Réponse : ✍

10.9 Situation carrousel

Dans cette dernière situation, on vérifie que chacun prend la main à tour de rôle dans un ordre prédéfini.

Nous partons du source :

```
maxTime=5
player () {
    echo -n "${BASHPID} " > pipe
    sleep 1
    for  (( i=0 ; i - ${maxTime} ; i++ ))
    do
  echo -n "$1"
    done
}
```

```
mkfifo pipe
( player "Universite ")&
( player "de ")&
( player "Bretagne-")&
( player "Sud
")&

sleep 1
read pids < pipe
rm pipe

wait ${pids}
```

Listing 14 – Question carrousel

Sans synchronisation, les processus écrivent n'importe quand.

```
de Universite de Universite de Bretagne-de Universite de Sud
Sud
Sud
Sud
Sud
```

Listing – Trace

Question : Ajoutez le code nécessaire en utilisant les fonctions mkNamedSem, namedSemV, namedSemP et rmNamedSem pour que chacun travaille à son tour.

Réponse : ✍

Page de notes personnelles : ✍

11 TP1 : Ressources

- Votre compte-rendu est à rendre en fin de TP.
- Vous ne devez pas rendre un listing, mais un document répondant aux questions posées.
- Vous préparerez le compte-rendu à l'avance (copier-coller les questions) pour gagner du temps.
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois.
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (uniquement la fenêtre concernée).
- N'oubliez pas de rendre votre TP imprimé en 2 colonnes de texte par côté de papier et de faire le TP en binôme.

Ce TP traite de processus en parallèle pour lesquels on ne peut garantir l'ordre d'exécution. Il se peut que des résultats qui devraient être aléatoires soient ordonnés par hasard. Reproduisez plusieurs fois les tests pour être sûr qu'aucun biais ne soit apparu.

11.1 Pratiques interactives

Voici quelques excercices pour la prise en main du shell. Donnez les commandes en shell et décrire le résultat. Vous pourrez utiliser l'option "-update 1" pour visualiser une trotteuse.

Question : Lancez xclock en avant-plan puis fermez la fenêtre xclock.

Réponse : ✍

Question : Lancez xclock en arrière-plan puis fermez la fenêtre xclock.

Réponse : ✍

Question : Lancez xclock en avant-plan puis ^Z puis fermez la fenêtre xclock. Et ensuite saisissez "fg".

Réponse : ✍

Question : Lancez xclock en avant-plan. Dans un autre terminal, suspendez l'exécution de xclock

Réponse : ✍

Question : Créez un zombie avec le code fourni en exercice de TD.

Réponse : ✍

Question : Lancez une copie d'un répertoire r1 sur r2 dans 2 sous-shell avec la commande "tar".

Réponse : ✍

11.2 Mise en place des sémaphores

Question : Récupérez ou recopiez le code des programmes C : mkNamedSem, namedSemV, namedSemP et rmNamedSem. (Commment avez-vous fait ?)

Réponse : ✍

Question : Compilez les programmes. (Donnez les commandes)

Réponse : ✍

Question : Expliquez comment tester leur bon fonctionnement. (Explication et traces)

Réponse : ✍

11.3 Application sur limitation de ressources

Question : Récupérez ou recopiez le code du TD. (Comment . . .)

Réponse : ✍

Question : Nous voulons limiter à 4 le nombre de fils simultanés. Expliquez la démarche (nombre, rôle et initialisation des sémaphores).

Réponse : ✍

Question : Expliquez les endroits de synchronisation (avec la place des “P” et “V”).

Réponse : ✍

Question : Donnez le code.

Réponse : ✍

Question : Expliquez comment valider que la solution est juste.

Réponse : ✍

Question : Donnez la trace de la validation.

Réponse : ✍

11.4 Application au carrousel

Question : Récupérez ou recopiez le code du TD.

Réponse : ✍

Question : Expliquez la démarche (nombre, rôle et initialisation des sémaphores).

Réponse : ✍

Question : Expliquez les endroits de synchronisation (avec la place des “P” et “V”).

Réponse : ✍

Question : Donnez le code.

Réponse : ✍

Question : Expliquez comment valider que la solution est juste.

Réponse : ✍

Question : Donnez la trace de la validation.

Réponse : ✍

11.5 Application lecteurs/écrivains

Question : Récupérez ou recopiez le code du TD.

Réponse : ✍

Question : Expliquez la démarche (nombre, rôle et initialisation des sémaphores).

Réponse : ✍

Question : Expliquez les endroits de synchronisation (avec la place des “P” et “V”).

Réponse : ✍

Question : Donnez le code.

Réponse : ✍

Question : Expliquez comment valider que la solution est juste.

Réponse : ✍

Question : Donnez la trace de la validation.

Réponse : ✍

Troisième partie

Système de gestion de fichiers

Les premiers systèmes de gestion de fichiers sont apparus dans les années 60 (voir https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_fichiers). Il y en a aujourd'hui un nombre important. Chaque système d'exploitation cherche souvent à définir sa gestion de ses fichiers en fonction de ses caractéristiques propres.

12 Besoin de fichiers

Un Système de Gestion de Fichiers (SGF) répond à plusieurs besoins :

- désignation compréhensible par un humain (utilisation de l'alphabet)
- hiérarchisation des informations (arbre, voire graphe)
- typage des informations (extension et/ou étiquette embarquée)
- intégrité (droits d'écriture)
- confidentialité (droits de lecture)
- fiabilité (redondance des données, stabilité du système)
- disponibilité (réplication en réseau)

Il n'est pas toujours nécessaire de mettre en place toutes ces caractéristiques en fonction de l'usage du système d'exploitation. À quoi sert de définir un système complexe de propriétés et de droits d'accès quand il n'y a qu'un seul utilisateur ? C'est le cas pour un enregistreur numérique de film de salon, un appareil photo ou un appareil électroménager.

À quoi sert d'utiliser des lettres de l'alphabet pour désigner des fichiers quand les noms donnés automatiquement sont des nombres ? De nouveau, c'est le cas d'enregistreurs automatiques, qui n'ont pas le temps d'interagir avec l'utilisateur et vont donner un nom en fonction de la date ou d'un numéro d'ordre.

12.1 Désignation

Dans la mesure où un utilisateur doit désigner des fichiers, on préférera utiliser des symboles signifiants. Il vaut mieux désigner le répertoire d'un utilisateur par son nom ("jean.martin") plutôt que par "e42".

Les systèmes peuvent utiliser différents alphabets (pour nous le latin accentué). Parfois, ils permettent l'utilisation des majuscules et des minuscules. Il reste cependant des caractères interdits qui sont utilisés dans la syntaxe du langage de commande du système.

Certains caractères spéciaux sont réservés comme :

- ? remplace un caractère
- * remplace plusieurs caractères
- / séparateur de chemin
- $ nom de variable
- & lancement en arrière-plan
- ...

En règle générale, évitez tous les caractères de ponctuation. Par ailleurs, évitez les caractères accentués qui ne seront pas nécessairement encodés de la même manière d'un système à l'autre et compliquent la sauvegarde de répertoire ou l'échange entre systèmes.

Il existe également des systèmes de gestion de fichiers qui limitent la taille des noms (par exemple 8 caractères pour les noms et 3 pour les extensions).

Enfin, certains fichiers de configuration sont présents dans un contexte (un répertoire) mais nuisent à la lecture du contenu du répertoire. Sous Unix, les fichiers commençant par un point sont dit "cachés" (mais pas secrets). C'est le cas de "." le répertoire courant et ".." le répertoire précédent dans l'arbre. Par défaut, la commande "ls" ne les affiche pas. Il suffit d'ajouter l'option "-a" pour qu'ils apparaissent (`ls -a`).

12.2 Typage

Certains systèmes ont besoin de connaître la nature des données pour réaliser des opérations appropriées.

Principalement les systèmes distinguent la

nature des données contenues dans les fichiers par :

- L'extension. C'est une suite de caractères qui suit le dernier point dans le nom d'un fichier (.c pour un fichier source C, .out pour un exécutable ...).
- Un "nombre magique" (voir https://fr.wikipedia.org/wiki/Nombre_magique_%28programmation%29). C'est une suite de caractères, mais cette fois incluse en tête du fichier (#! pour les scripts, GIF pour les images GIF ...)

- Un répertoire dédié. Des données particulières sont regroupées de façon à ce que le système les identifie (des variables dans /proc/, des scripts dans cgi-bin/ ...).

12.3 Hiérarchie

La représentation hiérarchique est de loin la plus utilisée par les systèmes de gestion de fichiers.

Parfois elle se limite à 2 niveaux : les répertoires ne peuvent contenir que des fichiers. Le plus souvent sans limitation logique (il y a toujours une limitation physique de capacité de stockage).

- Les feuilles sont les fichiers (d'accord, sous Unix les répertoires sont également des fichiers).
- Les nœuds de l'arbre sont appelés : dossier, catalogue ou répertoire.

Un fichier est désigné par un chemin (en anglais *path*).

Le chemin débute par la racine de l'arbre sur la mémoire de masse concernée (il n'y a qu'une racine sous Unix "/").

On indique la succession des noms des nœuds (des répertoires) jusqu'au fichier inclus. Les noms sont séparés par un caractère indiquant le changement de répertoire ("/" sous Unix).

Définition 24 (arbre)**.** Un arbre est un "graphe" "orienté" "connexe" "sans cycle" avec une seule "racine".

En réalité, les systèmes de gestion de fichiers autorisent des liens (ou raccourcis) qui rompent le caractère sans cycle. Il peut donc exister plusieurs chemins pour désigner une même donnée. Il peut également exister des boucles lorsqu'un lien fait référence à un nœud plus proche de la racine.

Sous Unix on distingue 2 types de liens :

- physique (commande `ln`). Qui ne peuvent se faire qu'au sein d'un même volume (pour simplifier une même partition d'un disque dur). Nous verrons que dans ce cas, deux chemins désignent le même "inode" sous Unix.
- logique (commande `ln -s`). Qui peuvent se faire y compris avec des espaces disques non présents au moment de la création du lien. Il s'agit simplement d'un fichier qui contient un chemin absolu ou relatif et étiqueté comme étant un lien.

12.4 Intégrité

Définition 25 (intégrité)**.** L'intégrité est la propriété de garantir la cohérence, la fiabilité et la pertinence des données. Un moyen consiste à interdire la modification des fichiers.

Il serait logique de penser que les fichiers sont faits pour être écrits. Pourtant, il arrive que l'on souhaite les protéger de toute modification, y compris de la part de leurs propriétaires. C'est par exemple le cas des fichiers gérés par "subversion" (un logiciel de gestion de versions de développement). Il crée un répertoire ".svn" qui nous appartient, mais que l'on ne peut pas modifier.

Pour trouver les fichiers protégés sur votre compte vous pouvez lancer la commande :

```
1 find  ~ ! -perm /a+w -print
```

Listing 15 – Recherche de fichiers non modifiables

Vous trouverez par exemple les fichiers ".svn/entries"

Un autre moyen est de déléguer le traitement des données à un utilisateur fictif ("mail" pour la messagerie, "root" pour la gestion des mots de passe ...). Seul cet utilisateur pourra réaliser les modifications en votre nom sous le contrôle des règles mises en place. Par exemple, le fichier "/etc/passwd" ne peut être écrit directement par l'utilisateur.

On écrit alors des programmes qui font ce travail au nom de l'utilisateur fictif. Sous le système Unix, c'est le "bit s" qui indique cette faculté. Bien entendu, seul le propriétaire peut indiquer que l'on prend sa personnalité en lançant ce programme. Par exemple, lorsque vous exécutez le programme "/usr/bin/passwd", vous devenez "root"**!** (mais uniquement pour ce pourquoi le programme a été écrit ☺.)

```
1 $ ls -l /etc/passwd /usr/bin/passwd
-rw-r--r-- 1 root root  1548 sept.  1 08:32 /etc/passwd
-rwsr-xr-x 1 root root 42824 sept. 13  2012 /usr/bin/passwd
```

Listing 16 – Bit s

Il est indispensable que les programmes avec "bit s" ne puissent pas être modifiables par tout le monde.

12.5 Confidentialité

Certaines données ne doivent par être lues par d'autres utilisateurs. C'est le cas des mots de passe ou des clefs secrètes de communication. Par exemple, les clefs utilisées par SSH ("Secure Shell") ne doivent pas être lues par d'autres (fichier "~/.ssh/id_dsa").

Cette confidentialité est toute relative, puisque l'administrateur a toujours la possibilité de lire tous les fichiers. Si vous avez un secret à garder, le mieux est de ne jamais le numériser.

12.6 Fiabilité

Un système de gestion de fichiers doit être fiable, y compris en cas de coupure de courant.

Le système d'exploitation passe son temps à modifier des fichiers. On peut distinguer des différences de fréquences d'utilisation en fonction des parties de l'arbre des fichiers :

- Le "boot". Utilisé en lecture est nécessaire au démarrage, il évolue rarement. Mais, sa dégradation empêche tout démarrage.
- Le système. Utilisé principalement en lecture, il n'est mis à jour qu'au gré des changements de versions.
- Les programmes. Utilisés principalement en lecture, ils sont mis à jour au gré des changements de versions. Ils peuvent être conservés, même en cas de mise à jour du système, mais ne nécessitent pas de sauvegarde.
- Les données des utilisateurs. Fréquemment écrites, il faut leur adjoindre un système de sauvegarde.
- Les données des programmes. Contiennent principalement des informations de configurations que l'on peut souvent reconstruire.
- Le répertoire temporaire. En écriture permanente, il sera le premier à être en état instable en cas de coupure de courant.
- Le "swap". Zone d'extension de la mémoire vive. Il est donc inutile de la conserver d'un arrêt à l'autre.

C'est pourquoi, on peut conseiller de les séparer physiquement dans des partitions distinctes du disque :

- /boot
- /
- /opt
- /home
- /var
- /tmp
- swap

Le mieux est bien entendu de ne pas perdre les données en cas de crash du système. Un système de gestion de fichiers met en place des stratégies pour que les écritures se fassent de façon atomique. Par exemple, lors de la modification de fichier, le système conserve pendant un temps très court les deux versions du fichier. Il prévoit une modification minime (un pointeur, voire un bit) pour passer de l'un à l'autre. De cette façon, les fichiers restent dans un état stable.

Il existe également des mécanismes de journalisation, où le système décrit les modifications qu'il s'apprête à effectuer. En cas d'interruption, il reprend la même séquence (elle donnera le même résultat). Quand elle est terminée, il efface le journal.

12.7 Disponibilité

D'autres mécanismes, se fondant sur la réplication des disques, permettent de pallier à des défaillances, y compris de disque et de les remplacer en cours de fonctionnement du système. Cela nécessite de multiplier les disques durs. Les fichiers (ou parties de fichiers) se retrouvent répliqués sur plusieurs disques. En cas de panne, on arrête le disque défectueux (mais pas le système), on le remplace et le système recopie les éléments manquants pendant que l'utilisateur les modifie.

C'est le cas de regroupements redondants de disques indépendants (ou RAID pour "Redundant Array of Independent/Inexpensive Disks", voir https://fr.wikipedia.org/wiki/RAID_%28informatique%29)

- RAID 1 : Disques en miroir
- RAID 5 : Volume agrégé par bandes à parité répartie

Ces mécanismes se trouvent également dans des caches, comme avec le système NFS (pour "Network File System"), où les fichiers se trouvent sur un serveur distant. Ils sont copiés sur une machine locale et verrouillés. Les modifications sont renvoyées en fin d'utilisation.

13 Support physique

Nous venons de décrire les principes logiques de fonctionnement. Il faut les appliquer avec des contraintes physiques (voir https://fr.wikipedia.org/wiki/Disque_dur).

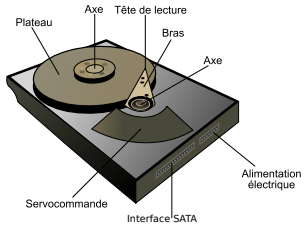


FIGURE 7 – Architecture d'un disque

La gestion de masse de données utilise encore des supports mobiles (tournant) car plus efficaces en rapport de leur coût. Les disques magnétiques et optiques (voire les deux) sont structurés en blocs de données (1 Ko ou 1/2 Ko). Ces blocs forment de façon logique une suite linéaire d'informations.

Ils doivent avoir un diamètre raisonnable. Les données sont situées sur des cercles concentriques, leurs circonférences augmentent avec le rayon. Il existe donc une relation entre vitesse de rotation, espacement des données et vitesse de lecture.

Les concepteurs se sont trouvés face aux choix suivants :

- soit avoir des vitesses (rotation et lecture) constantes, mais dans ce cas les bits sont plus longs et espacés à mesure que l'on s'éloigne du centre.
- soit distance régulière entre bits et lecture constante sur le périmètre du cercle et donc faire varier la vitesse de rotation des disques.
- soit enfin, distance régulière entre bits et rotation constante et avoir une vitesse de rotation constante mais une vitesse de lecture/écriture différente.

L'inertie mécanique rend difficile la variation instantanée des disques (déplacement des têtes du centre vers la périphérie). La modification de vitesse d'écriture rend aléatoire le temps d'accès aux données. Le choix a été fait de conserver la même quantité de données par rayon. C'est donc plus facile à gérer pour les informaticiens. C'est en revanche une perte considérable de support et donc une dépense inutile pour le propriétaire des données.

Un disque est donc divisé (voir https://fr.wikipedia.org/wiki/Cylindre/T%C3%AAte/Secteur)

en :

- Tête. Elle correspond à une surface d'écriture.
- Cylindre. Il correspond à l'empilement à rayons constants de cercle des différentes têtes.
- Secteur. Il correspond à un arc de cylindre (le périmètre d'une portion de camembert pour être imagé).

L'intersection d'une tête et d'un cylindre fournit une piste. L'intersection d'une piste et d'un secteur fournit un bloc de données.

À noter, qu'autrefois (époque des disquettes à une face), on parlait plutôt de piste et secteur. Il y avait confusion entre un bloc et un secteur (potentiellement sur plusieurs faces).

Définition 26 (bloc de données)**.** Le bloc est un composant élémentaire de conservation de données. Sur un support tournant, il est l'intersection d'une tête d'un cylindre et d'un secteur. Ils ont en général une taille de 1 024 ou 512 octets (1 Ko ou 1/2 Ko).

Vous pouvez déterminer la géométrie de votre disque avec les commandes `/sbin/fdisck` ou `gparted`.

Il y a deux façons de désigner un bloc :

- CHS (cylinder-head-sector) comme son nom l'indique
- LBA (Logical Block Addressing) utilisant le numéro du bloc dans une vision linéaire.

L'adressage LBA a remplacé celle du CHS, car elle généralise la désignation des blocs sur tous les supports (bandes, CD, clefs usb, . . .).

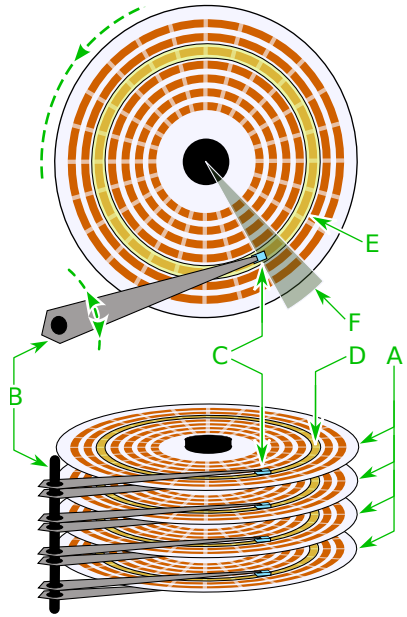


FIGURE 8 – Géométrie d'un disque. A : plateaux - B : bras - C : tête - D : cylindre - E : piste - F : secteur

14 Partition

Définition 27 (partition)**.** Une partition est une partie logique du disque dur.

Le fait de pouvoir découper le disque dur en différentes partitions permet d'isoler des arborescences de fichiers (comme vu précédemment). Cela permet également de disposer de plusieurs

systèmes de gestion de fichiers. C'est indispensable si l'on veut héberger plusieurs systèmes d'exploitation différents sur la même machine.

Les plus répandus sur les PC sont :

- FAT pour File Allocation Table utilisé par Windows
- NTFS pour New Technology File System utilisé par Windows à partir de 1993
- ext pour Extended File System Linux
- swap zone d'échange de la mémoire vive sous Unix.

Nous détaillerons le système de fichiers d'Unix par la suite.

15 Table de partition

Un système d'exploitation ne peut pas deviner combien, ni de quelle nature sont les partitions sur un disque. Il faut une structure qui organise le disque.

Définition 28 (partitionnement)**.** Le partitionnement est une structure qui permet de retrouver toutes les partitions d'un disque : soit par une table regroupant tous les descripteurs, soit par chaînage des descripteurs en tête des partitions.

Les plus répandus sont :

- MBR pour Master Boot Record (système propriétaire)
- GPT pour GUID Partition Table (récent et qui se généralise)

Pourquoi faire évoluer les structures anciennes ? Le MBR a pour défaut d'être limité à 4 partitions enregistrées dans une table. Une évolution a permis d'étendre cette table avec une partition appelée "étendue". On parle alors de 3 partitions primaires. Les autres étant des partitions logiques définies dans la partition étendue.

GPT lève cette contrainte. Il n'y a plus de différences entre les partitions qui se trouvent chaînées.

16 Structure d'un système de gestion de fichiers

À partir de maintenant, on considère un support comme une suite linéaire de blocs d'octets.

Un système de gestion de fichiers va devoir gérer des :

- données contenues dans les fichiers. Elles seront découpées pour être enregistrées dans des unités d'allocation
- méta-données qui décriront l'organisation du système de fichiers.
  - la table d'implantation des blocs du fichier
  - le descripteur de fichiers
  - les catalogues (répertoires) représentant l'arborescence de fichiers
  - la table d'allocation des blocs disponibles sur le disque.

16.1 Unité d'allocation

Un fichier ne peut être écrit directement sur le disque. Il faut le découper en blocs qui ne correspondent pas nécessairement au format du disque. Certains blocs sont utilisés pour enregistrer des informations propres au système de gestion, d'autres pour les fichiers.

Définition 29 (unité d'allocation)**.** Une unité d'allocation est un bloc d'enregistrement élémentaire du point de vue du système de gestion de fichiers qui contient une partie d'un fichier.

Les blocs sur les disques font en général 1/2 Ko (512 octets). Un bloc du système de gestion de fichiers peut faire 1 Ko. Cela signifie que si un fichier fait 2050 octets, les 2048 premiers octets seront sur deux blocs, et les deux derniers sur un troisième. Cela fait 1022 octets de perdus. Mettre en place une gestion des octets perdus ferait perdre bien plus de place que la somme des bouts de fichiers perdus. Ce serait également une perte d'efficacité en temps. La taille choisie par le système est un compromis qui se vérifie d'expérience en fonction de l'usage du système. De nouveau, c'est donc plus facile à gérer pour les informaticiens. C'est en revanche une perte considérable de support et donc une dépense inutile pour le propriétaire des données.

16.2 Descripteur de fichiers

Rappelons les informations essentielles à conserver pour garantir toutes les propriétés désirées :

- Le nom du fichier
- Son type
- Son propriétaire et son groupe
- Les droits
- Les dates (de création, modification, accès)
- Une table d'implantation sur le disque
- ...

Ces informations diffèrent d'un système à l'autre.

Définition 30 (descripteur de fichier)**.** Un descripteur de fichier ne contient pas le fichier, mais ses méta-données, c'est-à-dire l'ensemble des informations nécessaires à sa gestion comme le propriétaire et les droits, mais aussi son occupation sur le disque.

16.3 Table d'implantation

Définition 31 (table d'implantation)**.** La table d'implantation est la liste des unités d'allocation utilisées pour enregistrer un fichier.

Sa place occupée par la table se calcule en fonction de la taille :

- du fichier,
- des unités d'allocation,
- des tailles des adresses nécessaires pour désigner les unités d'allocation.

Pour donner un exemple en simplifiant les calculs, supposons qu'un fichier fait 1 Go, que nos blocs fassent 1 Ko, il devrait être découpé en 1024 × 1024 blocs de 1 Ko. Ce nombre nécessite un entier sur 20 bits. Partons sur des entiers de 24 bits (3 octets pouvant adresser jusqu'à 16 Go).

Si l'on souhaite adresser un fichier de 1Go, il faut 1024 × 1024 × 3 octets (donc 3 Mo = 0,3%).

Vous comprenez d'une part la démesure entre la taille du fichier et la taille de la table d'implantation. Plus le bloc sera grand, moins on perdra de place en table d'implantation.

Vous comprenez également que si l'on fixe une taille constante moyenne pour tous les descripteurs, on aura rempli le disque avec uniquement des descripteurs.

De nouveau, un compromis est nécessaire entre pouvoir désigner les plus gros fichiers possibles et ne pas perdre trop de place. Nous verrons qu'Unix a une solution élégante de répondre à ce problème.

16.4 Catalogue

Définition 32 (catalogue)**.** Un catalogue (répertoire) va enregistrer la liste des fichiers dans un nœud de l'arbre. Ce sont les références entre catalogue qui forment la structure arborescente du système de fichier.

Le système de gestion de fichiers met en place un mécanisme pour différencier les catalogues des fichiers.

16.5 Table d'allocation

Définition 33 (table d'allocation)**.** Une table d'allocation est un résumé compact de l'occupation des unités d'allocation sur le disque.

Lorsqu'on demande la création d'un fichier, il serait fastidieux de parcourir l'ensemble du disque pour chercher quelle unité est disponible. On utilise alors des bits qui représentent l'occupation d'un ensemble d'unité d'allocation. Le système gère également un pointeur sur le premier élément de la table qui indique une zone libre. À chaque création ou suppression de fichier le pointeur et la table sont mis à jour.

17 Le SGF Unix

Nous allons illustrer la description générale qui vient d'être faite avec la solution proposée par Unix.

La taille des différents éléments est déterminée au moment de la création du système de fichier. La commande `mkfs.ext4` choisira automatiquement les paramètres les plus appropriés en fonction de la taille de la partition (compromis exposé plus haut).

Pour connaître les caractéristiques de votre système vous pouvez utiliser la commande

`tune2fs -l`. Les valeurs peuvent être :

- Bloc size : 4096 (4 Kb)
- Inode size : 256 (1/4 Kb)

17.1 Unité d'allocation

Une unité d'allocation correspond à plusieurs blocs sur le disque dur (8 blocs de 512 dans notre exemple).

17.2 Inode

Le descripteur de fichier se nomme inode. La particularité d'Unix est qu'il ne contient pas le nom du fichier. L'avantage est qu'un fichier peut posséder plusieurs noms dans plusieurs répertoires. Il suffit de faire référence au même inode.

Un inode contient :

- la taille du fichier en octets
- l'UID : identifiant du propriétaire du fichier
- le GID : identifiant du groupe auquel appartient le fichier
- le mode du fichier qui détermine quel utilisateur peut lire, écrire et exécuter le fichier
- des chronomarques (timestamp)
  - ctime : date de dernière modification de l'inode
  - mtime : date de dernière modification du fichier
  - atime : date de dernier accès à l'inode
- le nombre de liens physiques sur cet inode
- la table d'implantation des unités d'allocation

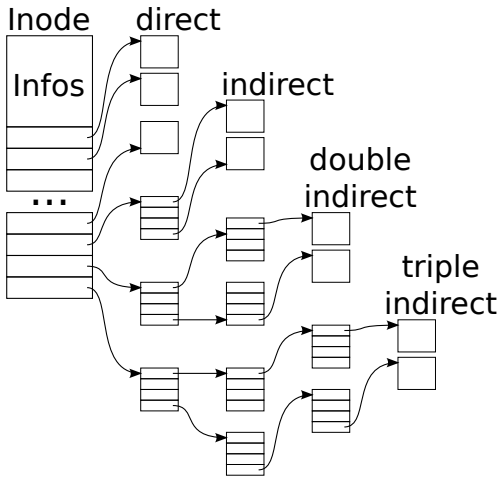


FIGURE 9 – Table d'implantation

Il existe également des informations supplémentaires comme pour les fichiers spéciaux (connectés à un périphérique avec le numéro majeur et mineur).

À chaque nouveau nom, le compteur de lien est augmenté. À chaque suppression de nom le compteur de lien est décrémenté. Le fichier est supprimé quand le compteur tombe à zéro.

Le compteur prend également en compte l'utilisation par la mémoire. Un programme qui s'exécute verra son compteur passer à 2. Il peut se supprimer de son répertoire (compteur de nouveau à 1) et continuer son exécution. À sa terminaison, il est délié de la mémoire (son compteur passe à 0) et réellement supprimé.

Remarquez qu'un fichier supprimé et simplement déréférencé, son contenu n'est pas rempli de zéro. On peut donc éventuellement retrouver des informations, mais l'on ne sait pas où les trouver.

17.3 Table d'implantation

La taille de la table d'implantation dépend de la taille totale de l'inode. Si nous avons :

- N la taille de la table d'allocation
- TI la taille des inodes
- TD la taille des informations dans l'inode (autre que la table d'allocation)
- TA taille des adresses pour référencer les unités d'allocation
- TUA taille des unités d'allocation

Nous obtenons : $N = (TI - TD)/TA$

La taille maximum des fichiers devrait donc être : $N * TUA$

Pour dépasser cette limite, la table fonctionne avec des niveaux d'indirection de références :

- Les $N - 3$ premiers éléments sont des références directes à des unités d'allocation.
- le suivant désigne une unité d'allocation qui contient des références vers des unités d'allocation (référence indirecte)
- l'avant-dernier désigne une unité d'allocation de référence indirecte (double indirection)
- le dernier désigne une unité d'allocation de référence doublement indirecte (triple indirection)

Si R est le nombre de références d'unité d'allocation que l'on peut mettre dans une unité d'allocation, le nombre d'unités maximum d'un fichier est : $(N - 3) + R + R^2 + R^3$. La taille maximum du fichier est ce nombre multiplié par la taille d'une unité d'allocation.

Une unité de 4 Ko peut contenir 512 pointeurs de 64 bits. Dans ce cas $R = 512$. Imaginez la taille des fichiers que l'on peut adresser.

L'avantage de cette approche est que pour les petits fichiers un inode suffit pour désigner tous les blocs. Pour les gros fichiers, il y a au maximum 3 indirections pour accéder à 4 Ko de données.

17.4 Répertoire

Les catalogues sous Unix se nomment répertoires. Ils sont stockés par le système comme des fichiers. En revanche on y accède par des commandes spécifiques (`mkdir`, `rmdir` ...) C'est un exemple concret d'une donnée qui appartient à un utilisateur, mais qui ce dernier ne peut modifier directement pour garantir son intégrité.

Ils ne contiennent que des couples

- inode
- nom de fichier

Un répertoire contient toujours au

minimum 2 entrées :

- "." le répertoire lui-même
- ".." le répertoire précédent dans l'arbre.

Puisqu'un répertoire possède un nom et qu'il se désigne lui-même, son compteur de liens est au moins de 2. Plus s'il possède d'autres noms dans d'autres répertoires (par exemple, dans ses fils il se nomme "..").

17.5 Volume Unix

Une partition Unix contient :

- un super-bloc qui décrit la structure de la partition, la taille des éléments et des informations de montage
- des informations de journalisation
- une table d'allocation
- une table des inodes
- des unités d'allocation

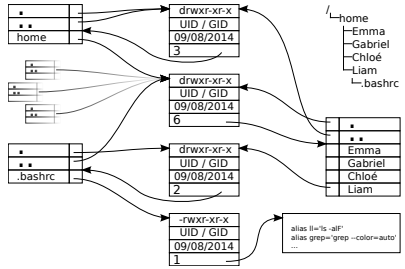


FIGURE 10 – Exemple d'inodes

Définition 34 (superbloc)**.** Un superbloc est un enregistrement des caractéristiques d'un système de fichiers, dont sa taille, celle des blocs, les blocs libres ou occupés, leur nombre...

En TD et TP, nous traiterons celui de Minix en exemple.

18 Montage

Certains systèmes gèrent des forêts de noms (une racine par partition). Unix unifie l'ensemble des partitions en les raccordant sur un arbre principal.

Définition 35 (montage)**.** Le montage (commande "`mount`") permet de superposer la racine d'un système de fichiers sur un nœud de l'arbre du système en cours d'exécution.

Il faut donc nécessairement créer un répertoire qui représentera la racine de la partition montée. Le répertoire de montage peut ne pas être vide. Durant la période de montage, les informations de ce répertoire sont masquées, la consultation du répertoire étant automatiquement redirigée vers la partition montée. Après le démontage de la partition, les informations réapparaissent.

Unix est capable de gérer plusieurs systèmes de fichiers. On peut donc construire un arbre en faisant succéder au long d'une branche des systèmes de natures différentes.

Les paramètres d'un montage comprennent :

- le type de montage
- la partition, désignée par le périphérique ou mieux par l'identifiant de la partition (indépendant de l'ordre de connexion des disques sur la carte mère).
- le répertoire de montage

Unix prévoit également des options importantes.

- L'option "`bind`" permet de monter une partie de l'arbre de nom à un autre endroit de l'arbre. Existant dès les premières versions d'Unix. L'option permet la mise au point du développement du système en lançant d'autres versions sur lui-même.
- L'option "`loop`" permet également de remonter des données ailleurs mais sous un autre format (par exemple un fichier ISO monté comme un CD). Le système est ainsi averti quand des tentatives d'écriture sur un même octet par deux moyens différents ont lieu (via le fichier ISO ou via le répertoire de montage).

Unix dispose également de mécanismes de montage via le réseau.

Il est possible de spécifier des auto-montages qui font apparaître le répertoire monté au moment où vous y accédez. Cela peut être pratique mais difficile à manier en cas de montage croisé, car il devient difficile de démonter l'ensemble.

19 Type de fichier

Tout dans Unix est fichier, y compris les périphériques. On les distingue par leur type et l'inode contient les informations nécessaires à leurs utilisations. Avec la commande `ls -l` le premier caractère désigne le type.

Caractère	Mode	Type de fichier
p	1	file (ou "named pipe")
c	2	spécial en mode caractère
d	4	répertoire (directory)
b	6	spécial en mode bloc
-	8	fichiers ordinaires
l	a	lien symbolique
s	c	socket

TABLE 4 – Les types des fichiers sous Unix

20 Les commandes shell

Voici maintenant les commandes utiles en rapport avec ce que nous venons de décrire dans ce chapitre.

Définition 36 (fdisk)**.** fdisk (*format disk*) est un outil en mode texte pour gérer la géométrie d'un disque et créer des partitions

Exemple :

```
/sbin/fdisk /dev/sda
```

Définition 37 (gparted)**.** gparted (*GNU Parted* donc interface graphique Gnome) est un outil graphique pour faire la même chose que *fdisk* (les deux sont complémentaires).

Exemple :

```
gparted /dev/sda
```

Définition 38 (mkfs)**.** mkfs (*make file system*) formate une partition (ou un fichier) suivant le système de fichier indiqué. mkfs fera appel à d'autres programmes adaptés en fonction du type de système de fichiers.

Exemple :

```
mkfs.ext4 /dev/sdaX
```

Définition 39 (fsck)**.** fsck (*file system check*) vérifie l'état d'un système de fichiers et sa fragmentation. Il peut réaliser des opérations de réparation automatique.

Exemple :

```
fsck /dev/sdaX
```

Définition 40 (dd)**.** dd (*disk to disk copy* copie bit à bit de disques) permet la recopie d'une partition sur une autre ou d'un fichier vers un autre. On l'utilise également pour allouer un fichier continu qui peut servir de travail sur un système de fichiers en TD.

Exemple :

```
dd if=/dev/zero of=monFichier64Ko bs=1024 count=64
```


Définition 41 (mount)**.** mount (*monte*) associe une partition dans l'arbre de fichier existant. En premier on indique le périphérique, en second le point de montage (un répertoire qui doit exister).

Exemple :

```
sudo mount -t iso9660 -o ro,loop /backup/kubuntu.iso /mnt/
```

Définition 42 (mkdir)**.** mkdir (*make directory*) crée des répertoires. L'option `-p` crée en une fois une arborescence.

Exemple :

```
mkdir -p r1/r2/r3
```

Définition 43 (touch)**.** touch permet de créer un fichier ou de changer sa date s'il existe déjà.

Exemple :

```
touch nouveauFihcier
```

Définition 44 (ln)**.** ln (*link*) crée un lien "physique" (partage du même inode) entre deux fichiers. Attention : il fonctionne comme "cp" : en premier le fichier qui existe, ensuite le fichier à créer (ou le répertoire dans lequel faire le lien).

L'option `-s` crée un lien symbolique.

Exemple :

```
ln -s fichierExistant ouPas monRepertoireDeLiens/
```

Définition 45 (rm)**.** rm (*remove*) supprime des fichiers. L'option `-r` supprime en une fois une arborescence.

Exemple :
Avez-vous vraiment besoin d'un exemple ?

Définition 46 (tar)**.** tar (*tape archiver*) outil conçu à l'origine pour l'archivage de bande magnétique (comme son nom l'indique). Il conserve les droits, dates et liens.

Exemple local :

```
(cd r1 ; tar -cf - .)|(cd r2; tar -xf -)
```

Exemple distant :

```
tar -cjf - monRepertoire|ssh moi@autreMachine: "cd r2; tar -xjf -"
```

Définition 47 (rsync)**.** rsync (*remote synchronization*) est un outil extrêmement performant pour la copie de fichier ou la sauvegarde. Il possède des options pour réaliser des sauvegardes incrémentales. Il copie en un temps record en occupant toute la bande passante. Il conserve les droits, les dates et la correspondance entre inodes (par exemple pour les liens physiques).

Exemple :

```
rsync --progress -rlogpt -e ssh /home/chezMoi moi@autreMachine:/home/
```

Exemple 1^{re} archive :

```
TODAY=$(date -I)
rsync -a /home/src /svg/$TODAY
```

Exemple achives suivantes :

```
TODAY=$(date -I)
YESTERDAY=$(date -I -d "1 day ago")
rsync -a /home/src --delete --link-dest=/svg/$YESTERDAY /svg/$TODAY
```

Définition 48 (kompare)**.** kompare est un outil de comparaison graphique de fichiers. Il visualise les portions de texte ajoutées/modifiées/supprimées et permet de valider ou refuser des changements.

21 TD2 : Fichiers

Dès votre arrivée en TD, pensez à déposer votre QCM rempli à votre encadrant. La partie à rendre est la dernière page de votre document (à découpe suivant les pointillés).

Après l'appel, c'est l'occasion de poser les questions concernant des éléments non compris du cours. Pensez à recopier le corrigé du QCM sur votre document (partie à concerver à la page précédante). Vous pourrez l'utiliser le jour de l'examen. Une page de notes personnelles est à votre disposition en fin de TD.

L'objectif du TD est de retrouver la table des inodes d'un système Unix.

21.1 Compréhension shell

Question : Construisez une arborescence de test constituée de 4 répertoires imbriqués, chacun possédant un fichier "readMe".

```
r1/
|--- r2/
|   |--- r3/
|   |   |--- r4/
|   |   |   \--- readMe
|   |   \--- readMe
|   \--- readMe
\--- readMe
```

Listing – Trace

Réponse : ✍

Question : Écrivez une commande qui prend des fichiers ou répertoires et affiche une paire constituée de l'inode et du fichier.

```
15228957 r1
```

Listing – Trace

Réponse : ✍

Question : Écrivez une commande qui prend des répertoires et affiche pour toutes ces arborescences les paires constituées de l'inode et du fichier.

Réponse : ✍

Question : Reprennez la commande précédente et triez les inodes en ordre croissant.

```
15208272 r1/readMe
15208276 r1/r2/readMe
15208281 r1/r2/r3/readMe
15208283 r1/r2/r3/r4/readMe
15228957 r1
15228958 r1/r2
15228959 r1/r2/r3
15228962 r1/r2/r3/r4
```

Listing – Trace

Réponse : ✍

Question : Les numéros ne suivent pas l'ordre des commandes. Pourquoi ?

Réponse : ✍

Question : On déplace le fichier readMe de r1 vers r2 en le renommant leReadMeDeR1. Que peut-on dire des inodes des fichiers et des répertoires déplacés ?

```
15208272 r1/r2/leReadMeDeR1
15208276 r1/r2/readMe
15208281 r1/r2/r3/readMe
15208283 r1/r2/r3/r4/readMe
15228957 r1
15228958 r1/r2
15228959 r1/r2/r3
15228962 r1/r2/r3/r4
```

Listing – Trace

Réponse : ✍

21.2 Pratique de MinixFS

Le système de gestion de fichiers de Minix est très rudimentaire. Les noms de fichiers sont limités à 30 caractères, il n'y a pas de journalisation, ... Bref, il n'est pas fiable et vous ne devez pas l'utiliser. En revanche, comme il est simple nous allons le prendre en exemple pour des travaux pratiques.

Nous travaillerons avec un fichier composé de 64 blocs de 1 Ko.

Nous ignorerons la partie super-bloc et les cartes d'occupation du disque. Nous ne considérerons que la table des inodes (0x1000) et les données qui commencent au bloc 5 (5*1 Ko = 5*0x400 = 0x1400). Nous n'utiliserons donc que 2 structures (voir http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux/include/linux/minix_fs.h).

La structure des inodes :

```
1  struct minix_inode {
           __u16 i_mode;
           __u16 i_uid;
           __u32 i_size;
5          __u32 i_time;
           __u8  i_gid;
           __u8  i_nlinks;
           __u16 i_zone[9];
   };
```

Listing 17 – Structure C des inodes Minix

Définition 49 (boutisme ou en anglais *Endianness*)**.** Les fichiers minix sont en "petit-boutiste" (*little-endian* en englais),
voir https://fr.wikipedia.org/wiki/Endianness.

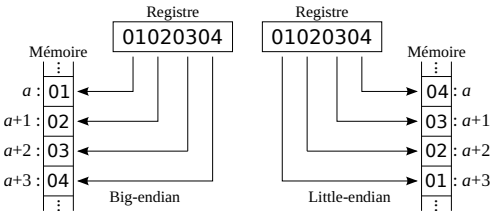


FIGURE 11 – Boutisme *(Endianness)*

Exmple sur :

```
1  00001020: fd41 4200 a000 0000 ed89 dc53 4303 0600  .AB........SC...
   00001030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Listing 18 – inode 2

On obtient :

i_mode	i_uid	i_size	i_time	i_gid	i_nlinks	i_zone
41fd	0042	000000a0	53dc89ed	43	03	0006 0000 ....

Le mode se découpe de la façon suivante (4 = dossier / 8 = fichier) :

type de fichier	droits
4	1fd

Soit en binaire :

Hexa	Octal	Binaire	s	s	t	r	w	x	r	w	x	r	w	x
1fd	0775	000111111101	0	0	0	1	1	1	1	1	1	1	0	1
1b4	0664	000110110100	0	0	0	1	1	0	1	1	0	1	0	0

La date est donnée en secondes depuis le début de l'univers, c'est à dire le 1/1/70 au sens d'Unix (le bug de la fin de l'univers est prévu pour le 7 février 2106 à 06 :28 :15 GMT). Pour la conversion de la date, vous pouvez commencer par convertir en décimales puis effectuer l'opération suivante : time/86400+"1/1/70" (dans une feuille de calcul par exemple).

Petit tracas suplémentaire, les numéro d'inode commencent à 1 (et non pas 0). donc :

numéro d'inode	position	adresse dans la table
1	0	0x00001000
2	1	0x00001020
3	2	0x00001040
...		

Enfin notez que, pour i_zone, les 9 entiers sur 2 octets pointent sur des blocs de données (Z->D), sauf :

- l'avant dernier, qui pointe sur un bloc des pointeurs sur des blocs données (Z->Z->D)
- le dernier, qui pointe sur un bloc qui pointe sur des blocs de même nature que le précédent (Z->Z->Z->D).

La structure des répertoires est plus simple :

```
1 struct minix_dir_entry {
        __u16 inode;
        char name[0];
  };
```

Listing 19 – Structure C des répertoires Minix

Pour réduire le listing les suites d'octets identiques sont représentées par "...".

```
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
...   0000
00000400: 2000 4000 0100 0100 0500 0000 001c 0810   .@.............
00000410: 8f13 0100 0000 0000 0000 0000 0000 0000  ................
...   0000
00000800: ef01 0000 feff ffff ffff ffff ffff ffff  ................
...   ffff
00000c00: ffff ff0f 0000 00f0 ffff ffff ffff ffff  ................
...   ffff
00001000: ed41 4200 a000 0000 ed89 dc53 4303 0500  .AB........SC...
00001010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001020: fd41 4200 a000 0000 ed89 dc53 4303 0600  .AB........SC...
00001030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001040: fd41 4200 6000 0000 ed89 dc53 4302 0700  .AB.`......SC...
00001050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001080: b481 4200 0004 0000 ed89 dc53 4301 0800  ..B........SC...
00001090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000010a0: b481 4200 0050 0000 ed89 dc53 4301 0900  ..B..P.....SC...
000010b0: 0a00 0b00 0c00 0d00 0e00 0f00 1000 0000  ................
000010c0: b481 4200 0900 0000 ed89 dc53 4301 1e00  ..B........SC...
000010d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000010e0: b481 4200 0900 0000 ed89 dc53 4301 1f00  ..B........SC...
000010f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
...   0000
00001400: 0100 2e00 0000 0000 0000 0000 0000 0000  ................
00001410: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001420: 0100 2e2e 0000 0000 0000 0000 0000 0000  ................
00001430: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001440: 0200 7231 0000 0000 0000 0000 0000 0000  ..r1............
00001450: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001460: 0500 316b 2e74 7874 0000 0000 0000 0000  ..1k.txt........
00001470: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001480: 0600 3230 6b2e 7478 7400 0000 0000 0000  ..20k.txt.......
00001490: 0000 0000 0000 0000 0000 0000 0000 0000  ................
...   0000
00001800: 0200 2e00 0000 0000 0000 0000 0000 0000  ................
00001810: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001820: 0100 2e2e 0000 0000 0000 0000 0000 0000  ................
00001830: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001840: 0800 6631 0000 0000 0000 0000 0000 0000  ..f1............
00001850: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001860: 0000 6475 6d79 0000 0000 0000 0000 0000  ..dumy..........
00001870: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001880: 0300 7232 0000 0000 0000 0000 0000 0000  ..r2............
00001890: 0000 0000 0000 0000 0000 0000 0000 0000  ................
...   0000
00001c00: 0300 2e00 0000 0000 0000 0000 0000 0000  ................
00001c10: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001c20: 0200 2e2e 0000 0000 0000 0000 0000 0000  ................
00001c30: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001c40: 0700 6632 0000 0000 0000 0000 0000 0000  ..f2............
00001c50: 0000 0000 0000 0000 0000 0000 0000 0000  ................
...   0000
00002000: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
...   4141
00002400: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
...   4242
00004000: 1100 1200 1300 1400 1500 1600 1700 1800  ................
00004010: 1900 1a00 1b00 1c00 1d00 0000 0000 0000  ................
...   0000
00004400: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
...   4242
00007800: 626f 6e6a 6f75 7232 0a00 0000 0000 0000  bonjour2........
...   0000
00007c00: 626f 6e6a 6f75 7231 0a00 0000 0000 0000  bonjour1........
...   0000
0000fff0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Listing 20 – Exemple Minix

Question : Donnez le descriptif complet de l'inode 1.

Réponse : ✍

Question : A quoi correspond cet inode ? Comment pouvez-vous l'affirmer ?

Réponse : ✍

Question : Que penser de l'inode 4 ?

Réponse : ✍

Question : Donnez le descriptif complet de l'inode 6.

Réponse : ✍

Question : Indiquez le contenu du fichier pointé par l'inode 6 ?

Réponse : ✍

Question : Donnez toute l'arborescence avec pour chaque fichier une ligne de la forme :
– [l'inode lesDroits IUD GID] nom (contenu éventuel des fichiers) .

Réponse : ✍

Page de notes personnelles : ✍

22 TP2 : Fichiers

- Votre compte-rendu est à rendre en fin de TP.
- Vous ne devez pas rendre un listing, mais un document répondant aux questions posées.
- Vous préparerez le compte-rendu à l'avance (copier-coller les questions) pour gagner du temps.
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois.
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (uniquement la fenêtre concernée).
- N'oubliez pas de rendre votre TP imprimé en 2 colonnes de texte par côté de papier et de faire le TP en binôme.

L'objectif du TP est d'écrire des logiciels permettant l'interprétation d'un système de fichiers Minix. En pratique nous écrirons :

- un ensemble de fonction constituant le pilote d'un périphérique de gestion du système Minix
- la commande ls
- la commande cat.

Il est conseillé de séparer les sources par rubriques ainsi que par programme :

- **convFunct.py** : bibliothèque de conversion d'adresse
- **testConv.py** : programme de test des conversions
- **testReadFs.py** : programme de test de lecture du système de fichiers
- **inodeFunct.py** : bibliothèque de traitement des inodes
- **testInode.py** : programme de test des inodes
- **dirFunct.py** : bibliothèque de traitement des répertoires
- **testDir.py** : programme de test des répertoires
- **ls.py** : programme d'affichage du contenu d'un répertoire
- **cat.py** : programme d'affichage du contenu d'un fichier

Pour chaque fonction nous donnerons le commentaire python et un résultat d'exécution.

22.1 Fonction de conversion

Commençons par des conversions numériques entre identifiants et adresses.

Question : Donnez les fonctions de correspondance entre unité d'allocation et adresse.

```
def ua2addr (ua):
    """
    Conversion de l'identifiant d'une unité d'allocation en adresse.
      :param ua : l'identifiant de l'unité d'allocation
      :type ua  : int
      :return   : adresse de l'unité d'allocation
      :rtype    : int
    """
```

Listing 21 – Entête convFunct.py : ua2addr

```
def testUa2addr ():
    """
    Teste la conversion des identifiants d'unité d'allocation 4 et 5.
    """
```

Listing 22 – Entête testConv.py : ua2addr

```
$ ./testConv.py ua2addr
ua: 4 =>  0x1000
ua: 5 =>  0x1400
```

Listing 23 – Traces testConv.py : ua2addr

Réponse : ✍

Question : Donnez les fonctions de correspondance entre numéro d'inode et adresse.

```
def i2addr (id):
    """
    Conversion de l'identifiant d'un inode en adresse.
      :param id : l'identifiant de l'inode
      :type id  : int
      :return   : adresse de l'inode
      :rtype    : int
    """
```

Listing 24 – Entête convFunct.py : i2addr

```
def testI2addr ():
    """
    Teste la conversion des inodes 1, 2 et 3.
    """
```

Listing 25 – Entête testConv.py : i2addr

```
$ ./testConv.py i2addr
addr: 1 =>  0x1000
addr: 2 =>  0x1020
addr: 3 =>  0x1040
```

Listing 26 – Traces testConv.py : i2addr

Réponse : ✍

22.2 Affichage des données d'un inode

Cette rubrique traite des différents formats d'affichage.

Il faut en premier déterminer le type de fichier (d ou -).

Question : Donnez la fonction de conversion d'un octet en type de fichier.

```
def modeFile2str (mode):
    """
    Retourne le type de fichier.
      :param mode : le type de ficher
      :type mode  : int
      :return     : information lissible par un humain
      :rtype      : char
    """
```

Listing 27 – Entête inodeFunct.py : modeFile2str

```
def testFileMode ():
    """
    Teste la traduction du type de fichier pour les valeurs 4 et 8.
    """
```

Listing 28 – Entête testInode.py : testFileMode

```
$ ./testInode.py fileMode
fileMode:
  4 =>  d
  8 =>  -
```

Listing 29 – Traces testInode.py : fileMode

Réponse : ✍

L'affichage des droits se limite dans un premier temps à uen catégorie (rwx).

Question : Donnez la fonction de conversion d'un octet droits. On pourra dans un premier temps ne pas considérer la gestion des bits "s".

```
def rightMode2str (mode, sMode):
    """
    Représente les droits pour une catégorie (utilisateur, groupe ou autre) en
        tenant compte du bit s.
      :param mode  : un entier entre 0 et 7 (donc 3 bits). r est à gauche (la plus
          grande valeur)
      :type mode   : int
      :param sMode : une chaîne de caractères des valeur possible pour x.
                     "-x" s'il n'y a pas de bit s
                     "Ss" s'il y a un bit s pour le propriétaire ou le groupe (su
                         bit)
                     "Tt" s'il y a un bit s pour le les autres (sticky bit)
      :type sMode : string
      :return     : une chaîne de 3 caractères de "---" à "rwx" ou "sst" ou "SST"
      :rtype      : string
    """
```

Listing 30 – Entête inodeFunct.py : rightMode2str

```
def testRightMode ():
    """
    Teste les droits d'une catégorie pour les valeurs 0, 1, 5, 6 et 7 avec ou sans
        bit "s".
    """
```

Listing 31 – Entête testInode.py : testRightMode

```
$ ./testInode.py rightMode
rightMode:
  0   -x  =>  ---
  1   -x  =>  --x
  5   -x  =>  r-x
  6   -x  =>  rw-
  7   -x  =>  rwx
  0   Tt  =>  --T
  1   Tt  =>  --t
  5   Tt  =>  r-t
  6   Tt  =>  rwT
  7   Tt  =>  rwt
  0   Ss  =>  --S
  1   Ss  =>  --s
  5   Ss  =>  r-s
  6   Ss  =>  rwS
  7   Ss  =>  rws
```

Listing 32 – Traces testInode.py : rightMode

Réponse : ✍

Question : Donnez la fonction de conversion d'un entier de 4 octets en droits.

```
def propMode2str (mode):
    """
    Converti les 4 octets des droits d'un fichier en une représenation lisible.
      :param mode : droits du fichier.
      :type mode  : int
      :return     : une chaîne de 10 caractères de "----------" à "drwxrwxrwx"
      :rtype      : string
    """
```

Listing 33 – Entête inodeFunct.py : propMode2str

```
def testPropMode ():
    """
    Teste les droits d'un fichier pour les valeurs 00775, 07775, 00664, 07664, 0
        x41ed et 0x81b4.
    """
```

Listing 34 – Entête testInode.py : testPropMode

```
$ ./testInode.py propMode
  00775  =>
Traceback (most recent call last):
  File "./testInode.py", line 97, in <module>
    switch [sys.argv[1]] ()
  File "./testInode.py", line 43, in testPropMode
    print "\t00775  => ", getPropMode (00775)
NameError: global name 'getPropMode' is not defined
```

Listing 35 – Traces testInode.py : propMode

Réponse : ✍

Question : Donnez la fonction de conversion d'un entier en date.

```
def date2str (timeStamp):
    """
    Converti une date unix (depuis le 1er janvier 1970) en une représenation
        lisible.
      :param timeStamp : le nombre de secondes écoulées depuis la référence unix.
      :type timeStamp  : int
      :return          : une chaîne de caractère au forma "jour. jour-du-mois mois
          année heure:minutes:secondes"
      :rtype           : string
    """
```

Listing 36 – Entête inodeFunct.py : date2str

```
def testDate2str ():
    """
    Teste le format de date pour la valeur 1407002500.
    """
```

Listing 37 – Entête testInode.py : testDate2str

```
$ ./testInode.py date2str
sam. 02 août 2014 20:01:40
```

Listing 38 – Traces testInode.py : date2str

Réponse : ✍

22.3 Lecture des données

Cette partie permet de vérifier que le système de fichier d'exemple est bien exploitable.

Question : Récupérez le fichier : td2.MinixFS et vérifiez que son contenu correspond à l'étude faite en TD. Indiquez votre façon de faire.

Réponse : ✍

Question : Donnez une fonction de lecture d'une suite d'octets dans le système de fichier

```
def readFs (fs, addr, len):
    """
    Lecture d'une suite d'octets
      :param fs    : système de fichiers
      :type fs     : fichier ouvert en lecture
      :param addr  : adresse absolue dans le fichier
      :type addr   : int
      :param len   : nombre d'octets à lire
      :type len    : int
      :return      : la zonne mémoire lue
      :rtype       : bytes
    """
    fs.seek (addr, 0)
    return fs.read (len)
```

Listing 39 – Entête readFs.py

```python
def testRead (fs):
    """
    Teste la lecture du système de fichiers
      * 32 octets à l'adresse 0x1000
      * 9 octets à l'adresse 0x7c00.
      :param fs    : système de fichiers
      :type fs     : fichier ouvert en lecture
    """
```

Listing 40 – Entête test readFs.py

```
$ ./testReadFs.py
read: 0x1000
['\xed', 'A', 'B', '\x00', '\xa0', '\x00', '\x00', '\x00', '\x84', "'", '\xdd', 'S
    ', 'C', '\x03', '\x05', '\x00', '\x00', '\x00', '\x00', '\x00', '\x00', '\x00'
    , '\x00', '\x00', '\x00', '\x00', '\x00', '\x00', '\x00', '\x00', '\x00', '\
    x00']
read: 0x7c00
bonjour1
```

Listing 41 – Traces testReadFs.py

Réponse : ✍

Pour la lecture des strcures C, nous utiliserons la bibliothèque “struct”.

https://docs.python.org/3/library/struct.html

```python
# struct minix_inode {
#    __u16 i_mode;
#    __u16 i_uid;
#    __u32 i_size;
#    __u32 i_time;
#    __u8 i_gid;
#    __u8 i_nlinks;
#    __u16 i_zone [9];
# };

"""
Format de la structure inode :
petit boutisme, unsigned short, unsigned short, unsigned int, unsigned int,
    unsigned char, unsigned char, unsigned short [9]
"""
inode_fmt = '<HHIIbb9H'
"""
Taille de la structure inode.
"""
inode_len = struct.calcsize (inode_fmt)
"""
Convertion de la structure inode
"""
inode_unpack = struct.Struct (inode_fmt).unpack_from
```

Listing 42 – Structure inode en python

22.4 Fonctions “inode”

Nous reprennons l'écriture des fonctions de traitement d'inode qui nécessitent la lecture du système de fichier.

Question : Donnez la fonction de lecture d'un inode en fonction de son numéro. Dans un premier, on se limitera à afficher les octets.

```python
def getInode (fs, iNum):
    """
    Lit un inode dans un système de fichier
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
      :param iNode : numéro de l'inode (commençant à 1 pour indiquer que 0 est une
          valeur vide)
      :type iNode  : int
      :return      : l'inode
      :rtype       : struct inode
    """
```

Listing 43 – Entête inodeFunct.py : getInode

```python
def testInode (fs):
    """
    Teste l'affiche du premier inode du système de fichier.
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
    """
```

Listing 44 – Entête test testInode.py : testInode

```
$ ./testInode.py inode
inode:  (16877, 66, 160, 1407002500, 67, 3, 5, 0, 0, 0, 0, 0, 0, 0, 0)
```

Listing 45 – Traces testInode.py : inode

Réponse : ✍

Question : Donnez la fonction de conversion d'un inode.

```python
def inode2str (inode):
    """
    Converti une structure inode en une chaîne de caractère lisible.
      :param inode : un inode
      :type inode  : struct inode
      :return      : une chaîne au forma proche de "ls -l"
      :rtype       : string
    """
```

Listing 46 – Entête inodeFunct.py : inode2str

```
$ ./testInode.py inode
inode:  (16877, 66, 160, 1407002500, 67, 3, 5, 0, 0, 0, 0, 0, 0, 0, 0)
drwxr-xr-x 3 u:66 g:67   160 sam. 02 août 2014 20:01:40
```

Listing 47 – Traces testInode.py : testInode

Réponse : ✍

Il ne faut pas hésiter à écrire des fonctions utiles : getRoot, getMode, isDir, getUser, getSize, getDate, getLinkCount, getUaList.

22.5 Fonctions “répertoire”

La lecture des inodes ne suffit pas pour comprendre l'arborescence du système de fihcier. Il est nécessaire de consulter le contenu des répertoires.

Nous commençont par la lecture d'une entrée puis nous constiturons une liste de tous les éléments pour représenter le contenu complet du répertoire.

Question : Donnez la fonction de lecture d'une entrée en fonction de sa position dans une unité d'allocation du répertoire.

```
def getDirEntry (fs, ua, idx):
    """
    Lit la nième entrée dans un répertoire.
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
      :param ua    : unité d'allocation contenant le contenu du répertoire
      :type ua     : l'identifiant de l'unité d'allocation
      :type ua     : int
      :return      : une entrée [inode, nom]
      :rtype       : struct dirEntry
    """
```

Listing 48 – Entête dirFunct.py : getDirEntry

```
########################################
def testDirEntry (fs):
    """
    Teste la lecture des 4 premières entrées du répertoire de l'unité d'allocation
         5.
```

Listing 49 – Entête test testDir.py : testDirEntry

```
$ ./testDir.py dirEntry
dirs:
[1, '.']
[1, '..']
[2, 'r1']
[5, '1k.txt']
```

Listing 50 – Traces testDir.py : dirEntry

Réponse : ✍

Question : Donnez la fonction de lecture d'un répertoire. On se limitera dans un premier temps d'afficher les valeurs non formatées.

```
def getDir (fs, inode):
    """
    Retourne le contenu d'un répertoire.
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
      :param inode : un inode
      :type inode  : struct inode
      :return      : une liste d'entrées
      :rtype       : [struct inode]
    """
```

Listing 51 – Entête dirFunct.py : getDir

```
def testGetDir (fs):
    """
    Teste la lecture du répertoire racine.
    """
```

Listing 52 – Entête test testDir.py : testGetDir

```
$ ./testDir.py getDir
root:
drwxr-xr-x 3 u:66 g:67   160 sam. 02 août 2014 20:01:40 .
drwxr-xr-x 3 u:66 g:67   160 sam. 02 août 2014 20:01:40 ..
drwxrwxr-x 3 u:66 g:67   160 sam. 02 août 2014 20:01:40 r1
-rw-rw-r-- 1 u:66 g:67  1024 sam. 02 août 2014 20:01:40 1k.txt
-rw-rw-r-- 1 u:66 g:67 20480 sam. 02 août 2014 20:01:40 20k.txt
```

Listing 53 – Traces testDir.py : getDir

Réponse : ✍

Il pourrat être utile dans un second temps d'écrire une fonction d'affichage.

```python
def printDir (fs, dirEntries):
    """
    Affiche le contenu d'un répertoire.
    Attention : les numéros d'inode à zéro correspondent à des entrées effacées.
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
      :param dirEntries : contenu d'un répertoire
      :type dirEntries  : [struct dirEntry]
    """
```

Listing 54 – Entête dirFunct.py : printDir

Question : Donnez la fonction de conversion d'un chemin en un inode.

```python
def path2inode (fs, path):
    """
    Retourne l'inode correspondant au chemin en paramètre.
    Attention : les numéro d'inode à zéro correspondent à des entrées effacées.
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
      :param path : le chemin composé de nom séparé par des "/"
      :type path   : string
      :return      : l'inode ou une exception
      :rtype       : struct inode
    """
```

Listing 55 – Entête dirFunct.py : path2inode


```python
def testPath2inode (fs):
    """
    Teste la convertion d'un chemin en inode
      :param fs    : système de fichier ouvert en lecture
      :type fs     : file
    """
```

Listing 56 – Entête test testDir.py : testPath2inode

```
$ ./testDir.py path2inode
drwxrwxr-x 2 u:66 g:67    96 sam. 02 août 2014 20:01:40 /r1/r2
```

Listing 57 – Traces testDir.py : path2inode

Réponse : ✍

22.6 La commande "ls"

Nous avons tous les éléments pour écrire la commande "ls".

Question : Donnez le programme qui prend en argument un chemin.

```
$ ./ls.py /r1
drwxrwxr-x 3 u:66 g:67   160 sam. 02 août 2014 20:01:40 .
drwxr-xr-x 3 u:66 g:67   160 sam. 02 août 2014 20:01:40 ..
-rw-rw-r-- 1 u:66 g:67     9 sam. 02 août 2014 20:01:40 f1
drwxrwxr-x 2 u:66 g:67    96 sam. 02 août 2014 20:01:40 r2
```

Listing 58 – Traces ls.py

Réponse : ✍

22.7 La commande "cat"

Dernier programme du TP est la commande "cat".

Question : Donnez le programme qui prend en argument un chemin.

```
$ ./cat.py /r1/f1
bonjour1
$ ./cat.py /20k.txt | wc -c
20480
```

Listing 59 – Traces cat.py

Réponse : ✍

Il pourrat être utile d'écrire une fonction "readListUa".

```python
def readListUa (fs, size, uaList):
  """
  Lit une suite d'unité d'allocation dans la limite du nombre d'octets à lire.
      :param fs      : système de fichier ouvert en lecture
      :type fs       : file
      :param size    : nombre d'octets à lire
      :type size     : int
      :param uaList  : liste d'unité d'allocation
      :type uaList   : [int]
      :return        : le nombre d'octets qu'il reste à lire
      :rtype         : int
  """
```

Listing 60 – Entête cat.py : readListUa

22.8 Conclusion

Vous venez de réaliser un travail considérable. Les fonctions de traitements sur les inodes et les répertoires constituent un pilote de périférique de gestion du système de fichiers Minix.

Certes, il est limité (pas d'écriture, pas de prise en compte de la géométrie du disque), mais c'est ce genre de logiciel qui se trouve dans le noyeau d'un système d'exploitation.

De même vous avez réalisé 2 commandes d'un interpréteur de commande.

La compléxité des systèmes actuels impose d'être nombreux à contribuer à un tel travail. Mais vous avez maintenant une idée plus précise de comment en réaliser un.

Quatrième partie

La mémoire

La mémoire d'un ordinateur revêt des formes multiples :

- mécanique (parties mobiles d'un disque dur)
- physique (encombrement)
- thermique (chauffe)
- électrique (consomme)
- rémanente (persiste après extinction)
- usage (écriture ou non)
- capacité (taille stockée)
- rapidité (temps d'accès) ...

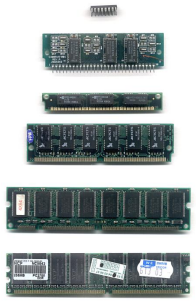


FIGURE 12 – RAM

Citons quelques exemples : RAM, ROM EEPROM, USB, disque, CD, DVD, vive, virtuelle, segmentée, paginée, partagée, protégée, d'échange ...

Nous allons aborder différents aspects de la mémoire, mais ce chapitre sera loin d'en faire le tour.

Malgré tout, Portons une attention particulière sur une mémoire à *tore magnétique* qui pendant 20 ans (de 1955 à 1975) permettait de conserver des données sans consommation électrique (voir https://fr.wikipedia.org/wiki/Tore_magn%C3%A9tique). Comme quoi, il est possible de trouver des solutions si on décide de se donner des contraintes (zéro consommation).

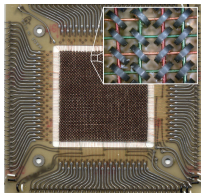


FIGURE 13 – Tore magnétique

Un carré de 11 cm de côté pouvait contenir 512 octets !

Profitons de l'occasion pour rappeler qu'à l'origine la mémoire de masse se trouvait sur bande magnétique donc d'un accès séquentiel. Pour lire une information en bout de bande, il fallait dérouler toute la bande. La mémoire vive a permis un accès dit aléatoire (non contraint d'être séquentiel). C'est ce qui a donné leur nom. Elles sont aujourd'hui électroniques.

Pour rappel, une liste de mémoires électroniques

- RAM (*Random Access Memory*) mémoire à accès aléatoire en opposition à mémoire à accès séquentiel.
- SRAM (*Static Random Access Memory*) mémoire chère et volumineuse, mais rapide et économique en énergie. Elle est utilisée dans les caches des microprocesseurs.
- DRAM (*Dynamic Random Access Memory*) s'appuyant sur un pico condensateur et un transistor, cette mémoire nécessite d'être rafraîchie au bout de quelques millisecondes à cause de fuite du condensateur.
- ROM (*Read-Only Memory*) mémoire morte dont les informations sont figées (peuvent être câblées).
- PROM (*Programmable Read Only Memory*) mémoire programmable une seule fois. Les bits sont stockés par des fusibles. Une surtension les grille. Ils passent à 0 définitivement.
- EPROM (*Erasable Programmable Read Only Memory*) mémoire programmable et effaçable. L'effacement peut se faire par UV (Ultra Violet), ce qui peut amener à la débrocher pour être effacé.
- EEPROM (*Electrically Erasable Programmable Read Only Memory*) mémoire programmable et effaçable (on dit aussi E^2PROM). En clair, on peut inscrire une information (souvent avec une tension électrique plus forte qu'à la lecture) et débrancher ensuite son alimentation. Elle conserve ses données.
- Flash est une variété de mémoire EEPROM rapide et effaçable par secteur complet. On les trouve dans les périphériques devant être personnalisés (BIOS sur une carte mère, adresse MAC sur une carte réseau ...).

23 Caractéristique de mémoires

23.1 Mémoire vive

Définition 50 (mémoire vive)**.** La mémoire est dite vive si elle n'existe que tant qu'elle est alimentée électriquement.

23.2 Mémoire des processus

Historiquement, dans un processus, on distingue 4 zones de mémoire en fonction de leurs usages :

- code contient les instructions de programme
- données contient les données globales du programme
- pile contient les données intermédiaires de calcul
- tas contient des données dont la vie n'est pas liée à la durée d'exécution d'une opération

Définition 51 (codes)**.** La zone de code contient

les instructions d'un programme. Elle est composée des fonctions C, des méthodes Java... Elle comprend des constantes qui se retrouvent dans les instructions assembleur. Sa taille est connue à la compilation. Au moment de l'exécution elle doit être protégée en écriture.

Définition 52 (données)**.** La zone de données contient

les données globales du programme. Elle est composée des variables hors fonction C ou des variables statiques dans les fonctions, des attributs statiques de classe Java... Elle comprend des constantes élaborées (initialisées par un calcul au démarrage). Sa taille est connue à la compilation.

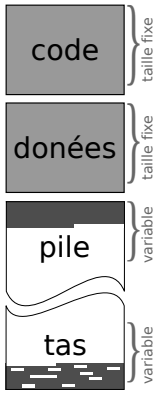


FIGURE 14 – Zone mémoire

Définition 53 (pile)**.** La zone de pile contient

les données intermédiaires de calcul. Elle est composée des variables automatiques (locales) des fonctions, des paramètres formels, des valeurs de retour. Sa taille n'est pas connue. En revanche, sa taille croît à chaque appel de fonction ou de méthode et revient au même endroit au retour.

Définition 54 (tas)**.** La zone de tas contient

des données dont la vie n'est pas liée à la durée d'exécution d'une opération. Ces données seront créées par une fonction et continueront d'être accessibles après l'appel. Elle est composée des variables obtenues par la fonction *malloc* en C, l'opérateur *new* en Java ... Pour éviter qu'elle ne croisse indéfiniment, on cherche à mettre des solutions de ramasse-miettes pour recycler les cellules mortes.

Au moment du lancement d'un processus, nous pouvons influencer sur la taille des mémoires. Par exemple, les arguments fournis à la JVM (Machine Virtuelle Java) fixent les limites. 1er caractère : s pour Stack m pour Memory et 2d caractère : s pour minimum et x pour maximum. Voici des exemples :

- `-Xss512M` taille minimum de 512 Mo pour la pile
- `-Xsx1G` taille maximum de 1 Go pour la pile
- `-Xms6G` taille minimum de 6 Go pour le tas
- `-Xmx36G` taille maximum de 36 Go pour le tas

Vérifiez que vous disposez de la mémoire vive suffisante pour les paramètres.

Les premiers systèmes d'exploitation allouaient le maximum de mémoire exigée par le processus et plaçaient tête-bêche la pile et le tas, en espérant qu'ils ne grandissent pas en même temps.

Aujourd'hui, la mémoire est plus souple (ce sera l'objet des sections suivantes), il n'y a plus de contraintes de placement.

Concernant l'accès en écriture par d'autres processus, on comprend que si un programme pouvait connaître, au moment de l'exécution d'un processus, la plage d'adresse qu'il utilise en mémoire, il pourrait modifier le contenu des variables et modifier les traitements. Nous verrons qu'aujourd'hui les adresses sont translatées et que les processus manipulent des adresses virtuelles (voir `https://fr.wikipedia.org/wiki/M%C3%A9moire_virtuelle`). Ils n'ont pas eux-mêmes accès à cette connaissance.

Définition 55 (mémoire virtuelle)**.** Mise au point dans des années 1960, la mémoire virtuelle se fonde sur la traduction à la volée (dispositif électronique) des adresses utilisées par les processus. Elle peut être segmentée ou paginée.

La mémoire virtuelle permet une diminution de besoin en mémoire réelle, car elle offre :

- le recours à la mémoire d'échange (extension de mémoire)
- le partage de code entre processus (économie de mémoire)
- le partage de données entre processus (possibilité algorithmique).

Par ailleurs, elle garantit l'étanchéité entre programme. Ils ont tous des adresses qui commencent par 0 et ne savent pas désigner la mémoire réelle.

Il est cependant possible de partager des informations entre différents processus. Nous avons déjà abordé cette question dans notre chapitre sur les ressources du point de vue de la communication.

Revenons sur le partage du point de vue de l'espace mémoire.

Au moment de la création ou de la récupération de la mémoire partagée, le processus reçoit un pointeur sur sa zone de tas. En réalité, la mémoire associée réelle est ailleurs.

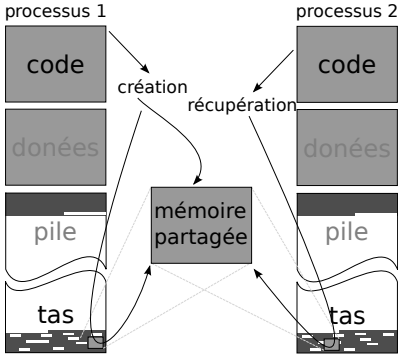


FIGURE 15 – Mémoire partagée

La mémoire partagée se trouve physiquement dans un espace alloué et géré par le système d'exploitation au moment de la demande création du processus. Les adresses dans l'espace d'adressage du processus sont détournées comme si la mémoire partagée était superposée au-dessus.

Si l'on devait fournir des adresses différentes à chaque processus, sachant qu'ils ne sont pas tous en même temps en mémoire, on pourrait craindre de devoir utiliser des adresses qui se chevauchent au moment d'un accroissement de l'un d'eux. Nous le verrons par la suite, qu'il n'y a pas de mémoire continue et donc pas d'adresse mémoire perdue.

23.4 Mémoire d'échange

Tous les programmes ne sont pas actifs au même moment en mémoire. Certains sont bloqués en attente de conditions qui n'arriveront peut-être jamais.

L'idée est donc d'être optimiste et de penser que l'on peut servir plus de processus que la mémoire vive réelle ne permet d'en héberger en même temps. Pour cela, quand un processus dort (et que l'on a besoin de place), les données du processus sont recopiées dans une mémoire étendue.

Définition 56 (mémoire d'échange)**.** Une mémoire d'échange (en anglais *swap*) est une extension sur un périphérique de mémoire de masse de la mémoire vive. Son utilisation est coûteuse en temps.

Il faut réaliser que les choix seront sûrement différents d'une époque à une autre. C'est parce que la mémoire vive rapide économiquement coûte cher, que l'on a recours à une mémoire de masse lente pour l'étendre.

Bien entendu, des euristiques sont développés pour déterminer quelle mémoire est à écarter de l'accès du processeur.

Il y a des limites à cet optimisme. Il ne serait pas raisonnable d'accepter l'accueille des milliers de processus sur un processeur en se disant qu'il y a une mémoire d'échanges ; donc cela peut tenir même si chaque processus n'aura que quelques nanosecondes par heure d'accès aux ressources.

L'inconvénient majeur est le temps d'accès au swap. Même si les mémoires du disque et de la RAM étaient équivalentes, il faudrait dans tous les cas inclure le temps de transfert sur les bus de données. À l'heure, où les mémoires centrales se comptent en giga octets, ce temps est loin d'être négligeable.

En revanche, il existe une autre utilisation qui pousse à déclarer une mémoire d'échange sur un système d'exploitation. C'est l'hibernation. Un ordinateur personnel peut être éteint pour consommer moins. Mais le redémarrage est coûteux en temps. On peut vouloir sauvegarder l'état de la mémoire vive pour revenir dans les mêmes conditions de travail après un sommeil. Ce sommeil peut être simple "mise en veille" et dans ce cas la mémoire vive reste alimenté (donc il reste une consommation). On peut vouloir une veille prolongée, sans consommation électrique. Dans ce dernier cas, la mémoire vive est enregistré dans la zone d'échange. On conseille alors de la dimensionner à 1,5 fois la taille de la mémoire vive. Si vous disposez de 64 Go de mémoire vive, il faut prévoir 96 Go de partition d'échange sur votre disque. Éteindre est dans ce cas plus rapide que la mise en veille prolongée.

23.5 Morcellement de la mémoire

Nous nous rappelons le problème d'inter-blocage posé par le dîner des philosophes. Nous allons présenter un cas semblable de "famine" (les processus n'ayant plus de mémoire à "manger" pour pouvoir vivre).

Prenons maintenant la situation suivante. Nous disposons d'un serveur avec une mémoire de taille 12 unités. 2 groupes (G1 et G2) de 4 étudiants lancent des processus qui vont s'alterner sur ce serveur. Le premier groupe (G1) a besoin de mémoire de taille 1 et le second (G2) de taille 2.

La moitié des étudiants terminent leur processus. Une partie de mémoire se libère. Sa taille est de $2 * (1 + 2) = 6$.

Un nouveau groupe de 2 étudiants veut lancer un processus (respectivement A et B) qui utiliseront chacun une mémoire de taille 3.

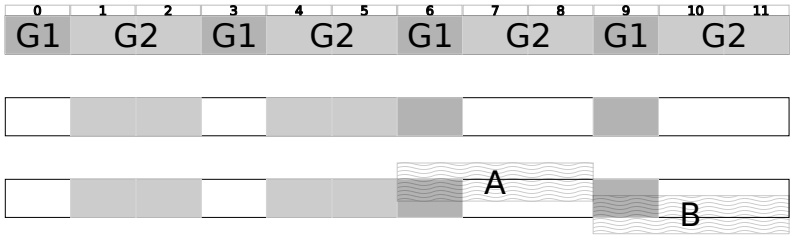


FIGURE 16 – Exemple de répartition de mémoire

En théorie, il y a assez de place pour accueillir les processus A et B. En pratique, les espaces libres contigus ne suffisent pas.

Ne peut-on pas découper la mémoire ?

Ce n'est pas si simple. Imaginez que vous avez écrit un document de 10 pages. Vous devez y ajouter un graphique de 10 cm, mais il ne reste plus que 1 cm de libre par page. Allez-vous découper le graphique en 10 morceaux de 1 cm ?

Il en est de même pour les processus. Au moment de la compilation les sauts conditionnels, les sauts de boucle et les appels à des sous-routines ont été figés. Le début du programme est supposé commencer à l'adresse 0. (Nous passons sous silence le problème des données ou du tas).

Lorsque l'adresse 0 est prise, on choisira un autre bloc de mémoire continue (capable d'accueillir le processus). On peut repérer tous les sauts dans un programme et au moment de l'exécution les décaler.

Le problème est bien plus complexe, si le processus est morcelé.

C'est pourtant ce que l'on envisage de faire avec 2 techniques :

- la segmentation,
- la pagination

24 Mémoire segmentée

Le problème de découpage d'un processus ne se pose pas si on trouve un emplacement mémoire suffisamment grand pour répondre à ses besoins. Seul ceux qui "débordent" nous intéressent.

La première approche est de découper au minimum le processus en commençant par la place la plus adaptée (la plus proche de sa taille et qui pourrait le contenir). Si le processus dépasse, on le découpe en deux et on réitère l'opération avec ce qui reste à placer. Si le reste rentre on a terminé.

Pendant ce découpage on maintient une table de correspondance. Voici celle de l'exemple donné :

processus	adresse processus	taille segment	adresse mémoire
1 (G2)	0	2	1
2 (G1)	0	1	6
3 (G2)	0	2	4
4 (G1)	0	1	9
5 (A)	0	2	7
5 (A)	2	1	0
6 (B)	0	2	10
6 (B)	2	1	3

TABLE 5 – Table de segmentation

Ce qui nous donne pour le même exemple simplifié une organisation comme celle de la figure suivante.

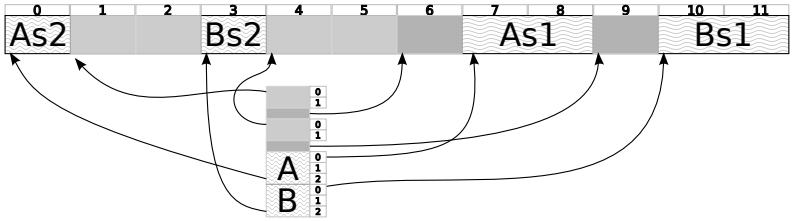


FIGURE 17 – Mémoire segmentée

Nous n'avons placé que quelques chiffres dans cet exemple. Cela ne doit pas cacher la complexité de l'algorithme pour retrouver une adresse de processus en mémoire. Pour trouver une mémoire à l'adresse A du processus P

- Recherche dans la table de segment S qui possède la plus grande "adresse processus" et qui soit inférieur à A
- La mémoire se trouve à "adresse mémoire" de S + A - "adresse processus"

En voici une illustration :

processus	adresse processus	taille segment	adresse mémoire
P	0x10543	0x567	0x67321

Table 6 – Calcul de segment

Pour le calcul de l'adresse $0x12345$ on vérifie $0x12345 > 0x10543$ et $0x12345 < (0x12345 + 0x567)$ puis on calcule $0x67321 + 0x12345 - 0x10543$.

Faisons quelques remarques.

- Cet exemple est trivial. En réalité, les adresses de processus sont représentées par des nombres hexadécimaux sur 32 voire 64 bits. La taille des programmes est quelconque. Les trous qu'ils feront en disparaissant ou en allant en zone d'échange, sont également de taille quelconque.
- Les segments qui résultent des trous peuvent avoir n'importe quelle taille eux aussi (impair, non multiple d'une puissance de 2 ...).
- lors de l'utilisation de la mémoire d'échange, nous imaginons sauver un segment complet. Mais au moment de la récupération en mémoire vive, il se peut qu'il n'y ait plus l'espace continu suffisant pour remettre le segment et qu'il faille segmenter à nouveau le processus.

Bref, la mémoire peut devenir rapidement un vrai gruyère. En envisageant le pire, on pourrait imaginer des segments de l'ordre de l'octet. Ce ne serait vraiment pas efficace.

Le système a une charge supplémentaire, il doit lors de l'adressage d'un processus calculer sa translation. Pour cela, il doit en premier déterminer quel segment est concerné. Il doit donc parcourir tous les segments précédents pour déterminer en fonction du cumul des tailles, la position du segment.

25 Mémoire paginée

La pagination part du même constat que la segmentation : il faut découper les processus.

La différence majeure est qu'en pagination tous les "bout de mémoire" ont la même taille. On parle alors de "page".

Définition 57 (cadre)**.** Un cadre est une portion de mémoire réelle qui contient une page.

Tous les processus sont systématiquement découpés en page et l'on maintient une table de correspondance. Voici celle de l'exemple donné :

processus	adresse processus	adresse mémoire
1 (G2)	0	4
1 (G2)	1	5
2 (G1)	0	6
3 (G2)	0	1
3 (G2)	1	2
4 (G1)	0	9
5 (A)	0	7
5 (A)	1	8
5 (A)	2	0
6 (B)	0	10
6 (B)	1	11
6 (B)	2	3

Table 7 – Table de pagination

Ce qui nous donne pour le même exemple simplifié une organisation comme celle de la figure suivante.

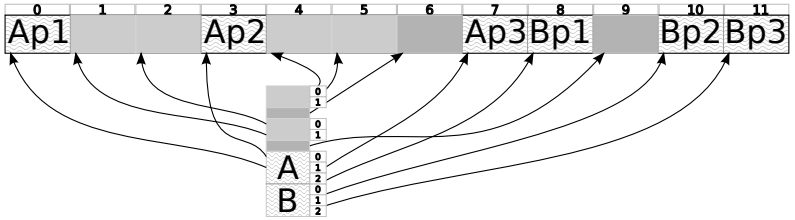


Figure 18 – Mémoire paginée

De nouveau, nous comprenons qu'il y a un compromis entre rapidité et perte de mémoire. En segmentation, la taille mémoire de l'ensemble des segments est exactement celle demandée par le processus. En pagination, la taille mémoire de l'ensemble des pages est l'arrondi supérieur à la page de celle demandée par le processus.

A priori, il y aurait un peu de perte avec la pagination. Mais ce sur-coût sera vite compensé.

Suivant la base utilisée pour l'adressage (donc en base hexadécimale), on utilisera les 2 derniers chiffres pour le contenu de la page et les précédents

pour désigner la page. En voici une illustration :

adresse processus	découpage	numéro de page	décalage dans la page
0xXXXYY	0xXXX 0xYY	0xXXX	0xYY
0x12345	0x123 0x45	0x123	0x45

TABLE 8 – Calcul de page

Nous remarquons la simplicité de l'algorithme de calcul par rapport à celui de la segmentation.

En réalité, la taille des pages n'est pas nécessairement de 8 bits (1/4 de Ko). Il suffit qu'elle soit une puissance de 2 (mots de 9, 11 ou 12 bits).

Faisons quelques remarques.

- Il y a des pertes de mémoire pour arrondir à des pages entières (négligeables par processus).
- La recherche de pages libres est rapide (il suffit qu'elles ne soient pas utilisées dans la table).
- Il n'y a pas d'apparition de "trou" qui ne pourrait pas être utilisé comme une page.
- Il n'y a pas de sur-segmentation lors d'un passage en mémoire d'échange.

26 Optimisations

Le passage à la mémoire virtuelle qu'elle soit segmentée ou paginée a apporté des caractéristiques nouvelles.

26.1 Taille mémoire des processus

Par exemple, la mémoire physique peut être plus petite que la mémoire utilisée par le processus.

Imaginons que l'on demande au système la réservation de mémoire dans le tas pour des objets de la taille d'une page. La libération de certains objets va libérer les pages correspondantes en mémoire qui pourront être réutilisées pour d'autres processus. Cependant, il n'est pas utile de compacter la mémoire (il est même impossible de décaler les adresses des objets déjà utilisés par le processus). La mémoire des processus peut donc être plus grande que la mémoire réellement utilisée.

Nous pouvons imaginer généraliser le procédé. Lorsqu'un processus demande la réservation d'un objet de plusieurs pages (implicitement initialisé à 0), ce ne sera qu'au moment de l'accès à l'objet que les pages seront créées.

26.2 Calcul d'adresse

Des composants électroniques spécialisés participent au calcul à la volée de la translation d'adresse de la mémoire. Le processeur ne fait en réalité aucun calcul qui retarderait de plusieurs cycles d'horloge la manipulation des données. L'ordonnanceur qui gère l'affectation des processus va déclarer auprès de composants la répartition de la mémoire. La translation des adresses se fera sur le bus de données de manière transparente pour le processeur.

26.3 Partage de codes

Lorsque des pages ne sont accessibles qu'en lecture par plusieurs processus, il devient pratique, avec le mécanisme de pointeurs de la mémoire virtuelle vers la mémoire réelle, de partager des informations communes.

C'est le cas pour l'instruction "exec". Le système d'exploitation tient à jour la liste des programmes déjà lancés et quels processus les utilisent. Il va compter le nombre d'utilisations.

- À la première demande le pointeur est initialisé et le compteur mis à 1.
- À chaque lancement supplémentaire du même programme, le pointeur est réutilisé et le compteur incrémenté.
- À chaque mort de processus, le compteur est décrémenté.
- Arrivé à 0 la mémoire est libérée.

26.4 Chargement à la demande

Il n'est pas toujours nécessaire de charger tout un processus en mémoire. Les parties de code d'un programme sont utilisées différemment tout au long de la vie d'un processus. Certaines ne sont utilisées qu'au début (initialisation), d'autres se voient occupées par période (interaction à l'utilisateur, échange par fichiers. . .).

Les pointeurs de mémoire virtuelle ne réfèrent pas toujours la mémoire vive, ils peuvent faire référence à une information sur disque (partie de code non encore chargée) ou sur la mémoire d'échange (données libérées pour un autre processus).

Au moment de l'accès, l'électronique va produire une interruption "défaut de page" qui sera interceptée par le système pour résoudre la recherche d'informations selon les stratégies qu'il a implantées.

26.5 Partage de données

Il n'y a pas que le code qui puisse être partagé entre processus. Étonnamment, bien qu'elles soient modifiables et potentiellement uniques, les données aussi peuvent être partagées pendant une partie de leur vie. En effet, l'instruction "fork" doit virtuellement cloner (dupliquer) tout un processus : code, donnée, pile, tas.

Plutôt que de passer du temps à recopier réellement les informations, le système va marquer les pages de code comme étant utilisées une fois de plus. Il va aussi marquer toutes les pages de données comme nécessitant une copie à l'écriture et augmenter le compteur de copies. Supposons qu'un programme déclare une variable "i" puis fork 2 fois. Il y aura sur la page de données un compteur de copies en écriture à 3. Le premier processus qui modifiera sa variable voyant que le compteur n'est pas unique, va recréer une page par copie et mettre le compteur à 1. Il va également décrémenter le compteur de la page d'origine. Le dernier processus qui modifiera sa variable aura un compteur nul et ne fera donc aucune copie.

Vous comprenez que les concepts les plus coûteux en théorie (copie de tout un processus utilisant beaucoup de mémoire) peuvent dans les faits être efficaces.

26.6 Mise en mémoire secondaire

Nous avons vu qu'il est possible de placer des informations en mémoire d'échange (zone de *swap*). C'est particulièrement pertinent si l'information se trouve dans une portion du processus qui ne sera plus utilisée avant longtemps. En réalité, cette réflexion pose un problème générique. Comment choisir l'information qu'il faut faire passer d'une mémoire de grande capacité mais lente, vers une mémoire réduite mais très rapide. C'est la préoccupation de tous les mécanismes de mémoire cache d'un processeur.

La difficulté est bien entendu pour disposer de la page utile à l'instant "t", de l'échanger avec une autre page que l'on n'utilisera plus avant longtemps. Ou autrement dit, de ne conserver que les pages qu'on utilisera le plus souvent par la suite.

Définition 58 (Défaut de page)**.** On appelle "défaut de page" le fait de ne pas disposer de l'information indispensable en mémoire principale et de devoir la charger à partir d'une mémoire secondaire.

Il existe plusieurs algorithmes pour choisir la page à supprimer. Nous allons en comparer 4 avec la séquence suivante de 24 demandes de page de processus :

ordre	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
proc.	6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2

TABLE 9 – Séquence de pages de référence pour comparaison

- **optimal** : "on refait le match". On enregistre toutes les demandes de page et on réfléchit en fonction de l'avenir que l'on peut prédire (puisque l'on a tout enregistré). Lorsque l'on doit faire sortir une page, on choisit celle qui ne sera plus utilisée pendant la plus grande période de temps. C'est un cas théorique que l'on ne peut jamais atteindre (la divination n'est pas une démarche très scientifique).

proc.	6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2
def. **9** sup.	6	7	4				3 4					5 3		2 6					1 7		4 1		3 5	
Optimal	6 X X	6 7 X	6 7 4	6 7 4	6 7 4	6 7 4	6 7 3	6 7 3	6 7 3	6 7 3	6 7 3	6 7 5	6 7 5	2 7 5	2 7 5	2 7 5	2 7 5	2 7 5	2 1 5	2 1 5	2 4 5	2 4 5	2 4 3	2 4 3

TABLE 10 – Algorithme "Optimal"

- **FIFO** : "premier arrivé, premier sorti". Les pages sont placées dans l'ordre d'arrivée et sont sorties dans le même ordre.

proc.	6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2
def. **13** sup.	6	7	4				3 6		6 4			5 7		2 3				7 6	1 5	5 2	4 7		3 1	2 5
FIFO D P	X X 6	X 6 7	6 7 4	6 4 7	6 4 7	6 4 7	4 7 3	4 7 3	7 3 6	7 3 6	7 3 6	3 6 5	3 6 5	6 5 2	6 5 2	6 5 2	6 5 2	5 2 7	2 7 1	7 1 5	1 5 4	1 5 4	5 4 3	4 3 2

TABLE 11 – Algorithme "FIFO"

- **LRU** (en anglais *Least Recently Used*) : "la moins récemment utilisée". Si la page est déjà dans le cache, on la remet en tête de file. Sinon on supprime la dernière page de la file et on place en tête la nouvelle page qui manquait. Cet algorithme est le plus souvent utilisé.

proc.		6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2
def. **13**		6	7	4				3		6			5		2				7	1	5	4		3	2
sup.								6		7			4		3				6	5	2	7		1	5
LRU	A	X	X	6	6	6	6	7	7	4	4	4	6	3	5	6	2	6	5	2	5	2	7	1	5
		X	6	7	4	4	7	4	4	3	3	6	3	5	6	2	5	5	2	7	1	5	4	4	3
	N	6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2

TABLE 12 – Algorithme “LRU”

- **aléatoire** : comme son nom l’indique. Si la page est dans le cache on l’utilise. Sinon on tire aléatoirement la page à supprimer pour la remplacer par la nouvelle. Évidemment, conserver des pages aléatoires est plus efficace que de ne pas conserver de pages. Et curieusement, on ne perd pas à tous les coups statistiquement. Cependant, les résultats ne sont pas reproductibles.

proc.		6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2
def. **12**		6	7	4				3					5		2				7	1	5	4		3	2
aléa		1	2	1	1	0	2	1	2	1	0	2	1	1	0	2	1	0	1	0	2	1	2	2	0
sup.								7					3		6				5	2	4	7		5	1
Aléa		X	X	6	6	6	6	6	6	6	6	6	6	6	2	2	2	2	2	1	1	1	1	1	2
		X	6	7	7	7	7	3	3	3	3	3	5	5	5	5	5	5	7	7	7	4	4	4	4
		6	7	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	3	3

TABLE 13 – Algorithme “Aléatoire”

- **LFU** (en anglais *Least Frequently Used*) : “la moins souvent utilisée”. Plus complexe, il faut un compteur à chaque page du cache. Si la page est dans le cache, on incrémente son compteur. Sinon on choisit de supprimer le cadre correspondant au compteur le plus faible (il peut y en avoir plusieurs) et on met la page à la place en remettant le compteur à zéro. L’inconvénient est que des pages peuvent être fréquemment utilisées au lancement du processus et que leur compteur soit si élevé qui ne soit plus possible de la remplacer.

proc.		6	7	4	7	4	4	3	3	6	6	3	5	6	2	5	2	2	7	1	5	4	4	3	2
def. **14**		6	7	4				3		6			5	6	2	5	2		7	1	5				2
sup.								6		7			6	5	6	2	5		2	7	1				5
compteur	1	1	1	1	1	1	1	2	2	1	2	2	3	1	1	3	1	1	3	2	1	3	1	1	4	4
	0	1	1	2	2	2	2	2	1	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1
	0	0	1	1	2	3	3	3	3	3	3	3	1	1	3	1	1	3	1	1	3	1	4	5	5	5
LFU		6	6	6	6	6	6	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
		X	7	7	7	7	7	7	7	6	6	6	5	6	2	5	2	2	7	1	5	5	5	5	2
		X	X	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

TABLE 14 – Algorithme “LFU”

26.7 Anomalie de Bélády

Il ne faut pas croire qu’en augmentant la mémoire réelle on résout systématiquement le problème de défaut de page virtuelle. Bien sûr, en allouant plus de mémoire virtuelle que vos processus en ont besoin, vous devriez ne pas avoir de défaut de page (peut-on encore parler de cache dans ce cas ?).

Cependant, László Bélády (informaticien hongrois) a démontré que des situations particulières contredisaient l’utilisation de l’algorithme FIFO.

En effet, la séquence (3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4) produit 9 défauts de page avec 3 cadres et 10 défauts de page avec 4 cadres.

Voici une illustration de l’anomalie avec la même représentation que précédemment :

demandé		3	2	1	0	3	2	4	3	2	1	0	4
défaut	**9**	3	2	1	0	3	2	4			1	0	
supprimée					3	2	1	0			3	2	
FIFO 3	D	X	X	3	2	1	0	3	3	3	2	4	4
		X	3	2	1	0	3	2	2	2	4	1	1
	P	3	2	1	0	3	2	4	4	4	1	0	0
défaut	**10**	3	2	1	0			4	3	2	1	0	4
supprimée								3	2	1	0	4	3
FIFO 4	D	X	X	X	3	3	3	2	1	0	4	3	2
		X	X	3	2	2	2	1	0	4	3	2	1
		X	3	2	1	1	1	0	4	3	2	1	0
	P	3	2	1	0	0	0	4	3	2	1	0	4

TABLE 15 – Anomalie de Bélády

27 TD3 : Mémoire

Dès votre arrivée en TD, pensez à déposer votre QCM rempli à votre encadrant. La partie à rendre est la dernière page de votre document (à découpe suivant les pointillés).

Après l'appel, c'est l'occasion de poser les questions concernant des éléments non compris du cours. Pensez à recopier le corrigé du QCM sur votre document (partie à concerver à la page précédante). Vous pourrez l'utiliser le jour de l'examen. Une page de notes personnelles est à votre disposition en fin de TD.

L'objectif du TD est de comparer différentes gestions de mémoire virtuelle.

Nous partons du shéma suivant : un disque dur échange avec une mémoire vive qui échange avec une mémoire cache qui échange avec le processeur.

Il ne serait pas raisonnable de travailler des Go de données. Nous partirons sur une taille fictive de mémoire vive de 4Ko. La mémoire cache est constituée de 4 pages indépendantes de 256 octets.

Nous noterons les adresses et taille en hexadécimal (c'est plus facile).

Il y a déjà des processus chargés en mémoire. Voici la liste des zones occupées :

début	fin	taille	(libre)
0121	01DF	BF	(01E0)
0403	043A	38	(043B)
0460	0497	38	(0498)
04C5	04FC	38	(04FD)
061E	06DD	C0	(06DE)
0A1D	0A5E	42	(0A5F)
0A82	0AC3	42	(0AC4)
0E04	0EC5	C2	(0EC6)

Nous voulons placer 4 processus :

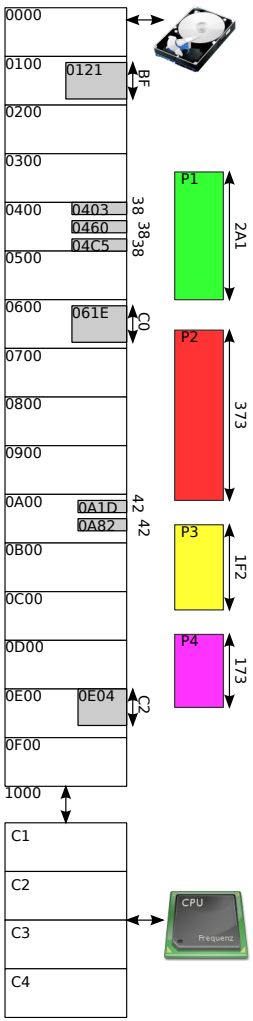


FIGURE 19 – Zone mémoire

processus	taille
P1	2A1
P2	373
P3	1F2
P4	173

28 Placement de processus

Le 1er exercice consiste à placer des processus

Placez les nouveaux processus dans une mémoire non virtuelle suivant le format : ✍

processus	@ mémoire	taille
P1		
P2		
P3		
P4		

Question : Que se passe-t-il ?

Réponse : ✍

Placez les nouveaux processus dans une mémoire segmentée et donnez la liste des segments suivant le format : ✍

processus	@ processus	@ mémoire	taille
P1	0000	0000	121
P1	0121		

Placez les nouveaux processus dans une mémoire paginée et donnez la liste des pages suivant le format : ✍

processus	@ processus	n° page	taille
P1	0000	0	100
P1	0100		

29 Calcul d'adresse mémoire

Donnez l'adresse mémoire non-virtuelle correspondant à l'adresse processus. ✍

proc.	P1	P1	P2	P2	P2	P3	P4
@ proc.	0000	0140	0000	0040	00C0	0000	0000
@ mem.							

Donnez l'adresse mémoire segmentée correspondant à l'adresse processus (avec le décalage dans le segment). ✍

proc.	P1	P1	P2	P2	P2	P3	P4
@ proc.	0000	0140	0000	0040	00C0	0000	0000
@ seg.							
@ off.							
@ mem.							

Donnez l'adresse mémoire paginée correspondant à l'adresse processus. ✍

proc.	P1	P1	P2	P2	P2	P3	P4
@ proc.	0000	0140	0000	0040	00C0	0000	0000
@ mem.							

30 Calcul d'adresse processus

Donnez le nom et l'adresse processus correspondant à une mémoire non-virtuelle. ✍

@ mem.	0100	0300	0500	0900	0B00	0F00
proc.						
@ proc.						

Donnez le nom et l'adresse processus correspondant à l'adresse mémoire segmentée ✍

@ mem.	0100	0300	0500	0900	0B00	0F00
@ seg.						
@ off.						
proc.						
@ proc.						

Donnez le nom et l'adresse processus correspondant à l'adresse mémoire paginée ✍

@ mem.	0100	0300	0500	0900	0B00	0F00
proc.						
@ proc.						

30.1 Comparaison

Combien de mémoire reste-t-il pour la mémoire : ✍

- non-virtuelle
- segmentée
- paginée

Question : Quels sont les avantages et inconvénients des différents types de mémorie ?

Réponse : ✍

Question : Qui écrit les algorithmes et qui supporte le coût de la mémoire ?

Réponse : ✍

31 Défaut de page

Le taux de défaut de page est le nombre de défauts sur le nombre de pages accédées.

Le processeur accéder aux adresses mémoire suivantes : 0858 0385 028A 0D00 0243 0233 06A6 02BB 025A 0E18 0B32 0951 0B32 075F 07C0 0DF4 012C 005A 0535 0F5D

Donnez la liste des pages correspondantes aux adresses. ✍

858	385	28A	D00	243	233	6A6	2BB	25A	E18	B32	951	B32	75F	7C0	DF4	12C	05A	535	F5D

Question : Donnez le taux de défaut de page en OPT. ✍

page.																			
def. sup.																			
- - - -																			

Réponse : ✍

Question : Donnez le taux de défaut de page en FIFO. ✍

page.																			
def. sup.																			
- - - -																			

Réponse : ✍

Question : Donnez le taux de défaut de page en LRU. ✍

page.																			
def. sup.																			
- - - -																			

Réponse : ✍

Question : Donnez le taux de défaut de page en LFU. ✍

page.																			
def. sup.																			
c c c c																			
- - - -																			

Réponse : ✍

Page de notes personnelles : ✍

32 TP3 : Mémoire

- Votre compte-rendu est à rendre en fin de TP.
- Vous ne devez pas rendre un listing, mais un document répondant aux questions posées.
- Vous préparerez le compte-rendu à l'avance (copier-coller les questions) pour gagner du temps.
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois.
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (uniquement la fenêtre concernée).
- N'oubliez pas de rendre votre TP imprimé en 2 colonnes de texte par côté de papier et de faire le TP en binôme.

L'objectif du TP est de faire des mesures statistiques sur différentes techniques de gestion de cache mémoire.

On prendra les valeurs suivantes :

- C1 = 24 (taille mémoire en page)
- C2 = 3 (taille du cache en page)
- C3 = 128 (nombre d'accès de page)
- C4 = 12 (nombre de test d'un algorithme)

Les programmes seront écrits en Java.

On commencera par une réalisation par tirage aléatoire.

Question : Écrivez une classe pour préparer les tests avec les caractéristiques suivantes :

- un ensemble de numéros de pages (représentant la mémoire cache)
- un constructeur paramétré pour limiter l'ensemble précédant (C2)
- une méthode tirant un entier positif suivant une valeur limite (C1)
- méthode de choix de page à supprimer. Elle prend en argument un numéro de page (C1) et vérifie qu'elle n'est pas en cache. Si elle l'est elle retourne -1 sinon elle tire une position de cache au hasard (inférieur à C2).
- une méthode mettant à jour le cache si une page est à supprimer. Au départ les caches sont vides, ils ne contiennent pas de numéro de page (valeur -1).

Donnez le code et les traces.

Réponse : ✍

Question : Ajoutez une fonction de test qui réalise C3 demande de pages consécutives et qui compte le nombre de défauts de page (donc différent de -1). Le taux de défauts de page est le nombre de défaut sur le nombre de demandes.

Donnez le code et des traces.

Réponse : ✍

Question : Faites une mesure statistique du taux de défaut de page en répétant C4 fois l'opération. Vous retirerez la valeur minimum et maximum avant de donner la moyenne.

Donnez le code et des traces.

Réponse : ✍

Question : Dérivez la classe pour implanter l'algorithme FIFO.

Donnez le code et les traces.

Réponse : ✍

Question : Dérivez la classe pour implanter l'algorithme LRU.

Donnez le code et les traces.

Réponse : ✍

Question : Comparez les résultats

Réponse : ✍

Question : Refaites les tests avec C2 = 4. Comparez les résultats.

Réponse : ✍

Cinquième partie

Les tâches

Nous avons déjà abordé la question des activités avec la notion de processus. Dans ce chapitre, il s'agit d'un niveau plus fin d'activité. On parle alors : de tâches, de processus légers ou allégés, de fil d'activité, d'exécution, d'instructions ou bien encore d'unité de traitement ou d'exécution.

Définition 59 (tâche)**.** Une tâche (en anglais *thread*) est une suite séquentielle d'instructions qui s'écoulent dans le même temps (donc indépendamment) que d'autres traitements au sein d'un même espace d'adressage.

En première approximation les tâches sont des "processus" dans un processus. Alors, à quoi servent-elles et quelles sont les différences avec des processus ?

33 Comparaison entre tâches et processus

Il est fréquent d'avoir besoin de réagir à des évènements désynchronisés. Habituellement, nous envisageons des déroulements en séquence pour un programme. Mais, l'environnement extérieur impose d'autres rythmes. Par exemple, en cas de sollicitations du réseau, un serveur n'a pas connaissance a priori du nombre ni de la fréquence des requêtes des clients. De même, lors de développement d'une interface graphique, les interactions avec l'utilisateur (clavier, souris) et avec le gestionnaire de fenêtres (recouvrement, transparence avec d'autres applications) s'entrelacent dans une logique différente de celle du programme.

On trouvera donc fréquemment des tâches pour gérer les :

- demandes de connexions
- requêtes de clients connectés
- évènements graphiques
- modifications du modèle de données

Une différence importante entre tâche et processus provient du mode de communication entre eux.

- Entre processus, les espaces d'adressage sont clairement isolés. Le système est l'intermédiaire incontournable et contrôle les interactions. Nous les avons cités au chapitre "Partage des ressources" : IPC, pipe, signaux, fichiers . . .
- Entre tâches d'une même application, il y a partage du même espace d'adressage. Le système ne contrôle pas l'utilisation simultanée des échanges d'informations entre tâches. L'accès est plus rapide, mais c'est au développeur d'en gérer le contrôle et la cohérence. Nous aborderons donc dans ce chapitre les questions de synchronisation.

33.1 Mécanismes historiques

Les mécanismes permettant de gérer des tâches en parallèle sont les mêmes que pour les processus. Ils sont apparus en même temps que les systèmes d'exploitation.

- En premier le plus simple, la notion de co-routines. Il s'agit en fait d'un traitement dont on pense qu'il se termine en un temps court. Au lieu de faire une boucle sans fin sur le traitement d'un évènement graphique, on ne traite que le premier et on retourne. L'application passera son temps à donner la main à un ensemble de co-routines. Le fait de ne pas bloquer induit une attente active.
- Vient ensuite la notion de préemption. On déclare auprès du système une alarme déclenchée par un chronomètre. Il faut alors sauvegarder le contexte d'exécution interrompue et vérifier que les données sont dans un état stable compatible à un changement de traitement. L'attente active est réduite par des périodes de sommeil. Mais dans le cas où il ne se passe rien, l'application est tout de même réveillée.
- En dernier, on a souhaité bénéficier de l'ordonnanceur de processus du système et de sa performance. Ce phénomène s'accroît avec l'arrivée des multi-processeurs et multi-cœurs. Lorsque vous êtes seuls sur une machine octo-processeurs, que vous avez conçu un programme en Java avec des traitements parallèles, il serait dommage de ne pas jouir pleinement de cette architecture.

À titre d'exemple, l'ordonnanceur d'une Machine Virtuelle Java (JVM) peut déclarer au système Linux, les tâches engendrées par un processus (tâches elles-mêmes décrites dans un programme Java ou Scala).

Il existe plusieurs niveaux d'intégration des tâches pour le développeur.

- Extérieur au système. Le développeur écrit des co-routines. Mais cette solution est inefficace du fait de l'attente active.

- Au niveau d'appels au système d'exploitation. Le développeur écrit des fonctions et doit les enregistrer auprès du système. Il devra mettre en place le contrôle nécessaire et prendre les décisions au niveau le plus bas.
- Au niveau de librairie. Dans ce cas, des composants logiciels ont été réalisés pour les cas les plus fréquents. Le développeur doit alors spécialiser des mécanismes génériques.
- Au niveau du langage. Là, le traitement parallèle est intégré au développement. C'est le cas de Java ou de Scala, deux langages qui produisent du "bytecode" interprété par une même Machine Virtuelle Java (JVM). Des mots clefs et une syntaxe facilitent la rédaction et permettent la manipulation de concepts au plus haut niveau (exceptions, piles séparées d'exécution, synchronisations, priorités, ramasse-miettes ...).

Sous Linux, vous pouvez voir les tâches avec l'option *-T* de la commande *ps*. L'option fait apparaître une colonne supplémentaire "SPID". Lorsque que le processus n'a qu'une tâche le "PID" et "SPID" sont identiques. On peut considérer que le "SPID" est un entier unique obtenu à partir d'un compteur et que le "PID" prend la valeur du "SPID" de la tâche principale.

Pour une approche plus concrète, la suite du chapitre sera illustrée par des exemples en Java. Pour voir les processus et tâches Java vous pouvez lancer la commande Unix : "jps".

34 Création de tâches

Il n'existe qu'une seule façon de créer une tâche en Java :

- créer un objet de la classe *Thread*
- invoquer sur cet objet la méthode *start*

La méthode `start` n'est bien évidemment pas bloquante. L'appelant n'attend donc pas le résultat du nouveau traitement engagé. Aussi la méthode `run` ne retourne aucune valeur (type `void`).

De façon symétrique, pour que la méthode `start` soit générique, la méthode `run` ne prend également aucun paramètre formel. Il est possible d'utiliser des attributs dans la classe définissant la méthode `run` pour initialiser le traitement réalisé dans la tâche.

La séquence d'instructions que doit réaliser l'activité est rassemblée dans la méthode de nom *run*

Méthode run

Vous ne devez jamais invoquer directement la méthode *run* d'un objet réalisant une tâche. Seule, la méthode *start* fera cette invocation dans une tâche séparée.

En revanche, il existe plusieurs façons d'attacher une méthode *run* à une tâche :

- par héritage de la classe *Thread*
- par réalisation de l'interface *Runnable*

Nous allons maintenant détailler ces différentes façons.

34.1 La classe Thread

La façon la plus directe de créer une tâche est de spécialiser la classe "Thread".

Dans le listing 61, la classe *SimpleThread* est une nouvelle classe d'activités. Son comportement est spécialisé à la ligne 4. Dans l'exemple, il n'y a qu'une instruction. Bien évidemment l'ensemble du corps de la méthode *run* est exécuté dans la tâche quelle que soit sa longueur et sa durée.

Pour tester la tâche, nous avons placé dans une méthode *main* les 2 étapes :

- création de l'objet tâche en ligne 9 et
- lancement de l'activité proprement dite en ligne 12.

```
1  public class SimpleThread extends Thread {
2
3      public void run () {
4    System.err.println ("Here is "+getName ());
5      }
6
7      static public void main (String[] arg) {
8    System.err.println ("new");
9    Thread thread = new SimpleThread ();
10
11   System.err.println ("start");
12   thread.start ();
13     }
14 }
```

Listing 61 – Héritage de Thread

34.2 L'interface Runnable

Java n'offre pas d'héritage multiple. Si l'on souhaite qu'un objet ait une activité, il est toujours possible de lui faire implanter l'interface "Runnable".

Dans le listing 65, la classe *SimpleRunnable* n'est pas une activité. Elle n'est donc pas contrainte par un héritage spécifique. En revanche, l'implantation de l'interface *Runnable* lui impose de décrire une méthode *run* qui sera éventuellement invoquée par une tâche.

Pour tester notre tâche, nous avons placé dans une méthode *main* les 3 étapes :

- création de l'objet de notre application qui pourra dérouler une tâche en ligne 9,
- création d'une tâche générique qui activera le comportement de notre objet en ligne 10 et
- lancement de l'activité proprement dite sur la tâche qui déroulera le comportement de notre objet en ligne 13.

```
1  public class SimpleRunnable implements Runnable {
2
3      public void run () {
4    System.err.println ("Here is "+Thread.currentThread ().getName ());
5      }
6
7      static public void main (String[] arg) {
8    System.err.println ("new");
9    Runnable runnable = new SimpleRunnable ();
10   Thread thread = new Thread (runnable);
11
12   System.err.println ("start");
13   thread.start ();
14     }
15 }
```

Listing 62 – Implantation de Runnable

Nous pouvions lancer directement une tâche générique anonyme avec l'instruction : `(new Thread (runnable)).thread.start ();`

34.3 Les tâches anonymes

Une solution intermédiaire consiste à ne pas hériter de la classe "Thread" et ne pas créer une nouvelle classe dans l'arbre de nos développements. On peut utiliser une classe anonyme spécialisée, qui hérite de la classe "Thread" mais redéfinit à la volée la méthode "run".

Dans le listing 63, commençons par décrire la ligne 5. Les parenthèses vides indiquent un appel au constructeur d'une tâche. Nous aurions pu fournir des paramètres au constructeur (pour donner un nom à la tâche). La particularité de cette syntaxe est que juste après la liste des paramètres réels on trouve une paire d'accolades qui permet de spécialiser l'objet créé. Ce qui est mis dans les accolades correspond à ce que l'on aurait mis dans une classe dérivée : attributs et méthodes. La classe anonyme ne contiendra qu'un seul objet.

```
1  public class AnonymousThread {
2      static public void main (String[] arg) {
3    System.err.println ("new & start");
4
5    (new Thread () {
6      public void run () {
7          System.err.println ("Here is "+getName ());
8      }
9        }).start ();
10     }
11 }
```

Listing 63 – Tâche en classe anonyme

L'avantage de la construction est de pouvoir disposer d'un nombre illimité de tâches différentes pour un même objet ou une classe d'objets. Cette technique est particulièrement adaptée à la définition à la volée de rétroaction (en anglais *callback*) sur des éléments d'interface graphique (boutons, champs de saisie . . .).

34.4 Autres propriétés

La classe "Thread" dispose d'autres propriétés :

- un constructeur et une méthode *setName* qui permettent de fixer un nom symbolique de la tâche.
- une méthode *getName* qui permet de connaître le nom de la tâche.
- une méthode de classe *sleep* qui permet de demander à s'endormir pendant un temps approximatif.
- une méthode de classe *currentThread* permettant de connaître la tâche qui exécute cette invocation.

Il arrive souvent que l'on souhaite ne voir qu'un seul exemplaire de la tâche que l'on vient de lancer. Par exemple, lorsque l'utilisateur clique frénétiquement sur sa souris, il faut savoir réagir en temps réel.

Définition 60 (temps réel)**.** Pour qu'un développement soit "temps réel", il faut qu'il ignore une action devenue obsolète. L'application est "temps réel", si elle abandonne ce qui ne peut pas être réalisé dans les temps.

Bien entendu, la rapidité du traitement doit être le plus proche possible de ce qui est nécessaire dans le cas général. En revanche, il ne sert à rien de savoir traiter une image d'un film en moins de $1/20^e$ de seconde lorsque la fréquence de défilement passe de 20 à 60 images par seconde. Il est plus utile de savoir abandonner le traitement une fois sur 3 pour conserver la fluidité du mouvement, autrement dit un rendu "en temps réel".

Voici un exemple de code (listing 64) pour ne pas lancer plusieurs fois une même tâche. Le principe est de mémoriser la dernière tâche lancée. Toutes les tâches ne poursuivent leurs traitements que si elles sont bien la dernière lancée.

```
1  public class SingleThread {
2
3      private Thread lastThread;
4      public void stopSingle () {
5    lastThread = null;
6      }
7      public void startSingle () {
8    (new Thread () {
9      public void run () {
10         for (lastThread = currentThread ();
11        lastThread == currentThread ();
12        ) {
13       System.err.println ("Here is "+getName ());
14       pause (1);
15         }
16     }
17       }).start ();
18     }
19
20     static public void pause (int seconds) {
21   try {
22       Thread.sleep (seconds*1000);
23   } catch (InterruptedException e) {
24   }
25     }
26     static public void main (String[] arg) {
27   SingleThread activeObjet = new SingleThread ();
28   for (int i = 0; i < 4 ; i++) {
29       activeObjet.startSingle ();
30       pause (1);
31   }
32   activeObjet.stopSingle ();
33     }
34 }
```

Listing 64 – Limitation à une tâche

Vous percevez tout l'intérêt de la tâche anonyme. En effet, *SingleThread* ne peut hériter de *Thread* sans impliquer l'unicité de la tâche. Car, un nouvel appel à *start* n'aurait aucun effet. Nous souhaitons ici recommencer tout le processus en initialisant la tâche avec les informations du moment (mises à jour par l'utilisateur) et en demandant l'arrêt des anciennes versions.

Enfin, le recours à une implantation de l'interface *Runnable* aurait rendu complexe la phase de lancement d'une nouvelle tâche, en initialisant des identifiants uniques pour comprendre de quelle filiation provenait la tâche pour l'associer à un objet de contrôle.

Notons pour finir, que la ligne 32 non seulement termine les tâches créées, mais arrête du même coup le programme Java.

Terminaison Java

Le processus d'une application Java se termine lorsque toutes ses tâches sont terminées.

La tâche principale d'une application Java invoque la méthode *main* de la classe principale. Après l'exécution de la méthode *main*, elle attend que toutes les tâches secondaires se terminent. Lorsque l'application demande la création d'une fenêtre, une tâche de gestion des évènements graphiques est créée. Comme on ne peut déterminer quand l'utilisateur cessera de jouer avec une application, cette tâche ne se termine jamais et l'application Java de même. Il faut prévoir une action spécifique (bouton *quitter*) pour mettre un terme au programme.

35 Synchronisation

La création de tâche va de pair avec le contrôle de sections critiques. Nous avons vu au chapitre "partage des ressources" comment gérer de telles sections à l'aide de sémaphores. Java a fait un choix plus fin que celui des sémaphores pour synchroniser les méthodes ou l'accès à des attributs d'un objet. Il utilise le concept de "moniteur". L'équivalence entre moniteur et sémaphore a été démontrée. Il ne s'agit donc que d'un choix de facilité d'usage et d'un niveau d'intégration avec le langage.

Commençons par parler de synchronisation Java hors moniteur.

35.1 L'instruction join

Il y a un premier niveau de synchronisation en Java, celui de la vie des tâches : au démarrage et à la fin.

- La synchronisation de démarrage est la conséquence de la méthode "start".
- La synchronisation de terminaison se fait explicitement avec l'appel de la méthode "join".

```java
public class Join {
    static public void pause (int seconds) {
  try {
      Thread.sleep (seconds*1000);
  } catch (InterruptedException e) {
  }
    }
    static public void main (String[] arg) {
  Thread thread = new Thread () {
    public void run () {
        System.err.println ("Here is "+getName
            ());
        pause (2);
    }
      };

  thread.start ();
  pause (1);
  try {
      thread.join ();
  } catch (InterruptedException e) {
  }
    }
}
```

Listing 65 – Synchronisation sur la vie d'une tâche

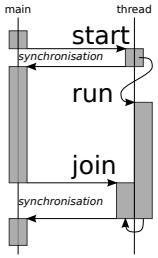


FIGURE 20 – Synchronisation

35.2 Moniteur Java

Définition 61 (moniteur)**.** Un moniteur est un dispositif permettant le contrôle de l'exécution de méthodes particulières d'un objet, en vue de résoudre des problèmes posés par des sections critiques. Les méthodes ne doivent pas s'exécuter simultanément. Pour autant, le dispositif prévoit d'interrompre l'une des méthodes en donnant la main à une autre, puis de revenir sous certaines conditions à la première.

Dans une première approximation, nous dirons que les moniteurs permettent la déclaration de méthodes en exclusions mutuelles. Nous verrons ensuite que cela peut être affiné.

Définition 62 (synchronized)**.** Le mot clef "synchronized" permet de déclarer des traitements en exclusion mutuelle à l'exception de la tâche qui entre en section critique.

Lorsqu'une synchronisation est contrôlée par un moniteur la tâche qui prend la main a un usage exclusif sur l'objet. Cela a deux conséquences :

- Les autres tâches se trouvent bloquées au moment d'invoquer une exécution synchronisée sur le même objet.
- La tâche ayant la main dans le moniteur peut invoquer librement d'autres exécutions synchronisées sur le même objet. En d'autres termes les méthodes synchronisées sont ré-entrantes (elles peuvent être récursives).

C'est en particulier indispensable pour un mécanisme lecteurs/écrivains. Imaginons un objet synchronisé sur lequel le nous avons développé 2 méthodes :

- *insert* en exclusion mutuelle En écriture, elle ne doit s'exécuter avec aucune autre.
- *dicoSearch* partageable En lecture, elle est utilisable par plusieurs lecteurs simultanément.

Nous souhaitons bien un mécanisme lecteurs/écrivains. Malheureusement, nous souhaitons réutiliser la méthode dicoSearch pour savoir où réaliser l'insertion. C'est exactement ce que permettent les moniteurs en autorisant une tâche déjà autorisée à poursuivre ses traitements sur l'objet. Le contrôle lecteurs/écrivains étant différent pour chaque objet (plusieurs objets ayant plusieurs lecteurs en même temps que d'autres objets ont un écrivain).

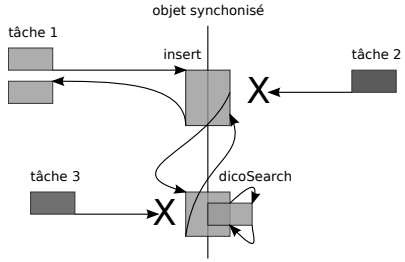


FIGURE 21 – Ré-entrance

35.3 Le qualificatif synchronized

Il y a deux syntaxes différentes : objet complet ou liste de méthode.

Au niveau de l'objet complet, la synchronisation permet de demander au moniteur de l'objet un usage exclusif. Certains parlent de synchronisation par bloc, mais c'est bien le moniteur d'**un** objet qui verrouille l'exécution et pas un bloc. Il n'y a aucune obligation à ce que les méthodes de l'objet soient synchronisées. Si le moniteur est libre, la tâche prend l'exclusivité. Si le moniteur est déjà pris par une autre tâche (synchronisation d'objets ou de méthodes) le moniteur place la tâche en attente. Elle sera réveillée dès que le moniteur sera libéré.

```java
synchronized (this) {
    synchronized (unAutreObjet) {
        // suite d'instruction en exclusion mutuelle
        // aucune autre tache ne pourra utiliser this et unAutreObjet
    }
}
```

Listing 66 – Synchronisation d'objets

Au niveau d'une méthode, la synchronisation permet de demander au moniteur un usage exclusif pour l'objet sur lequel la méthode est invoquée. L'exclusion est donc mutuelle entre toutes les méthodes déclarées *synchronised* dans la classe de l'objet. Si le moniteur et libre, la tâche prend l'exclusivité. Si le moniteur est déjà pris par une autre tâche (synchronisation d'objets ou de méthodes) la tâche est en attente. Elle sera réveillée dès que le moniteur sera libéré.

```
1 public class UneCLasse {
      public void methodeNonSynchrone1 () { /*..*/ }
      public void methodeNonSynchrone2 () { /*..*/ }
      public synchronized void methodeEnExclusionMutuelle1 () { /*..*/ }
5     public synchronized void methodeEnExclusionMutuelle2 () { /*..*/ }
  }
```

Listing 67 – Synchronisation de méthodes

Il existe un 3e cas, la synchronisation des méthodes de classe. Elle revient à la synchronisation en considérant la classe comme un objet. Dans ce cas, le moniteur n'est pas attaché à un exemplaire de la classe mais à la classe elle-même.

Le mot clef *synchronised* seul ne permet que l'exclusion mutuelle, or nous avons vu en TD du chapitre "Partage des ressources" qu'il existe bien d'autres configurations de sections critiques. C'est là qu'intervient le dialogue avec le moniteur d'un objet pour un réglage plus fin.

35.4 Les instructions wait, notify et notifyAll

Lorsque qu'une tâche déroule en exclusivité une méthode d'un objet, elle a le loisir de vérifier les conditions particulières d'utilisation de la ressource. Le calcul peut révéler que son traitement est prématuré. La tâche peut alors demander au moniteur de la mettre en sommeil en section critique.

Le moniteur fait donc la distinction entre :

- **La** tâche active qui a autorisation d'exécution sur un objet
- Les tâches en sommeil qui seront réveillées à l'endroit où elles se sont endormies, à la condition de se retrouver seule tâche synchronisée active sur l'objet.
- Les tâches à évaluer qui font la demande d'entrer en section critique mais qui sont bloquées en attente de terminaison ou de sommeil de la tâche active.

Pour communiquer avec le moniteur d'un objet, il existe 3 méthodes : *wait*, *notify* et *notifyAll*.

Définition 63 (wait)**.** La méthode *wait* permet à une tâche entrée en synchronisation de s'endormir pour laisser une autre tâche entrer en section synchronisée. Dans tous les cas, il n'y a qu'une seule tâche active en section synchronisée.

Une tâche endormie en section synchronisée ne sera jamais réveillée par la terminaison de la tâche active de la même section synchronisée.

Il est possible d'indiquer un délai de garde à la méthode *wait*. C'est à déconseiller car cela a une fâcheuse tendance à sombrer dans l'attente active. Il faut toujours préférer la production d'un évènement de réveil à l'endroit où les conditions changent.

Définition 64 (notify)**.** La méthode *notify* permet de signaler au moniteur d'un objet qu'il devra réveiller *une* tâche en sommeil en section synchronisée. Dans tous les cas, il n'y a qu'une seule tâche active en section synchronisée.

L'instruction notify peut être en premier ou en dernier dans une méthode. Dans tous les cas, l'effet du *notify* n'aura lieu que lorsqu'il n'y aura plus de tâche en section synchronisée sur l'objet qui reçoit le notify.

Notez que *wait* et *notify* peuvent être invoqués en désignant explicitement un objet (qui n'est donc pas nécessairement *this* de la méthode qui invoque).

Définition 65 (notifyAll)**.** La méthode *notifyAll* permet de signaler au moniteur d'un objet qu'il devra réveiller *toutes* les tâches en sommeil en section synchronisée. Dans tous les cas, il n'y a qu'une seule tâche active en section synchronisée. Donc les tâches seront réveillées à tour de rôle.

35.5 Une classe sémaphore avec un moniteur

Puisqu'il y a correspondance entre moniteur et sémaphore, nous donnons le listing 70 qui présente une réalisation d'un sémaphore avec un moniteur.

```
1  public class Semaphore {
2
3      private int value;
4
5      public Semaphore (int value) {
6    if (value < 0)
7        throw new  IllegalArgumentException ("Negative value !");
8    this.value = value;
9      }
10
11     public synchronized void p () {
12   if (value <= 0)
13       try {
14     wait ();
15       } catch (InterruptedException e) {
16       }
17   value--;
18     }
19
20     public synchronized void v () {
21   value++;
22   notify ();
23     }
24 }
```

Listing 68 – Sémaphore avec un moniteur

Notez que l'exclusion mutuelle fournie par le mot clef *synchronised*, sans les méthodes *wait* et *notify*, rend impossible le blocage de la méthode *p* tant que la condition `value > 0` n'est pas remplie.

Pour l'utilisation de *notify* et *notifyAll*, toutes les situations sont à regarder au cas par cas. Le tableau suivant est purement indicatif. Il donne une tendance sur les utilisations les plus probables.

	notify	notifyAll
if – wait	OK	risqué
while – wait	très prudent	prudent

Table 16 – Comparaison notify/notifyAll

Imaginez les différentes configurations avec la classe *Sémaphore* précédente :

- En commentant la bonne ligne dans la méthode *p*

```
11     public synchronized void p () {
12   /* if || while*/
13   // if
14   while
15       (value <= 0)
16       try {
17     wait ();
18       } catch (InterruptedException e) {
19       }
20   value--;
21     }
```

Listing 69 – Réglage *wait* de la classe *Semaphore*

- En commentant la bonne ligne dans la méthode *v*

```
20     public synchronized void v () {
21   value++;
22   /* notify || notifyAll*/
23   // notify ();
24   notifyAll ();
25     }
```

Listing 70 – Réglage *notify* de la class *Semaphore*

36 TD4 : Tâches

Dès votre arrivée en TD, pensez à déposer votre QCM rempli à votre encadrant. La partie à rendre est la dernière page de votre document (à découpe suivant les pointillés).

Après l'appel, c'est l'occasion de poser les questions concernant des éléments non compris du cours. Pensez à recopier le corrigé du QCM sur votre document (partie à concerver à la page précédante). Vous pourrez l'utiliser le jour de l'examen. Une page de notes personnelles est à votre disposition en fin de TD.

L'objectif du TD est de manipuler des tâches au sein d'une même application. La création de tâches qui n'auraient aucune interaction (sauf exception) manque d'intérêt. Nous allons donc mettre en place des synchronisations. Pour gagner du temps en compréhension, nous réutiliserons des exemples vus aux chapitres précédents.

En prenant comme support Java, nous insisterons sur la compréhension du concept de moniteur et de notification.

Question : Ecrivez une tâche qui compte jusqu'à 10 et fait une pause d'une seconde entre chaque iteration(avec une trace).

Réponse : ✍

Question : Est-ce utile que le programme principal attende la fin de la tâche ?

Réponse : ✍

Question : Modifiez le programme principal pour que toutes les 2 secondes, il affiche la valeur du compteur de la tâche.

Réponse : ✍

Question : Comment faire pour que le programme se termine lorsque la tâche est terminée ?

Réponse : ✍

On considère un magasin qui gère des objets avec une certaine valeur (poids, volume...). Nous symboliserons les objets par leur valeur et les gérerons dans une file. La file sera gérée par une structure de taille variable (*Vector*, *ArrayList*...).

Question : Ecrivez une file d'entiers avec une méthode ajouter et une méhode supprimer. Ecrivez une méthode consulter qui retourne la somme des entiers.

Réponse : ✍

Question : Ecrivez les méthodes déposer et retirer et inventaire qui utilisent les méthodes précédentes et qui soient en exclusion mutuelle.

Réponse : ✍

Question : Ecrivez un observateur, un consommateur et un producteur de cette file. L'observateur consulte la somme des valeurs des objets, mais ne modifie pas la file. Le consommateur retire le premier élément. Le producteur ajoute en dernier.

Réponse : ✍

Question : Que se passe-t-il si un consommateur souhaite retirer alors que la file est vide ? Modifiez la file pour résoudre le problème.

Réponse : ✍

Le magasin a des capacités limitées. Il fixe une limite à la somme des objets qui peuvent être déposés.

Question : Modifiez la liste pour qu'elle puisse fixer une valeur maximum pour l'ensemble des objets. Attention, on ne limite pas le nombre des objets.

Réponse : ✍

Pour éviter les erreurs, l'inventaire du magasin est réalisé par plusieurs personnes en même temps.

Question : Comment permettre que la liste supporte plusieurs observateurs simultanément sans qu'il y ait de modification pendant leurs inventaires.

Réponse : ✍

Question : Modifiez la liste en conséquence.

Réponse : ✍

Page de notes personnelles : ✍

37 TP4 : Tâches

- Votre compte-rendu est à rendre en fin de TP.
- Vous ne devez pas rendre un listing, mais un document répondant aux questions posées.
- Vous préparerez le compte-rendu à l'avance (copier-coller les questions) pour gagner du temps.
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois.
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (uniquement la fenêtre concernée).
- N'oubliez pas de rendre votre TP imprimé en 2 colonnes de texte par côté de papier et de faire le TP en binôme.

L'objectif du TP est de manipuler des tâches Java en animant des objets sur un panneau.

Question : Ecrivez une classe panneau qui crée un espace pour afficher des rectangles (largeur, hauteur).

Donnez le code et les traces.

Réponse : ✍

Question : Ecrivez une classe pour créer un rectangle (couleur, largeur, hauteur). Et affichez-en plusieurs sur le paneau.

Donnez le code et les traces.

Réponse : ✍

Question : Créez une activité pour déplacer chaque rectangle (vitesse horizontale, vitesse verticale). Lorsque le rectangle touche le bord gauche (ou haut) sa vitesse horizontale (ou verticale) devient positif, et réciproquement sur le bord droit (ou bas) elle devient négative.

Donnez le code et les traces.

Réponse : ✍

Question : Ajoutez un identifiant à chaque rectangle et mémorisez les pixels occupés par chacun dans le panneau. Faites en sorte qu'un rectangle ne se déplace que si les pixels correspondants sont libres.

Donnez le code et les traces.

Réponse : ✍

Question : Faites de même pour la création. Le rectangle ne doit apparaître que quand il n'y aura plus de rectangle là où on souhaite le créer.

Donnez le code et les traces.

Réponse : ✍

Question : Permettrez l'arrêter et la relance d'une activité (clic, touche clavier). Il ne devra y avoir qu'au maximum une activité par rectangle.

Donnez le code et les traces.

Réponse : ✍

Sixième partie

Les entrées-sorties

La communication entre sa partie calculatrice et le monde extérieur est indispensable pour donner un sens à l'utilisation d'un ordinateur. Les premiers modes d'échanges consistaient en des cartes perforées. Elles étaient perforées sur des sortes de machines à écrire. Il y eut ensuite des imprimantes, des écrans (uniquement en texte). Aujourd'hui, nous sommes à l'ère de l'USB et du sans-fil.

Les entrées/sorties peuvent être minimalistes. Pour l'asservissement optique d'un appareil photo, les entrées peuvent se résumer à un interrupteur et un capteur donnant une valeur numérique, et les sorties à l'alimentation d'un moteur pas à pas.

Nous parlerons dans ce chapitre des entrées/sorties d'un système d'exploitation classique. Pour autant, nous ne nous limiterons pas à la communication entre l'ordinateur vu comme une boîte noire et le reste du monde. Mais nous engloberons dans cette notion d'échanges toutes les communications avec un processus. Rappelons celles que nous avons déjà mentionnées précédemment :

- les communications entre processus en commençant par les signaux et en incluant les IPC (sémaphores, files de messages, mémoires partagées)
- les échanges avec un être humain : écran, clavier, souris et tous les dispositifs de pointage (surface tactile, manette de jeux . . .)
- les communications via les réseaux filaires ou électromagnétiques
- tous les périphériques en général
- et enfin tous les fichiers en général

La mémoire est sans doute le premier média d'échanges. Pour le grand public, notons que la mémoire vive est passée de 1 Ko en 1981 (avec le ZX81 de 8 bits à 3 MHz) à aujourd'hui plusieurs Go sur des architectures multi-cœurs (64 bits à plusieurs GHz). De même, dans les années 80, pour diminuer les coûts de la micro-informatique grand public, on trouvait des entrées/sorties utilisant la prise Péritel du tube cathodique du poste de télévision et des lecteurs de cassettes audio pour la mémoire de masse.

38 Les périphériques

Bien que les développeurs manipulent les entrées/sorties de manière logique, il existe nécessairement une connexion physique des périphériques.

Certaines connexions sont cachées dans l'unité centrale. À commencer par les composants implantés sur la carte mère qui utilisent des connecteurs (en anglais *slot*) pour :

- les processeurs
- les bancs de mémoire vive
- une pile bouton (pour le fonctionnement d'une horloge temps réel)
- les alimentations (carte mère elle-même, CPU)
- le contrôle des ventilateurs (châssis, CPU)
- une carte graphique de base
- des émetteurs/récepteurs réseaux sans fils
- des extensions génériques (ISA, PCI)
- des extensions graphiques (AGP, PCI express)
- des mémoires de masse (IDE, SATA)
- des prises USB

FIGURE 22 – Connecteurs

D'autres connecteurs sont apparents (principalement au dos de l'unité centrale). Ils servent aux branchements des périphériques extérieurs.

La connaissance des connexions physiques permet la manipulation via le système des périphériques.

Les fichiers spéciaux par caractères (*c*) ou par bloc (*b*) identifient les périphériques par 2 numéros : le majeur et le mineur.

Le n° majeur correspond à un périphérique ou type de connecteur. Il change d'un système à l'autre. Par exemple :

n° majeur	type de périphérique
1	carte mère
4	terminaux virtuels
5	terminaux réels
8	disques durs
11	lecteur CD
29	vidéo
99	port parallèle

TABLE 17 – Exemple de n° majeur

Le n° mineur correspond à une subdivision ou un sous-adressage du périphérique. De même, il varie d'un système à l'autre. La table 18 en donne un exemple.

n° majeur	n° mineur	périphérique
1	1	mémoire
1	3	"dev nul"
1	5	produit des zéros
8	0	sda : le 1er disque en entier
8	1	sda1 : la 1re partition du 1er disque
8	2	sda2 : la 2e partition du 1er disque
8	16	sdb : le 2e disque en entier

TABLE 18 – Exemple de n° majeur/mineur

Utilisation des périphériques

Attention ! Les périphériques par caractère ou par bloc fournissent un accès direct à l'information. Si vous modifiez un octet sur le disque et que dans le même temps le système modifie un fichier, cette dernière aura une répercussion sur une unité d'allocations et donc sur le disque. Il y aura incohérence entre les données que vous et le système modifiez par des canaux différents.

Le système ne peut gérer cette situation, qui revient à synchroniser 2 représentations, dont pour l'une d'elle vous ne donnez pas la signification de vos interventions.

Nous avons vu un cas où le système gère une double représentation : lorsqu'une image ISO (élément de l'arbre des fichiers) est montée sur un répertoire du même arbre avec la commande `mount -loop`. Mais dans cette situation, le système est maître des actions. Il peut les séquencer et mettre à jour l'une en fonction de l'autre.

Heureusement, habituellement nous n'utilisons pas des périphériques directement. Cela peut arriver avec la commande *dd* (vu au chapitre III), qui permet la copie entre périphériques pour des sauvegardes de disque ou la création de clef USB bootable.

39 Lecture/Ecriture sous Unix

Un système d'exploitation met en place des pilotes adaptés à chaque type de périphérique. Il présente à l'utilisateur un haut niveau d'abstraction pour que l'interface ne se limite pas à lire ou écrire des bits dans des mots de communication avec le matériel.

Sous Unix cette encapsulation se fait en utilisant des fichiers. Il n'y a que de rares cas où la communication est différente (répertoire, échange de datagramme par socket).

Pour limiter les accès disques, qui sont lents et qui peuvent faire vieillir prématurément le support (nombre de réécritures limitées pour certaines mémoires), Unix n'écrit pas les données lorsqu'elles sont modifiées.

Par exemple, les fichiers temporaires (justement nommés) sont créés pour un traitement intermédiaire. Leur durée de vie peut être de quelques fractions de seconde. La plupart est détruite avant qu'Unix ait décidé de les enregistrer véritablement sur la partition */tmp* du disque.

La commande `sync` indique au système qu'il doit synchroniser la représentation des disques en mémoire avec les périphériques (écriture des inodes et des unités d'allocation).

39.1 Bufferisation

De même, les modifications intermédiaires d'un fichier ne sont pas immédiatement répercutées. On parle de "bufferisation" (en anglais *bufferization*). Les modifications d'un fichier peuvent ainsi être stockées par ligne (jusqu'à l'arrivée d'un caractère *CR*) ou par bloc d'unités d'allocation (habituellement 1 Ko).

Sous Unix, la fermeture des fichiers est automatique à la fin d'un processus. Les tampons (en anglais *buffer*) sont vidés.

Bufferisation et fork

Le fork est un clônage fidèle de l'ensemble du processus. Les tampons mémoires en font partie. Il est donc utile de s'assurer que tous les tampons mémoires sont vides avant d'invoquer la fonction fork.

Remarquez la ligne 8 du programme au listing 71. Elle permet de choisir entre vider les tampons ou non.

```
1  int main (int argc, char** argv, char** envp) {
   int doFlush = strcasecmp ("avec", argv[1]);
   fprintf (stdout, "%s flush : ", doFlush ? "Avec" : "Sans");
   fflush (stdout);
5
   for (int i = 0; i < 3; i++) {
     fprintf (stdout, "%d", i);
     if (doFlush)
       fflush (stdout);
10   if (fork () < 0)
       error ();
   }
   return 0;
}
```

Listing 71 – Effet de bord des tampons avec fork

Le résultat suivant illustre bien le problème décrit.

```
Sans flush : 012012012012012012012012
Avec flush : 0112222
```

Listing – Trace

Figure 23, lorsque nous lançons un programme (sans redirection dans un pipe ou un fichier) la sortie standard est connectée à l'écran. Elle est donc bufferisée en ligne (attente de retour chariot). Le processus réserve donc dans ses données un espace pour stocker ces caractères. Au moment du *fork*, le buffer est recopié avec son contenu. Comme il n'y a pas de *println*, les tampons seront vidés à la fin des processus. Paradoxalement, à la dernière itération après le *fork*, les processus ne font rien avant de quitter. C'est pourtant le simple fait de duplication qui fait passer le nombre de "012" de 4 à 8.

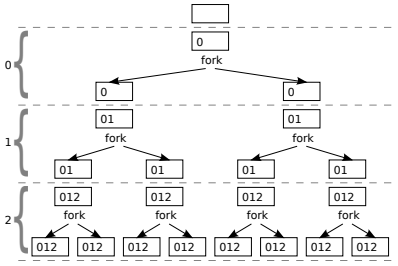


FIGURE 23 – Fork avec tampon (sans *flush*)

Figure 24, lorsque nous prenons garde à vider les tampons avant le clônage, il n'y a aucune question à se poser sur le mode de bufferisation. Bien qu'il y ait au final 8 processus (1 père, 3 fils, 3 petits-fils et 1 arrière-petit-fils), il n'y aura que 7 chiffres écrits.

Pour compter rapidement le nombre de processus, il suffit de se souvenir qu'un fork les double. Si on double 3 fois de suite on en obtient huit fois plus ($2^3 = 8$).

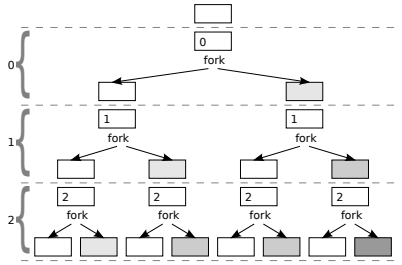


FIGURE 24 – Fork sans tampon (avec *flush*)

Dans le cas où les tampons sont vidés, le dernier *fork* qui ne fait rien n'aura aucun effet de bord sur le résultat.

39.2 Les fichiers

La manipulation d'un fichier s'effectue en 3 étapes :

- l'ouverture qui initialise une position dans le fichier
- les lectures/écritures qui modifient la position
- la fermeture qui videra les tampons d'écriture s'ils existent

L'ouverture se fait suivant un mode qui déterminera entre autres la position du pointeur de lecture. Ce mode se retrouvera dans l'encapsulation faite par des langages de programmation comme C ou PHP. On ne le retrouve pas en Java qui cherche à masquer la dépendance au système d'exploitation sous-jacent.

mode	droit	création	pointeur
r	lecture seule	doit exister	au début
r+	lecture/écriture	doit exister	au début
w	lecture/écriture	efface ou crée	au début
w+	lecture/écriture	efface ou crée	au début
a	lecture/écriture	création éventuelle	à la fin
a+	lecture/écriture	création éventuelle	lecture au début / écriture à la fin

TABLE 19 – Mode d'ouverture de fichier

La section 2 du manuel fournit les explications sur les fonctions de lecture, d'écriture et de modification du pointeur dans le fichier : *read*, *write* et *lseek*, ainsi que sur les caractéristiques du fichier (taille, droits) avec *stat*.

Cela a déjà été noté dans le document à plusieurs reprises. Il ne faut jamais recourir à une attente active même si elle est encadrée par une mise en sommeil. D'ailleurs quelle valeur donner ? Trop courte, on passe du temps processeur pour rien et l'on retarde l'utilisateur. Trop longue, on retarde le traitement et donc l'utilisateur.

Il est souvent possible de faire une lecture bloquante (sur un fichier ou un socket). Mais l'on peut vouloir en langage C attendre un évènement sur un ensemble de fichiers sans pour autant mettre en œuvre des tâches.

Unix prévoit un appel système spécifique : *select*. Il prend en paramètres des ensembles de fichiers dont on scrute la modification ou la disponibilité d'écriture.

Depuis plusieurs décennies, chaque problème rencontré lors de développement a amené des solutions et des mécanismes appropriés. Il faut avoir une bonne connaissance du système et de ses capacités pour développer des programmes efficaces.

39.3 Les répertoires

La manipulation de répertoires est particulière. Cela provient du fait que même en faisant abstraction des droits d'accès utilisateurs au répertoire, le système conserve la maîtrise du contenu de ceux-ci.

- La lecture est libre, car elle ne modifie pas l'arborescence de fichiers.
- Les modifications se font de manière explicite : création de nouveau fichier, re-nommage, modification de structure.

Commençons par les modifications des feuilles de l'arbre. Elles sont la conséquence d'effets de bord d'autres commandes.

Lorsque l'on demande à accéder à un fichier en écriture qui n'existe pas encore, on provoque la création d'une entrée dans un répertoire.

La suppression d'une entrée fichier dans un répertoire se fait par l'appel système *unlink*, au travers de la commande `shell` *rm*.

La modification d'un nom de fichier dans un répertoire se fait par l'appel système *rename* au travers de la commande *mv*.

Regardons maintenant les modifications de structure. La création et la suppression d'un nœud de l'arbre des fichiers se font avec les appels *mkdir* et *rmdir*.

Il ne reste plus qu'à détailler la lecture d'un répertoire. De nouveau, c'est le même principe dans la plupart des langages (*C*, *PHP*, *Java*). Une commande fournit un descripteur, en l'occurrence l'appel système *opendir*. Ensuite, un pointeur géré par ce descripteur permettra de lister une à une chaque entrée avec l'appel système *readdir*.

40 Les formats de fichier

Il y a en général deux manières de lire un fichier au niveau le plus bas :

- caractère par caractère. C'est long et demande de faire une analyse (doit-on découper en fin de ligne ?) à chaque endroit du programme.
- en fournissant un tableau de caractères à remplir. Mais d'autres questions se posent : le tableau est-il assez grand ou trop grand ? Tombe-t-il à cheval sur plusieurs structures (plusieurs lignes) ?

En général, les langages fournissent des fonctions de plus haut niveau pour la lecture de fichiers textes. Les fichiers binaires nécessitent plus de précisions : quelle taille de structure ? Quelle décomposition ?

Il n'est pas possible d'utiliser l'une pour l'autre. La lecture texte permet de gérer la fin des lignes et en particulier la différence entre l'utilisation d'un marqueur simple ou d'un marqueur double.

marqueur	contrôle	hexadécimal	signification
\n	^J	0x0A	nouvelle ligne
\r \n	^M ^J	0x0D 0x0A	retour chariot / nouvelle ligne

Table 20 – Marqueur de fin de ligne

Pour illustrer cette différence, nous allons lire en mode texte un fichier binaire en utilisant *readLine* en Java

```
    static public void main (String[] arg) {
  try {
      File file = new File ("/etc/alternatives/emacs-48x48.png");
      BufferedReader in =
    new BufferedReader (new InputStreamReader
          (new FileInputStream (file)));
      try {
    for (;;) {
        String line = in.readLine ();
        if (line == null)
      break;
        System.out.println (line);
    }
      } finally {
    in.close ();
      }
  } catch (IOException e) {
```

Listing 72 – Code Java utilisant *readLine*

et en utilisant *fscanf* en C.

```c
int main (int argc, char** argv, char** envp) {
  char tmp [8192];
  while (fscanf (stdin, "%s", tmp) != EOF)
    fprintf (stdout, "%s", tmp);
  return 0;
}
```

Listing 73 – Code C utilisant *fscanf*

Voici le résultat de la lecture du fichier "/etc/alternatives/emacs-48x48.png"

```
/etc/alternatives/emacs-48x48.png
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
00000010: 0000 0030 0000 0030 0806 0000 0057 02f9  ...0...0.....W..

readline en Java
00000000: efbf bd50 4e47 0a1a 0a00 0000 0a49 4844  ...PNG.......IHD
00000010: 5200 0000 3000 0000 3008 0600 0000 5702  R...0...0.....W.

fscanf en C
00000000: 8950 4e47 1a49 4844 5249 e8ee f79e adcf  .PNG.IHDRI......
00000010: fb7d e75e dd2b 300e 8e73 675e 9d73 cf3d  .}.^.+0..sg^.s.=
```

Listing – Trace

On remarque que Java a tenu compte des retours chariots mais les a ignorés à la recopie. Des caractères hors ASCII (au-delà de la valeur 128) ont été transcrit en UTF8.

En C, tous les caractères nuls sont ignorés puisqu'il s'agit dans ce langage du marqueur de fin de chaîne de caractères.

À l'inverse la lecture binaire de fichiers textes impose un traitement supplémentaire. Il faut donc toujours utiliser le mode de lecture et d'écriture le plus adapté. Par exemple pour la lecture en Java choisir entre : *BufferedReader*, *DataInputStream*, *FileImageInputStream*, *ObjectInputStream* . . .

41 Fichiers textes

La lecture de caractères est plus complexe qu'il n'y paraît.

41.1 Codage des caractères

Dans le monde, nous n'utilisons pas tous le même alphabet. Ce document est écrit en latin. Nous ne sommes pourtant qu'une minorité à utiliser cette représentation (la majorité utilise les caractères chinois).

Avant d'invoquer une fonction de lecture ou d'écriture sur un fichier il faut donc préciser le codage utilisé. Du fait qu'une machine ne pense pas, elle ne peut donner de sens à ce qui est lu et ne sera pas faire la différence entre un texte correctement décodé et un autre.

La phrase Gaëlle, où êtes vous ? encodée en *UTF8* donnera GaÃ«lle, oÃ¹ Ãªtes vousÂ ? si on croit qu'elle avait été encodée en *latin1*. Ce genre de chose arrive souvent lorsque les méta-données d'un page HTML ne décrit pas correctement l'encodage utilisé.

Exemple	alphabet	codage
This text is writen in latin	latin	ASCII
Ce texte est écrit en latin accentué	latin accentué	ISO 8859-1
Этот текст написан на кириллице	cyrillique	KOI8-R

Table 21 – Codage de caractères

ISO-8859-1 est devenu ISO-8859-15 après l'euro.

La raison de disposer de plusieurs encodages est de réduire la taille des caractères. Entre le nombre de lettres (majuscules et minuscules), le nombre de chiffres, les symboles et les divers caractères de contrôle, il faut environ un octet. C'est suffisant pour un alphabet mais pas tous en même temps. Il y a recouvrement des différents encodages.

La norme UTF (pour *Unicode Transformation Format*) regroupe tous les alphabets. Elle fonctionne en encodant les caractères les plus fréquents sur les bits de poids faible et en utilisant le dernier pour un chaînage vers une extension. En UTF8 le code ASCII prend 1 octet (compatible avec l'ASCII), mais cela peut aller jusqu'à 4 octets avec des alphabets plus grands. Il existe des variantes UTF16 et UTF32 plus constantes dans l'interprétation des valeurs (même taille pour tous les caractères) mais qui peuvent quadrupler la taille du fichier texte. De préférence, utilisez UTF8.

En cas de difficulté, Unix propose un outil permettant toutes les conversions d'encodage pour des fichiers avec la commande *recode*.

41.2 1 champ par ligne

On pourrait croire qu'un fichier binaire est structuré et qu'un fichier texte ne l'est pas. En réalité, il y a la même diversité d'organisation. Un premier niveau de structuration des fichiers textes consiste à reconnaître les lignes.

Pour cela, on empile des filtres qui ajoutent des fonctionnalités à un objet. *FileInputStream* offre la lecture d'un ensemble d'octets `int read (byte[] b, int off , int len);`.

À partir de cette fonctionnalité, *InputStreamReader* offre la lecture d'un ensemble de caractères `int read (char[] cbuf, int offset, int length);`.

À partir de cette fonctionnalité, *BufferedReader* peut offrir la lecture d'une ligne `String readline ();`

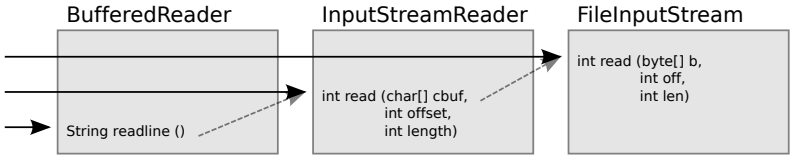


FIGURE 25 – Empilement de fonctionnalité

41.3 2 champs par ligne

Les lignes sont parfois utilisées comme déclaration : variable = valeur. C'est le cas des fichiers *properties* de Java. Ces fichiers sont utilisés pour enregistrer des données de configuration ou de sauvegarde. Ils peuvent être inclus dans l'arbre des classes d'un fichier ".jar".

Le listing 74 donne un exemple de création de propriété.

```
1     static public void main(String[] args) {
    try {
        Properties prop = new Properties();
        prop.setProperty ("universite", "Bretagne-Sud");
5       prop.setProperty ("diplome", "DUT Informatique");
        prop.setProperty ("module", "M3101");
        OutputStream out = null;
        try {
      out = new FileOutputStream ("demo.properties");
10    prop.store (out, null);
        } finally {
      out.close ();
        }
    } catch (FileNotFoundException e) {
15  } catch (IOException e) {
    }
      }
```

Listing 74 – Création d'un fichier propriété Java

Il produit le fichier propriété (voir listing 75).

```
1  #Sat Aug 16 18:23:28 CEST 2014
diplome=DUT Informatique
universite=Bretagne-Sud
module=M3101
```

Listing 75 – Création d'un fichier propriété Java

Le listing 76 donne un exemple de lecture.

```
1     static public void main(String[] args) {
    try {
        Properties prop = new Properties();
        prop.load (new FileInputStream ("demo.properties"));
5       for (Object key : prop.keySet ())
      System.out.println (key+"="+prop.getProperty ((String)key));
    } catch (FileNotFoundException e) {
    } catch (IOException e) {
    }
10    }
```

Listing 76 – Création d'un fichier propriété Java

41.4 n champs par ligne

La structure peut être une table de base de données avec un délimiteur de champs pour constituer des "n-uplet". Unix utilise ce format pour ses propres bases par exemple avec le caractère " :" comme délimiteur (passwd, group, shadow, gshadow).

```
root:x:0:0:root:/root:/bin/bash
```

Listing – Trace

La section 1 du manuel fournit les explications sur les commandes *cut*, *sort*, *grep*, *head*, *tail*, *wc* ...

Par exemple, la liste des shell de connexion est obtenue avec la commande :

```
cut -d : -f7 /etc/passwd | sort -u
```

et donne le résultat suivant :

```
/bin/bash
/bin/false
/bin/sh
/bin/sync
/usr/sbin/nologin
```

Listing – Trace

41.5 Découpage en jetons

Les éléments d'analyse ne sont pas forcément limités à une ligne. Le fichier en entier peut être une succession de mots. C'est le cas d'un programme informatique que l'on souhaite découper en éléments lexicaux. On appelle alors ces éléments : des jetons (en anglais *token*). Les jetons sont : des identifiants, des chaînes de caractères, des nombres, des ponctuations ...

Pour reconnaître les jetons on réalise une analyse lexicale. Pour comprendre le texte suivant un langage, il faut définir en plus une grammaire et réaliser une analyse syntaxique.

Java fournit le premier niveau d'analyse (lexical) avec deux classes :

- *StringTokenizer* qui s'applique à une chaîne de caractères. Elle découpe de façon rudimentaire des phrases en donnant pour chaque jeton la chaîne de caractères reconnue.
- *StreamTokenizer* qui s'applique à un texte entier. Elle est plus élaborée et reconnaît des commentaires, des nombres, des textes entre guillemets... Elle fournit pour chaque jeton son type et sa valeur.

41.6 Modèle de texte

Un fichier de textes est certes constitué d'une suite de mots, mais il peut être chargé d'informations que l'on peut formater (en anglais *format*) pour une meilleure compréhension ou que l'on peut extraire (en anglais *parse*).

Java fournit un ensemble de classes et de méthodes dans ce but dans le paquetage `java.text`.

Voici les classes les plus significatives :

- `DateFormat`, pour les dates et les heures.
- `NumberFormat`, pour tous les nombres (int, long, float, double).
- `DecimalFormat`, pour les nombres à virgule.
- `ChoiceFormat`, pour la correspondance entre des intervalles de nombres et du texte.
- `MessageFormat`, pour des messages élaborés incluant des formats simples.

Le listing 77 donne une rapide utilisation de l'utilisation de quelques classes.

```
1     static public void main(String[] args) {
  try {
      SimpleDateFormat sdf = new SimpleDateFormat ("dd MM HH:mm ");
      System.out.println (sdf.format (new Date ()));
5
      ChoiceFormat cf = new ChoiceFormat ("-1# negatif|0#nul|1#un|1<plusieurs");
      for (int i : new int[]{-1, 0, 1, 2}) {
    System.out.println (i+" => "+cf.format (i));
      }
10
      MessageFormat mf =
    new MessageFormat ("Phrase avec "+
          "un nombre {0,number,integer}, "+
          "un mot {2} une date {1,date}, "+
15        "une heure {1,time}.");
      Object[] objs = {42, new Date (), "super"};
      String s = mf.format (objs);
      System.out.println (s);

20    Object[] result = mf.parse (s);
      for (Object obj : result)
    System.out.println (" => "+obj);
  } catch (ParseException e) {
25  }
    }
```

Listing 77 – Exemple de message formaté en Java

Notez qu'en lignes 14 et 15 , un même objet "temps" peut être affiché en tant que date (ligne 14) et en tant qu'heure.

Le résultat est assez surprenant :

```
1  17 08 11:51
   -1 =>  negatif
   0 => nul
   1 => un
5  2 => plusieurs
   Phrase avec un nombre 42, un mot super une date 17 aout 2014, une heure 11:51:59.
    => 42
    => Thu Jan 01 11:51:59 CET 1970
    => super
```

Listing – Trace

Notez qu'en ligne 8 de cette trace d'exécution, le même objet "temps" a été interprété

- une 1re fois en tant que date (mais sans heure il a fourni "Sun Aug 17 2014 00 :00 :00") et
- une 2de fois en tant qu'heure (mais sans date il a fourni la date d'origine d'Unix "Thu Jan 01 1970")

La 2de a écrasé la 1re.

Pour l'interprétation d'un texte en langage naturel ce mécanisme est rudimentaire. Il ne fait qu'une analyse lexicale et non syntaxique. Ne lui demandez pas l'impossible. Il ne pourra pas lever des ambiguïtés, telle l'analyse d'un mot suivi d'un nombre si le 1er mot est nombre lui-même.

41.7 Expression régulière

Au niveau le plus paramétrable des outils proposés par Java, se trouve un paquetage de gestion des expressions régulières (*java.util.regex*).

Il est composé de 2 classes :

- *Pattern* qui permet de définir un modèle avec des sélections de caractères et des répétitions.
- *Matcher* qui permet d'appliquer le modèle sur une chaîne à analyser. Elle fournit toutes les interprétations possibles du modèle sur la chaîne sous forme de groupe d'analyse.

41.8 Internationalisation

Nous voulons des programmes génériques qui séparent fond et forme.

Comment permettre l'internationalisation (en anglais *localization*) d'une application ?

Nous devons rechercher toutes les occurrences des chaînes de caractères et les remplacer par un message qui puisse être paramétré par la langue.

En clair, au lieu d'écrire : `System.out.println ("Bonjour");`
Il faut écrire : `System.out.println (localized (msgHello));`

Et au lieu de : `System.out.println (MessageFormat.format ("Il y a {0} objets", nbObjs));`
Il faut écrire : `System.out.println (localized (msgNbObj, nbObjs));`

C'est à vous d'écrire la fonction de traduction `localized`.

Rechercher toutes les chaînes du programme veut dire : tous les titres de fenêtres, les titres des boîtes de dialogue, les champs de formulaires, les textes des boutons ...

Ce travail paraît fastidieux a posteriori, quand le programme est terminé et que rien n'a été prévu pour le multi-linguisme.

En réalité, rien n'empêche de prévoir des classes qui allègent la réécriture et qui en fin de compte factorise le formatage des textes. Nous venons de voir des composants élémentaires qu'il suffit d'assembler.

Les messages sont à enregistrer dans des propriétés que Java sait associer avec la langue choisie (voir la classe *ResourceBundle*). Ainsi, en même temps que les messages de l'application sont mis à jour, la machine virtuelle Java change le format de la date et des décimaux.

41.9 XML

Les fichiers textes peuvent enfin s'éloigner du langage naturel pour avoir une structure à la syntaxe rigoureuse pour accueillir des bases de données. C'est le cas de XML. Ce chapitre est trop court pour inclure une présentation complète d'XML. Nous supposons ce concept déjà acquis.

Les listings 78 et 79 donnent respectivement des exemples de manipulation d'XML en PHP pour la navigation dans la structure ou sa création. Le listing complet est donné en fin de chapitre.

```
$requette = new DOMDocument ("1.0", "utf8");
$requette->loadXML ($xml);
$racineRequete = $requette->documentElement;
$info = $racineRequete->getAttribute ("info");
$nom = $racineRequete->getElementsByTagName ("personne")->item (0)->nodeValue;
$texte = $racineRequete->getElementsByTagName ("tache")->item (0)->nodeValue;
```

Listing 78 – Navigation XML en PHP

```
$reponse = new DOMDocument ("1.0", "utf8");
$racineReponse = $reponse->createElement ("reponse");
$reponse->appendChild ($racineReponse);

if ($info == "nouvelle")
  $racineReponse->setAttribute ("action", "cree");
else
  $racineReponse->setAttribute ("action", "supprime");

$racineReponse->appendChild ($reponse->createElement ("personne", $nom));
$racineReponse->appendChild ($reponse->createElement ("tache", $texte));
```

Listing 79 – Création XML en PHP

De même en Java, les listings 80 et 81 fournissent des exemples de lecture et d'écriture de structure XML.

```
  // ========================================
  static public Document readDocument (InputStream stream)
throws java.io.IOException {
try {
    DocumentBuilder documentBuilder =
  DocumentBuilderFactory.newInstance ().newDocumentBuilder ();
    Document document =
  documentBuilder.parse (new NoWaittingNoCloseInputStream (stream));
    document.normalizeDocument ();
    return document;
} catch (javax.xml.parsers.ParserConfigurationException e) {
    throw new IOException (e);
} catch (org.xml.sax.SAXException e) {
```

Listing 80 – Exemple de lecture XML en Java

```
  }

  // ========================================
  static public void writeDocument (Document document, OutputStream stream) {
try {
          Source source = new DOMSource (document);
          Result result = new StreamResult (stream);
    Transformer xformer =
  TransformerFactory.newInstance ().newTransformer ();
    xformer.setOutputProperty (OutputKeys.INDENT, "yes");
    xformer.setOutputProperty
  ("{http://xml.apache.org/xslt}indent-amount", "2");
          xformer.transform (source, result);
    stream.write ("\n".getBytes ());
```

Listing 81 – Exemple d'écriture XML en Java

Le listing 82 fournit un exemple réel d'utilisation des méthodes précédentes.

```
  public void print (File file)
      throws FileNotFoundException {
      try {
          DocumentBuilder documentBuilder =
```

```
            DocumentBuilderFactory.newInstance ().newDocumentBuilder ();
        Document challengeDoc = documentBuilder.newDocument ();
        challengeDoc.setXmlStandalone (true);
        Element challengeTag = challengeDoc.createElement (challengeToken);

        // creation du document XML ...

        XML.writeDocument (challengeDoc, new FileOutputStream (file));
      } catch (ParserConfigurationException e) {
        Log.keepLastException ("XMLChallenge::print", e);
      }
  }

  public void parse (File file)
      throws FileNotFoundException, IOException {
      Document challengeDoc = XML.readDocument (new FileInputStream (file));
      Element challengeTag = challengeDoc.getDocumentElement ();

      challengeName = challengeTag.getAttribute (nameToken);
      // lecture du document XML ...
```

Listing 82 – Exemple d'utilisation XML en Java

La lecture d'un fichier XML peut poser problème sur certaines versions de Java. Le listing 84 donne un moyen de contourner le blocage en fin de fichier.

```
  /** une idee de Kohsuke Kawaguchi
http://weblogs.java.net/blog/kohsuke/archive/2005/07/socket_xml_pitf.html */
  static public class NoWaittingNoCloseInputStream
extends java.io.FilterInputStream {
public NoWaittingNoCloseInputStream (InputStream in) { super (in); }

public int read (byte[] b, int off, int len) throws IOException {
    if (super.available () <= 0)
  return -1;
    int nb = super.read (b, off, len);
    return nb;
}
public void close () throws IOException {}
```

Listing 83 – XML non bloquant en Java

```
  /** une idee de Kohsuke Kawaguchi
http://weblogs.java.net/blog/kohsuke/archive/2005/07/socket_xml_pitf.html */
  static public class NoWaittingNoCloseInputStream
extends java.io.FilterInputStream {
public NoWaittingNoCloseInputStream (InputStream in) { super (in); }

public int read (byte[] b, int off, int len) throws IOException {
    if (super.available () <= 0)
  return -1;
    int nb = super.read (b, off, len);
    return nb;
}
public void close () throws IOException {}
```

Listing 84 – XML non bloquant en Java

42 Fichiers binaires

La lecture des fichiers binaires dépend de la structure du fichier. Il n'y a pas de solution générique pour toutes les situations. Cependant, dans certains cas des classes de solutions ont été développées.

Les sons sont intégrés à Java (voir le paquetage *javax.sound*).

Les images simples sont également intégrées dans le langage (voir le paquetage *javax.imageio.ImageIO*). Certains cas peuvent poser problème, comme les formats multi-spectraux ou des pixels sur 48 bits. Des extensions au langage sont disponibles avec le paquetage *javax.media.jai*. JAI (pour *Java Advanced Imaging*) qui pourront être trouvées à cette URL http://www.oracle.com/technetwork/java/iio-141084.html.

Les vidéos ne sont pas dans le langage mais un paquetage *com.xuggle.mediatool* permet la manipulation image par image d'un film. Le développement est disponible à l'URL http://www.xuggle.com/. Il s'appuie sur le logiciel libre FFmpeg (voir http://www.ffmpeg.org/).

Il y a enfin un moyen d'écrire et de relire des objets Java, grâce à des mécanismes de sérialisation (mise en série des attributs). De nouveau ce n'est pas si simple. Il ne faut pas que l'objet enregistré forme un graphe qui tire l'ensemble des Gigas de données de l'application. Les attributs de l'objet sont annotés de façon à indiquer, si la sauvegarde de leurs attributs est pertinente ou si par exemple on peut les recalculer à partir d'autres attributs. Un attribut qui ne participe pas à la persistance de l'objet est dit *transient*.

Le listing listing:readObjectJavaest extrait de la javadoc de l'API Java.

```
        FileInputStream fis = new FileInputStream ("objects.sav");
        ObjectInputStream ois = new ObjectInputStream (fis);
        int i = ois.readInt ();
        String today = (String) ois.readObject ();
        Date date = (Date) ois.readObject ();
        ois.close();
```

Listing 85 – Exemple de lecture d'objets Java

43 Annexe : code source

Voici le code complet des extraits donnés dans ce chapitre.

```
<?php
/*
  (c) F. Merciol
  Plus d'informations sur http://m3101.merciol.fr
*/

{
```

```php
  // envoi de fichier MXL
  header("Content-Type: application/xml");

  // lecture du fichier XML du client
  $xml = trim (file_get_contents ("php://input"));

  $requette = new DOMDocument ("1.0", "utf8");
  $requette->loadXML ($xml);
  $racineRequete = $requette->documentElement;
  $info = $racineRequete->getAttribute ("info");
  $nom = $racineRequete->getElementsByTagName ("personne")->item (0)->nodeValue;
  $texte = $racineRequete->getElementsByTagName ("tache")->item (0)->nodeValue;

  // simule un temps de traitement du serveur
  sleep (mt_rand (1, 2));

  // creation du document XML
  $reponse = new DOMDocument ("1.0", "utf8");
  $racineReponse = $reponse->createElement ("reponse");
  $reponse->appendChild ($racineReponse);

  if ($info == "nouvelle")
    $racineReponse->setAttribute ("action", "cree");
  else
    $racineReponse->setAttribute ("action", "supprime");

  $racineReponse->appendChild ($reponse->createElement ("personne", $nom));
  $racineReponse->appendChild ($reponse->createElement ("tache", $texte));

  $reponse->formatOutput = true;
  $reponse->save("php://output") . "\n";
}
?>
```

Listing 86 – XML en PHP


```
package io;
import misc.*;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Hashtable;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;

public class XML {

    // ========================================
    /** une idee de Kohsuke Kawaguchi
  http://weblogs.java.net/blog/kohsuke/archive/2005/07/socket_xml_pitf.html */
    static public class NoWaittingNoCloseInputStream
extends java.io.FilterInputStream {
public NoWaittingNoCloseInputStream (InputStream in) { super (in); }

public int read (byte[] b, int off, int len) throws IOException {
    if (super.available () <= 0)
  return -1;
    int nb = super.read (b, off, len);
    return nb;
}
public void close () throws IOException {}
  }

  // ========================================
  static public Document readDocument (InputStream stream)
throws java.io.IOException {
try {
    DocumentBuilder documentBuilder =
  DocumentBuilderFactory.newInstance ().newDocumentBuilder ();
    Document document =
  documentBuilder.parse (new NoWaittingNoCloseInputStream (stream));
    document.normalizeDocument ();
    return document;
} catch (javax.xml.parsers.ParserConfigurationException e) {
    throw new IOException (e);
} catch (org.xml.sax.SAXException e) {
    throw new IOException (e);
}
  }

  // ========================================
  static public void writeDocument (Document document, OutputStream stream) {
try {
          Source source = new DOMSource (document);
          Result result = new StreamResult (stream);
    Transformer xformer =
  TransformerFactory.newInstance ().newTransformer ();
    xformer.setOutputProperty (OutputKeys.INDENT, "yes");
    xformer.setOutputProperty
  ("{http://xml.apache.org/xslt}indent-amount", "2");
          xformer.transform (source, result);
    stream.write ("\n".getBytes ());
    stream.flush ();
      } catch (Exception e) {
    Log.keepLastException ("XML::writeDocument", e);
}
  }

  // ========================================
  static public void writeElement (Element element, OutputStream stream) {
try {
    DocumentBuilder documentBuilder =
  DocumentBuilderFactory.newInstance ().newDocumentBuilder ();
    Document document = documentBuilder.newDocument ();
    document.setXmlStandalone (true);
    document.appendChild (document.importNode (element, true));
    XML.writeDocument (document, stream);
    stream.flush ();
} catch (Exception e) {
    Log.keepLastException ("XML::writeElement", e);
}
  }

  // ========================================
  public final static void putToken (Hashtable<String, String> hashtable,
```

```
                String token, String value) {
  hashtable.put (token, (value == null) ? "" : value);
    }

    // ========================================
    public final static Hashtable<String, String> node2hashtable (Node child) {
  Hashtable<String, String> hashtable = new Hashtable<String, String> ();
  for (; child != null; child = child.getNextSibling ()) {
      if (child.getNodeType () == Node.ELEMENT_NODE) {
    Element elementTag = (Element) child;
    String token = child.getNodeName ();
    NodeList nodeList = ((Element) child).getChildNodes ();
    if (nodeList.getLength () > 0)
        hashtable.put (token,
            ((Text) nodeList.item (0)).getWholeText ());
      }
  }
  return hashtable;
    }

    // ========================================
    public final static void hashtable2node (Document document, Element container,
                Hashtable<String, String> hashtable) {
  for (String token : hashtable.keySet ()) {
      Element tag = document.createElement (token);
      tag.appendChild (document.createTextNode (hashtable.get (token)));
      container.appendChild (tag);
  }
    }

    // ========================================
}
```

Listing 87 – XML en Java

44 TD5 : Entrées/sorties

Dès votre arrivée en TD, pensez à déposer votre QCM rempli à votre encadrant. La partie à rendre est la dernière page de votre document (à découpe suivant les pointillés).

Après l'appel, c'est l'occasion de poser les questions concernant des éléments non compris du cours. Pensez à recopier le corrigé du QCM sur votre document (partie à concerver à la page précédante). Vous pourrez l'utiliser le jour de l'examen. Une page de notes personnelles est à votre disposition en fin de TD.

L'objectif du TD est de manipuler et de formater des données contenues dans du texte.

Nous allons nous entraîner sur un premier ensemble de chaînes de caractères suivantes :

- "send A 1"
- "answer B 3"
- "guest bob"
- "return A"
- "close"
- "full"
- "quit"

Nous supposerons que le premier mot est un mot clef (limité en valeur) et que les suivants peuvent avoir différentes valeurs mais rester de même nature (un entier pour un entier, ...).

Question : Comment récupérer tous les mots d'une chaîne (on utilisera StringTokenizer) ?

Réponse : ✍

Voici un extrait de la documentation java sur les expressions régulières :

```
Characters
x                  The character x
\\                 The backslash character
\0ooo              The character with octal value 0ooo
\xhh               The character with hexadecimal value 0xhh
\uhhhh             The character with hexadecimal value 0xhhhh
```

```
\t              The tab character ('\u0009')
\n              The newline (line feed) character ('\u000A')
\r              The carriage-return character ('\u000D')
\f              The form-feed character ('\u000C')
\a              The alert (bell) character ('\u0007')
\e              The escape character ('\u001B')
\cx             The control character corresponding to x

Character classes
[abc]           a, b, or c (simple class)
[^abc]          Any character except a, b, or c (negation)
[a-zA-Z]        a through z or A through Z, inclusive (range)
[a-d[m-p]]      a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]    d, e, or f (intersection)
[a-z&&[^bc]]    a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]   a through z, and not m through p: [a-lq-z](subtraction)

Predefined character classes
.               Any character (may or may not match line terminators)
\d              A digit: [0-9]
\D              A non-digit: [^0-9]
\s              A whitespace character: [ \t\n\x0B\f\r]
\S              A non-whitespace character: [^\s]
\w              A word character: [a-zA-Z_0-9]
\W              A non-word character: [^\w]

POSIX character classes (US-ASCII only)
\p{Lower}       A lower-case alphabetic character: [a-z]
\p{Upper}       An upper-case alphabetic character:[A-Z]
\p{ASCII}       All ASCII:[\x00-\x7F]
\p{Alpha}       An alphabetic character:[\p{Lower}\p{Upper}]
\p{Digit}       A decimal digit: [0-9]
\p{Alnum}       An alphanumeric character:[\p{Alpha}\p{Digit}]
\p{Punct}       Punctuation: One of !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph}       A visible character: [\p{Alnum}\p{Punct}]
\p{Print}       A printable character: [\p{Graph}\x20]
\p{Blank}       A space or a tab: [ \t]
\p{Cntrl}       A control character: [\x00-\x1F\x7F]
\p{XDigit}      A hexadecimal digit: [0-9a-fA-F]
\p{Space}       A whitespace character: [ \t\n\x0B\f\r]

Boundary matchers
^               The beginning of a line
$               The end of a line
\b              A word boundary
\B              A non-word boundary
\A              The beginning of the input
\G              The end of the previous match
\Z              The end of the input but for the final terminator, if any
\z              The end of the input

Greedy quantifiers
X?              X, once or not at all
X*              X, zero or more times
X+              X, one or more times
X{n}            X, exactly n times
X{n,}           X, at least n times
X{n,m}          X, at least n but not more than m times

Reluctant quantifiers
X??             X, once or not at all
X*?             X, zero or more times
X+?             X, one or more times
X{n}?           X, exactly n times
X{n,}?          X, at least n times
X{n,m}?         X, at least n but not more than m times

Possessive quantifiers
X?+             X, once or not at all
X*+             X, zero or more times
X++             X, one or more times
X{n}+           X, exactly n times
X{n,}+          X, at least n times
X{n,m}+         X, at least n but not more than m times

Logical operators
XY              X followed by Y
X|Y             Either X or Y
(X)             X, as a capturing group

Back references
\n              Whatever the nth capturing group matched
\k<name>        Whatever the named-capturing group "name" matched

Special constructs (named-capturing and non-capturing)
(?<name>X)          X, as a named-capturing group
(?:X)               X, as a non-capturing group
(?idmsuxU-idmsuxU)  Nothing, but turns match flags i d m s u x U on - off
(?idmsux-idmsux:X)  X, as a non-capturing group with the given flags i d m s u x
    on - off
(?=X)               X, via zero-width positive lookahead
(?!X)               X, via zero-width negative lookahead
(?<=X)              X, via zero-width positive lookbehind
(?<!X)              X, via zero-width negative lookbehind
(?>X)               X, as an independent, non-capturing group
```

Listing 88 – Extrait documentation expression régulière

Mais aussi sur un second ensemble :

```
- "140824 07:09:30 alice: guest bob"
- "140824 07:09:31 alice: guest grpA:bob"
- "140824 07:09:32 alice: guest grpA:"
- "140824 07:09:33 bob: welcome alice@10.0.0.1:8080"
- "140824 07:09:34 alice: welcome bob@10.0.0.2:8081, chalie@10.0.0.1:8082"
- "140824 07:09:35 alice: send 10.0.0.1:8080-A"
- "140824 07:09:36 alice: answer 10.0.0.1:8082-C"
- "140824 07:09:37 alice: reject 10.0.0.1:8080-A"
- "140824 07:09:38 alice: close"
- "140824 07:09:39 bob: full"
- "140824 07:09:40 bob: done 10.0.0.2:8081-[ACD] 10.0.0.2:8082-[BD] 10.0.0.2:8083-F"
```

Question : En utilisant les expressions régulières, comment reconnaître des groupes de mots : chiffres, date, lettres, adresses IP, adresses IP + port, participants ?

Réponse : ✍

Question : Comment retrouver le contenu des groupes de mots ?

Réponse : ✍

Question : En utilisant les messages formatés, comment reconnaître tous les modèles puis retrouver les contenus des groupes de mots ?

Réponse : ✍

Question : Donnez un moyen de franciser les messages.

Réponse : ✍

Voici un extrait de la documentation java sur le format des dates :

```
Letter  Date or Time Component          Presentation    Examples
    G   Era designator                  Text            AD
    y   Year                            Year            1996; 96
    Y   Week year                       Year            2009; 09
    M   Month in year                   Month           July; Jul; 07
    w   Week in year                    Number          27
    W   Week in month                   Number          2
    D   Day in year                     Number          189
    d   Day in month                    Number          10
    F   Day of week in month            Number          2
    E   Day name in week                Text            Tuesday; Tue
    u   Day number of week (0 = Sunday) Number          1
    a   Am/pm marker                    Text            PM
    H   Hour in day (0-23)              Number          0
    k   Hour in day (1-24)              Number          24
    K   Hour in am/pm (0-11)            Number          0
    h   Hour in am/pm (1-12)            Number          12
    m   Minute in hour                  Number          30
    s   Second in minute                Number          55
    S   Millisecond                     Number          978
    z   Time zone                       General         GMT-08:00
    Z   Time zone                       RFC 822         -0800
    X   Time zone                       ISO 8601        -08; -0800; -08:00
```

Listing 89 – Extrait documentation format de date

Page de notes personnelles : ✍

45 TP5 : Entrées/sorties

- Votre compte-rendu est à rendre en fin de TP.
- Vous ne devez pas rendre un listing, mais un document répondant aux questions posées.
- Vous préparerez le compte-rendu à l'avance (copier-coller les questions) pour gagner du temps.
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois.
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (uniquement la fenêtre concernée).
- N'oubliez pas de rendre votre TP imprimé en 2 colonnes de texte par côté de papier et de faire le TP en binôme.

L'objectif du TP est de réaliser des extractions d'information dans des chaînes de caractères. Ce travail est à réaliser en Java et sera réutilisé pour le dernier TP du module.

Question : Ecrivez l'analyse du premier ensemble d'exemples du TD en utilisant StringTokenizer.

Donnez le code et les traces.

Réponse : ✍

Question : Ecrivez un automate qui analyse une chaîne de caractères et invoque une méthode correspondant à chaque modèle avec les bons types d'arguments.

Donnez le code et les traces.

Réponse : ✍

Question : Ecrivez l'analyse pour reconnaître une chaîne de la forme "date heure joueur : ... \n"

Donnez le code et les traces.

Réponse : ✍

Question : Ecrirez l'analyse d'une chaîne de la forme "ddd.ddd.ddd.ddd :pppp-l" ou "ddd.ddd.ddd.ddd :pppp-[lll]"

Donnez le code et les traces.

Réponse : ✍

Question : Extrayez toutes les informations au format minimal "ddd.ddd.ddd.ddd :pppp-l".

Donnez le code et les traces.

Réponse : ✍

Question : En se limitant aux actions “send”, “answer” et “reject” écrivez l'automate qui reconnait les chaînes suivant le second ensemble d'exemples et invoque une méthode correspondante avec les bons types d'arguments.

Donnez le code et les traces.

Réponse : ✍

Question : Complétez l'automate pour les actions "close", "full" et "done"

Donnez le code et les traces.

Réponse : ✍

Septième partie

Introduction à la programmation réseau

46 Brève histoire de l'Internet

- 1961 MIT : communication par paquets
- 1962 MIT : application possible entre ordinateurs
- 1962 DARPA : création d'ARPANET
- 1970 : TCP/IP UDP/IP
- 1971-1978 : Cyclades en France
- NFS généralise ARPANET en l'Internet.
- ˜1990 BBS
- 2000 l'Internet devient le support commercial

D'où vient le réseau mondial que nous connaissons aujourd'hui ?

Le MIT rédigea en 1961 un texte théorique sur la communication par paquets, puis en 1962 sur des applications possibles entre ordinateurs.

Le département de la défense étasunienne (au DARPA) s'empara du projet en 1962. Le réseau militaire portera le nom d'ARPANET (adresse IP 10.0.0.0).

Dans les années 1970, TCP et UDP sont définis.

A noter le réseau expérimental français Cyclades (dirigé par Louis Pouzin) se crée à la suite de ARPANET. Les échanges entre Cyclades et ARPANET furent nombreux. «Les travaux de Pouzin nous ont beaucoup apporté, explique Vinton chef de ARPANET. Nous avons utilisé son système de contrôle de flux pour le protocole TCP/IP. C'était motivant de parler avec lui.»

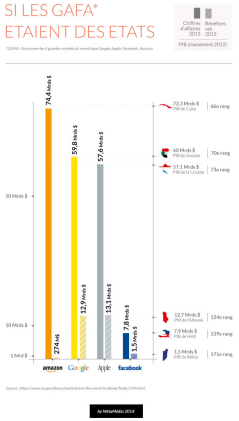


FIGURE 26 – Chiffre d'affaires des GAFAM (source Mediapart)

La NSF (Fondation Nationale pour la Science étasunienne) généralise l'accès à ARPANET, qui devient l'Internet (l'interconnexion des réseaux : le réseau des réseaux).

Dans les années 1990, il y eut un projet alternatif : les BBS (Bulletin Board System) qui consistait en une multitude de serveurs, avec un ou des modems offrant les services d'échanges de messages et de stockage et d'échange de fichiers. Le modem (modulateur-démodulateur) est un dispositif traduisant les éléments d'informations (bits) en sons, pour être transmis par des lignes téléphoniques. Dans les années 2000, l'Internet supplanta les BBS.

Le réseau est mondial. En France, il existe plusieurs opérateurs privés et publics. Ils sont interconnectés par le "SFINX". Tous les échanges inter-opérateurs y transitent en France. L'université utilise son propre réseau : Rénater (**RÉ**seau **NA**tional de télécommunications pour la **T**echnologie, l'**E**nseignement et la **R**echerche, voir la figure 27). Rénater n'est pas l'Internet. C'est un des réseaux de l'Inter-connexion des Réseaux.

En 2013, l'Internet représente 10% de la consommation électrique mondiale, soit 1 000 Tw (l'équivalent de 90 réacteurs nucléaires de 1 300 Mw). L'électricité n'est qu'un vecteur de transport de l'énergie (pas une source). La source d'énergie de l'Internet est nucléaire en France. Dans le reste du monde, l'Internet utilise principalement du charbon et des gaz de schiste. Les calories dégagées par les centres de données participent évidemment également au réchauffement climatique.

En 2013, les GAFAM (Google, Apple, Facebook, Amazon) représentent un chiffre d'affaires de 200 milliards de $ (figure 26 source `http://meta-media.fr/2014/02/07/si-les-gafa-etaient-des-etats-infographie.html`). Ils réduisent le commerce de proximité. Et par un modèle économique fondé sur la publicité, ils augmentent le prix des denrées, ce qui touche les pays pauvres et nous-mêmes. Les principales bénéficiaires sont des compagnies étasuniennes.

C'est enfin une source d'informations sur la vie privée des citoyens, mise en place sous forme ludique (moteur de recherche, exhibition de la vie privée, commerce en ligne). L'Internet représente un pouvoir politique.

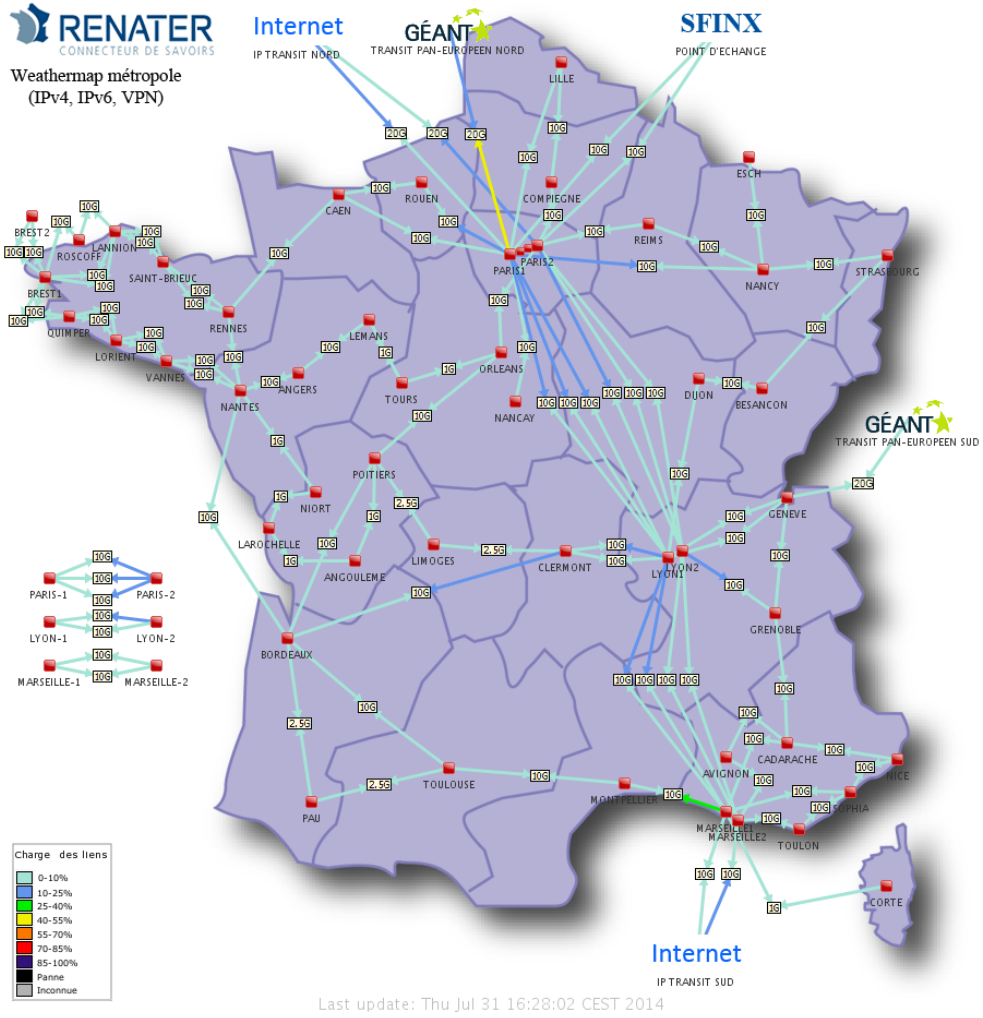


FIGURE 27 – Rénater (source https://www.renater.fr/au 31/7/2014)

L'Internet est un standard (il n'est pas régi par des accords entre nations). Dans le même temps, les nations définissent des normes. L'ISO normalise les communications informatiques. La table 22 représente une correspondance entre le standard de l'Internet et la norme ISO.

ISO	couche ISO	couche IP	structure	Protocoles IP
7	Application	Application	données	SMTP, POP3, telnet, FTP
6	Présentation			
5	Session			
4	Transport	Transport	segments	TCP, UDP
3	Réseau	Internet	paquets	IP
2	Liaison	Ethernet	trames (frames)	
1	Physique		bits	

TABLE 22 – Correspondance ISO / l'Internet

47 Ethernet

La couche de liaison fonctionne sur un bus d'échanges de trames d'information. Sur le même brin d'un bus, toutes les machines envoient leurs trames de façon asynchrone.

Elles écoutent en même temps qu'elles envoient. Lorsqu'il y a collision, elles arrêtent l'émission et reprennent plus tard.

Pour limiter la ré-émission en même temps, elles attendent un temps aléatoire avant de recommencer.

En théorie... cela ne fonctionne pas.

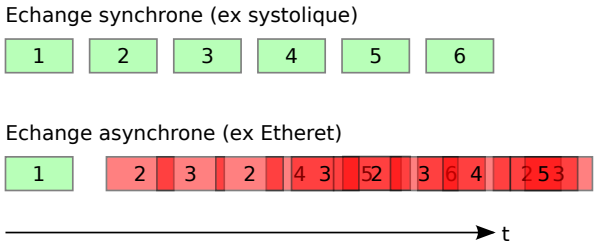


FIGURE 28 – Exemple de congestion de trames

La probabilité de collisions augmente au fur et à mesure que des collisions apparaissent. Le phénomène devient exponentiel.

Habituellement, les choses "tombent en panne" (on ne sait pas pourquoi la panne survient). Ici, les choses "tombent en marche" (on ne sait pas pourquoi la marche survient).

En fait, lorsqu'il y a saturation, un signal particulier est envoyé pour réinitialiser les tirages aléatoires. L'envoi systématique et automatique est une solution à ce problème.

ARP (Address Resolution Protocol) est un protocole permettant de découvrir les adresses des autres machines sur le brin (commande `arp -a`).

L'identification se fait par l'adresse MAC (Media Access Control). Elle est composée de 48 bits (6 octets affichés en chiffres hexadécimaux séparés pas " :") :

- 1 bit : individuel ou groupe
- 1 bit : universel ou local
- 22 bits : identifiant du constructeur
- 24 bits : adresse unique pour un constructeur donné

Les trames Ethernet sont constituées de l'adresse MAC du destinataire, de l'adresse MAC de l'émetteur, de 2 octets de protocole (par exemple IP) et des données.

48 IP

Ethernet permet la connexion de proche en proche. L'Internet permet la connexion de bout en bout (du réseau).

La désignation se fait par une adresse IP composée de 4 octets (en IP V4) écrits sous forme décimale.

- Le premier bit à 0 désigne la classe
  - classe A 128 réseaux de 16 777 214 machines
  - classe B 16 384 réseaux de 65 534 machines
  - classe C 2 097 152 réseaux de 254 machines
  - classe D multidiffusion
  - classe E réservée pour usage futur
- Les suivants, l'identifiant de réseau
- Les derniers, l'identifiant des machines dans le réseau

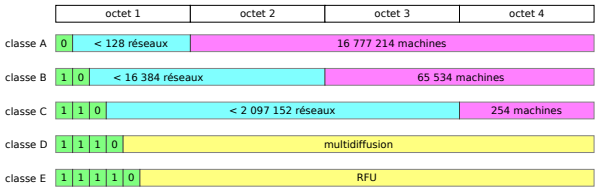


FIGURE 29 – Classes IPV4

Le masque réseau permet de séparer les 2 éléments d'une adresse et de réaliser du sous-adressage. Il s'agit d'une suite de 1 à gauche (points forts) pour le réseau et de 0 à droite (points faibles) pour les machines.

Il existe 2 adresses de machine spécifiques :

- tous les bits à zéro : désigne le réseau
- tous les bits à un : désigne la diffusion sur le sous-réseau.

On notera par exemple le masque générique d'une classe C `255.255.255.0`. Ou plus rapidement un "/" suivi du nombre de 1. "/24" en classe C. Il est possible de découper une classe en augmentant le nombre de bits réseau. On perd alors deux adresses (réseau et diffusion).

Certains réseaux sont réservés pour chaque classe pour un usage privé. Tout le monde peut les utiliser à condition de ne pas sortir sur l'Internet (de toute façon ces adresses sont bloquées) :

- classe A : 10/8 de 10.0.0.0 à 10.255.255.255
  le réseau de défense étasunienne (donc non routé sur l'Internet)
- classe B : 172.16/12 de 172.16.0.0 à 172.31.255.255
- classe C : 192.168.0.0/16 de 192.168.0.0 à 192.168.255.255

49 TCP/IP UDP/IP

Il n'est pas possible de monopoliser le bus pour l'envoi d'une trame. Les trames sont limitées en taille. Pour l'envoi de données volumineuses les données contenues dans une trame sont découpées en segment.

Les segments sont numérotés et peuvent être ré-émis en cas de corruption. Ils sont automatiquement réordonnés à leur arrivée. Dans les 2 cas, il y a échange entre les deux machines tout au long de l'échange des segments.

Il existe 2 sortes de segments.

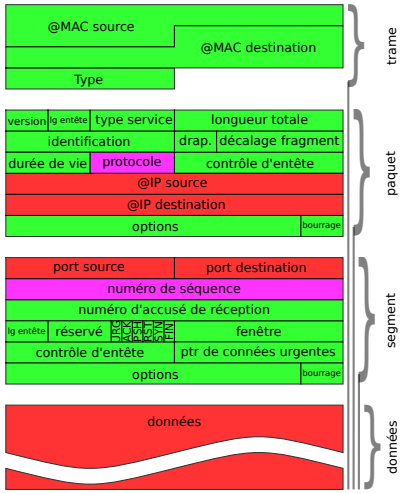


FIGURE 30 – Trames IP

- TCP pour l'envoi en mode dit connecté.
- UDP pour l'envoi en mode dit déconnecté ou datagramme.

La différence entre les deux modes provient du fait qu'en TCP les ressources restent réservées pour des échanges au niveau applicatif. Dans le cas d'UDP, il faudra établir une nouvelle connexion à chaque envoi.

50 Les Sockets

Pour échanger des informations entre applications, celles-ci utilisent des "prises" (en anglais *sockets*) avec soit TCP soit UDP. C'est l'interface logiciel de communication. Bien que l'on utilise le même terme de *socket* et les mêmes trames IP, les manipulations sont différentes.

Un socket possède un identifiant sur 2 octets : le "port". Chaque port correspond à un usage d'application. L'usage se nomme service et la correspondance entre port et service se trouve dans le fichier `/etc/services`

Port inférieur à 1024

Les numéros inférieurs à 1024 ne sont utilisables que par l'administrateur. Cela garantit que le service n'est pas rendu par un utilisateur quelconque. Par exemple, il est préférable que les messages ne puissent pas être reçus par un invité sur une machine multi-utilisateur.

Lorsque l'utilisateur n'a pas de préférence de numéro, il utilise la valeur 0, un port disponible (au-dessus de 1024) lui est fourni.

L'application serveur web (par exemple apache) a un port (en général 80). L'application navigateur (par exemple firefox) en a un également (il changera à chaque connexion).

50.1 TCP/IP

En mode connecté (TCP), l'application crée un socket particulier : le serveur de sockets (une sorte de standard d'appel).

Au moment de la connexion, le standard va créer une nouvelle liaison pour pouvoir libérer la ligne. Elle aura le même numéro de port côté serveur. Les liaisons sont différenciées par le fait qu'elles associent un numéro de port client et un numéro de port serveur.

Il y a 3 fonctions pour établir une connexion client/serveur :

- "bind" une seule fois côté serveur pour lier l'application à un port.
- "accept" à chaque nouveau client pour établir la liaison.
- "connect" côté client pour établir la connexion vers le serveur.

En Java c'est la classe *Socket* qui représente un socket. La méthode *connect* est invoquée en fournissant l'adresse IP et le numéro de port sur la machine serveur.

La classe *ServeurSocket* représente le "standard" d'appel. Le *bind* est réalisé à la construction en fournissant le numéro de port (l'adresse IP est trouvée automatiquement localement). La méthode *accept* est bloquante. Elle sera libérée par une connexion d'un client. Elle retourne un *socket* déjà connecté.

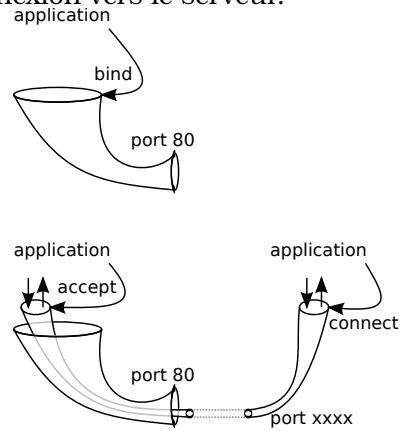


FIGURE 31 – TCP/IP

Voici un exemple de code Java pour un serveur qui attend un client unique. Il attend une ligne puis reste connecté pour lui envoyer l'entrée standard.

```
1       int port = Integer.parseInt (args [0]);
        ServerSocket serverSocket = new ServerSocket (port);
        System.err.println ("Start server on port "+port);

5       for (;;) {
```

```
    Socket call = serverSocket.accept ();
    System.err.println ("Client comming from "+
            call.getRemoteSocketAddress ());
    BufferedReader in =
        new BufferedReader
        (new InputStreamReader (call.getInputStream ()));
    PrintStream out =
        new PrintStream (call.getOutputStream (), true);
    BufferedReader sysin =
        new BufferedReader (new InputStreamReader (System.in));
    for (;;) {
        String line1 = in.readLine ();
        if (line1 == null)
      break;
        System.out.println (line1);
        System.out.flush ();
        String line2 = sysin.readLine ();
        out.println (line2);
        out.flush ();
    }
    System.err.println ("Socket closed");
      }
```

Listing 90 – Code Java d'un serveur de sockets

La ligne 2 crée le serveur. La ligne 6 reçoit une connexion. Les lignes 9 et 12 récupèrent les entrées/sorties du socket.

Voici un exemple de code Java pour un client de socket. Il se connecte et envoie l'argument du programme. Ensuite, il reste connecté tant que le serveur lui envoie du texte.

```
      Socket clientSocket =
    new Socket (args [0], Integer.parseInt (args [1]));
      PrintStream out =
    new PrintStream (clientSocket.getOutputStream (), true);
      out.println (args[2]);
      out.flush ();
      clientSocket.shutdownOutput ();
      BufferedReader in =
    new BufferedReader
    (new InputStreamReader (clientSocket.getInputStream ()));
      for (;;) {
    String line = in.readLine ();
    if (line == null)
        break;
    System.out.println (line);
    System.out.flush ();
    out.close ();
      }
```

Listing 91 – Code Java d'un client de socket

La ligne 1 établit une connexion. Les lignes 3 et 8 récupèrent les entrées/sorties du socket.

50.2 UDP/IP

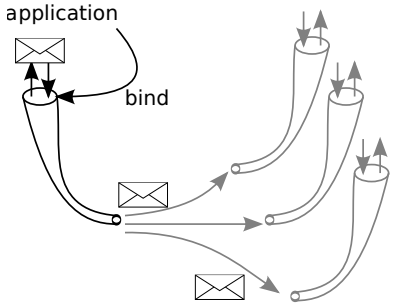


FIGURE 32 – UDP/IP

En mode non-connecté (UDP), l'application créée un socket particulier : le serveur de datagramme (message de données).

C'est le même type de socket pour les serveurs et les clients.

L'adresse IP et le port de l'émetteur des messages se trouvent dans les messages reçus eux-mêmes.

Voici un exemple de code Java pour la réception d'un datagramme. Il récupère le port dans l'argument de la commande. Il faut créer un tampon mémoire capable de recevoir le paquet de données en entier. Les données reçues sont sous forme d'octets. Il faut prévoir un mécanisme d'interprétation.

```
      int port = Integer.parseInt (args [0]);
      DatagramSocket socket = new DatagramSocket (port);

      byte[] buf = new byte[256];
      DatagramPacket packet = new DatagramPacket (buf, buf.length);
      socket.receive (packet);

      String received = new String (packet.getData(), 0, packet.getLength ());
      String srcIP = packet.getAddress ().getHostAddress ();
```

Listing 92 – Code Java de réception d'un datagramme

La ligne 2 crée le socket d'échanges de datagramme. La ligne 5 crée un datagramme d'accueil. La ligne 6 puise dans le datagramme reçu depuis la création. La ligne 9 affiche le message reçu et son expéditeur.

Voici un exemple de code Java pour l'envoi d'un datagramme. Il récupère les arguments, crée le datagramme et l'envoi.

```
      InetAddress address = InetAddress.getByName (args[0]);
      int port = Integer.parseInt (args [1]);
      byte[] buf = args[2].getBytes ();
      DatagramPacket packet =
    new DatagramPacket (buf, buf.length, address, port);

      DatagramSocket socket = new DatagramSocket ();
      socket.send(packet);
```

Listing 93 – Code Java de l'envoi d'un datagramme

La ligne 4 crée le datagramme. La ligne 8 l'envoie.

51 Protocoles IP

La plupart des protocoles de l'Internet datent des origines. À l'époque, on a choisi d'échanger les informations en mode texte. C'est donc des codes ASCII qui transitent par le réseau. Cela facilite la compréhension et la mise au point. L'origine de l'Internet étant étasunienne, les mots sont en anglais.

Un outil particulièrement simple a été développé. Il s'agit de la commande `telnet host port`. Elle permet d'établir une connexion avec un serveur donné en argument, de connecter l'entrée du socket au clavier et la sortie sur l'écran.

Les protocoles comme : SMTP (envoi de message), POP3 (consultation de message), FTP (transfert de fichiers), HTTP (navigation web)... échangent en clair sur le réseau.

Les outils graphiques que vous utilisez aujourd'hui, ne sont que des interfaces pour masquer ces échanges.

Vous pouvez sans difficulté vous connecter et discuter directement avec l'un de ces serveurs.

52 Annexe : code source

Voici le code complet des extraits donnés dans ce chapitre.

```java
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
package network;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;

public class RecvCmd {

    static public void main (String[] args) {
  if (args.length != 1) {
      System.err.println ("Usage: RecvCmd port");
      System.exit (1);
  }
  try {
      int port = Integer.parseInt (args [0]);
      ServerSocket serverSocket = new ServerSocket (port);
      System.err.println ("Start server on port "+port);

      for (;;) {
    Socket call = serverSocket.accept ();
    System.err.println ("Client comming from "+
            call.getRemoteSocketAddress ());
    BufferedReader in =
        new BufferedReader
        (new InputStreamReader (call.getInputStream ()));
    PrintStream out =
        new PrintStream (call.getOutputStream (), true);
    BufferedReader sysin =
        new BufferedReader (new InputStreamReader (System.in));
    for (;;) {
        String line1 = in.readLine ();
        if (line1 == null)
      break;
        System.out.println (line1);
        System.out.flush ();
        String line2 = sysin.readLine ();
        out.println (line2);
        out.flush ();
    }
    System.err.println ("Socket closed");
      }
  } catch (Exception e) {
      e.printStackTrace ();
  }
  System.err.println ("Server closed");
  System.err.flush ();
      }
}
```

Listing 94 – Réception de datagramme en Java

```java
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
package network;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintStream;
import java.net.Socket;

public class SendCmd {

    static public void main (String[] args) {
  if (args.length != 3) {
      System.err.println ("Usage: SendCmd host port request");
      System.exit (1);
  }
  try {
      Socket clientSocket =
    new Socket (args [0], Integer.parseInt (args [1]));
      PrintStream out =
    new PrintStream (clientSocket.getOutputStream (), true);
      out.println (args[2]);
      out.flush ();
      clientSocket.shutdownOutput ();
      BufferedReader in =
    new BufferedReader
    (new InputStreamReader (clientSocket.getInputStream ()));
      for (;;) {
    String line = in.readLine ();
    if (line == null)
        break;
    System.out.println (line);
    System.out.flush ();
    out.close ();
      }
  } catch (IOException e) {
```

```
      System.err.println ("Socket closed");
      System.err.flush ();
  } catch (Exception e) {
      e.printStackTrace ();
  }
    }
}
```

Listing 95 – Envoi de datagramme en Java

```
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
package network;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class RecvData {
    static public void main (String[] args) {
  if (args.length != 1) {
      System.err.println ("Usage: RecvData port");
      System.exit (1);
  }
  try {
      int port = Integer.parseInt (args [0]);
      DatagramSocket socket = new DatagramSocket (port);

      byte[] buf = new byte[256];
      DatagramPacket packet = new DatagramPacket (buf, buf.length);
      socket.receive (packet);

      String received = new String (packet.getData(), 0, packet.getLength ());
      String srcIP = packet.getAddress ().getHostAddress ();
      int scrPort = packet.getPort ();
      System.out.println (srcIP+":"+scrPort+" send : "+received);
  } catch (SocketException e) {
      e.printStackTrace ();
  } catch (IOException e) {
      e.printStackTrace ();
  }
    }
}
```

Listing 96 – Réception de datagramme en Java

```
// (c) F. Merciol
// Plus d'informations sur http://m3101.merciol.fr
package network;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

public class SendData {
    static public void main (String[] args) {
  if (args.length != 3) {
      System.err.println ("Usage: SendData host port data");
      System.exit (1);
  }

  try {
      InetAddress address = InetAddress.getByName (args[0]);
      int port = Integer.parseInt (args [1]);
      byte[] buf = args[2].getBytes ();
      DatagramPacket packet =
    new DatagramPacket (buf, buf.length, address, port);

      DatagramSocket socket = new DatagramSocket ();
      socket.send(packet);
  } catch (UnknownHostException e) {
      e.printStackTrace ();
  } catch (SocketException e) {
      e.printStackTrace ();
  } catch (IOException e) {
      e.printStackTrace ();
  }
    }
}
```

Listing 97 – Envoi de datagramme en Java

La théorie c'est quand on sait tout et que rien ne fonctionne.
La pratique c'est quand tout fonctionne et que personne ne sait pourquoi.
Ici, nous avons réuni théorie et pratique : rien ne fonctionne . . .
et personne ne sait pourquoi !

Albert Einstein

53 TD6 : Réseau

Dès votre arrivée en TD, pensez à déposer votre QCM rempli à votre encadrant. La partie à rendre est la dernière page de votre document (à découpe suivant les pointillés).

Après l'appel, c'est l'occasion de poser les questions concernant des éléments non compris du cours. Pensez à recopier le corrigé du QCM sur votre document (partie à concerver à la page précédante). Vous pourrez l'utiliser le jour de l'examen. Une page de notes personnelles est à votre disposition en fin de TD.

L'objectif du TD est de réaliser des commandes au travers du réseau. Il reprend les acquis des TD précédents : synchronisation, tâche, expression régulière . . . Le résultat sera la réalisation d'un logiciel de jeu implantant le jeu du "Romancier Scilof" dont les règles sont données en fin de TD.

54 Compréhension du problème

54.1 A vous de jouer !

Des cartes vous seront distribuées pour pouvoir vous exercer. Observer bien les différentes phases du jeu.

Faites des groupes de 6 (si possible). Commencer par faire une partie (sans variante). Vérifiez que vous respectez scrupuleusement les règles. Notez les incompréhensions ou les ambiguïtés.

Réféchissez à une implantation informatique. L'objectif est de réaliser un logiciel sans serveur.

54.2 Compréhension des règles

Question : Pourquoi dire "fermeture du bureau de poste" ? Qui cela concerne-t-il ?

Réponse : ✍

Question : Pourquoi dire "boîte pleine" ? Qui cela concerne-t-il ?

Réponse : ✍

Question : Comment vérifier qu'il n'y a pas eu de tricherie ?

Réponse : ✍

54.3 Compréhension des éléments du programme

Question : A quoi correspond le bureau de poste ?

Réponse : ✍

Question : A quoi correspond les numéros des joueurs ?

Réponse : ✍

Question : Comment réaliser le dépôt de la carte "enveloppe fermée" sur le bureau de poste ?

Réponse : ✍

Question : Comment réaliser le retournement de la carte enveloppe ?

Réponse : ✍

Question : A quoi correspond l'expression "à n'importe quel moment" ?

Réponse : ✍

Question : Qu'implique le fait que des évènements arrivent "à n'importe quel moment" ?

Réponse : ✍

Dans un contexte de réseau, les joueurs ne se voient pas. Il n'est donc pas aisé de déterminer le nombre a priori de joueurs. Nous allons donc réaliser la variante "fusion de communes".

Reprenez le jeu uniquement dans sa phase d'ajout de joueurs. Vérifiez que le fonctionnement permet la connexion des joueurs dans n'importe quel ordre d'appariement.

Question : Pourquoi 2 sortes d'invitation ?

Réponse : ✍

55 Réalisation

Pour rendre plus lisible les traces, les joueurs entreront un pseudo que nous afficherons à la place de l'adresse IP.

La réalisation se fera en mode texte.

Question : Imaginez une forme de saisie.

Réponse : ✍

Page de notes personnelles : ✍

Romancier Scilof

Règle du jeu

56 “Romancier Scilof” : le jeu

De façon concise, le jeu consiste à poster une lettre dans une enveloppe adressée à un autre joueur via le bureau de poste. Le but de chaque joueur est de réunir dans sa main une collection complète de lettres de valeur.

56.1 Origine

L’auteur a réalisé un travail de recherche remarquable pour trouver les éléments fondamentaux de ce jeu.

La légende prétend que *Romancier Scilof* aurait vécu en Transylvanie au 18e siècle. Il y aurait produit de nombreux ouvrages sur sa région des Carpathes (peut-être inspirât-il de Jules Vernes pour son roman gothique “Le Château des Carpathes” publié en 1892).

Il aurait formé un cercle de poètes (dont les ouvrages sont aujourd’hui disparus, car détruits lors de la bataille de “la passe de Dukla” en 1944). Ce cercle se réunissait nocturnement dans une grotte pour lire et exprimer des vers. Le groupe produisait également des textes de manière épistolaire, en envoyant au hasard des mots d’un membre vers les autres. Une fois qu’un membre trouvait que sa liste de mot formait une phrase suffisamment incongrue, il indiquait aux autres qu’il avait “fermé” (sa phrase), que chacun devait retrouver son “poste”, pour tenir un “bureau” du cercle.

Cette pratique, dont les traces ne subsistent que par tradition orale, serait à l’origine du jeu des surréalistes nommé “Cadavre exquis”.

Cette règle du jeu *Romancier Scilof* s’inspire de souvenirs lointains et mystérieux. L’auteur vous propose de revivre ces instants passés.

56.2 Contenu du jeu

Le jeu est constitué de deux paquets de cartes (voir la table 23) contenant :

- des lettres. Elles représentent les échanges entre membres du cercle de poètes. Une lettre de l’alphabet est inscrite sur chacune d’elle (symbolisant un contenu épistolaire).
- des enveloppes. Elles servent principalement à contenir les lettres qui sont échangées. Elles sont :
  - soit adressées à un autre joueur. Le recto pour l’envoi et le verso permet de retourner une réponse. Ces échanges sont confidentiels et ne concernent que le binôme communiquant.
  - soit utilisées pour mettre en place des rencontres. Dans ce cas, elles servent à organiser un cercle fermé d’échange parmi les joueurs potentiels autour de la table (variante “fusion de communes”). Ces cartes disposent de deux faces : le recto sert à inviter des joueurs à rejoindre le groupe, le verso sert à valider l’accroissement du nombre de participants dans le groupe.

  La couleur du rabat de la lettre (ainsi que le chiffre) permette d’identifier d’un coup d’œil le destinataire de l’envoi.

Les cartes sont toutes nominatives. Elles disposent au recto, du numéro du joueur dans chaque coin et d’un liseré de sa couleur. En revanche, le verso est anonyme.

Le centre de la table de jeu est appelé le bureau de poste. Il n’y a pas de tapis fourni pour matérialiser cet espace, et il n’est pas indispensable au déroulement de la partie.

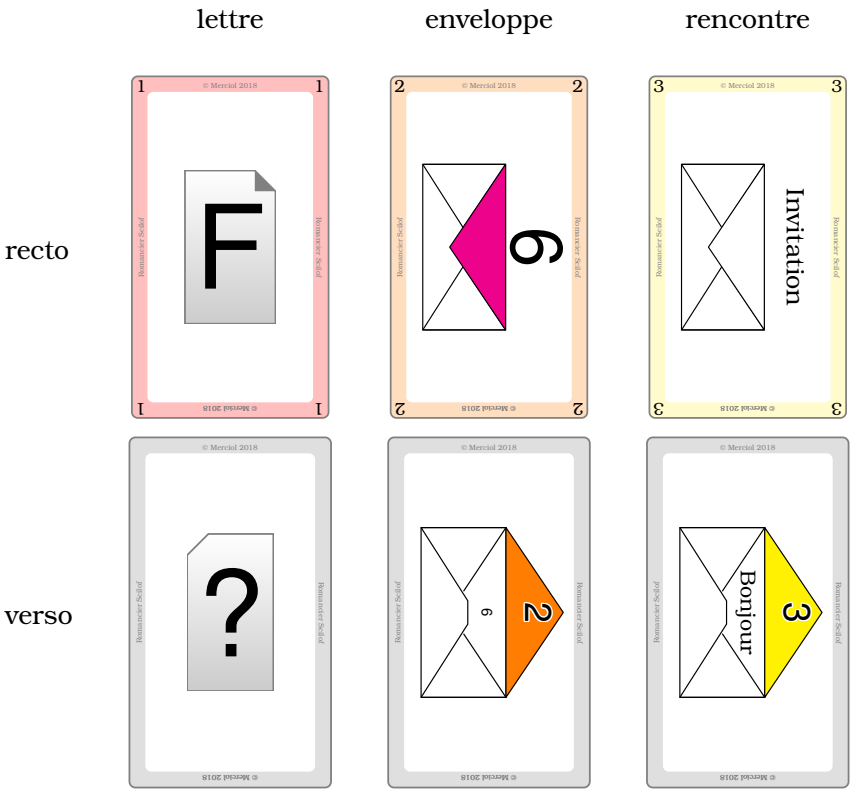


Table 23 – Cartes du jeu “Romancier Scilof”

56.3 Nombre de joueurs

Il est techniquement possible de joueur à partir de 2. Mais dans ce cas, deux échanges suffisent pour que les deux joueurs gagnent en même temps. À 3 joueurs, l'intérêt est discutable. Le jeu devient intéressant à partir de 4 joueurs.

Il n'y a en théorie pas de limite au nombre de joueur. Plus il y a de joueurs, plus longues sont les parties. À partir de 6 joueurs, il faut développer des stratégies. Si l'on souhaite jouer à grand nombre, il est préférable de limiter le nombre de cartes (par exemple 6 : de A à F). Les cartes de ce jeu sont imprimées pour au maximum 6 joueurs. Chaque joueur reçoit un chiffre (entre 1 et 6) et une couleur (voir la table 24)

1	2	3	4	5	6
rouge	orange	jaune	vert	ciel	magenta

Table 24 – Couleurs des joueurs du jeu “Romancier Scilof”

56.4 Début de partie

On définit en début de partie le nombre de joueurs. On distribue à chaque joueur ses cartes. Les joueurs n'utiliseront que le nombre de lettre et d'enveloppes suivant la table 25 :

- en commençant par la lettre A pour les lettres,
- en commençant par le chiffre 1 pour les enveloppes.

Nombre de joueurs	3	4	5	6
Lettres jusqu'à	C	D	E	F
Enveloppes jusqu'aux adresses	3	4	4	6

Table 25 – Cartes utilisées par joueur “Romancier Scilof”

La carte “Invitation/Bonjour” ne sert que dans la variante “fusion de communes”. Les autres cartes sont placées dans la réserve de chaque joueur.

56.5 Échange de courrier

Les joueurs peuvent poster du courrier quand ils le souhaitent. Il n'y a pas de tour. On dira que le mode de communication est asynchrone.

Avant la fermeture du bureau de poste (voir le paragraphe “Fin de partie”), chaque joueur peut poster autant de courrier qu'il le souhaite à la fois. Cependant, il peut être ennuyeux de se démunir de toutes ses lettres car (comme nous le décrirons par la suite) elles seront utiles également pour répondre.

Pour poster un courrier, un joueur choisit dans sa main une lettre et la pose face cachée (ou doit voir le dos avec son “ ?”) sur le bureau de poste (centre de la table). Il dépose ensuite sur celle-ci une enveloppe du destinataire voulu. On trouve alors deux cartes empilées : la lettre cachée (face contre table) en dessous, l'enveloppe fermée au-dessus montrant le destinataire.

Le destinataire doit faire son possible pour répondre à un courrier (même après fermeture du bureau de poste sauf pour ceux qui ont déclaré “fermeture du bureau”). Il ne doit pas découvrir la lettre avant de répondre (on ne peut pas sélectionner le courrier qu’on reçoit).

Pour répondre, le destinataire choisit une lettre dans sa main et la place (face cachée) à côté du courrier qui lui est adressé. Avec un geste auguste (n’oublions pas les nobles origines de ce jeu), il retourne l’enveloppe de l’émetteur sur la carte qu’il vient de mettre au bureau de poste. L’enveloppe a visiblement été ouverte et doit être retournée à l’envoyeur (la couleur du rabat a changé). Il récupère la lettre, dont il ne connaissait pas encore le contenu.

Les joueurs doivent obligatoirement et prioritairement reprendre leurs enveloppes ouvertes avec la lettre de réponse (même après fermeture du bureau de poste)

Un joueur peut récupérer un courrier fermé si visiblement son destinataire l’a oublié dans le bureau. À l’inverse un joueur peut refuser un courrier en retournant simplement l’enveloppe qui le recouvre. C’est le “retour à l’envoyeur”.

Ces situations peuvent arriver si un destinataire boude un écrivain qu’il sait ne pas receler des lettres qu’il convoite, ou si plusieurs écrivains se liguent pour polluer un adversaire. Mais, sachons en toutes occasions jouer un “jeu juste” (“fair play”).

56.6 Fin de partie

Lorsque qu’un joueur récupère dans sa main toutes les lettres de même valeur, il dit “fermeture du bureau de poste”. Sportivement, il réalise des “retour à l’envoyeur” (il refuse son courrier en souffrance, en retournant les enveloppes qui lui sont adressées) pour aider les écrivains à retrouver leurs correspondances.

Le moment est crucial dans la partie. A cet instant, les autres joueurs ne peuvent plus envoyer ni lettre, ni invitation (une fermeture de bureau en cours de phase d’invitation pose évidemment problème). En revanche, ils doivent répondre aux courriers envoyés.

Les joueurs qui ont toutes leurs cartes disent “boîte pleine”. Ces déclarations permettront de savoir quand les échanges cesseront. Rappelons qu’un joueur qui possède le nombre de cartes initial en main, n’est pas dispensé de répondre. Il pourra ainsi compléter une collection pendant cette phase de fin de partie.

La partie est terminée quand il n’y a plus de courrier en attente au bureau de poste. C’est-à-dire quand tout le monde aura dit soit “fermeture de bureau de poste” soit “boîte pleine”.

Les joueurs découvrent alors leurs mains aux autres. Les gagnants sont ceux qui ont complété leur collection (toutes les cartes ont la même valeur).

56.7 Technique de jeu

Comme chaque joueur doit constituer une collection, il est donc préjudiciable de divulguer cette information. A contrario, les enveloppes ne sont utiles que comme outil de transport des lettres, leur stockage est public.

Une bonne pratique consiste pour les joueurs à prendre en main leurs lettres pour en cacher le contenu. Ils peuvent placer devant eux les enveloppes fermées (on doit voir la face avec la bordure colorée). Ils conservent à l’écart une réserve de lettres et d’enveloppes non encore utilisées.

56.8 Variante nouveau joueur

Lorsqu’un nouveau joueur entre dans une partie, chaque joueur ajoute dans ses mains une lettre prise dans ses lettres supplémentaires et dans l’ordre alphabétique permettant d’atteindre le nombre de joueurs.

Le nouveau joueur part directement avec le bon nombre de lettres pris dans sa propre réserve.

56.9 Variante fusion de communes

Cette variante prend son sens quand il y a de nombreux joueurs. Elle permet de définir des communes possédant chacune un bureau de poste propre. Il est possible d’imaginer que deux groupes de joueur n’utilisent qu’une partie de la table. Par exemple, quatre joueurs sur la partie gauche et quatre joueurs sur la partie droite. Ils ont chacun un bureau de poste distinct (suivant leurs “communes”).

L’un des joueurs d’un groupe peut vouloir inviter un joueur de l’autre groupe à le rejoindre. Si l’autre accepte, les deux parties en cours seront fusionnées.

Dans le cas où un joueur lance une invitation. Il place sur un bureau de poste (le plus lisible de l’invité) une enveloppe “Invitation” avec l’enveloppe désignant le joueur invité (dans sa réserve d’enveloppes supplémentaires). Elle n’est pas encore ajoutée à sa main.

Le joueur invité peut décliner l’invitation. Pour cela, soit il ne fait rien, soit il retourne l’enveloppe portant son numéro dans le paquet d’invitation. Dans ce cas, l’invitant reprend ses cartes d’invitation est les replace dans sa réserve.

Si l'invité accepte, il ajoute dans sa main l'enveloppe de sa réserve correspondant à l'invitant et complète la lettre manquante (dans l'ordre alphabétique) pour atteindre le nombre de joueurs. Il place alors devant lui une enveloppe "Bonjour" avec toutes les enveloppes qu'il utilise (hors celles encore en réserve).

Tous les joueurs qui sont concernés par ce "Bonjour" vérifient qu'ils connaissent déjà tous les participants. Si c'est le cas, ils n'ont rien à faire et reprennent leurs cartes "Invitation" et "Bonjour" qu'ils auraient déposés sur bureau de poste. Sinon, ils ajoutent à l'ensemble des enveloppes utilisées celles qu'ils ne possédaient pas encore. Ils placent devant eux une carte "Bonjour" avec toutes les enveloppes qu'ils possèdent (donc avec les nouveaux arrivants). Cette étapes est répétée tant que de nouveaux joueurs apparaissent (les amis des amis qui étaient en train de jouer).

Chaque joueur complète le nombre de lettres pour correspondre au nombre de joueurs.

Les "Invitation" et "Bonjour" sont repris du bureau de poste lorsqu'elles ne sont visiblement plus utiles.

Dans le cas de la variante avec abandon de joueur, les "paniers de lettres abandonnées" des bureaux de postes concernés sont fusionnés.

56.10 Variante abandon de joueur

Dans cette variante on ajoute un "panier des lettres abandonnées" au bureau de poste.

Lorsqu'un joueur abandonne une partie, il ne poste plus de courrier. Mais la partie continue. Plusieurs joueurs peuvent partir en même temps.

Tous les autres joueurs ne peuvent plus poster les lettres dont la bordure indique un joueur parti.

Les joueurs qui abandonnent doivent :

- réaliser un "retour à l'envoyeur" sur le bureau de poste les enveloppes qui leur étaient destinées.
- reprendre dans le bureau poste toutes les enveloppes (avec les lettres qu'elles contiennent) qui leur appartiennent (suivant la bordure).
- placer dans le panier des lettres abandonnées les lettres (face cachée) dont la bordure correspondant aux joueurs actifs.

Les autres joueurs doivent :

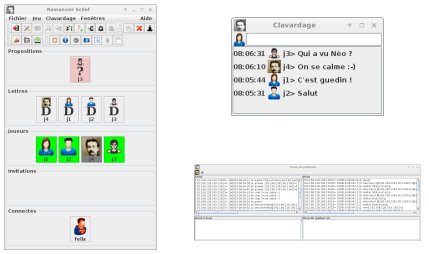
- rendre à ceux qui abandonnent les lettres qui leur appartiennent (suivant la bordure).
- retirer les lettres au-delà de celles permises en fonction du nombre de joueurs (voir table 25).
- limiter le nombre de lettre dans leur main au nombre de joueur. Les cartes supplémentaires sont placées (face cachée) dans le "panier des lettres abandonnées".
- reprendre les enveloppes qui leur sont destinées. Si leur main est déjà pleine, il place les lettres supplémentaires sans les regarder dans le "panier des lettres abandonnées"

Si un joueur ne dispose pas suffisamment de lettre en main en comptant les enveloppes qu'il a dans le bureau de poste, il doit compléter sa main en piochant au hasard dans le "panier des lettres abandonnées" (ne vous battez pas !).

56.11 Variante équipe

Les joueurs peuvent constituer des équipes. Le nombre des parties gagnées est mis en commun.

"Romancier Scilof" c'est également un logiciel à retrouver sur :
`http://romancierscilof.parlnet.org/`.



57 TP6 : Réseau

- Votre compte-rendu est à rendre en fin de TP.
- Vous ne devez pas rendre un listing, mais un document répondant aux questions posées.
- Vous préparerez le compte-rendu à l'avance (copier-coller les questions) pour gagner du temps.
- Vous rappelerez vos prénoms et noms, groupe TD, date et identifierez clairement le TP.
- Vous recopierez le texte des questions (et numéro) à chaque fois.
- Vous répondrez par des phrases.
- Vous recopierez les commandes que vous avez saisies pour obtenir un résultat et plus généralement ce que vous saisissez dans un terminal.
- Quand c'est nécessaire, vous pouvez faire une copie d'écran (uniquement la fenêtre concernée).
- N'oubliez pas de rendre votre TP imprimé en 2 colonnes de texte par côté de papier et de faire le TP en binôme.

L'objectif du TP est de réaliser un système communicant non centralisé en implantant le jeu du "Romancier Scilof" avec des datagrammes.

Le format des échanges (commandes) avec l'utilisateur est libre. En revanche pour permettre l'interopérabilité, il faudra respecter le protocole réseau. Les datagrammes contiennent des chaînes de caractères commençant par une date (AAMMJJ hh :mm :ss) et finissant par un retour à la ligne (\n). Les mots clefs sont ceux vus en TD.

Question : Vérifiez que les programmes sur les datagrammes du cours fonctionnent.

Dîtes comment vous les avez mis en œuvre et donnez les traces.

Réponse : ✍

Question : Chargez les fichiers jar et vérifiez que l'IHM fonctionne.

Dîtes comment vous les avez mis en œuvre et donnez les traces.

Réponse : ✍

Question : Vérifiez le fonctionnent en mode texte. Pour cela déclarez un autre utilisateur dans le terminal

192.168.128.161 :9999 Alice : guest Bob@192.168.128.161 :13623

Puis acceptez l'invitation avec l'IHM.

Déposez une carte

Regardez la trace

Repondrez avec une autre carte

Donnez le code et les traces.

Réponse : ✍

Question : Dévelopez la classe NetCom.java

Testez

Astuce : Vous pouvez lancer 2 jeux sur votre machine avec des ports différents

Donnez le code et les traces.

Réponse : ✍

Question : Testez entre 2 postes

Donnez le code et les traces.

Réponse : ✍

Question : Testez votre réalisation avec les autres.

Vérifiez l'interopérabilité.

Réponse : ✍

Question : Vérifiez qu'il n'y a pas eu de triche.

Donnez le code et les traces.

Réponse : ✍

QCM

(page gauche vide)

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 Quel est la marque de fin de ligne sous Unix ?

☐ \n ☐ \n\r ☐ \r\n ☐ \r

Question 2 Pour Unix, un numéro majeur désigne

☐ une version stable
☐ une expression régulière
☐ un développement important
☐ un dispositif matériel

Question 3 "`StringTokenizer`" découpe en

☐ paragraphes ☐ anneaux magiques ☐ mots ☐ lignes

Question 4 L'électricité est un vecteur de transport d'énergie, mais dans le monde, quelle source d'énergie est la moins utilisée par l'Internet ?

☐ Le renouvelable ☐ Les gaz de schiste ☐ Le nucléaire ☐ Le charbon

Question 5 Pour ne pas céder d'informations aux réseaux commerciaux de collecte de données personnelles (comme FaceBook), il faut

☐ bloquer les icônes "f" ☐ interdire les cookies ☐ protéger son profil ☐ faire un vœu

Question 6 Concernant l'emploi hors États-Unis Amérique, l'Internet

☐ repeuple les campagnes
☐ créer des libraires
☐ détruit des emplois
☐ augmente les circuits-courts

Question 7 Le SFINX est

☐ un routeur aléatoire
☐ l'interconnexion des réseaux en France
☐ le protocole d'échange égyptien
☐ un animal mythique

Question 8 Sous Unix, un processus démon ne peut jamais être

☐ tué ☐ suspendu ☐ défunt ☐ créé par un *moldu*

Question 9 Un processus Unix terminé dont le père n'a pas encore lu le testament se nomme

☐ en attente ☐ orphelin ☐ accidenté ☐ zombie

Question 10 Un `docker` c'est

☐ une carte d'extension ☐ un type musclé ☐ de la virtualisation ☐ 8 points au Scrabble

Question 11 On qualifiera une application de “temps réel” si des actions peuvent être

☐ abandonnées ☐ en moins d’1 ns ☐ en moins d’1 s ☐ en moins d’1 ms

Question 12 Un programme Java qui ouvre une fenêtre se termine

☐ après le “`main`” ☐ en temps déterminé ☐ par un banquet ☐ après “`System.exit`”

Question 13 En Java, un moniteur

☐ contrôle des tâches ☐ s’oppose au Mutex ☐ est un sémaphore ☐ prévient de la noyade

Question 14 Les objets Java ou C++ créés dynamiquement lors de l’exécution sont placés la mémoire nommée

☐ tas ☐ code ☐ données ☐ pile

Question 15 La mémoire flash est

☐ une mesure routière ☐ éphémère ☐ une EEPROM ☐ ineffaçable

Question 16 L’algorithme de suppression de page LRU :

☐ tient compte du dernier utilisé ☐ vide le cache
☐ prend en file ☐ prive de ressources

Question 17 Les premières mémoires électroniques (sans consommation d’énergie), utilisaient des

☐ dauphins ☐ anneaux magnétisés ☐ petites réserves d’eau ☐ rayons de lumière

Question 18 En informatique, l’acronyme RAID désigne un(e)

☐ algo de recherche ☐ PC en acier renforcé ☐ groupe de disques ☐ unité de police

Question 19 On décrit la structure des fichiers et répertoires par

☐ un filet ☐ un arbre ☐ un anneau ☐ une file

Question 20 Les premiers systèmes de gestion de fichiers sont apparus dans les années

☐ 70 ☐ 80 ☐ 90 ☐ 60

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 Sous Unix, un processus démon ne peut jamais être

☐ créé par un *moldu* ☐ suspendu ☐ tué ☐ défunt

Question 2 Un processus Unix terminé dont le père n'a pas encore lu le testament se nomme

☐ en attente ☐ accidenté ☐ orphelin ☐ zombie

Question 3 Un `docker` c'est

☐ 8 points au Scrabble ☐ une carte d'extension ☐ un type musclé ☐ de la virtualisation

Question 4 Sous Unix, un processus gelé se poursuit avec

☐ `^Z` ☐ `fg` ☐ `jobs` ☐ `kill -CONT`

Question 5 "`fork`" est une fonction qui

☐ clone un processus ☐ change de processeur ☐ lance un programme ☐ double le processeur

Question 6 En Shell, "`trap`"

☐ place un piège ☐ arrête un programme ☐ intercepte un signal ☐ démarre un processus

Question 7 Un `docker` c'est

☐ un type musclé ☐ une carte d'extension ☐ de la virtualisation ☐ 8 points au Scrabble

Question 8 Le dispositif qui attribue régulièrement des processus aux processeurs est

☐ une pile ☐ un ordonnanceur ☐ un sémaphore ☐ une horloge

Question 9 Un inter-blocage se résout par

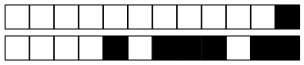
☐ réflexion humaine ☐ invocation mystique ☐ test combinatoire ☐ test aléatoire

Question 10 Sous Unix, un "*pipe*" est

☐ un "ou" logique ☐ une pile ☐ un "et" logique ☐ une file

Question 11 Dans un processeur, la virtualisation permet

☐ la visualisation en 3D ☐ la création de pseudo
☐ l'envoi de messages ☐ la cohabitation de système d'exploitation



Question 12 Un programme possède de façon déterminée

- ☐ une date de début
- ☐ une date de fin
- ☐ un ordre de passage
- ☐ des instructions

Question 13 Dans une section critique non protégée, la valeur des informations dépend

- ☐ de la mémoire utilisée
- ☐ de l'entrelacement des instructions
- ☐ des langages
- ☐ la vitesse du processeur

Question 14 Sous Unix, "`kill`"

- ☐ bloque les E/S
- ☐ arrête un programme
- ☐ envoie un signal
- ☐ libère la mémoire

Question 15 Un processeur

- ☐ règle les inter-blocages
- ☐ ordonne des programmes
- ☐ fixe les priorités
- ☐ interprète des processus

Question 16 Le nombre de processus actifs est —— nombre de processeurs.

- ☐ indéfinissable au
- ☐ supérieur au
- ☐ indépendant du
- ☐ inférieur ou égal au

Question 17 Quelle commande n'est pas nécessaire à l'écriture de la fonction "*système*" est

- ☐ `exit`
- ☐ `wait`
- ☐ `fork`
- ☐ `exec`

Question 18 Une section critique identifie des

- ☐ écriture concomitantes
- ☐ alertes vocales
- ☐ sémaphores
- ☐ moniteurs

Question 19 "`exec`" est une fonction qui dans un processus

- ☐ substitut son programme
- ☐ termine les fils
- ☐ exécute un processus
- ☐ termine le processus

Question 20 Un périphérique est un

- ☐ composant matériel
- ☐ logiciel
- ☐ algorithme
- ☐ contournement de la circulation des données

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ⬛5 ⬛6 ⬛7 ⬛8 ⬛9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 En informatique, l'acronyme RAID désigne un(e)

☐ algo de recherche ☐ unité de police ☐ PC en acier renforcé ☐ groupe de disques

Question 2 On décrit la structure des fichiers et répertoires par

☐ un filet ☐ une file ☐ un anneau ☐ un arbre

Question 3 Les premiers systèmes de gestion de fichiers sont apparus dans les années

☐ 90 ☐ 80 ☐ 70 ☐ 60

Question 4 Le boutisme est une

☐ religion ☐ option de démarrage ☐ terminaison de fichier ☐ représentation d'entier

Question 5 En informatique, MBR veux dire

☐ Master Boot Record ☐ Memory Boot Recover
☐ Mon Bien Réutilisable ☐ Manual Before Repair

Question 6 La table d'allocation contient la liste des

☐ blocs d'un fichier ☐ processus orphelins ☐ blocs du disque ☐ droits d'un fichier

Question 7 Pour les mémoires de masses, une partition est une fraction de

☐ temps ☐ fichiers sonores ☐ disque ☐ matériel

Question 8 Sous Unix, quelle partition est la plus stable ?

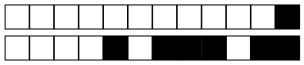
☐ /home ☐ /boot ☐ /tmp ☐ /

Question 9 Les blocs de même densité forment

☐ un cylindre ☐ un plateau ☐ un secteur ☐ une tête

Question 10 Sous Unix, le "*swap*" sert à

☐ augmenter la vitesse de tous les processus ☐ changer de programme
☐ nettoyer le disque ☐ augmenter la mémoire virtuelle



Question 11 En informatique, GPT est

- ☐ un ensemble de processus
- ☐ une architecture de processeur
- ☐ l'ordonnancement de tâche
- ☐ une structure de disque

Question 12 Avec la commande `ls -l` le "s" qui apparaît en tête du "mode" désigne un

- ☐ lien symbolique
- ☐ "*pipe*" nommé
- ☐ périphérique spécial
- ☐ "*socket*"

Question 13 Pour un débit régulier des données avec le disque, la vitesse de rotation d'un disque est

- ☐ plus rapide au milieu
- ☐ plus rapide au centre
- ☐ constante
- ☐ plus rapide à l'extérieur

Question 14 Le choix de représentation des données sur disque permet de

- ☐ maximiser la capacité de stockage
- ☐ simplifier la gestion
- ☐ diminuer le bruit du mécanisme
- ☐ diminuer le prix aux consommateurs

Question 15 Quelle unité est à la fois utilisée en coordonnées cylindriques et linéaires du disque ?

- ☐ le cylindre
- ☐ la piste
- ☐ le mètre
- ☐ le secteur

Question 16 En informatique, un "*inode*" est un

- ☐ débordement
- ☐ fichier
- ☐ routeur de l'Internet
- ☐ attribut d'un fichier

Question 17 Pour les mémoires de masses, CHS et LBA ont rapport avec des

- ☐ adresses
- ☐ type de support
- ☐ quantités de mémoire
- ☐ mesures de temps

Question 18 Un super-bloc contient les caractéristiques du

- ☐ système de fichiers
- ☐ repère Marvel
- ☐ répertoire
- ☐ montage

Question 19 Dans un disque, la tête désigne

- ☐ une face
- ☐ un arc de cercle
- ☐ un plateau
- ☐ un cercle

Question 20 Quel média utilise au maximum son support de mémorisation ?

- ☐ Le disque flash
- ☐ L'éléphant d'Asie
- ☐ Le CD/DVD
- ☐ Le disque dur

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 Les objets Java ou C++ créés dynamiquement lors de l'exécution sont placés la mémoire nommée

☐ pile ☐ code ☐ données ☐ tas

Question 2 La mémoire flash est

☐ éphémère ☐ ineffaçable ☐ mesure routière ☐ une EEPROM

Question 3 L'algorithme de suppression de page LRU :

☐ vide le cache ☐ prive de ressources
☐ prend en file ☐ tient compte du dernier utilisé

Question 4 Les premières mémoires électroniques (sans consommation d'énergie), utilisaient des

☐ rayons de lumière ☐ dauphins ☐ petites réserves d'eau ☐ anneaux magnétisés

Question 5 L'une des mémoires suivantes n'utilise pas la page comme granularité.

☐ Le disque ☐ Le swap ☐ La mémoire virtuelle ☐ Le cache processeur

Question 6 Le "code" est une zone mémoire contenant

☐ les règles de piraterie en Amérique centrale ☐ les instructions du programme
☐ les données des fonctions ☐ les règles de routage

Question 7 Quelle est l'option fixant le maximum de mémoire alloué au "tas" d'une JVM (machine Java) ?

☐ -Xms ☐ -Xmt ☐ -Xmx ☐ -Mg

Question 8 Une "pile" fonctionne sur le principe

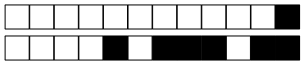
☐ LRU ☐ LIFO ☐ d'oxydoréduction ☐ FIFO

Question 9 Quelle mémoire ne peut être modifiée ?

☐ ROM ☐ Celle d'une huître ☐ DRAM ☐ EPROM

Question 10 L'algorithme le plus utilisé intervenant dans les défauts de page est

☐ OPT ☐ FIFO ☐ aléatoire ☐ LRU



+1/2/59+

Question 11 Pourquoi l'algorithme de défaut de page le plus efficace n'est-il jamais implanté ?

- ☐ Il a été perdu dans la poche d'un druide
- ☐ Par manque de place
- ☐ Par manque de temps
- ☐ Car il est difficile de prévoir l'avenir

Question 12 Début des années 80, les ordinateurs grands publics (Commodore, Atari, Amstrad, Amiga, Sinclair...) avaient une capacité de quelques

- ☐ péta octets
- ☐ méga octets
- ☐ octets
- ☐ kilo octets

Question 13 Les mémoires paginées et segmentées sont des mémoires

- ☐ de base de donnée
- ☐ virtuelles
- ☐ de masse
- ☐ de livre audio

Question 14 La mémoire électronique peut s'effacer par

- ☐ la maladie
- ☐ l'usage répété
- ☐ le froid
- ☐ la nature du contenu

Question 15 Bélády a montré

- ☐ l'efficacité de la segmentation
- ☐ qu'on peut mettre des accents partout
- ☐ l'efficacité des caches
- ☐ des lenteurs avec plus de mémoire

Question 16 La mémoire segmentée permet

- ☐ résoudre des calculs géométriques
- ☐ d'utiliser un cache processeur
- ☐ un calcul rapide d'adresse
- ☐ d'utiliser toute la mémoire

Question 17 ROM désigne

- ☐ Une mémoire morte
- ☐ Un alcool des Antilles
- ☐ Un effacement par surtension
- ☐ Un effacement par UV

Question 18 La zone de données pour le langage "C" contient les variables dites

- ☐ statiques
- ☐ volatiles
- ☐ automatiques
- ☐ météorologique

Question 19 Les données d'une mémoire DRAM

- ☐ s'effacent rapidement
- ☐ s'effacent par UV
- ☐ sont permanentes
- ☐ sont d'accès lent

Question 20 L'algorithme de LFU choisit de supprimer la page

- ☐ la moins utilisée
- ☐ dernièrement utilisée
- ☐ la plus utilisée
- ☐ raturée

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 On qualifiera une application de "temps réel" si des actions peuvent être

☐ en moins d'1 ms ☐ en moins d'1 s ☐ en moins d'1 ns ☐ abandonnées

Question 2 Un programme Java qui ouvre une fenêtre se termine

☐ par un banquet ☐ en temps déterminé ☐ après le "`main`" ☐ après "`System.exit`"

Question 3 En Java, un moniteur

☐ s'oppose au Mutex ☐ est un sémaphore ☐ prévient de la noyade ☐ contrôle des tâches

Question 4 Une tâche qui entre dans une méthode Java synchronisée peut appeler

☐ aucune méthode synchronisée ☐ aucune méthode non-synchronisée
☐ aucune méthode synchronisée d'autres objets ☐ toutes les méthodes

Question 5 Un serveur doit recourir à des tâches pour

☐ des clients simultanés ☐ accroître sa précision ☐ servir le café ☐ moins polluer

Question 6 L'invocation de la méthode "notifyall" d'un objet Java indique au moniteur

☐ d'annuler la synchronisation ☐ de réactiver toutes les tâches en attentes
☐ de libérer les sémaphores ☐ la fin de baignade surveillée

Question 7 En Java, un sémaphore s'écrit

☐ avec "join" ☐ sans "wait" ☐ avec "synchronized" ☐ sans "notify"

Question 8 Quelles sont les méthodes que peut bloquer le moniteur d'un objet Java ?

☐ Non synchronisées ☐ Toutes ☐ Récursives ☐ Synchronisées

Question 9 Dans quelle classe Java se trouve la méthode "sleep" ?

☐ `Thread` ☐ `Miller (meunier)` ☐ `Runnable` ☐ `Monitor`

Question 10 On peut visualiser les tâches ("threads") d'une Machine Virtuelle Java avec la commande "*shell*"

☐ c'est impossible ☐ `ttop` ☐ `lsthread` ☐ `ps`

Question 11 En Java, l'invocation "wait" dans une méthode synchronisée implique pour un objet de cette classe

- ☐ la présence de tâches en concurrence
- ☐ l'utilisation de `notifyall`
- ☐ l'utilisation d'attributs
- ☐ la possibilité de changer de tâche active

Question 12 En Java, pour lancer une nouvelle tâche, il ne faut jamais

- ☐ créer un "`Runnable`"
- ☐ créer un "`Thread`"
- ☐ jeter l'encre
- ☐ invoquer "`run`"

Question 13 On peut tuer une tâche ("*thread*") Java sans quitter la JVM avec la commande *Shell*

- ☐ `rmthread`
- ☐ non, c'est impossible
- ☐ `kill`
- ☐ `stop`

Question 14 En Java, l'instruction "`wait`"

- ☐ bloque les méthodes synchronisées
- ☐ attend une notification
- ☐ signale une anomalie
- ☐ bloque toutes les méthodes

Question 15 La méthode "`run`" d'un objet Java

- ☐ lance une activité
- ☐ ne peut être récursive
- ☐ doit être dans le "`main`"
- ☐ ne prend pas de paramètre

Question 16 Le développeur décide du démarrage d'une tâche par

- ☐ création d'un "`Runnable`"
- ☐ invocation de "*run*"
- ☐ création d'un "`Thread`"
- ☐ invocation de "`start`"

Question 17 L'attente de la fin d'une tâche Java se fait en invoquant la méthode

- ☐ "`join`"
- ☐ "`RDV`"
- ☐ "`wait`"
- ☐ "`p`"

Question 18 La préemption d'un processus

- ☐ se fait entre 2 cycles d'horloge
- ☐ attend la fin des boucles
- ☐ via un acte notarié
- ☐ attend la fin des écritures

Question 19 Java crée automatiquement une tâche pour une application qui

- ☐ affiche une fenêtre
- ☐ connecte un *socket*
- ☐ fait des calculs de précision
- ☐ gère des exceptions

Question 20 De quelle manière l'invocation de la méthode "notify" d'un objet Java agit-elle sur les tâches ?

- ☐ Pas d'effet avant la fin de la méthode
- ☐ Réveille immédiatement la 1^{re}
- ☐ Réveille les autres objets
- ☐ Réveille toutes les tâches

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 Quel est la marque de fin de ligne sous Unix ?

☐ \n\r ☐ \r\n ☐ \r ☐ \n

Question 2 Pour Unix, un numéro majeur désigne

☐ une expression régulière
☐ un développement important
☐ une version stable
☐ un dispositif matériel

Question 3 "`StringTokenizer`" découpe en

☐ anneaux magiques ☐ paragraphes ☐ lignes ☐ mots

Question 4 Sous Unix, l'ouverture d'un fichier en mode "`w`"

☐ limite à l'écriture seule
☐ limite à la lecture seule
☐ conserve le contenu
☐ efface le contenu

Question 5 En Java, la classe la plus indiquée pour afficher une date est

☐ "`SimpleDateFormat`" ☐ "`Affaire`" ☐ "`MessageFormat`" ☐ "`NumberFormat`"

Question 6 Sous Unix, l'ouverture d'un fichier en mode "a+"

☐ place l'écriture au début
☐ place la lecture en fin
☐ place la lecture au début
☐ est réservé aux fichiers bien notés

Question 7 Pour Unix, un numéro mineur désigne

☐ une version instable
☐ un développement négligeable
☐ un périphérique particulier
☐ une expression irrégulière

Question 8 Quelle classe permet l'utilisation de la lecture de texte en ligne ?

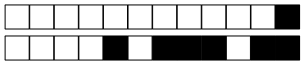
☐ "`InputStreamReader`" ☐ "`BufferedReader`" ☐ "`Wikipedia`" ☐ "`FileReader`"

Question 9 Les conditions du montage d'un fichier ISO dans une partition visible dans l'arbre nommage

☐ requièrent l'option "loop"
☐ nécessitent un chef d'orchestre habile
☐ sont impossibles
☐ imposent une partition différente

Question 10 En PHP objet, pour créer un document XML vierge, il suffit de faire "`new`"

☐ "`XMLEmptyDocument`" ☐ "`XMLDocument`" ☐ "`<xml>`" ☐ "`DOMDocument`"



Question 11 En java, un objet "`Matcher`" est rendu par un objet "`Pattern`" suite à l'invocation de la méthode

☐ "`pattern` ☐ "`toString`" ☐ "`group`" ☐ "`matcher`"

Question 12 Lorsqu'un processus Unix se divise ("`fork`"), les données dans les mémoires tampons d'écriture sont

☐ envoyées juste avant ☐ effacées ☐ placées dans le père ☐ dupliquées

Question 13 En Java, "`StreamTokenizer`" ne gère pas

☐ les commentaires C++ ☐ les commentaires C
☐ la priorité des opérateurs ☐ les nombres

Question 14 La référence ISO 8859-15 désigne

☐ l'alphabet latin sans € ☐ l'alphabet latin avec €
☐ le format de CD ☐ l'alphabet cyrillique

Question 15 Sous Unix, l'ouverture d'un fichier en mode "r+"

☐ place la lecture en fin ☐ place l'écriture en fin
☐ limite à la lecture seule ☐ place l'écriture en début

Question 16 La lecture d'un fichier binaire en mode texte

☐ est plus lente ☐ est impossible
☐ impose une conversion ☐ perd de l'information

Question 17 Sous Unix, la base de données des mots de passe utilise comme délimiteur

☐ : ☐ \t ☐ ! ☐ ,

Question 18 En Java, "`ChoiceFormat`" est

☐ une classe ☐ une méthode ☐ un attribut ☐ juste une illusion

Question 19 En Java, il y a un objet "Pattern" par

☐ expression régulière ☐ ligne ☐ chaîne à analyser ☐ stop

Question 20 Le formatage utilisant la chaîne de caractère "%d", ne peut être utilisé dans le langage

☐ lisp ☐ Python ☐ Java ☐ C

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ indiquez votre numéro d'étudiant (un chiffre par ligne), et votre nom ci-dessous.

Prénom et Nom :

....

Groupe TD : ☐A ☐B ☐C ☐D

Question 1 L'électricité est un vecteur de transport d'énergie, mais dans le monde, quelle source d'énergie est la moins utilisée par l'Internet ?

☐ Le charbon ☐ Le nucléaire ☐ Les gaz de schiste ☐ Le renouvelable

Question 2 Pour ne pas céder d'informations aux réseaux commerciaux de collecte de données personnelles (comme FaceBook), il faut

☐ faire un vœu ☐ interdire les cookies ☐ protéger son profil ☐ bloquer les icônes "f"

Question 3 Concernant l'emploi hors États-Unis Amérique, l'Internet

☐ repeuple les campagnes ☐ créer des libraires
☐ augmente les circuits-courts ☐ détruit des emplois

Question 4 Le SFINX est

☐ un animal mythique ☐ un routeur aléatoire
☐ le protocole d'échange égyptien ☐ l'interconnexion des réseaux en France

Question 5 Quel était l'usage du premier réseau de l'Internet ?

☐ Militaire ☐ Poétique ☐ Mathématiques ☐ Physique

Question 6 Pour les connexions TCP/IP, le port de l'émetteur

☐ n'existe pas ☐ est tiré au hasard par le système
☐ est supérieur à 65 535 ☐ est pris dans une ferme de wiki

Question 7 Quel est le cycle énergétique le plus court ?

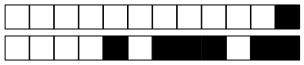
☐ le pétrole ☐ le nucléaire ☐ le papier ☐ corriger des copies

Question 8 UDP signifie

☐ Union Départementale des Prestaires numérique ☐ Unify Data Protocol
☐ User Datagram Protocol ☐ Union Départementale des Prolétaires

Question 9 Le nombre maximum de machines identifiables d'une classe A en IP V4 est de

☐ 16 777 214 ☐ 2 097 152 ☐ 65 534 ☐ 254



Question 10 Le chiffre d'affaire des GAFAM en 2013 est environ

☐ $< 66^e$ PIB mondial ☐ $<$ PIB du Soudan ☐ 20 millions de $ ☐ 200 milliards de $

Question 11 Concernant le climat, l'Internet

☐ lutte contre la déforestation ☐ diminue la température en Californie
☐ régule la fonte des glaces ☐ augmente l'effet de serre

Question 12 En TCP/IP, une application serveur récupère le "*socket*" de communication avec un client

☐ avec des sandales ☐ jamais ☐ au hasard ☐ via le serveur "*socket*

Question 13 Les compagnies étasuniennes dites "gratuites" (cotées en bourse) revendent les données personnelles

☐ des extraterrestres ☐ de tous ☐ des inscrits ☐ des visiteurs

Question 14 Sous IP, un datagramme permet

☐ la mesure des paquets ☐ l'envoi asynchrone
☐ des messages limités à 256 octets ☐ l'échange de données urgentes

Question 15 La vente des données personnelles sur l'Internet

☐ diminue les prix ☐ aide les pays pauvres ☐ enrichit les étudiants ☐ enrichit les États-Unis

Question 16 Qui imagina en premier la communication par paquet de l'Internet ?

☐ La NSA ☐ Le gouvernement ☐ Des pacifistes ☐ Le MIT

Question 17 La taille des messages échangés en UDP/IP (en octets) est

☐ quelconque ☐ égale à 263 ☐ inférieure à 256 ☐ inférieure à 512

Question 18 Par rapport à l'achat d'un DVD chez un marchand (transport inclus), l'empreinte carbone d'un film téléchargé est

☐ $>$ de 270 % ☐ $<$ de plus 50 % ☐ négligeable ☐ $<$ de 30% à 50 %

Question 19 Renater est un

☐ réseau public de la recherche ☐ opérateur privé de l'Internet
☐ réseau départemental des entreprises ☐ réseau reliant les bateaux à "Rennes"

Question 20 Qui payent les services dits "gratuits" de l'Internet ?

☐ les plus pauvres ☐ les utilisateurs ☐ les plus riches ☐ les PDG des GAFAM