Capes Informatique

Polycopié d'exercices - Système

Table des matières

Résumé

Ce cahier Ce cahier d'exercices a été réalisé pour l'enseignement Système du Capes Informatique, proposé par l'université Lyon1.

Intervenants Les personnes suivantes ont été impliquées dans l'écriture de ces exercices et des supports associés, et l'enseignement correspondant :

— 2018/2019 : Laure Gonnord, Nicolas Louvet, Fabien Rico.

`http://master-info.univ-lyon1.fr/CAPES/`

TD 1

Linux ligne de commande, Arborescence

1.1 Arborescence de fichiers : chemins

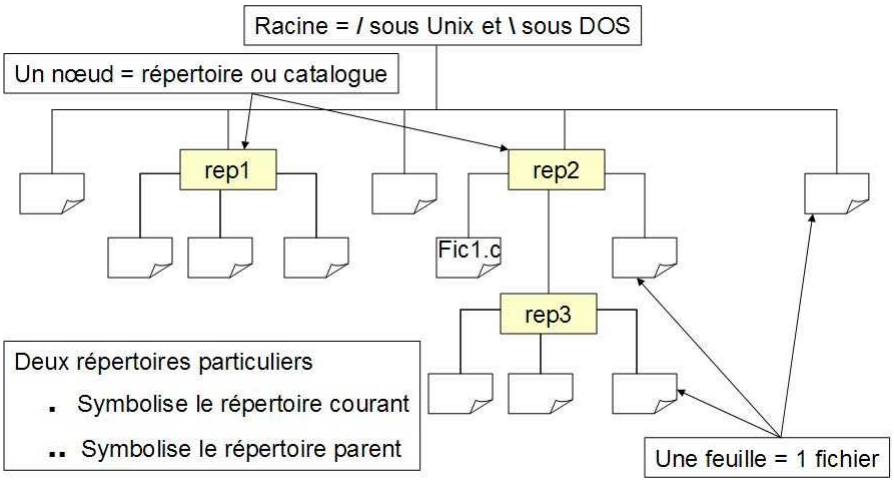

FIGURE 1.1 – Arborescence et chemin sous Linux

La figure 1.1 résume les concepts à retenir.

— *Chemin absolu* d'un fichier : c'est le nom "complet" du fichier, donné par son chemin d'accès depuis la racine. Exemple, sous Unix : **`/rep/fic1.c`**.

— *Répertoire de travail* (répertoire courant) : répertoire sur lequel un utilisateur est positionné à un moment donné. Ce répertoire est nommé "`.`"

— *Chemin relatif* d'un fichier : référence d'un fichier à partir du répertoire de travail.

— *Répertoire personnel* (sous Unix) : répertoire sur lequel est positionné un utilisateur suite à son authentification. Ce répertoire est nommé "`~`" (tilde). On parle aussi de "home utilisateur" ou de "racine personnelle". Sous Unix c'est souvent un sous-répertoire de `/home`.

EXERCICE #1 ▶ **Chemins (TD)**

La figure 1.2 représente l'arborescence présente dans un ordinateur sous Unix :

(Les fichiers sont en grisé, les autres noeuds sont des répertoires). Supposons que vous êtes l'utilisateur `alan`. Donner des commandes sucessives pour :

Q.1.1) - Vous placer dans votre répertoire d'accueil

Q.1.2) - Y créer un répertoire nommé `TP`.

Q.1.3) - Faire une copie du fichier `hosts.conf` dans le répertoire `TP`. Puis la renommer `myhosts.conf`.

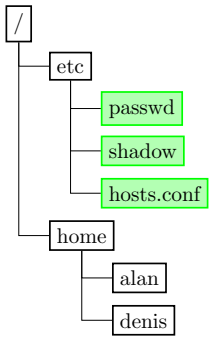

FIGURE 1.2 – Arborescence 1

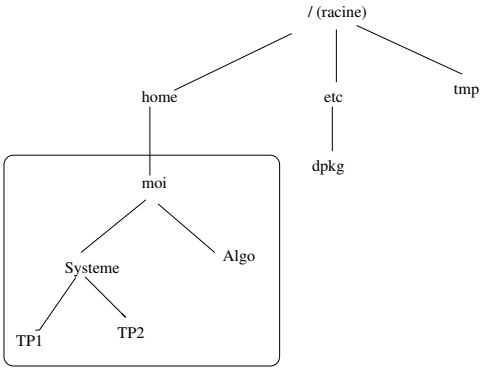

FIGURE 1.3 – Arborescence à réaliser

EXERCICE #2 ▶ **Navigation relative et absolue - TD ou TP**
Donner les commandes ou réaliser les manipulations suivantes :

Q.2.1) - Réaliser la hiérarchie de répertoires et de fichiers décrite dans la figure 1.3
Si ce n'est pas le cas, effectuer les modifications nécessaires (à l'aide des commandes `mv` et `mkdir`?)

Q.2.2) - Se placer dans le répertoire TP2. Pour copier un fichier vers le répertoire TP1, vous pouvez au choix (réaliser chacune de ces manipulations) :

- utiliser le chemin relatif (du répertoire TP1 par rapport à où je suis, ie TP2) : `cp nomfichier ../TP1/`
- utiliser le chemin absolu du répertoire TP1 : `cp nomfichier /home/moi/Systeme/TP1/`
- utiliser le raccourci de votre *home* : `cp nomfichier ~/Systeme/TP1/`

Q.2.3) - Se placer à la racine de votre répertoire personnel. *Sans se déplacer*, et en s'inspirant des manipulations précédentes :

- copier un des fichiers du répertoire `/etc/dpkg` dans le répertoire `TP2`.
- lister le contenu du répertoire `TP1`.
- déplacer les fichiers qui n'ont rien à faire de `TP1` vers `/tmp`.

1.2 Droits posix

EXERCICE #3 ▶ **Droits sous unix**
Sur le système considéré, il y a 4 utilisateurs :

- `fontaine` qui fait partie du groupe `prof` et `user`;
- `elise` qui fait partie des groupes `etu` et `user`;
- `hippolyte` qui fait partie du groupe `prof`.
- `root` qui est l'administrateur et fait parti du groupe `root`

Q.3.1) - Représentez les possibilités d'accès des 4 utilisateurs aux fichiers `visioneurPDF`, `sujet.pdf`, `correction.pdf` et `notes.ods`.

```
drwxr-x--x 27      root  user   /bin/
-rwsr-xr-x  1      root   etu   /bin/visioneurPDF
drwxr-xr-x 80      root  user   /home/
drwxr-x--x 10  fontaine  prof   /home/fontaine/
drwx--x---  4  fontaine  prof   /home/fontaine/prive/
-rw-r-x---  1  fontaine  prof   /home/fontaine/sujet.pdf
-rw-r--r--  1 hippolyte  prof   /home/fontaine/prive/correction.pdf
-rw-rw----  1  fontaine  prof   /home/fontaine/prive/notes.ods
```

Attention, pour les fichiers vous devez tenir compte des droits des répertoires et sous répertoires.

TD 2

Linux ligne de commande, Arborescence

2.1 Le manuel

`man` (pour *manual*) est une commande disponible sur les systèmes d'exploitation de type Unix. Elle permet d'obtenir de l'aide sur la plupart des programmes disponibles sur le système, mais aussi sur les fonctions du langage C, et de certaines bibliothèques. Pour l'invoquer, la commande est :

```
man nom_de_la_commande
```

Par exemple,

```
man du
```

permet d'obtenir le manuel de la commande du.

Une fois que la page manuel est affichée, les touches de déplacement permettent de naviquer dedans. Pour quitter (revenir à la ligne de commande), il suffit de taper Q. Il y a beaucoup de subtilités que l'on passe ici sous silence, mais vous pouvez maintenant faire `man man` pour en apprendre plus sur `man`!

EXERCICE #1 ▸ **Man**

Voici ci-dessous un extrait de la page de manuel de la commande `ls`.

```
NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILEs (the current directory by default). Sort entries
    alphabetically if none of -cftuvSUX nor --sort is specified.

    Mandatory arguments to long options are mandatory for short options too.

    -c    with -lt: sort by, and show, ctime (time of last modification of file
        status
          information); with -l: show ctime and sort by name; otherwise: sort by
              ctime,
          newest first

    -h, --human-readable
          with -l and/or -s, print human readable sizes (e.g., 1K 234M 2G)

    --si  likewise, but use powers of 1000 not 1024

    -l    use a long listing format

    -r, --reverse
          reverse order while sorting

    -R, --recursive
          list subdirectories recursively
```

```
-s, --size
      print the allocated size of each file, in blocks

-S    sort by file size, largest first

-t    sort by modification time, newest first

-u    with -lt: sort by, and show, access time; with -l: show access time and
    sort
      by name; otherwise: sort by access time, newest first
```

Q.1.1) - Quelles options de `ls` utiliser pour lister le contenu du répertoire courant dans l'ordre inverse de l'ordre alphabétique ?

Q.1.2) - Quelles options utiliser pour lister le contenu du répertoire par ordre de date de dernière modification, avec les fichiers modifiés le plus récemment à la fin ?

Q.1.3) - Quelles options pour ranger les fichiers par ordre de taille décroissante ?

2.2 Gestion de la mémoire

EXERCICE #2 ▶ **Version compliquée**

De façon similaire à la mémoire des processeurs INTEL PENTIUM, la table 2.1 représente une mémoire paginée :

- Chaque adresse est codée sur 9 bits.
- Une adresse de 9 bits correspond à 1 mot mémoire de 32 bits
- Chaque page contient 8 mots mémoires (1 ligne du tableau 2.1).
- Il y a 2 niveaux d'indirection (table de répertoires de page et table de pages).
- Une adresse logique est composée de 9 bits, de gauche à droite : 3 pour le répertoire de pages, 3 pour la page et 3 pour le décalage dans la page.
- Dans les tables de pages, pour chaque page, 7 bits sont utilisés pour détailler les propriétés de la page :
  1. pour signaler l'existence de la page
  2. pour signaler la présence de la page en mémoire
  3. pour signaler le droit d'écriture
  4. pour signaler le droit d'exécution
  5. pour signaler le « copy-on-write »
  6. pour le bit d'accès
  7. pour le dirty bit
- Dans les tables de répertoire de pages, seul le premier bit d'information est utilisé (celui de l'existence du répertoire correspondant).

On considère 2 processus, la table des répertoires de pages du processus 1 est à l'adresse 0o01 et celle du processus 2 à l'adresse 0o17.

Q.2.1) - Quel est la capacité d'adressage ?

Q.2.2) - Pour chacun des deux processus, que contient la case d'adresse 0b101111001.

Q.2.3) - Que se passe-t-il si le processus 1 essaye de lire la valeur de la case 0b101101101

Q.2.4) - Même question si le processus 2 tente d'exécuter le code de l'adresse 0b000001011

Q.2.5) - Que remarque-t-on pour l'adresse 0b101010000 du processus 1 et l'adresse 0b000101000 du processus 2 ? Dans quelle cas cela peut-il arriver ?

Q.2.6) - Mêmes questions pour l'adresse 0b000011111 des 2 processus. Que se passe-t'il si l'un des processus écrit à cette adresse.

	00	01	02	03	04	05	06	07
00	000.00	100.16	100.05	101.03	00.03	110.17	000.14	010.03
01	110.17	001.17	001.00	110.02	101.14	000.12	001.13	001.00
02	qYRd	ZT7z	lBe2	1Shn	cGXa	7w4D	l63x	ZjvX
03	Itd	nzuZ	Dgjc	4DBr	EGjc	4X23	kISa	aGW4
04	9Xah	Hc8	Mlyl	B1i	hAXR	TrVX	ljRK	30wT
05	atch	ing	you	FV7H	RPn1	9fji	GoE2	fu i
06	s120	4xvs	Vcv	hlmZ	MODe	KjdG	ElD1	mNGi
07	D5I9	1v9N	P lH	ckgY	PU73	rgYr	fyxr	3RnN
10	c86H	oA	BZJL	ZRzM	G6be	CgX5	lb 8	VZd1
11	Nuhd	Y9Xw	PsH1	agP	LEPJ	4pXX	M8li	9jSI
12	i4dr	UEjk	kT9w	9nxB	iGBj	KY4W	fLUh	7JTs
13	rbZI	zdDg	UxpR	6gqq	oMKl	Kt9	rCy9	7xXm
14	Ulym	yg3f	UTBC	Setz	bLvq	Lvtf	oJeR	ghWC
15	LnDf	vy5T	8GAj	kPUg	cdki	LVyr	xF50	b0Jz
16	Pp2u	KFyA	Izer	DwKE	Bigb	roth	er i	s wh
17	CIs5	M5TV	zwx6	zes	iver	YMv1	MK4E	iT9i

FIGURE 2.1 – Mémoire simplifiée

EXERCICE #3 ▶ **Version simple**

De façon similaire à la mémoire des processeurs, la table 2.1 représente une mémoire paginée :

— Chaque adresse est codée sur 6 bits (2 chiffres entre 0 et 8).

— Une adresse de 6 bits correspond à 4 octets (4 caractères).

— Chaque page contient 8 mots mémoires (1 ligne du tableau 2.1).

— Il y a une indirection. À partir de l'adresse demander on calcul l'adresse réelle en regardant la position de la page dans la table des pages qui dépend du processus. Les 2 processus représentés n'ont donc pas les même données associées au même adresses.

— Une adresse logique est composée de 6 bits, de gauche à droite : 3 pour la page et 3 pour le décalage dans la page.

— Dans les tables de pages, pour chaque page, 3 bits sont utilisés pour détailler les propriétés de la page :

1. pour signaler l'existence de la page
2. pour signaler le droit d'écriture
3. pour signaler le droit d'exécution

On considère 2 processus, la table des répertoires de pages du processus 1 est à l'adresse 0o00 et celle du processus 2 à l'adresse 0o01.

Q.3.1) - Lisez les 28 octets de la chaîne de caracrète à l'adresse `bx001100` du processus 1.

Q.3.2) - Lisez les deux octets de la chaîne de caracrète à l'adresse `bx101011` du processus 1.

Q.3.3) - Lisez les deux octets de la chaîne de caracrète à l'adresse `bx000011` du processus 2.

Q.3.4) - Que remarquez-vous ?

adresse de la page	Décalage dans la page							
mem	0	1	2	3	4	5	6	7
00	70	48	22	142	32	126	215	128
01	1100000.11	0000000.13	0000000.07	0000000.02	0000000.21	1100000.33	0000000.23	0000000.11
02	107	63	71	241	171	91	93	223
03	29	167	27	139	154	87	59	79
04	33	54	197	180	13	112	71	168
05	130	15	233	70	8	206	76	139
06	178	227	4	159	116	23	58	200
07	204	28	123	120	61	173	51	11
10	185	63	13	130	24	18	117	87
11	1101010.05	1101000.06	1100110.25	1110110.37	0010100.57	0010010.77	0001100.66	0010000.74
12	157	115	237	19	159	219	30	187
13	0000000.16	0010100.37	0001111.67	0000000.37	0000000.03	0000000.34	0000000.34	1110000.20
14	197	248	206	145	47	28	197	249
15	129	126	122	137	205	70	220	127
16	1101010.05	1101000.06	1100110.25	1110110.37	0011000.46	1110000.14	0001010.06	1110000.27
17	1100000.16	0000000.07	0000000.25	0000000.20	0000000.25	1100000.13	0000000.07	0000000.06
20	122	17	153	217	242	151	150	209
21	205	201	173	237	183	208	31	78
22	147	148	236	235	243	118	195	249
23	145	42	235	41	148	181	30	83
24	193	36	2	240	167	207	147	192
25	69	69	216	223	163	41	58	141
26	41	93	176	16	99	10	178	207
27	223	135	55	26	205	211	248	24
30	193	26	8	3	42	191	181	232
31	165	162	122	104	67	191	78	123
32	219	55	208	116	148	253	55	143
33	0001010.57	0010100.32	1110010.14	0011000.31	0000000.63	0001111.31	0001010.06	1110011.23
34	191	142	215	171	241	121	112	180
35	1	94	177	206	225	148	86	2
36	111	5	242	120	32	44	143	201
37	0	0	0	0	0	0	0	0

TABLE 2.1 – Exemple de mémoire

TD 3

Linux : autres commandes utiles

Contenu

Cette section vous propose des exercices à réaliser avec vos élèves de classe de Terminale. Elle ne sera pas réalisée en séance.

3.1 Commandes

3.1.1 Afficher du texte

Utilisation des commandes `cat, less, more`.

EXERCICE #1 ▸ **cat, etc.**
Pour tester ces commandes on va manipuler les fichiers qui se trouvent dans `/etc/dictionaries-common`. Placez-vous dans ce répertoire.

1. Tapez :
   ```
   ls -l
   cat words
   cat words words
   ```
   Que fait la commande cat ? Peut-on visualiser de gros fichiers ?
2. Utilisation de `less` :
   ```
   less  words
   ```
   puis
   - tapez sur la touche "espace"
   - tapez sur la touche "espace"
   - tapez sur la touche "espace"
   - tapez sur la touche "b"
   - tapez sur la touche "b"
   - tapez sur la touche "flèche du haut"
   - tapez `/foo` puis la touche entrée ou saut de ligne)
   - tapez `/z` puis (la touche entrée ou saut de ligne)
   - tapez sur la touche "flèche du haut"
   - tapez sur la touche "h" (regardez 30 secondes et continuez)
   - tapez sur la touche "q" (pour sortir de help)
   - tapez sur la touche "q" (pour sortir de less)

   Que fait less quand on tape "/quelquechose" ?
3. Comparez `less` et `more`.

3.1.2 Commandes pour fouiller le texte

EXERCICE #2 ▸ **Utilisation simple de `wc` et `grep`**
Toujours dans le répertoire `/etc/dictionaries-common` :

1. Que fait la commande `wc <nomfichier>`, et ses options `-l` et `-w` ?
2. Que fait la commande `grep` :
   ```
   grep house words
   grep maison words
   ```

3.1.3 Éditer/modifier un fichier

Il existe des commandes pour modifier un fichier à la ligne de commmande, mais ici nous allons faire plus simple.

EXERCICE #3 ▶ **Fichier**
Créer un fichier texte dans le répertoire courant en utilisant un éditeur de texte :

```
gedit monjolifichiertexte.txt &
```

(on verra ce que signifie & plus tard)

1. Écrire vos noms et prénoms, sauver, et quitter l'éditeur.
2. Vérifier à l'aide de `less` que ce fichier contient bien ce que vous y avez écrit.
3. Quelle taille a ce fichier?

EXERCICE #4 ▶ **Commande `diff`**
À l'aide d'un éditeur de texte, dans le répertoire `/Systeme/TP2`

1. Créez un fichier1.txt avec 3 lignes différentes.
2. Créez un fichier2.txt qui est égal au fichier1.txt avec une ligne en moins.
3. Créez un fichier3.txt qui est égal au fichier1.txt avec en plus une ligne avec une phrase.
4. Créez un fichier4.txt qui est égal au fichier1.txt sauf que la deuxième ligne contient autre chose.
5. Faites `diff` entre les différents fichiers.
6. (Bonus pour les personnes en avance). Jouer avec la commande `patch` pour reconstruire le fichier2.txt à partir du fichier1.txt et de la sortie de `diff`.

On se contentera de retenir que `diff` n'imprime rien si deux fichiers sont identiques, et sinon donne des informations sur les différences entre les deux.

3.1.4 Chercher des fichiers dans un répertoire

Un petit exo pour découvrir la puissance de la commande `find` :

EXERCICE #5 ▶ **Découverte de find**
Exécuter :

1. `man find`
2. puis :
   ```
   find /etc
   find /etc -name "*.d"
   find /etc -name "*.cfg" -ls
   ls -R /etc
   ```
3. Que veut dire l'étoile ici?
4. À quoi sert l'option `-name`?
5. Commenter la différence entre `ls -R` et `find`.
6. Essayer la commande `find /etc -exec wc '{}' +` puis la commande
   `find /etc -name "*.cfg" -exec wc '{}' +`
   Que fait l'option `-exec`?
7. Que fait l'option `-iname`?

Noter qu'on parlera des caractères d'échappement plus tard .

3.1.5 Archiver/compresser

EXERCICE #6 ▶ **Commandes `gzip`, `tar`**
Dans cet exercice, on va commencer par mettre en pratique les connaissances du TP1.

1. À l'intérieur d'un répertoire `Systeme/TP2/`, **créer** un répertoire nommé `toto` contenant deux fichiers texte, `titi` (contenant le seul mot titi) et `tata`, (contenant le seul mot tata)
2. Vérifier avec `ls` et `less`.
3. Créer une archive du répertoire `toto` et de son contenu : [1]

1. les dièses sont des commentaires, ils ne sont pas à recopier

```
tar -cvf toto.tar toto # crée l'archive
ls -l toto.tar
cat toto.tar
tar -tvf toto.tar # donne le listing (le contenu)
```

4. Créer un autre répertoire (`Systeme/TP2/testtar`, par exemple), y-mettre l'archive et la désarchiver à cet endroit (`tar -xf`). Supprimer ensuite le répertoire `Systeme/TP2/toto`.
5. Compresser l'archive tar à l'aide de `gzip` (option `-9` ou `-1`). Vérifier que l'archive compressée est moins grosse que l'archive initiale.

Remarque : `tar` peut lui-meme faire appel à gzip (compression/décompression) en ajoutant l'option `-z` : `tar -cvzf toto.tgz toto` pour compresser, par exemple.

TD 4

Ordonnancement et contrainte

4.1 Section critique et deadlock

EXERCICE #1 ▸ **Problème d'accès concurrent**
Deux agences d'une banque veulent mettre à jour le même compte bancaire. Pour cela, l'agence de Nancy effectue :

```
1. courant = get_account(1867A)
2. nouveau = courant + 1000
3. update_account(1867A, nouveau)
```

et l'agence de Karlsruhe :

```
A. aktuelles = get_account(1867A)
B. neue = aktuelles -1000
C. update_account(1867A, neue)
```

1. En supposant que l'agence de Nancy commence en premier, quel sera le montant à l'issue des transactions ?

EXERCICE #2 ▸ **dead-lock (*interblocage* en français)** [1]
Toujours dans le cadre bancaire, nous allons utiliser deux nouvelles fonctions :

— `bloque(num_compte)`

— `debloque(num_compte)`

la fonction `bloque` réserve le compte passé en argument. Si ce dernier est déjà réservé, elle bloque le programme appelant jusqu'à ce que le compte soit libéré par la fonction `debloque`.

Votre programmeur vous propose d'implémenter le transfert d'argent comme cela :

```
1  function transfert(from, to, montant):
2      bloque(from)
3      bloque(to)
4
5      cour = get_account(from)
6      cour = cour -montant
7      set_account(from, cour)
8
9      cour = get_account(to)
10     cour = cour + montant
11     set_account(to, cour)
12
13     debloque(to)
14     debloque(from)
```

Lors des essais, vous constatez que certaines transactions restent irrémédiablement bloquées.

1. Expliquez ce qu'il peut se passer.
2. Donnez une correction.

1. Merci à `https://stackoverflow.com/questions/1385843/simple-deadlock-examples`

4.2 Ordonnancement

4.2.1 À priorités

EXERCICE #3 ▶ **Préemptif avec priorités**
Nous utilisons un ordonnancement préemptif avec priorité [2]. Nous allons utiliser un jeu de tâches qui mélange des tâches périodiques et des tâches ponctuelles.

Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0, 6, 12, 18, 24, 30 ...	10	1	Périodique
B	0, 10, 20, 30 ...	8	4	Périodique
C	21	9	6	Ponctuelle
D	0	1	7	Ponctuelle

1. Faire l'ordonnancement de ces tâches sur 32 unités de temps.
2. Quel est le temps de réponse de chaque tâche ?

EXERCICE #4 ▶ **Ordonnancement collaboratif**
Le système n'interrompt jamais une tâche, il ne reprend la main que si la tâche se termine ou fait une opération bloquante. Supposons le jeu de tâches suivant :

Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0	4	5	au bout de 3 unités de calculs, la tâche laisse la main (`yield`) elle est immédiatement prête
B	1	8	4	au bout de 2 unités de calculs, la tâche fait un read bloquant qui dure au moins 3 UC
C	10	9	3	
D	1	5	4	

1. Faire l'ordonnancement sur 16 unités de temps.

EXERCICE #5 ▶ **Ordonnancement avec liste de priorité**

— Le système gère des listes de priorités.
— On se place en mode préemptif c'est à dire que lorsqu'une tâche plus prioritaire arrive, le système lui donne immédiatement accès au processeur.
— Pour des tâches de même priorité, le système utilise le `Round Robin` avec un quantum de 2.

Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0	4	5	
B	1	4	4	
C	3	8	6	

1. Faire l'ordonnancement sur 15 unités de temps.

EXERCICE #6 ▶ **Ordonnancement collaboratif**
Le système n'interrompt jamais une tâche, il ne reprend la main que si la tâche se termine ou fait une opération bloquante. Supposons le jeu de tâches suivant :

Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0	6	5	au bout de 3 unités de calculs, la tâche laisse la main (`yield`) elle est immédiatement prête
B	1	8	4	au bout de 2 unités de calculs, la tâche fait un read bloquant qui dure au moins 3 UC
C	10	9	3	
D	1	5	4	

1. Faire l'ordonnancement sur 16 unités de temps.

2. Plus la valeur de priorité est importante plus la tâche est prioritaire

TD 5
Algorithmes d'ordonnancement statique, tests d'ordonnançabilité

Contexte Dans le cadre des systèmes temps-réel, il est important de produire des ordonnancements statiques, sur une période d'exécution. Les tâches se reproduiront ensuite de manière identique dans les périodes suivantes. Les algorithmes qui suivent (RM, EDF) suivent le même schéma de choix, il s'agira de les exécuter symboliquement sur la période en question. Pour la culture, ces algorithmes viennent aussi avec des CN, CS ou CNS d'ordonnançabilité, qui permettent de conclure quelquefois dans faire les calculs.

Ces exercices sont inspirés d'une feuille de TD de F. Singhoff, univ Brest, avec l'aimable autorisation de l'auteur. Ils ne seront pas effectués dans la semaine du bloc3, mais on vous les fournit gracieusement, pour aller plus loin.

5.1 Un bout de cours

Les deux algorithmes RM et EDF sont des algorithmes qui traitent le cas de tâches périodiques avec chacune une période (T_i ou P_i), avec une échéance souvent égale à la période, et elles peuvent être réveillées au début de leur période.

L'algorithme d'ordonnancement est le suivant :

```
runnable = [false;...;false] ; cpt = [0;...;0]

(* faire avancer le temps d'un pas *)
forall i
   if cpt(i) = 0 then
      if runnable(i) then Stop(SchedulingError)
   else
       runnable(i) <- true;
       cpt(i) <- T (i)
   cpt(i) <- cpt(i) - 1

(* election *)
let l = {i/runnable(i)} in
let next = select(l) in
if next <> current then current <- next;
Preemption(next)
```

Ensuite c'est la fonction `select` qui change : pour RM on choisit la tâche de période minimale; pour EDF on choisit la tâche dont la deadline est la plus proche.

5.2 Ordonnancement à priorité fixe avec affection de priorité RM

Fonctionnement

Le calcul de priorité consiste à associer à chaque tâche une priorité fixe inversement proportionnelle à sa périodicité (Rate Monotonic).

La phase d'élection consiste à élire la tâche de plus forte priorité (ordonnancement à priorité fixe).

Tests d'ordonnançabilité

Hypothèses : tâches indépendantes et périodiques. Algorithmes préemptifs.

1. Test sur le taux d'utilisation avec $\forall i : D_i = P_i : \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{(1/n)} - 1)$ condition suffisante mais non nécessaire.
2. Utilisation de la période d'étude : ordonnancement à comportement cyclique (propriété du modèle périodique). Période d'étude = $[0, PPCM(P_i)]$ (si $\forall i : S_i = 0$).
3. Test sur le temps de réponse de la tâche i (noté r_i), avec $D_i = P_i$:

$$r_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$$

. On calcule itérativement ce test en utilisant cet algorithme :

— $w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$.

— Init : avec $w_i^0 = C_i$.

— Conditions d'arrêt : échec si $w_i^n > P_i$, réussite si $w_i^{n+1} = w_i^n$.

Exercice 1 : ordonnancement à priorité fixe + Rate Monotonic

EXERCICE #1 ▸ **S**
oient trois tâches périodiques T1, T2 et T3 définies par les paramètres suivants : $S_1 = S_2 = S_3 = 0, P_1 = 29, C_1 = 7, P_2 = 5, C_2 = 1, P_3 = 10, C_3 = 2$. On suppose un ordonnancement à priorité fixe avec une affection Rate Monotonic des priorités. Les délais critiques sont égaux aux périodes (ie $\forall\ i\ :\ D_i = P_i$).

1. Calculez le taux d'utilisation pour RM. Le jeu de tâches est il ordonnançable ?
2. Dessinez, sur les 30 premières unités de temps, l'ordonnancement généré par RM, d'abord avec la version préemptive, puis, avec la version non préemptive (vous commencerez à la date zéro). Que constatez vous ?

5.3 Earliest deadline first (EDF)

Principe Le calcul de priorité consiste à déterminer une échéance. L'échéance $D_i(t)$ d'une tâche i à l'instant t est égale à la somme de la date de début de l'activation courante à l'instant t et du délai critique D_i.

La phase d'élection consiste à élire la tâche de plus *proche* échéance.

Test d'ordonnançabilité

1. Test sur le taux d'utilisation :
   — Cas $\forall i : D_i = P_i : \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ condition nécessaire et suffisante.
   — Cas $\exists i : D_i < P_i : \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$ condition suffisante seulement, $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ condition nécessaire uniquement.
2. Utilisation de la période d'étude (propriété du modèle périodique).

EXERCICE #2 ▸ **EDF**
Soient trois tâches périodiques T1, T2 et T3 définies par les paramètres suivants : $S_1 = S_2 = S_3 = 0, P_1 = 12, C_1 = 5, P_2 = 6, C_2 = 2, P_3 = 24, C_3 = 5$. Les délais critiques sont égaux aux périodes (soient $\forall\ i\ :\ D_i = P_i$).

1. Calculez le taux d'utilisation du processeur. Concluez sur l'ordonnançabilité du jeu de tâches.
2. Déterminer le nombre d'unités de temps libre sur la période d'étude (le ppcm des périodes, ie 24) (sans dessiner d'ordonnancement). On trouvera une formule liant ce nb à la période d'étude et le taux d'utilisation.
3. Confirmez les points précédents en dessinant, sur la période d'étude, l'ordonnancement généré par EDF, d'abord avec la version préemptive, puis, avec la version non préemptive.

TD 6
Programmation d'un générateur de questionnaire

6.1 Introduction

Le but de ce TP est de programmer en Python un prototype de générateur de questionnaire. Ce dernier sera capable de lire une série de questions et de solutions dans un répertoire. Chaque question demande à l'élève de fournir une commande, le générateur doit être capable de comparer le résultat de la réponse de l'élève avec celui de la correction. Bien sûr, il faut faire en sorte que l'élève ne puisse pas avoir la possibilité de lire la solution alors que le générateur doit en être capable.

Ce TP pourra vous permettre de manipuler :

— les fichiers en python ;

— les droits sur les fichiers et l'utilisation du bit `setuid` ;

— l'exécution de processus, les signaux et l'utilisation de pipe.

6.2 Lecture de la liste des questions

Vous devez faire un programme Python capable de lire la liste des questions depuis un répertoire. Pour chaque question, deux fichiers sont présents :

— le nom de l'un est de la forme `q-NOMDELAQUESTION.txt` et contient la question,

— le nom du fichier solution correspondant à cette question est `s-NOMDELAQUESTION.txt`

Pour l'instant, on ne se préoccupe pas des solutions : ce sera l'objet des parties suivantes. Vous devez commencer votre programme de façon à ce qu'il pose les questions, et pour cela, vous allez utiliser les biliothèques python : `os`, `os.path` et `re` (Expressions régulières).

Q.0.1) - Faire une fonction `listeQuestions(rep)` qui ouvre le répertoire `rep` qui contient les questions, cherche les fichiers dont le nom est `q-NOMDELAQUESTION.txt` et retourne la liste des noms de questions.

Q.0.2) - Faire une fonction `poseQuestion(rep, question)` qui teste si le fichier `rep/q-NOMQUESTION.txt` existe, pose la question qui est dedans et récupère la réponse donnée par l'étudiant.

Q.0.3) - Faire un programme, en utilisant les deux fonctions précédentes, qui prend en paramètre le nom du répertoire de questions (bibliothèque `argparse`), lit les questions et les pose dans un ordre aléatoire.

6.3 Sécurisation des solutions

Pour le moment, la liste des questions et les solutions sont visibles à l'utilisateur du questionnaire. Vous devez donc protéger l'accès aux solutions, en modifiant les droits d'accès aux fichiers.

Sur la machine que vous utilisez, vous disposez de deux utilisateurs : `profburp` est l'enseignant et `chaprot` l'élève. L'utilisateur `profburp` fait partie des groupes `prof`, `premiereS`, `users` et `lycee`. L'utilisateur `chaprot` fait partie des groupes `premiereS`, `eleve`, `lycee` et `users`.

Q.0.4) - Quels changements de droits faut-il faire pour que seuls les élèves de `premiereS` et `profburp` soient capables d'éxécuter le programme `questionnaire?`

Q.0.5) - Comment interdire à tous, à part à `profburp`, de lire les fichiers d'un questionnaire?

Q.0.6) - Comment permettre à tous les étudiants de `premiereS` d'utiliser tout de même le questionnaire?

6.4 Comparaison des résultats

Vous devez maintenant comparer la réponse de l'élève avec la solution (comme il y a beaucoup de façons de faire la même chose, vous allez comparer le résultat de la commande entrée par l'élève à celui de la commande de la solution).

Pour cela vous allez utiliser la bibliothèque `subprocess`.

Q.0.7) - Faire une fonction `executeCommande(com)`, qui prend en paramètre une commande shell, exécute et retourne un tableau associatif avec les champs :

- `fonctionne` : un booléen qui dit si la commande a eu une erreur ou non (elle retourne 0 en cas de fonctionnement).
- `out` : une chaîne de caractères `UTF8` contenant le texte affiché sur la sortie standard par la commande.
- `err` : une chaîne de caractères `UTF8` contenant le texte affiché sur la sortie d'erreur par la commande.

Q.0.8) - Sécurisez la fonction afin de faire en sorte qu'elle ne génère jamais une exception, qu'elle s'arrête forcément même si la solution proposée ne le fait pas et qu'elle récupère au maximum les informations générées. Pour cela utilisez des *timeouts* et des signaux.

Q.0.9) - Modifiez la fonction `poseQuestion` pour qu'elle teste la proposition de l'étudiant, la solution de l'enseignant et compare les résultats.