TD N°03 : Gestion des processus (Partie I)

Exercice 3.1 (Questions de Compréhension)

Q1) Quel est le rôle de l'ordonnanceur ?

Q2) Pourquoi l'algorithme d'ordonnancement SJF n'est-il pas réellement applicable ?

Q3) Les algorithmes d'ordonnancement basés sur des priorités peuvent engendrer la famine (non-exécution) des processus à faible priorité. Comment peut-on éviter ce problème ?

Q4) Décrire (sur 2 à 3 phrases) la différence entre les algorithmes d'ordonnancement de programmes préemptifs et non préemptifs. Lequel qui convient mieux pour un système à temps partagé?

Q5) Supposons qu'on a une machine avec 4 processeurs et 3 processus qui sont en état « prêt ». Quelle est d'après vous la politique d'ordonnancement de processus qui donnerait les meilleures performances pour le système ? Argumentez votre réponse.

Q6) Décrivez ce qui se passe, du côté du système d'exploitation, lorsqu'une touche de clavier est pressée ? (maximum 3 phrases)

Q7) La stratégie d'ordonnancement de processus la plus appropriée pour un système d'exploitation es temps partagé est : (Choisir la bonne réponse)

(a) Le Court-travail Premier (SJF). **(b)** Priorité. **(c)** Rond-Robin.
(d) Premier arrivée Premier Servi (FCFS). **(e)** tout ce qui précède.

Q8) Un processus était observé de commuter depuis l'état actif vers l'état prêt. L'ordonnancement (ou le scheduling) doit être : **(Choisir la bonne réponse)**

(a) Le plus court job le premier (SJF) **(b)** Non-préemptive. **(c)** Préemptive
(d) Round Robin **(e)** Aucune de ce qui précède

Q9) La différence entre l'ordonnancement de programmes avec préemption et sans préemption est : **(Choisir la bonne réponse)**

(a) Si ou non un processus prêt peut être involontairement terminé depuis l'état prêt.
(b) Si ou non un processus élu est involontairement enlevé de l'état actif.
(c) Si ou non un processus bloqué peut avoir ses ressources involontairement lui retirées.

Q10) Quel est l'intérêt du scheduling multi-niveaux ?

Exercice 3.2 (Gestion des Processus)

On considère un système monoprocesseur et les quatre processus P1, P2, P3 et P4 qui effectuent du calcul et des entrées/sorties avec un disque selon les temps donnés ci-contre. Les processus sont disponibles dès le début, dans cet ordre.

Q1) On considère que l'ordonnancement sur le processeur se fait selon une politique à priorité préemptive : le processus élu à un instant **t** est celui qui est le processus prêt de plus forte priorité.

On donne : priorité (P1) > priorité (P3) > priorité (P2) > priorité (P4). On considère que l'ordre de service des requêtes d'E/S pour le disque se fait toujours selon une politique FIFO.

Complétez **l'Annexe A**, et donnez le temps de rotation moyen obtenu.

	P1	P2	P3	P4
Temps d'exécution sur le CPU	3	4	2	7
E/S	7	3	3	
Temps d'exécution sur le CPU	2	2	2	
E/S	1	1		
Temps d'exécution sur le CPU	1	1		

Q2) La politique d'ordonnancement du processeur est inchangée, mais on considère maintenant que l'ordre de services des requêtes d'E/S pour le disque se fait également selon la priorité des processus : le processus commençant une E/S est celui de plus forte priorité parmi ceux en état d'attente du disque. Une opération d'E/S commencée ne peut pas être préemptée.

Complétez **l'Annexe B**, et donnez le temps de rotation moyen obtenu.

Q3) On considère que l'ordonnancement sur le processeur se fait selon une politique tourniquet avec un quantum de 2 unités de temps. On suppose que l'ordre d'arrivée a été P1 puis P2 puis P3 puis P4. On considère que l'ordre de services des requêtes d'E/S pour le disque se fait en FIFO.

Complétez **l'Annexe C**, et donnez le temps de rotation moyen obtenu.

Q4) Comparez les différents temps de rotation calculés précédemment, et interprétez le résultat.

<u>Exercice 3.3</u> (Gestion des Processus avec temps de commutation)

On considère l'exécution des cinq processus suivants.

processus	Date d'arrivée	Durée (ms)
P1	0	7
P2	1	4
P3	1	2
P4	2	2
P5	3	1

Q1) Donner les diagrammes de Gant et les temps de réponse moyen, en utilisant les algorithmes d'ordonnancement suivant :

a) FCFS (first come first served),

b) SJF (short job first),

c) SRTF (short remained time first) ,

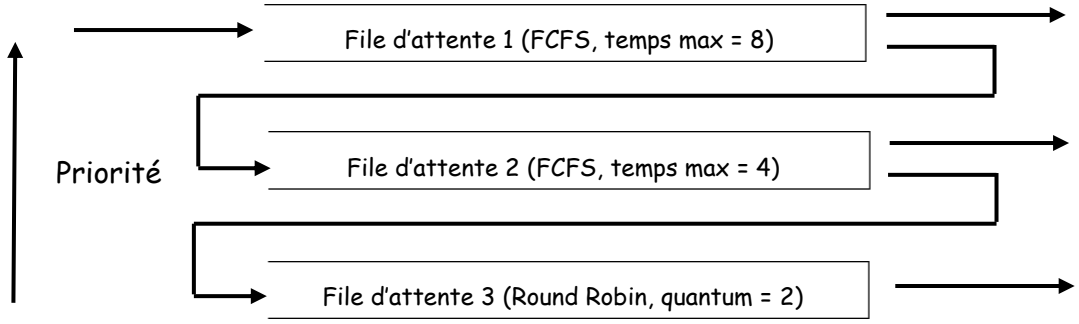
d) Round Robin (avec un quantum de **1 ms**).

Q2) Si le temps de commutation est de **0.5 ms**, quel est alors le temps de réponse moyen dans le cas d'un ordonnancement SRTF et d'un ordonnancement Round Robin. Qu'en déduisez-vous ?

Exercice 3.4 (scheduling multiniveau)

On considère la technique de scheduling multiniveau et feedback suivante :

- La **File 1** est la plus prioritaire. Un processus nouvellement créé est placé dans la **File 1** qui est gérée selon l'algorithme FCFS.
- Lorsqu'un processus de la **File 1** obtient le processeur, on lui accorde un temps max de **8** unités de temps, s'il ne termine pas, il est déplacé dans la **File 2**.
- La **File 2** est aussi gérée selon le scheduling FCFS, mais on donne un temps max de **4** unités de temps à chaque processus. Si le processus ne termine pas, il est déplacé dans la **File 3**.
- La **File 3** est gérée selon le scheduling Round Robin avec un quantum égal à **2**.



Q1) Expliquez l'intérêt de cette méthode de scheduling

Q2) Donnez le diagramme de Gantt pour le scénario suivant :

Processus	Durée d'exécution	Instant d'arrivée
P1	16	0
P2	14	0
P3	10	0
P4	20	0
P5	06	0

Q3) Donnez le temps de réponse, le temps de restitution et le temps d'attente de chaque processus.

Processus	Temps de Réponse	Temps de Restitution	Temps d'attente
P1	…	…	…
P2	…	…	…
P3	…	…	…
P4	…	…	…
P5	…	…	…

Exercice 3.5 (Gestion des processus avec deux processeurs)

On considère un système possédant deux processeurs et une seule file d'attente pour les processus prêts.

Q1) Avec ce système, quel problème peut-on avoir avec l'algorithme de scheduling « Plus haute priorité » ? Quelles solutions proposez-vous ?

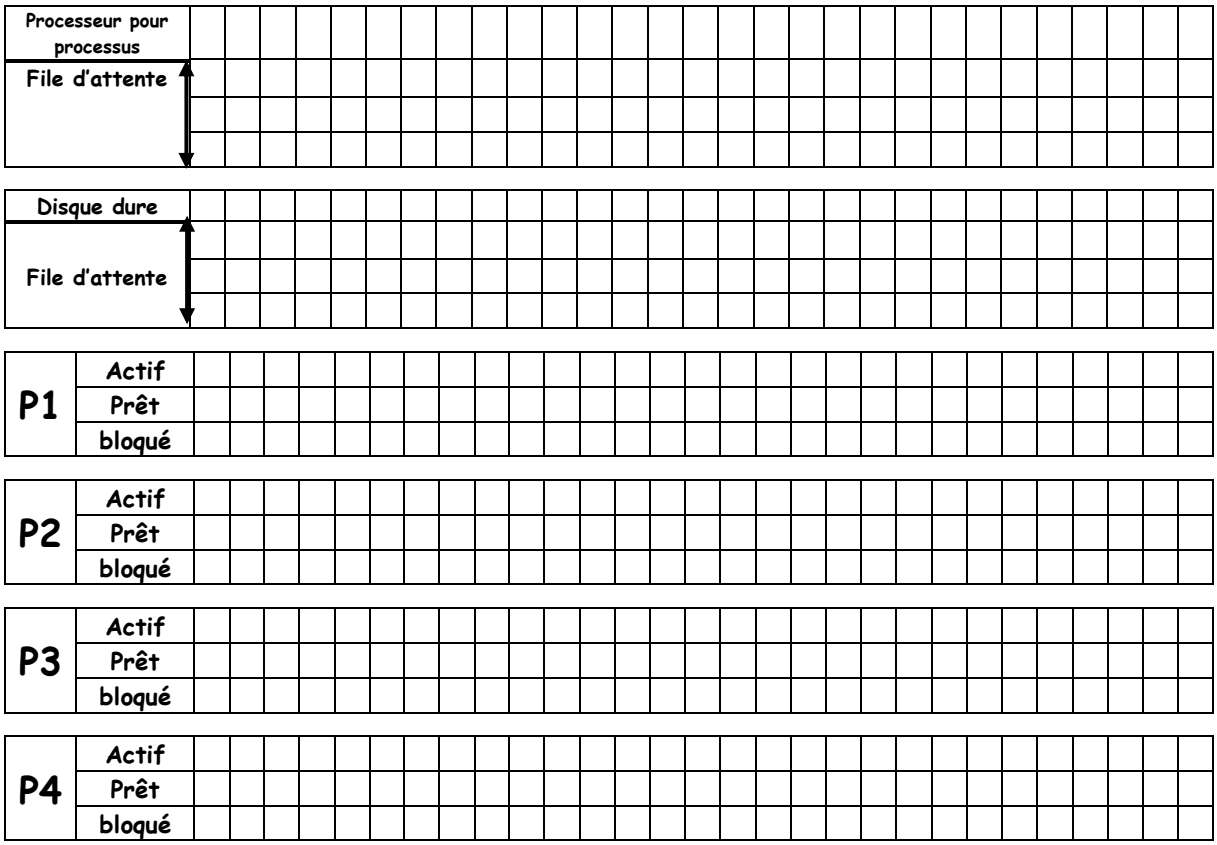
Q2) Soit le scénario d'arrivée des processus suivants : P1, P2, P3 et P4, ayant les caractéristiques suivantes (la priorité 1 correspond à la plus faible priorité). Pour chacun des algorithmes de scheduling suivants :

a) FCFS ,
b) Plus haute priorité (sans réquisition)
c) Plus haute priorité, (avec réquisition)
d) Round Robin (avec quantum=2)

Donnez les diagrammes de Gantt et les temps d'attente et de restitution des processus.

Processus	Priorité	Instant d'arrivée	Durée d'exécution
P1	2	0	4
P2	4	2	5
P3	3	0	6
P4	1	0	7

Annexe A



Annexe B

Processeur pour processus	
File d'attente	

Disque dure	
File d'attente	

P1	Actif
	Prêt
	bloqué

P2	Actif
	Prêt
	bloqué

P3	Actif
	Prêt
	bloqué

P4	Actif
	Prêt
	bloqué

Annexe C

Processeur pour processus	
File d'attente	

Disque dure	
File d'attente	

P1	Actif
	Prêt
	bloqué

P2	Actif
	Prêt
	bloqué

P3	Actif
	Prêt
	bloqué

P4	Actif
	Prêt
	bloqué

TD N°03 : Gestion des processus (Partie II)

L'objectif de la deuxième partie du TD est d'approfondir les notions relatives aux processus et de les appliquer au cadre spécifique d'Unix.

Exercice 3.6 (Création de processus avec l'appel système *fork*)

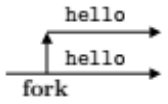
Q1) Qu'affiche l'exécution du programme suivant :

```
1 : int main() {
2 : pid_t pid;
3 : int x = 1;
4 :
5 : pid = fork();
6 : if (pid == 0) {
7 : printf("Dans fils : x=%d\n", ++x);
8 : exit(0);
9 : }
10 :
11 : printf("Dans père : x=%d\n", --x);
12 : exit(0);
13 : }
```

Q2) On considère les deux programmes suivants et leur schéma d'exécution. •

- Exemple : un clonage

```
1 : int main() {
2 : fork();
3 : printf("hello!\n");
4 : exit(0);
5   : }
```



a) Illustrer l'exécution des programmes suivants :

```
1 : int main() {
2 : fork();
3 : fork();
4 : printf("hello!\n");
5 : exit(0);
6 : }
```

```
1 : int main() {
2 : fork();
3 : fork();
4 : fork();
5 : printf("hello!\n");
6 : exit(0);
7 : }
```

b) Combien de lignes « hello ! » imprime chacun des programmes suivants ?

Programme 1	***Programme 2***	***Programme 3***	***Programme 4***

1 : int main() {
2 : int i;
3 :
4 : for (i=0; i<2; i++)
5 : fork();
6 : printf("hello!\n");
7 : exit(0);
8 : }
	1 : void doit() {
	2 : fork();
	3 : fork();
	4 : printf("hello!\n");
	5 : }
	6 : int main() {
	7 : doit();
	8 : printf("hello!\n");
	9 : exit(0);
	10 : }
		1 : int main() {
		2 : if (fork())
		3 : fork();
		4 : printf("hello!\n");
		5 : exit(0);
		6 : }
			1 : int main() {
			2 : if (fork()==0)
			3 : if (fork()) {
			4 : printf("hello!\n");
			5 : }
			5 : exit(0);
			6 : }

Exercice 3.7 (Arbre généalogique)

Dessiner l'arbre généalogique des processus engendrées par le programme suivant :

```
1 : #include <unistd.h>
2 :
3 : int main()
4 : {
5 :     fork() && (fork() || fork());
6 :     return 0;
7 : }
```

Exercice 3.8

Ecrire un programme qui engendre l'arbre généalogique de la **figure 3.1.**

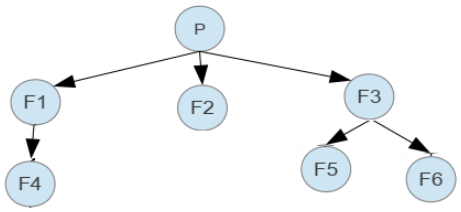


Figure 3.1 : l'arbre généalogique

Exercice 3.9

Préciser le nombre de processus créer et dessiner l'arbre généalogique des processus engendrées par les trois codes (programmes) suivants :

Code 1

```
int main() {
    fork();
    fork();
    fork();
}
```

Code 2

```
int main() {
    if (fork() > 0)
        fork();
}
```

Code 3

```
int main() {
    int cpt=0;
    while (cpt < 3) {
        if (fork() > 0)
            cpt++;
        else
            cpt=3;
}
```

Code 4

```
void main ()
{
pid_t p1, p2, p3, p4;
if ((p1 = fork ()) == 0)
    if ((p2 = fork ()) == 0)
      f2 ();
    else
      f1 ();
else
    if ((p3 = fork ()) == 0)
      f3 ();
    else
      if ((p4 = fork ()) == 0)
        f4 ();
sleep (3);
while (wait (NULL) > 0);
}
```

Exercice 3.10

Considérez le programme C suivant (**code 4**) :

Q1) Tracez l'arborescence des processus créés par ce programme si les fonctions f1, f2, f3 et f4 se terminent par **exit()**.