# ED 3
# Gestion du processeur

## Exercice 1 : Politiques d'ordonnancement des processus

## Exemple de Linux

Il existe trois politiques d'ordonnancement dans le système Linux : la première est utilisée pour ordonnancer des processus ordinaires et les deux autres pour ordonnancer des processus temps réel. Afin de connaître la politique d'ordonnancement à utiliser pour un processus, chaque processus possède un type qui peut être égal à :

`SCHED_FIFO` pour un processus temps réel non préemptible,

`SCHED_RR` pour un processus temps réel préemptible,

`SCHED_OTHER` pour un processus ordinaire (non temps réel).

Trois files différentes accueilleront les processus prêts appartenant aux trois types. Les processus de la file `SCHED_FIFO` sont plus prioritaires que ceux de la file `SCHED_RR` qui eux-mêmes sont plus prioritaires que ceux de la file `SCHED_OTHER`.

Quel que soit son type, un processus Linux possède une priorité et est inséré dans la file associée à son type dans l'ordre décroissant de sa priorité. A tout moment, le processus de type *x* le plus prioritaire se trouve en tête de la file du même type.

L'ordonnanceur gère la file `SCHED_FIFO` en utilisant la stratégie non préemptive. Il choisit d'élire le processus de type `SCHED_FIFO` le plus prioritaire pour s'exécuter. N'étant pas préemptible, ce processus s'exécute jusqu'à la fin sans libération du processeur, excepté dans les cas suivants :

- un autre processus de type `SCHED_FIFO` plus prioritaire vient d'être inséré dans la file ; il est alors exécuté à la place du processus courant qui est inséré en fin de file.
- le processus demande à faire une entrée-sortie;
- le processus abandonne le processeur en appelant la primitive *`sched_yield()`*.

La file `SCHED_RR` est gérée par la technique du tourniquet avec une seule file d'attente. En effet, tout processus de type `SCHED_RR` est exécuté pour la durée d'un quantum de temps (égale à 60 ms). A l'expiration du quantum de temps, le processus le plus prioritaire parmi les processus de type `SCHED_FIFO` est choisi. Si la file d'attente associée est vide, le processus `SCHED_RR` le plus prioritaire est élu.

La priorité pour les processus des files `SCHED_FIFO` et `SCHED_RR` varie entre 1 et 99. Le plus prioritaire étant celui qui a la plus grande valeur.

La file `SCHED_OTHER` accueille les processus ordinaires c'est-à-dire non temps réel. Les processus temps réel étant plus prioritaires, les processus de type `SCHED_OTHER` ne pourront s'exécuter que si les files de type `SCHED_FIFO` et `SCHED_RR` sont vides.

La file `SCHED_OTHER` est elle aussi gérée avec la technique du tourniquet à une seule file d'attente. L'insertion des processus se fait en fonction de leurs priorités mais lorsque la priorité est identique (en général égale à 0), l'insertion se fait en FIFO.

La priorité d'un processus (quel que soit son type) se calcule à partir :

- d'une partie modifiable par l'utilisateur (à l'aide des primitives `nice` ou `setpriority`). Un processus utilisateur ne pourra que diminuer sa priorité; il ne pourra l'augmenter que s'il est un processus privilégié.

- d'une partie fixée par le système et qui baisse lorsque le processus consomme un certain nombre de cycles d'horloge ; ceci pour permettre à ceux de moindre priorité de s'exécuter rapidement.

Soit un programme C lancé sous Linux par le super-utilisateur, et qui crée presque en même temps 4 processus p1, p2, p3 et p4.

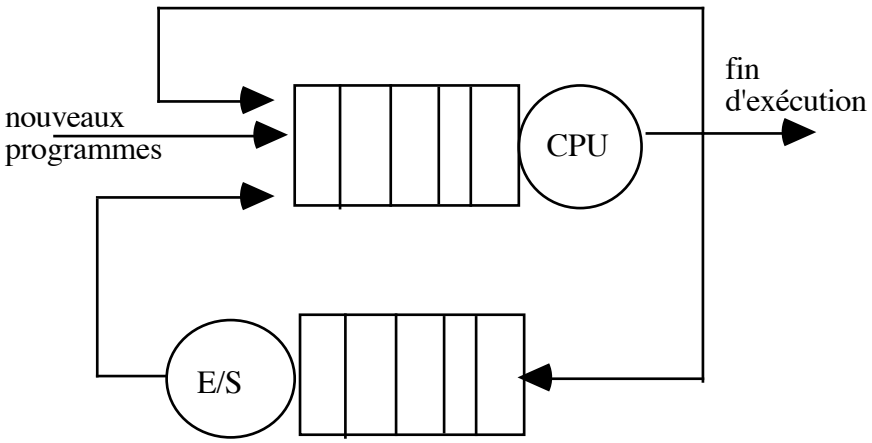
## Question

*En supposant que ces processus ne font pas d'entrée-sortie, que leurs priorités ne changent pas durant l'exécution et que leurs durées d'exécution sont celles décrites ci-dessous, déterminer les temps de réponses de chaque processus en appliquant les politiques d'ordonnancement adéquates.*

| Nom du processus | durée d'exécution | actions particulières réalisées |
|---|---|---|
| P0 | 60ms | à la fin de son calcul, il crée les 4 processus : p1, p2, p3 et p4<br>- met p1 dans la file `SCHED_RR` avec la priorité 2<br>- met p2 dans la file `SCHED_FIFO` avec la priorité 3<br>- met p3 dans la file `SCHED_RR` avec la priorité 4<br>- met p4 dans la file `SCHED_FIFO` avec la priorité 4 |
| | 30ms | calcul + attente de la fin des processus |
| P1 | 100ms | calcul |
| P2 | 90ms | calcul |
| | 30ms | exécute `usleep`(30) |
| | 10ms | calcul |
| P3 | 110ms | calcul + exécute `sched_setscheduler()` à la fin du calcul pour se mettre dans la file SCHED_OTHER avec la priorité 1 |
| | 20ms | calcul |
| P4 | 70ms | calcul + exécute `sched_yield()` en fin de calcul pour libérer le processeur |
| | 50ms | |

# Exercice 2 :
# Priorité des processus fonction du temps d'accès à l'UC

On considère un système dans lequel les seules ressources partagées sont un disque géré par un canal d'entrée sortie et un processeur.




Les requêtes disques sont gérées à l'ancienneté (FIFO). Par contre, l'allocation du processeur est réalisée selon la politique suivante : on définit un quantum de temps q. Un processus allocateur est activé à chaque début et fin de traitement d'une entrée-sortie disque ou après un quantum si une entrée-sortie ne s'est pas produite dans le dernier quantum.

L'allocateur alloue le processeur au processus Pi en attente qui a le plus fort rapport Ti/Tcpui. Ti représente la durée totale écoulée depuis le début de l'exécution du processus Pi (c'est-à-dire le temps écoulé depuis l'instant de première requête du processeur par le processus Pi).

Tcpui est le cumul des durées pendant lesquelles le processeur a été alloué au processus i.

Lorsque le rapport Ti/Tcpui est égal à 0/0, celui-ci est interprêté comme $+\infty$.

Lorsque plusieurs processus ont même priorité supérieure à celle de tous les autres processus, c'est le processus d'indice le plus fort qui obtient le processeur.

L'allocateur utilise un délai de garde (temporisation) de durée q. Il est activé :

- soit lorsque le délai est écoulé

- soit lorsque le processus actif fait une requête d'entrée-sortie

- soit lorsque l'exécution d'une telle requête se termine.

Dans tous les cas, il commence par réarmer le délai de garde puis procède à l'allocation quand celle-ci est possible. A l'instant initial trois processus sont présents dans le système et commencent à s'exécuter en faisant leur première requête à l'allocateur.

Le tableau suivant donne (en nombre de quantums) la durée totale de CPU nécessaire à leur exécution, le nombre d'entrée-sortie disque réalisé par chaque processus et la date, mesurée en temps CPU écoulé depuis le début d'exécution, des requêtes d'entrée-sortie.

| N° processus | Temps total CPU (quantum) | Nombre d'accès disque | Dates des accès (tps CPU relatif) |
|---|---|---|---|
| 1 | 5,5 | 1 | 1,5 |
| 2 | 5 | 0 | — |
| 3 | 1,5 | 2 | 0,5 puis 1 |

La durée d'un service disque est deux quantums et les demandes sont servies à l'ancienneté (FIFO).

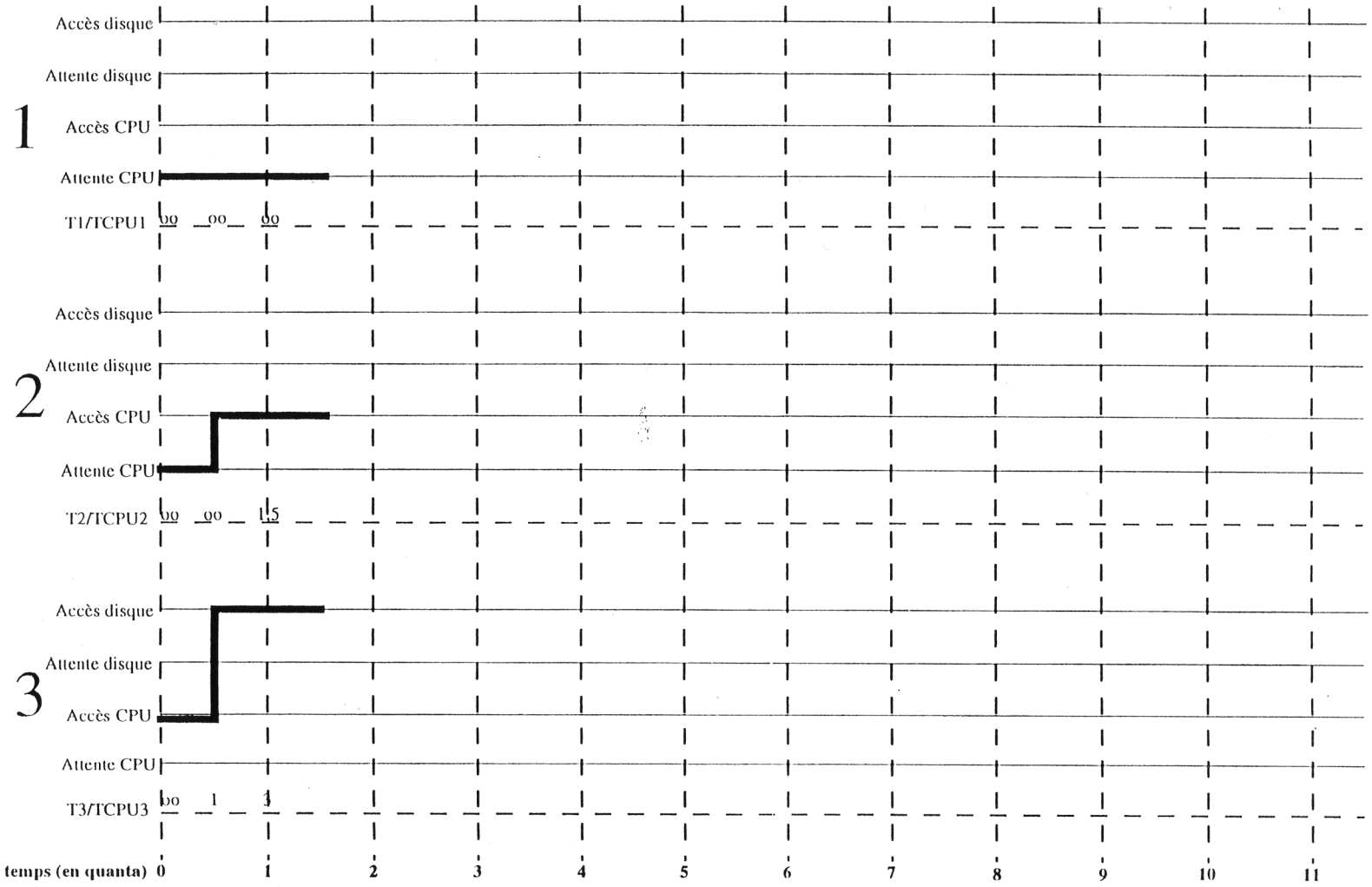
## Question 1

*Compléter le chronogramme en annexe qui donne l'état des différents processus à chaque instant ainsi que la valeur des rapports Ti/Tcpui lors de chaque changement d'état.*

Note : la durée de fonctionnement de l'allocateur est négligée.

## Question 2

*Commenter le chronogramme en donnant les avantages et inconvénients de cette gestion du processeur.*


1
Accès disque
Attente disque
Accès CPU
Attente CPU
T1/TCPU1
2
Accès disque
Attente disque
Accès CPU
Attente CPU
T2/TCPU2
3
Accès disque
Attente disque
Accès CPU
Attente CPU
T3/TCPU3
temps (en quanta)
0
1
2
3
4
5
6
7
8
9
10
11