TD N°03 : Gestion des processus (Supplémentaire)

Exercice 3.1 (Ordonnancement)

Six étudiants du département Informatique viennent voir l'enseignant X (nous tenons à garder son anonymat). L'enseignant X dispose d'une heure pour discuter avec les étudiants et passer d'un étudiant à un autre lui prend une minute. La table suivante donne l'ordre d'arrivée des étudiants et le temps dont ils ont besoin. Les étudiants arrivent tous au même moment.

Etudiant	Ordre d'arrivée	Temps requis
E1	1	45 min
E2	2	15 min
E3	3	10 min
E4	4	5 min
E5	5	6 min
E6	6	20 min

Q1) Sachant que l'enseignant X ne peut consacrer en tout qu'**une heure** à l'ensemble des étudiants, quel est l'algorithme d'ordonnancement qui lui permettra de traiter **le plus d'étudiants complètement**. Justifier à l'aide des diagrammes.

- **a)** premier arrivé, premier servi (FCFS)
- **b)** le plus court d'abord (SJF)
- **c)** Tourniquet ou Round Robin avec un quantum = **5 min**. (l'intervalle de **1min** reste valable même si l'enseignant X reste avec le même étudiant pendant **2** quantums de suite)

Q2) Un ordonnancement de type tourniquet peut utiliser différents quantums. Donner une raison d'utiliser un petit quantum ainsi qu'une raison d'utiliser un grand quantum.

Exercice 3.2

Supposez que le système d'exploitation est composé de deux unités de contrôle (deux processeurs CPU1 et CPU2) et d'une unité d'E/S. Tous les processus prêts sont dans une même file d'attente. La commutation de contexte est supposée de durée nulle.

Considérez trois processus A, B et C décrits dans le tableau suivant :

Processus	Instant d'arrivée	Temps d'exécution
A	0	4 unités cpu, 2 unités d'E/S, 2 unités cpu
B	2	3 unités cpu, 4 unités d'E/S, 2 unités cpu
C	3.5	5 unités cpu

La première ligne signifie que le processus A arrivé dans le système à l'instant 0, son exécution nécessite dans l'ordre 4 unités de temps CPU, 2 unités de temps d'E/S et 2 unités de temps CPU.

Au départ le processus A est élu par le processeur CPU1.

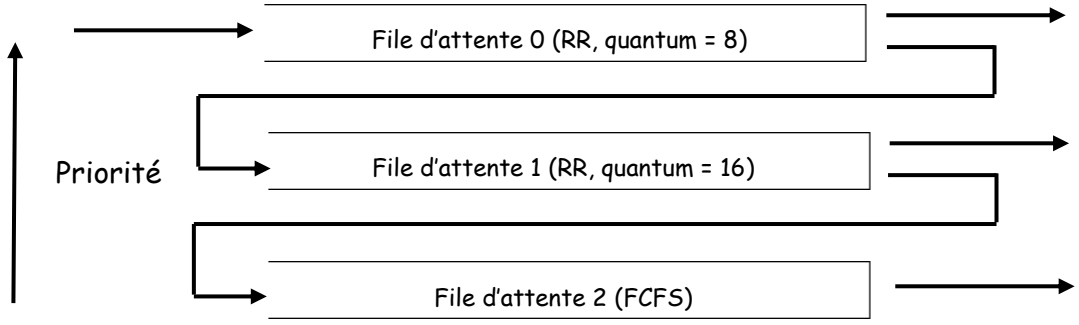
- Si plusieurs événements surviennent en même temps, vous supposerez les priorités suivantes :
- Le CPU1 a la priorité d'accès à la file des processus prêts par rapport au CPU2.
- A la fin d'un quantum, le processus en cours et non terminé est suspendu uniquement si la file des processus prêts n'est pas vide. Le traitement réalisé à la fin d'un quantum est plus prioritaire que celui d'une fin d'E/S qui, à son tour, est plus prioritaire que l'arrivée de nouveaux processus dans le système.

Q1) Donnez les diagrammes d'exécution des jobs A, B et C montrant l'allocation des deux processeurs, de l'unité d'E/S dans le cas des systèmes : multiprogrammé et temps partagé avec un quantum de temps égal à 3 unités CPU.

Q2) Calculez le temps moyen de réponse pour chaque système.

<u>Exercice 3.3</u> (Scheduling avec files d'attente multi-niveaux et feedback)

Dans cette stratégie de scheduling un processus peut basculer entre files. On considère Un système est doté de 3 files d'attentes multi-niveaux : **File 0**, **File 1** et **File 2**.



- La **File 0** est la plus prioritaire.
- Les **Files 0 et File 1** sont gérées selon la politique Round Robin.
- La **File 2** est gérée selon la technique FCFS.

Un processus entrant dans le système

- Sera rangé dans la **File 0** et on lui donne une tranche de temps de **8 ms** au processus.
- S'il ne finit pas, il est déplacé vers la **File 1**.
- Si la **File 0** est vide, on donne une tranche de temps de **16 ms** au processus en tête de la **File 1**.
- S'il ne termine pas, il est interrompu et il est mis dans la **File 2**.
- Les processus de la **File 2** sont exécutées seulement quand les **File 0** et **File 1** sont vides.

Q1) Donnez le diagramme de Gantt pour les processus suivants:

Processus	Temps d'arrivé	Durée exécution
P0	0	26
P1	3	6
P2	20	30

Q2) Donnez le temps de réponse, le temps de restitution et le temps d'attente de chaque processus.

Exercice 3.4

On considère l'ensemble des processus suivants :

N° Processus	Temps d'arrivé	Temps CPU	Priorité
1	7h 00	10 min	2
2	7h 00	15 min	3
3	7h 03	8 min	4
4	7h 10	18 min	5

Q1) On suppose qu'on utilise un algorithme d'ordonnancement basé sur la priorité (les priorités sont croissantes : 5 est le plus prioritaire).

Donnez le diagramme de Gantt pour les priorités données dans le tableau.

Q2) On voudrait que la priorité des processus soit dynamique au cours du temps. Ainsi, pour calculer la priorité d'un processus, on utilise la formule suivante :

$$\boldsymbol{Priorité} = \frac{\boldsymbol{Temps\ d'attente + Temps\ CPU\ Restant}}{\boldsymbol{Temps\ CPU}}$$

Q3) Donnez le diagramme de Gantt sachant que la priorité est recalculée toutes les 5 minutes.

Q4) Calculez le temps d'attente moyen ainsi que le temps de rotation moyen.

Q5) Comparez les résultats obtenus par rapport à ceux obtenus avec l'algorithme de priorité classique.

***Remarque** : Lors des calculs, on arrondira suivant l'exemple suivant :*

3.5 ou 3.6 ➔ 4

3.1 ou 3.4 ➔ 3.

Exercice 3.5

Un système fait appel à l'algorithme d'ordonnancement avec priorité préemptif (les processus au numéro de priorité élevé ont une priorité plus importante). Les processus sont introduits dans le système avec une priorité de 0. Lors de l'attente dans la file des processus prêt, la priorité d'un processus change au rythme α. Lors de l'exécution du processus, sa priorité est modifiée au rythme β.

Q1) Quel algorithme résulte de **β > α > 0**

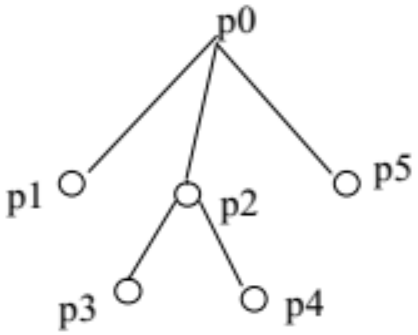
Q2) Quel algorithme résulte de **β < α < 0**

Exercice 3.6

Écrivez un programme qui lance, l'un à la suite de l'autre, les fichiers exécutables passés en paramètre. Un fichier exécutable est lancé uniquement si tous ceux qui le précèdent sont terminés. Avant de se terminer, le programme affiche, à l'écran, le nombre de lancements de fichiers exécutables qui ont échoués (parce que le fichier n'existe pas ou n'est pas exécutable).

Exercice 3.7

Considérons l'arbre des processus suivant :
P0 crée trois fils P1, P2 et P5. P1 est créé en premier suivi de P2. Le processus P5 est créé après la terminaison de P1 et P2. Le processus P2 crée P3 puis P4.
Écrire le programme qui permet de créer ces processus en respectant les liens père-fils et l'ordre de création des processus du schéma. Chaque processus doit se mettre en attente de ses fils avant de se terminer. Chaque processus **Pi** exécute une fonction **Fi()**.



TP N°03 : Gestion des processus (Supplémentaire)

Exercice 3.1

Ecrire un programme qui crée 2 processus l'un faisant la commande **ls -l**, l'autre **ps -l**. Le père devra attendre la fin de ses deux fils et afficher quel a été le premier processus à terminer.

Exercice 3.2

Écrivez un programme C qui affiche "bonjour", crée deux processus P1 et P2 et dit "au revoir". P1 doit attendre deux secondes puis afficher les 10 premiers entiers avant de se terminer. P2 doit attendre la fin de P1 puis lister le contenu du répertoire courant.

Exercice 3.3 (Fonction wait)

Ecrire un programme qui crée 2 processus l'un faisant la commande **ls-l**, l'autre **ps-l**. Le père devra attendre la fin de ses deux fils et afficher quel a été le premier processus à terminer.

Exercice 3.4 (Synchronisation des processus père et fils par la commande wait)

Ecrire un programme qui prenant une matrice de taille 2 en paramètre et crée quatre fils. Chaque fils calcule un élément du carré de la matrice initiale et le renvoi au processus père comme code de retour.

Exercice 3.5 (Signaux)

Écrivez un programme en C qui crée deux processus fils (P1 et P2). Le processus P1 envoie un signal SIGUSR1 au processus P2. Le processus P2 doit capturer le signal SIGUSR1 et afficher "Signal SIGUSR1 reçu". Ensuite, le processus P2 envoie un signal SIGUSR2 au processus P1. Le processus P1 doit capturer le signal SIGUSR2 et afficher "Signal SIGUSR2 reçu". Assurez-vous que les processus se terminent correctement après avoir échangé les signaux.

Exercice 3.6

Écrivez un programme en C qui crée trois processus fils (P1, P2 et P3). Le processus P1 envoie un signal SIGUSR1 au processus P2 et un signal SIGUSR2 au processus P3. Le processus P2 doit capturer le signal SIGUSR1 et afficher "Signal SIGUSR1 reçu". Le processus P3 doit capturer le signal SIGUSR2 et afficher "Signal SIGUSR2 reçu". Ensuite, le processus P2 doit envoyer un signal SIGUSR1 au processus P3. Le processus P3 doit capturer le signal SIGUSR1 et afficher "Signal SIGUSR1 reçu". Assurez-vous que les processus se terminent correctement après avoir échangé les signaux.

Exercice 3.7

Écrivez un programme en C qui crée un processus fils (P1). Le processus P1 doit créer deux processus fils supplémentaires (P2 et P3). Le processus P1 envoie un signal SIGUSR1 au processus P2 et un signal SIGUSR2 au processus P3. Le processus P2 doit capturer le signal SIGUSR1, afficher "Signal SIGUSR1 reçu", puis envoyer un signal SIGUSR2 au processus P3. Le processus P3 doit capturer le signal SIGUSR2, afficher "Signal SIGUSR2 reçu", puis envoyer un signal SIGUSR1 au processus P1. Le processus P1 doit capturer le signal SIGUSR1, afficher "Signal SIGUSR1 reçu", puis envoyer un signal SIGUSR2 au processus P2. Assurez-vous que les processus se terminent correctement après avoir échangé les signaux.

Exercice 3.8 (tubes ordinaires)

Ecrire un programme qui crée un processus fils. Le processus fils envoie un message "Je suis le fils !" au processus père qui le reçoit et l'affiche à l'écran, le tout en utilisant un tube ordinaire

Exercice 3.9 (Tubes nommés)

Ecrire deux programmes qui communiquent par tube nommé:
L'ordre de lancement des deux programmes est indifférent.

Expliquer :

1) Le premier effectue les actions suivantes:
- crée un tube nommé **myTube** en lecture et en écriture pour l'user et le groupe
- demande un descripteur en écriture
- écrit un message - libère le descripteur
- demande un descripteur en lecture
- lit un message et l'affiche - libère le descripteur
- supprime la référence **myTube**

2) Le deuxième effectue les actions suivantes:
- demande un descripteur en lecture sur le tube nommé **myTube**
- lit un message et l'affiche
- libère le descripteur
- demande un descripteur en écriture sur le tube nommé **myTube**
- écrit un message
- libère le descripteur.