TP 2 – Ordonnancement

L'objectif de ce TP est de se familiariser avec différentes facettes de l'ordonnancement des processus. Dans un premier temps, il s'agit d'implémenter les opérations nécessaires à la gestion d'une file d'attente de processus, puis dans un deuxième temps d'implémenter des stratégies d'ordonnancement.

1 Opérations sur une file de processus

La file d'attente est définie par une structure de données simple : une file simplement chaînée avec un pointeur sur la queue de la file, où chaque cellule contient un pointeur sur le bloc de contrôle d'un processus (champ `pcb` pour *process control block*) et un pointeur sur la cellule suivante. Une file d'attente est un pointeur sur la *dernière cellule* (qui elle-même pointe sur la première).

1. Dessinez la structure de données résultant de l'apparition de 3 processus qui se sont présentés dans l'ordre pid=4, pid=9 et pid=16.
2. Écrivez la fonction `pcb_create` (fichier `pcb.c`) qui crée un nouveau PCB avec un pid, une date de démarrage, une durée et une valeur de priorité (pour l'exercice suivant). La date est un entier qui sera incrémenté par le simulateur.
3. Écrivez la fonction `queue_insert` (fichier `schedrr.c`) qui insère un PCB existant en fin de la file d'attente.
4. Écrivez la fonction `queue_remove_any` (fichier `schedrr.c`) qui supprime une entrée de la file d'attente quelle que soit sa position dans la file ;
5. Écrivez la fonction `display_queue` (fichier `schedrr.c`) qui affiche la file d'attente.
6. Écrivez, dans un fichier de nom `testqueue.c`, un programme afin de tester votre implémentation : création de trois processus, affichage de la file, création d'un nouveau processus, affichage de la file, suppression de processus et affichage de la file.

Vous trouverez dans l'archive accompagnant ce TP les fichiers :
– `simul.h` : déclarations de la structure d'un descripteur de processus (PCB) et des opérations sur les PCB et sur la file d'attente ;
– `pcb.c` : implémentation des fonctions sur le PCB ;
– `schedrr.c` : implémentation des fonctions associées à la file d'attente.

2 Ordonnancement simple : le tourniquet

On souhaite maintenant réaliser un ordonnanceur simple basé sur l'algorithme du tourniquet (ou *round-robin*).

1. Le fichier `simul.c` fourni dans l'archive accompagnant ce TP est le simulateur. Il permet de :
   – simuler l'apparition de processus avec une priorité paramétrable ;
   – dérouler le temps et provoquer des ordonnancements avec une période donnée par un quantum ;
   – afficher des statistiques en fin d'exécution ;
   Analysez le code du simulateur et indiquez :
   (a) quelles sont les options et leur signification ; indiquez en particulier le rôle des options `-i` et `-t` ?
   (b) quelles sont les conditions pendant lesquelles un ordonnancement (appel à la fonction `schedule`) est effectué ?
   (c) quelle est la signification de « elapsed time » ?
   (d) en supposant que la fonction `schedule` (cf question suivante) est déjà écrite, quel serait le résultat de l'exécution avec les options : `-p 1 -c 4 -q 10` ?
2. Écrivez la fonction `schedule` (fichier `schedrr.c` qui réalise l'ordonnancement de la file d'attente suivant le principe du tourniquet.
3. Simulez votre programme en faisant varier les paramètres.

3 Ordonnancement avec priorités fixes

Nous nous intéressons maintenant à un ordonnanceur avec une gestion des priorités fixes. Le nouvel ordonnanceur doit avoir les propriétés suivantes :

- 32 niveaux de priorités : la plus forte priorité étant la priorité 0 ;
- les processus ayant les priorités les plus fortes sont exécutés en premier ;
- algorithme *HPF* (*Highest Priority First*) : pour que l'ordonnanceur choisisse les processus d'une priorité i, il faut que les listes de processus de priorité $j, j \in [0, i[$ soient vides ;
- pour chaque file, l'ordonnanceur choisit le prochain processus à exécuter en utilisant une politique FCFS (First Come, First Served).

On mémorisera la priorité d'un processus dans le champ `prio` de son PCB.

1. Définissez la structure de données et programmez le nouvel ordonnanceur dans un nouveau fichier de nom `schedprio.c`.
2. Quel est l'impact sur les performances ?

4 Question bonus : ordonnancement avec priorités dynamiques

L'ordonnanceur ici proposé, une version épurée de celui utilisé par des systèmes BSD, calcule de manière périodique les priorités des processus. Une interruption, appelée *top* dans cet exercice, provoque l'activation de l'ordonnanceur. La priorité du processus courant est recalculée tous les 4 tops, suivant la formule ci-dessous :

$$usrpri = 160 + \left[\frac{ucpu}{4}\right] + 2 \times nice$$

Notre scheduler se base sur 320 niveaux de priorités, soit 320 files de processus. Ainsi, dans cette formule, nous cantonnerons $usrpri$ à $usrpri \in [0.. + 320[$, 0 correspondant à la priorité la plus forte. La valeur de $nice$ correspond à la valeur ($\in [-20.. + 20]$) fournie par l'utilisateur, -20 correspondant à la priorité la plus forte.

La valeur de $ucpu$ correspond à une estimation de la consommation CPU du processus courant. Pour calculer cette estimation :

- le système incrémente cette valeur pour le processus courant à chaque top d'horloge
- tous les 50 tops, les valeurs de $ucpu$ de tous les processus sont réajustées avec un « filtre de decay » selon la formule suivante :

$$ucpu_{n+1} = \frac{2 \times load}{2 \times load + 1} ucpu_n + nice$$

  La variable $load$, représentant la charge CPU, est calculée en comptant le nombre courant de processus dans l'état « prêt ».

Proposez une implémentation de cet ordonnanceur avec calcul dynamique des priorités. Vos sources et un fichier README (contenant les étapes nécessaires à la mise en route de l'exécutable), devront nous être remis au plus tard le vendredi 19/10/2012.