TP3 et TP4 : Simulation du Multi-tâches

¿¿¿ CE TP DOIT ÊTRE RENDU.
Les modalités seront précisées la semaine prochaine (lors du TP 5).

1 Introduction du multi-tâches

Le but de cette séance de travaux pratiques est double : d'une part, ajouter des fonctions multi-tâches en temps partagé à notre mini système et d'autre part, utiliser ces nouvelles fonctions pour endormir les processus pendant un certain temps

1.1 Étape 1 : Codage des processus

Commencez par ajouter au simulateur les structures de données ci-dessous. Elles permettent de représenter un ensemble de processus et leur mot d'état processeur.

```
#define MAX_PROCESS  (20)   /* nb maximum de processus  */

#define EMPTY         (0)   /* processus non-pret       */
#define READY         (1)   /* processus pret           */

struct {
    PSW  cpu;               /* mot d'etat du processeur */
    int  state;             /* etat du processus        */
    }
    process[MAX_PROCESS];   /* table des processus      */

int current_process = -1;   /* nu du processus courant  */
```

Faites en sorte qu'au démarrage du système (interruption INIT), le système prépare la première case du tableau des processus. Il y a pour l'instant un seul processus.

1.2 Étape 2 : Un ordonnanceur simplifié

A chaque interruption d'horloge, le système va maintenant sauvegarder le PSW dans la case correspondante du tableau des processus et chercher un nouveau processus prêt pour lui redonner la C.P.U. (voir recherche ci-dessous).

```
<sauvegarder le processus courant si il existe>
do {
    current_process = (current_process + 1) % MAX_PROCESS;
} while (process[current_process].state != READY);
<relancer ce processus>
```

A ce stade, le simulateur doit fonctionner correctement. Étant donné qu'il n'y a qu'un seul processus, il est systématiquement sauvegardé puis choisi pour être exécuté.

Pour tester votre ordonnanceur, vous pouvez maintenant créer directement deux processus au démarrage du système (prenez l'exemple de boucle du TP précédent). Utilisez le même segment pour ces deux processus. Ce sont plus des *threads* que des *processus* puisqu'ils partagent leur code et leur données. Les sorties des deux processus devraient se mélanger pour illustrer le multi-tâches simulé.

2 Les appels systèmes

2.1 Création de threads

• Dans un premier temps, vous pouvez ajouter un nouvel appel système (`SYSC Ri,Rj,SYSC_NEW_THREAD`) qui duplique le thread courant pour en créer un nouveau. Chez le père (l'appelant), le système renvoie le numéro du thread fils dans le registre `Ri` (et `AC`). Chez le fils, ce même registre est forcé à zéro (ainsi que `AC`). Le père et le fils continuent leur exécution à la première instruction qui suit l'appel au système :

```
 0 : SYSC R1, R1, SYSC_NEW_THREAD
 1 : IFGT 10
 2 : code du fils
     ..........
10 : code du pere
     ..........
```

• Programmez un exemple dans lequel le fils incrémente le contenu d'une case mémoire et le père l'affiche sans la modifier. Vous aurez sans doute besoin d'une nouvelle instruction `STORE` :

```
instruction STORE Ri, Rj, k
   | AC = Rj + k
   | si (AC < 0) ou (AC >= SS) <erreur adressage>
   | mem[SB + AC] = Ri
   | AC = Ri
   | PC += 1
```

2.2 Endormir des threads

On se propose d'implanter l'appel système `SYSC Ri,Rj,SLEEP` qui va endormir le thread courant pendant `Ri` seconde(s). Pour ce faire, vous devez :

- ajouter un état endormi,
- ajouter une date de réveil (voir `man 2 time`),
- endormir le thread courant,
- faire en sorte de le réveiller.

Vous pouvez tester cette fonction en créant deux threads (le père et le fils). Le premier affiche toujours le même entier et le second un entier différent toutes les 3 secondes.

2.3 La fonction getchar

On se propose d'implanter l'appel système `SYSC Ri,Rj,GETCHAR` qui va lire un caractère sur l'entrée standard et le placer dans `Ri` ou attendre l'arrivée d'un caractère.

Nous ne pouvons pas réellement utiliser le clavier car cela impose de contrôler parfaitement les arrivées de caractères. Nous allons donc simuler la frappe au clavier en demandant au système de placer un caractère toutes les trois secondes dans le tampon.

Pour ce faire, vous devez :

- prévoir la définition du tampon (capacité un caractère) :

```
char tampon = '\0';       /* le '\0' indique le vide */
```

- ajouter un nouvel état `GETCHAR` (endormi en attente de caractère) et un compteur de processus dans cet état,

– endormir le processus courant si le tampon est vide,

– à l'arrivée d'un caractère, vous devez réveiller un processus qui serait dans un état `GETCHAR` ou stocker le caractère dans le tampon.

Vous pouvez tester cette fonction en créant un thread qui tente de lire un caractère toutes les secondes ou toutes les quatres secondes.

2.4 Création de processus

Nouvel exercice qui ne doit pas être rendu.

On se propose d'implanter l'appel système `SYSC Ri,Rj,FORK` qui va créer un nouveau processus par recopie de la mémoire d'un processus père et recopie de l'état processeur du *thread* courant. Pour ce faire, vous devez :

– ajouter à votre système un pointeur vers la dernière case mémoire utilisée,

– prévoir la duplication de la partition courante,

Partons du principe que toutes les zones mémoire **ont la même taille**. Comment gérer la fin d'un processus et la libération de sa mémoire ?

Vous pouvez tester cette fonction en créant un processus qui va créer un processus fils toutes les quatre secondes. Ce processus fils réalise deux affichages et meurt au bout de **4 secondes**.