UNIVERSITÉ DE ROUEN	ANNÉE 2004-2005
FACULTÉ DES SCIENCES	SYSTÈME D'EXPLOITATION
LICENCE INFORMATIQUE 3ème ANNÉE	CONTRÔLE CONTINU
30 NOVEMBRE 2004	2H00

Aucun document n'est autorisé.

Exercice 1

1) Soient A, B, C, D, E et F six processus de durées respectives de 12, 8, 20, 8, 4 et 18 secondes. Les temps d'arrivées respectifs des six processus sont : 0, 0, 2, 4, 4, 10. On suppose que ces processus ne font pas d'entrée-sortie.

Déterminez les temps de réponse de chaque processus, ainsi que le temps de réponse moyen, en appliquant les disciplines d'ordonnancement suivantes. Ne tenez pas compte du temps de commutation des processus. Vous expliquerez (par exemple par des schémas ou des diagrammes) les résultats obtenus.

1. FCFS (ou FIFO).
2. SJF (plus court d'abord).
3. Ordonnancement non préemptif avec priorité statique où $P(A) = 5$, $P(B) = 3$, $P(C) = 6$, $P(D) = 2$, $P(E) = 1$, $P(F) = 4$ (1 étant la priorité maximale).
4. Tourniquet avec un quantum de 4 secondes (sans priorités).
5. Tourniquet avec un quantum de 4 secondes avec priorités statiques. On prendra cette fois-ci $P(A) = 5$, $P(B) = 3$, $P(C) = 6$, $P(D) = 3$, $P(E) = 3$, $P(F) = 1$.
6. Tourniquet avec un quantum de 4 secondes avec priorités dynamiques avec : $P_0(A) = 5$, $P_0(B) = 3$, $P_0(C) = 6$, $P_0(D) = 2$, $P_0(E) = 1$, $P_0(F) = 4$ et $P_{i+1} = P_i + (\text{temps utilisé}/4)$. Ici le temps utilisé correspond au temps effectif utilisé par le processus depuis son arrivée. À la fin du quantum de temps, le processus est remis à la fin de la file d'attente par l'ordonnanceur avant que celui-ci ne choisisse un nouveau processus à élire.

Exercice 2 On souhaite réaliser une simulation de jeu de bataille pour deux joueurs à l'aide de deux processus.

La bataille se joue à deux joueurs en répartissant l'ensemble des cartes en deux tas, faces cachées. Chaque joueur obtient un tas. Chaque joueur pose ensuite la carte du sommet de son tas sur le tapis, face visible. Celui qui à posé la plus forte carte remporte le pli et le met à la fin de son tas. Lorsque les deux joueurs posent une carte ayant la même valeur, il y a «bataille» : chaque joueur pose alors une carte face cachée sur sa première carte puis une carte face visible par-dessus, c'est cette dernière qui va permettre de décider qui remporte les six cartes. Il peut y avoir de nouveau bataille si les deux cartes sont identiques, on recommence alors l'opération jusqu'à ce qu'une carte permette de l'emporter. Un joueur gagne la partie lorsque son adversaire n'a plus de cartes à poser. Il y a égalité s'il y a bataille et que les deux joueurs n'ont pas suffisemment de cartes pour jouer celle-ci.

Vous supposerez que le système est capable de gérer des variables partagées. Le premier processus distribuera les cartes ; puis, tant qu'il restera des cartes dans sa main, il posera une carte sur le tapis, la comparera à celle posée par le deuxième processus et donnera le pli au processus vainqueur. Le deuxième processus se contentera de poser ses cartes sur le tapis.

1) Donner un algorithme permettant de simuler ce jeu à l'aide de sémaphores. Vous pourrez supposer définies les fonctions

`distribuer_cartes, poser_une_carte_sur_le_tapis, donner_pli_a_joueur(i)`

ainsi que tout autre fonction permettant de manipuler les données partagées (la difficulté ne doit pas se trouver ici). Un soin tout particulier sera porté aux portions concernant la synchronisation des processus.

2) Même question mais en utilisant cette fois l'attente active.

Exercice 3 Écrire un programme C qui utilise 3 processus H, M, S qui incrémentent les 3 «aiguilles» d'une horloge. S reçoit un signal `SIGALRM` chaque seconde et émet un signal à M quand son compteur passe de 59 à 0. Quand M reçoit un signal, il incrémente son compteur. Quand son compteur passe de 59 à 0, M envoie un signal à H. Les paramètres correspondent aux valeurs d'initialisation des compteurs.

Pour que S reçoive un signal chaque seconde, on pourra utiliser le code suivant :

```
struct itimerval minuterie;

minuterie.it_interval.tv_sec=1;
minuterie.it_interval.tv_usec=0;
minuterie.it_value.tv_sec=1;
minuterie.it_value.tv_usec=0;
if (setitimer(ITIMER_REAL, &minuterie, NULL)==-1) {
  perror("setitimer");
  exit(1);
}
```

On supposera que le signal `SIGALRM` est envoyé et reçu exactement toutes les secondes (*i.e.* on ne gérera pas les éventuels retards pris par notre horloge).

De plus, nous rappelons les appels système qui pourront être utiles lors de l'écriture du programme :

```
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

pid_t fork(void);
int sigfillset(sigset_t *set);
int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
int kill(pid_t pid, int sig);
int pause(void);
```

La fonction `pause` renvoie toujours -1. Les autres fonction renvoient -1 en cas d'erreur.

De plus vous considèrerez que la structure `struct sigaction` est définie par :

```
struct sigaction {
    void     (* sa_handler ) (int);
    sigset_t    sa_mask;
    int         sa_flags;
}
```