Contrôle Systèmes d'exploitation, Réseaux

Vendredi 22 Mars 2013
9h - 12h
Aucun document n'est autorisé

Exercice 1 : Gestion de processus (10 = 2 + 1 + 1 + 2 + 1 + 3)

1. Rappeler ce que fait le système d'exploitation lors de l'appel à la fonction système fork(). On précisera en particulier ce que fait le système d'exploitation (gestion des tables, ...).

2. Rappeler en quelques lignes ce qu'est la commutation de contexte et à quoi sert l'ordonnanceur.

3. Les processus suivants doivent être exécutés sur un ordinateur ayant un seul processeur:

Processus	Date de début d'exécution	Durée supposée d'exécution
P1	0	6
P2	4	7
P3	8	2
P4	10	3
P5	12	2

   On supposera que le temps de commutation de contexte est négligeable. Donner le diagramme de Gantt (processus en exécution, liste des processus en attente à chaque instant) et le temps moyen de traitement lorsque l'algorithme d'ordonnancement de processus utilisé par le système d'exploitation utilise la méthode dite FIFO (premier arrivé, premier servi).

4. Même question avec la méthode PCTER (plus court temps d'exécution restant, ou encore nommé SRTF, shortest remaining time first). On expliquera pourquoi cette méthode permet un temps moyen de traitement optimal.

5. Même question avec la méthode du tourniquet (on prendra 2 comme durée d'un quantum). On expliquera brièvement pour chaque commutation de contexte le choix effectué.

6. On suppose maintenant que le système a p processus en exécution en permanence. On suppose de plus que l'ordonnanceur utilise la méthode tourniquet avec un quantum de q secondes, et que pour chaque changement de processus en exécution, ordonnancement et commutation prennent c secondes. On définit le temps de latence l d'un processus comme la durée maximale entre le moment où un processus finit un quantum d'exécution et le moment où ce même processus sera de nouveau en exécution. Donner et expliquer la relation entre p, c, q et l. Donner, en la justifiant, la valeur maximale du quantum pour que le temps de latence soit inférieur à $1/10s$ [on prendra $p = 10$; $c = 1/1000$].

Exercice 2 : Parallélisation (10 = 3 + 1 + 1 + 2.5 + 2.5)

On considère l'algorithme (dit de Gauss) séquentiel suivant:

```
FOR k:= 1 TO n-1 DO
   T_{kk};
   FOR j := k+1 TO n DO
      T_{kj};
   OD
OD
```

Les T_{kj} désignent des tâches dont nous ne précisons pas le contenu et dont nous nous limitons à donner les domaines de lecture et d'écriture ($M(i,j)$ est la cellule mémoire d'adresse (i,j)) :

Pour les tâches T_{kk} :
$L_{kk} = \{M(i,k) \text{ avec } k \leq i \leq n\}$
$E_{kk} = \{M(i,k) \text{ avec } k+1 \leq i \leq n\}$

Pour les tâches T_{kj} pour j variant de $k+1$ à n :
$L_{kj} = \{M(i,k) \text{ avec } k \leq i \leq n\} \cup \{M(i,j) \text{ avec } k \leq i \leq n\}$
$E_{kj} = \{M(i,j) \text{ avec } k+1 \leq i \leq n\}$

1. Pour $n = 3$, après avoir donné le système de tâches initial, construisez et justifiez le graphe de parallélisme maximal associé.
2. Ecrire un programme avec les primitives de Dijkstra (utilisant parbegin / parend, begin / end) pour réaliser ce graphe.
3. Ecrire un programme avec les primitives de Conway (utilisant fork / join / quit) pour réaliser ce graphe.
4. Ecrire un programme en C utilisant la création de processus et des tubes pour les communications.
5. Ecrire un programme en C utilisant la création de threads et des sémaphores pour le contrôle de la synchronisation.

Fonctions et en-tête utiles:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include <unistd.h>

pid_t fork(void)
int pipe(int pipefd[2])

int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void*), void *arg)
int pthread_join(pthread_t thread, void **value_ptr)
void pthread_exit(void *value_ptr)

int pthread_mutex_lock(pthread_mutex_t *mutex)
int pthread_mutex_unlock(pthread_mutex_t *mutex)
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)
int pthread_mutex_destroy(pthread_mutex_t *mutex)
```