Institut Galilée
Année 2013-2014

Systèmes d'exploitation, Réseaux
Licence 2

Contrôle Systèmes d'exploitation, Réseaux

Vendredi 14 Mars 2014 — 9h - 12h
Aucun document n'est autorisé

Exercice 1 (5 = 2 + 1,5 + 1,5)

1. Rappeler ce qu'est la commutation de contexte et à quoi sert l'ordonnanceur. On précisera comment s'effectue la commutation entre 2 processus ou threads.
2. Les processus suivants doivent être excutés sur un ordinateur ayant un seul CPU :

Processus	Date de début d'exécution	Durée supposée d'exécution
A	0	3
B	1	6
C	4	4
D	6	2
E	7	1

   On supposera que le temps de commutation de contexte est négligeable.

   Donner le diagramme de Gantt et le temps moyen de traitement lorsque l'algorithme d'ordonnancement de processus utilisé par le système d'exploitation utilise la méthode premier arrivé premier servi (FIFO).
3. Même question avec la méthode dite du plus court temps d'exécution restant (SRTF).

Exercice 2 (5 = 2,5+2,5)

Soient deux processus A et B prêts tels que A est arrivé en premier suivi de B, 2 unités de temps après. Les temps de CPU nécessaires pour l'exécution des processus A et B sont respectivement 15 et 4 unités de temps. Le temps de commutation est supposé nul. Calculer le temps de séjour de chaque processus A et B, le temps de traitement et le temps d'attente, le temps moyen de traitement et le temps moyen d'attente, et le nombre de changements de contexte pour les méthodes avec quantum (méthodes RR) suivantes:

1. un quantum de 3 unités de temps
2. un quantum de 10 unités de temps

Exercice 3 (4 = 1+3)

1. Décrivez le fonctionnement de l'appel système `pthread_create()`. Mentionnez en particulier ce qui est effectué relativement à la table de description des processus, à la mémoire centrale. Comparer avec l'appel système `fork()`.
2. Pour le système d'exploitation unix, écrire un programme C qui doit afficher sur la sortie standard le nombre de thread fils qu'il peut créer. Pour cela, ce programme doit créer de nouveaux threads jusqu'à ce que cette opération soit impossible. Attention à la terminaison des threads!

Exercice 4 (6 = 3+3)

La fonction TRI, donnée à l'annexe 1, effectue un tri rapide (quicksort).

1. Modifiez ce programme pour que chaque appel à la fonction `TRI` soit en fait effectué par un thread créé pour cela.
2. Sous Unix, les informations sur un fichier peuvent être obtenues en utilisant la fonction système suivante:
   `int stat(const char *nom_fichier, struct stat *stat_ptr)`
   La fonction `stat()` met les informations concernant le fichier `nom_fichier` dans l'argument `stat_ptr`. Le type `stat` de cet argument est donné à l'annexe 2.

   Ecrivez un programme en `C` qui prend en entrée une liste de noms de fichiers et crée deux threads. Le premier thread trie la liste en fonction de la taille des fichiers et écrit le résultat dans un fichier `tri.taille` et le second thread trie la liste en fonction de l'heure du dernier accès et sauvegarde le résultat dans un fichier `tri.acces`.

(Annexe 1) Fonction TRI effectuant un tri rapide (quicksort):

```
#include <stdio.h>

#define N 10
int T[N] = {2,4,8,5,1,9,10,4,14,11};

int PARTITION(int *,int,int);
void ECHANGER(int *,int,int);
void TRI(int *,int,int);

void TRI(int T[N], int  d, int f) {
   if (d < f) {
       int q = PARTITION( T, d, f);
       TRI( T, d, q-1);
       TRI( T, q+1, f);
} }

int PARTITION(int T[N], int  d, int f) {
   int pivot = T[f];
   int j, i = d-1;
   for (j=d ; j<f; j++) {
    if (T[j] <= pivot) {
        i = i+1;
        ECHANGER( T, i, j);
   } }
   ECHANGER( T, i+1, f);
   return i+1;
}

void ECHANGER(int T[N], int  i, int j) {
   int temp = T[i];
   T[i] = T[j];
   T[j] = temp;
}

void main () {
 int i;
 for (i=0;i<N;i++) printf("%d ",T[i]);
 printf("\n");
 TRI(T,0,N-1);
 for (i=0;i<N;i++) printf("%d ",T[i]);
 printf("\n");
}
```

(Annexe 2) Description du type `stat`:

```
struct stat {
 unsigned long   st_dev;     /* Peripherique.                        */
 unsigned long   st_ino;     /* Numero i-noeud.                      */
 unsigned int    st_mode;    /* Protection.                          */
 unsigned int    st_nlink;   /* Nb de liens materiels.               */
 unsigned int    st_uid;     /* User ID proprietaire.                */
 unsigned int    st_gid;     /* Group ID proprietaire.               */
 unsigned long   st_rdev;    /* Numero de peripherique.              */
 long            st_size;    /* Taille totale en octets.             */
 int             st_blksize; /* Taille de bloc pour E/S.             */
 long            st_blocks;  /* Nombre de blocs de 512 octets alloues. */
 long            st_atime;   /* Heure de dernier acces.              */
 long            st_mtime;   /* Heure de derniere modification.      */
 long            st_ctime;   /* Heure de dernier changement.         */
};
```

(Annexe 3) Fonctions et en-tête utiles:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

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 open(const char *pathname, int flags);
int close(int fd);
ssize_t write(int fd, const void *buf, size_t count);
```