Institut Galilée
Année 2011-2012

Systèmes d'exploitation, Réseaux
Licence 2

–CORRIGÉ– Contrôle Systèmes d'exploitation, Réseaux

Mercredi 9 Mars 2012
9h - 12h
Aucun document n'est autorisé

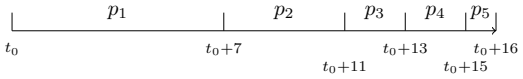
Exercice 1 : Ordonnancement de processus (6 = 3 + 3)

On considère les cinq exécutions de processus suivants (la durée est exprimée en seconde) :

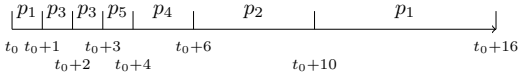
Processus	Date d'arrivée	Durée
P1	0	7
P2	1	4
P3	1	2
P4	2	2
P5	3	1

1. Donner les diagrammes de Gantt et les temps de traitement moyen obtenus à l'aide des algorithmes d'ordonnancement FIFO (Premier arrivé - Premier servi), PCTER (plus court temps d'exécution restant) et Tourniquet (avec un quantum de 1) en supposant un temps de commutation de contexte négligeable.

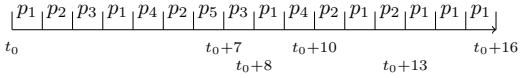
   Cas FIFO : $tmt = ((7-0)+(11-1)+(13-1)+(15-2)+(16-3))/5 = 11$

   Cas PCTER : $tmt = ((16-0)+(13-1)+(3-1)+(6-2)+(4-3))/5 = 7$

   Cas Tourniquet : $tmt = ((16-0)+(13-1)+(8-1)+(10-2)+(7-3))/5 = 9,4$

2. Si le temps de commutation est de 0,5 seconde, quel est alors le temps moyen de traitement dans le cas d'un ordonnancement PCTER et d'un ordonnancement tourniquet. Qu'en déduisez-vous ?

Cas PCTER : il y a 5 commutations de contexte (changement de processus élu). Le tmt est alors le suivant :

$$((18,5-0)+(12-1)+(3,5-1)+(7,5-2)+(5-3))/5 = 7,9$$

Cas Tourniquet : il y a 13 commutations de contexte (changement de processus élu). Le tmt est alors le suivant :

$$((22,5-0)+(19-1)+(11,5-1)+(14,5-2)+(10-3))/5 = 14,1$$

Tenir compte des comuutations ne change pas le fait que le protocole PCTER soit meilleure que les autres (et en particulier que le protocole tourniquet). Le temps dû aux commutations est toutefois sensible sur le tmt avec le protocole tourniquet, celui-ci doit donc rester à un niveau modéré.

Exercice 3 : Parrallélisation (3)

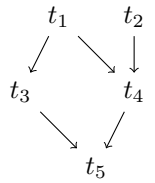
On considère l'ensemble de tâches suivant (qui calcule $(x+y+z)/u*v*(x+y)$) :

```
t1 : a = x + y ;
t2 : b = u * v ;
t3 : c = a + z ;
t4 : d = a * b :
t5 : R = c / d ;
```

Le système de tâches avec l'ordre $t1 < t2 < t3 < t4 < t5$ est-il déterminé ? Si oui, donner le système de parallélisme maximal équivalent et en déduire un programme utilisant les primitives de Conway fork/join/quit.

Le système avec l'ordre $t1 < t2 < t3 < t4 < t5$ est déterminé : l'ordre est total donc le calcul est parfaitement déterminé par la séquence des instructions.

Pour calculer le système de parallélisme maximal équivalent, on conserve les couples de tâches (de l'ordre initial) qui sont de domaine d'écriture non vide (c'est toujours le cas ici) et pour lesquels une des tâches a une variable en écriture qui est en lecture ou en écriture pour l'autre tâche. C'est le cas pour les paires de tâches $(t1,t3)$, $(t3,t5)$, $(t1,t4)$, $(t2,t4)$, $(t4,t5)$. Le graphe de précédence est le suivant :



Le programme peut s'écrire alors comme suit :

```
label   code
        m := 2 ; n := 2 ; fork p1 ; fork p2 ; quit ;
p1 :    a := x+y ; fork p3 ; join m,p4 ; quit ;
p2 :    b := u*v ; join m,p4 ; quit ;
p3 :    c := a + z ; join n,p5 ; quit ;
p4 :    d := a*b ; join n,p5 ; quit ;
p5 :    R := c/d ;
```

Exercice 4 : Threads et exclusion mutuelle (8 : 1 + 3 + 2 + 2)

On cherche à programmer un logiciel capable de compter le nombre de 'a' dans une chaîne de caractères. Concrètement il s'agit d'écrire un programme ayant les caractéristiques suivantes :

- Une fonction `int compter(char *s)` qui retourne le nombre de 'a' de la chaîne de caractères `s`.
- Le programme a une donnée globale qui est un tableau de chaîne de caractères nommé `char *requetes[]` de taille 5.
- Le programme crée un thread qui doit exécuter la fonction `void *accueil(void*s)` et 10 threads qui exécutent la fonction `void *executer(void *s)`, puis attend que tous les threads soient terminés.
- La fonction `void *accueil(void*s)` est une boucle infinie qui lit du clavier une chaîne de caractères et la met à la première case libre du tableau `requetes` si le tableau n'est pas plein.
- La fonction `void *executer(void *s)` récupère du tableau `requetes` une chaîne s'il y en a une, appelle la fonction `compter` et affiche le résultat.

1. Ecrire la fonction `compter`.
2. Sans considérer les problèmes d'exclusion mutuelle, écrire les fonctions `accueil` et `executer`.
3. Ecrire le reste du programme : fonction `main` et variables globales nécessaires.
4. Ajouter les mutex nécessaires au bon fonctionnement.

(compiler avec `gcc executer.c -lpthread -lreadline`)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <signal.h>
#include <sys/wait.h>
#include <readline/readline.h>
#include <readline/history.h>
```

```
14
15 #define TAILLE_TABLEAU 5
16
17 char *requetes[TAILLE_TABLEAU];
18 int caseVide = 0;
19 int casePleine = 0;
20 int nbcasespleines = 0;
21
22 pthread_mutex_t mutex_maitre, mutex_esclave, mutex_nbcasespleines;
23
24 int compter(char *s){
25    int n = 0;
26    while (*s != '\0') {
27       if (*s == 'a') n++;
28       s++;
29    }
30    return n;
31 }
32
33 void *accueil(void *s){
34    while(1) {
35       pthread_mutex_lock(&mutex_nbcasespleines);
36       if (nbcasespleines == TAILLE_TABLEAU){
37          pthread_mutex_unlock(&mutex_nbcasespleines);
38          pthread_mutex_lock(&mutex_maitre);
39       }
40       else {
41          requetes[caseVide] = readline("%>>");
42          caseVide = (caseVide + 1) %5;
43          nbcasespleines++;
44          pthread_mutex_unlock(&mutex_nbcasespleines);
45          pthread_mutex_unlock(&mutex_esclave);
46       }
47    }
48 }
49
50 void *executer(void *s){
51    while(1) {
52       pthread_mutex_lock(&mutex_nbcasespleines);
53       if (nbcasespleines == 0){
54          pthread_mutex_unlock(&mutex_nbcasespleines);
55          pthread_mutex_lock(&mutex_esclave);
56       }
57       else {
58          printf("Nombre␣de␣'a'␣dans␣%s␣=␣%d\n",
59             requetes[casePleine],
60             compter(requetes[casePleine]));
61          free(requetes[casePleine]);
62          casePleine = (casePleine + 1) %5;
63          nbcasespleines--;
```

```
64            pthread_mutex_unlock(&mutex_nbcasespleines);
65            pthread_mutex_unlock(&mutex_maitre);
66         }
67      }
68 }
69
70 int main(){
71
72    int i;
73    pthread_t thread_maitre ;
74    pthread_t tab_thread_esclave[10] ;
75
76    if( pthread_mutex_init(&mutex_esclave,NULL) < 0 ){
77       perror("Pas de creation du mutex esclave");
78       exit(-1);
79    };
80    if( pthread_mutex_init(&mutex_maitre,NULL) < 0 ){
81       perror("Pas de creation du mutex maitre");
82       exit(-1);
83    };
84    if( pthread_mutex_init(&mutex_nbcasespleines,NULL) < 0 ){
85       perror("Pas de creation du mutex nbcasespleines");
86       exit(-1);
87    };
88
89    if (pthread_create(&thread_maitre,NULL,accueil,NULL) < 0){
90       perror("Pas de creation du thread maitre");
91       exit(-1);
92    };
93    for (i=0;i<10;i++) {
94       if (pthread_create(&tab_thread_esclave[i],NULL,executer,NULL) < 0){
95          perror("Pas de creation du thread esclave");
96          exit(-1);
97       };
98    }
99    ( void ) pthread_join ( thread_maitre ,(void *) &i) ;
100   return 0;
101 }
```