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, ...).

   CORRECTION:¶
   pages 2-3 du poly et cours
   ¶

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

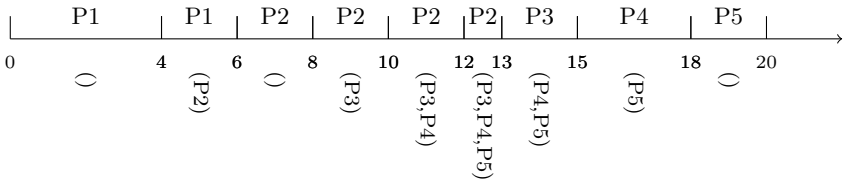
   CORRECTION:¶
   pages 4-5 et suivantes du poly
   ¶

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).

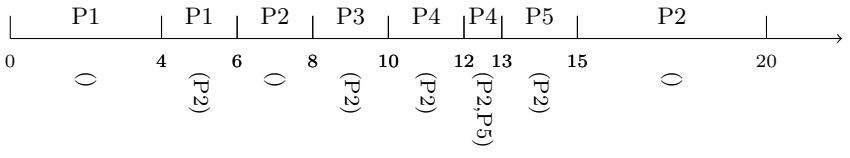
   CORRECTION:¶



   Le temps moyen de traitement (t.m.t.) est $((6-0)+(13-4)+(15-8)+(18-10)+(20-12))/5 = 38/5 = 7,6$ unités de temps.
   ¶

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.

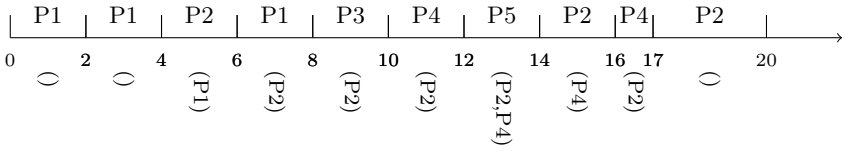
   CORRECTION:¶
   Voir TD.



Le temps moyen de traitement (t.m.t.) est $((6-0)+(20-4)+(10-8)+(13-10)+(15-12))/5 = 30/5 = 6$ unités de temps.
¶

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.

   CORRECTION:¶
   Un processus nouvellement créé est élu de préférence à ceux en liste d'attente. Lorsqu'un processus termine avant la fin de son quantum, un autre processus est immédiatement élu.



   Le temps moyen de traitement (t.m.t.) est $((8-0)+(20-4)+(10-8)+(17-10)+(14-12))/5 = 35/5 = 7$ unités de temps.
   ¶

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 $p = 10$; $c = 1/1000$].

   CORRECTION:¶
   Chaque exécution partielle de processus "prend" q secondes suivi de c secondes pour la commutation. Un processus après son quantum doit "attendre" l'exécution des quantums des $p-1$ autres processus. Donc le temps de latence est $l = c + (p-1) * (q+c)$.

   Pour avoir $l < 1/10$, il faut donc $q < 1/100$seconde. $1/100$ est donc la valeur maximale du quantum.
   ¶

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.

   CORRECTION:¶
   Système de tâches correspondant au programme initial : $(\{T_{11}, T_{12}, T_{13}, T_{22}, T_{23}\}, T_{11} < T_{12} < T_{13} < T_{22} < T_{23})$.

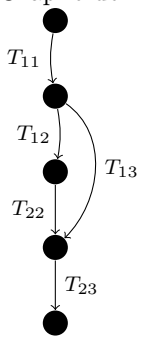
   On calcule les paires de tâches indispensables, c'est à dire les couples (T_{ij}, T_{kl}) tels que: $T_{ij} < T_{kl}$ et $E_{ij} \neq \emptyset$ et $E_{kl} \neq \emptyset$ et $(E_{ij} \cap L_{kl} \neq \emptyset$ ou $L_{ij} \cap E_{kl} \neq \emptyset$ ou $E_{ij} \cap E_{kl} \neq \emptyset)$.

Dans le cas présent:

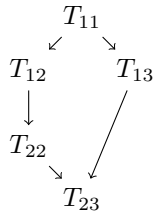
$$\begin{array}{ll} L_{11} = \{M_{11}, M_{21}, M_{31}\} & E_{11} = \{M_{21}, M_{31}\} \\ L_{12} = \{M_{11}, M_{21}, M_{31}, M_{12}, M_{22}, M_{32}\} & E_{12} = \{M_{22}, M_{32}\} \\ L_{13} = \{M_{11}, M_{21}, M_{31}, M_{13}, M_{23}, M_{33}\} & E_{13} = \{M_{23}, M_{33}\} \\ L_{22} = \{M_{22}, M_{32}\} & E_{22} = \{M_{32}\} \\ L_{23} = \{M_{22}, M_{32}, M_{23}, M_{33}\} & E_{23} = \{M_{33}\} \end{array}$$

Donc on garde $T_{11} < T_{12}$ et $T_{11} < T_{13}$ mais pas $T_{12} < T_{13}$. On garde aussi $T_{12} < T_{22} < T_{23}$ et aussi $T_{13} < T_{23}$ mais pas $T_{13} < T_{22}$. D'où les graphes de parallélisme maximal suivants:

Graphe de flot:



Graphe de précédence:



¶

2. Ecrire un programme avec les primitives de Dijkstra (utilisant parbegin / parend, begin / end) pour réaliser ce graphe.

CORRECTION:¶

```
1         begin
2                 T11
3                 parbegin
4                         begin T12 ; T22 end
5                         | T13
6                 parend;
7                 T23
8         end;
```

¶

3. Ecrire un programme avec les primitives de Conway (utilisant fork / join / quit) pour réaliser ce graphe.

CORRECTION:¶

```
label   code
        m:= 2; T11; fork p1; fork p2; quit;
p1:     T12; T22; join m,p3; quit;
p2:     T13; join m,p3; quit;
p3:     T23; quit;
```

¶

4. Ecrire un programme en C utilisant la création de processus et des tubes pour les communications.

CORRECTION:¶

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4
5 void (* tab_taches[5]) (void);
6
7 void T11(void),T12(void),T13(void),T22(void),T23(void);
8
```

```
9  int M11=11,M21=21,M31=31,M12=12,M22=22,M32=32,M13=13,M23=23,M33=33;
10 int tube1112[2],tube1222[2],tube2223[2],tube1113[2],tube1323[2],tube23fin[2];
11
12 int main(int argc,char *argv[]) {
13         int i;
14         pid_t pid;
15
16         if(pipe(tube1112)==-1) {perror("tube1112");exit(-1);}
17         if(pipe(tube1222)==-1) {perror("tube1222");exit(-1);}
18         if(pipe(tube2223)==-1) {perror("tube2223");exit(-1);}
19         if(pipe(tube1113)==-1) {perror("tube1113");exit(-1);}
20         if(pipe(tube1323)==-1) {perror("tube1323");exit(-1);}
21         if(pipe(tube23fin)==-1) {perror("tube23fin");exit(-1);}
22
23         tab_taches[0] = T11;tab_taches[1] = T12;tab_taches[2] = T13;
24         tab_taches[3] = T22;tab_taches[4] = T23;
25
26         for (i=0;i<5;i++) {
27                 pid = fork();
28                 if (pid==-1) {perror("fork");exit(-1);}
29                 if (pid==0) tab_taches[i]();
30         }
31
32         if (read(tube23fin[0],&i,sizeof (int) )==-1) {perror("read tube23fin");exit(-1);}
33         close (tube23fin[0]);
34         printf("Resultat : %d\n",i);
35 }
36
37 void T11(void) {          /*T11*/
38
39         printf("T11: M11, M21, M31 connues (%d,%d,%d)\n",M11,M21,M31);
40
41         printf("T11: calcul de M21 et M31\n");
42
43         printf("T11: envoi de M21, M31 vers T12\n");
44         close(tube1112[0]);
45         if(write(tube1112[1],&M21,sizeof(int))==-1) {perror("write tube1112");exit(-1);}
46         if(write(tube1112[1],&M31,sizeof(int))==-1) {perror("write tube1112");exit(-1);}
47         close(tube1112[1]);
48
49         printf("T11: envoi de M21, M31 vers T13\n");
50         close(tube1113[0]);
51         if(write(tube1113[1],&M21,sizeof(int))==-1) {perror("write tube1113");exit(-1);}
52         if(write(tube1113[1],&M31,sizeof(int))==-1) {perror("write tube1113");exit(-1);}
53         close(tube1113[1]);
54
55         exit(0);
56 }
57
58 void T12(void) {          /*T12*/
59
60         printf("T12: M11, M12, M22, M32 connues (%d,%d,%d,%d)\n",M11, M12, M22, M32);
61         if (read(tube1112[0],&M21,sizeof (int) )==-1) {perror("read tube1112");exit(-1);}
62         if (read(tube1112[0],&M31,sizeof (int) )==-1) {perror("read tube1112");exit(-1);}
63         close (tube1112[0]);
64         printf("T12: M21, M31 de T11 (%d,%d)\n",M21, M31);
65
66         printf("T12: calcul de M22 et M32\n");
67
68         printf("T12: envoi de M22, M32 vers T22\n");
69         close(tube1222[0]);
70         if(write(tube1222[1],&M22,sizeof(int))==-1) {perror("write tube1222");exit(-1);}
71         if(write(tube1222[1],&M32,sizeof(int))==-1) {perror("write tube1222");exit(-1);}
72         close(tube1222[1]);
73
```

```
74          exit(0);
75  }
76
77  void T13(void) {          /*T13*/
78
79          printf("T13: M11, M13, M23, M33 connues (%d,%d,%d,%d)\n",M11, M13, M23, M33);
80          if (read(tube1113[0],&M21,sizeof (int) )==-1) {perror("read tube1113");exit(-1);}
81          if (read(tube1113[0],&M31,sizeof (int) )==-1) {perror("read tube1113");exit(-1);}
82          close (tube1113[0]);
83          printf("T13: M21, M31 de T11 (%d,%d)\n",M21, M31);
84
85          printf("T13: calcul de M23 et M33\n");
86
87          printf("T13: envoi de M23, M33 vers T23\n");
88          close(tube1323[0]);
89          if(write(tube1323[1],&M23,sizeof(int))==-1) {perror("write tube1323");exit(-1);}
90          if(write(tube1323[1],&M33,sizeof(int))==-1) {perror("write tube1323");exit(-1);}
91          close(tube1323[1]);
92
93          exit(0);
94  }
95
96  void T22(void) {          /*T22*/
97
98          if (read(tube1222[0],&M22,sizeof (int) )==-1) {perror("read tube1222");exit(-1);}
99          if (read(tube1222[0],&M32,sizeof (int) )==-1) {perror("read tube1222");exit(-1);}
100         close (tube1113[0]);
101         printf("T22: M22, M32 de T12 (%d,%d)\n",M22, M32);
102
103         printf("T22: calcul de M32\n");
104
105         printf("T22: envoi de M22, M32 vers T23\n");
106         close(tube2223[0]);
107         if(write(tube2223[1],&M22,sizeof(int))==-1) {perror("write tube2223");exit(-1);}
108         if(write(tube2223[1],&M32,sizeof(int))==-1) {perror("write tube2223");exit(-1);}
109         close(tube1323[1]);
110
111         exit(0);
112 }
113
114 void T23(void) {          /*T23*/
115
116         if (read(tube2223[0],&M22,sizeof (int) )==-1) {perror("read tube2223");exit(-1);}
117         if (read(tube2223[0],&M32,sizeof (int) )==-1) {perror("read tube2223");exit(-1);}
118         close (tube1113[0]);
119         printf("T23: M22, M32 de T22 (%d,%d)\n",M22, M32);
120         if (read(tube1323[0],&M23,sizeof (int) )==-1) {perror("read tube1323");exit(-1);}
121         if (read(tube1323[0],&M33,sizeof (int) )==-1) {perror("read tube1323");exit(-1);}
122         close (tube1113[0]);
123         printf("T23: M23, M33 de T13 (%d,%d)\n",M23, M33);
124
125         printf("T23: calcul de M33\n");
126
127         printf("T23: envoi de M33 vers fin\n");
128         close(tube23fin[0]);
129         if(write(tube23fin[1],&M33,sizeof(int))==-1) {perror("write tube23fin");exit(-1);}
130         close(tube23fin[1]);
131
132         exit(0);
133 }
```

¶

5. Ecrire un programme en C utilisant la création de threads et des sémaphores pour le contrôle de la synchronisation.

CORRECTION:¶

```
1  /* gcc -o prog_thread prog_thread.c -lpthread */
2
3  #include<stdio.h>
4  #include<stdlib.h>
5  #include<unistd.h>
6  #include<pthread.h>
7
8  void *(* tab_taches[5]) (void*);
9
10 void *T11(void*),*T12(void*),*T13(void*),*T22(void*),*T23(void*);
11 int M11=11,M21=21,M31=31,M12=12,M22=22,M32=32,M13=13,M23=23,M33=33;
12
13 static pthread_mutex_t verrou1112,verrou1222,verrou2223,verrou1113,verrou1323,verrou23fin;
14
15
16 int main(int argc,char *argv[]) {
17         int i;
18         pthread_t thread[5];
19
20
21 /* initialisation des verrous */
22         if(pthread_mutex_init(&verrou1112, NULL)==-1) {perror("verrou1112");exit(-1);}
23         if(pthread_mutex_init(&verrou1222, NULL)==-1) {perror("verrou1222");exit(-1);}
24         if(pthread_mutex_init(&verrou2223, NULL)==-1) {perror("verrou2223");exit(-1);}
25         if(pthread_mutex_init(&verrou1113, NULL)==-1) {perror("verrou1113");exit(-1);}
26         if(pthread_mutex_init(&verrou1323, NULL)==-1) {perror("verrou1323");exit(-1);}
27         if(pthread_mutex_init(&verrou23fin, NULL)==-1) {perror("verrou23fin");exit(-1);}
28
29         pthread_mutex_lock (&verrou1112);pthread_mutex_lock (&verrou1222);pthread_mutex_lock (&ve
30         pthread_mutex_lock (&verrou1113);pthread_mutex_lock (&verrou1323);pthread_mutex_lock (&ve
31
32         tab_taches[0] = T11;tab_taches[1] = T12;tab_taches[2] = T13;
33         tab_taches[3] = T22;tab_taches[4] = T23;
34
35
36 /* creation des threads */
37         for (i=0;i<5;i++) {
38                 if (pthread_create(&thread[i], NULL, tab_taches[i], NULL) != 0)
39                         {perror("pthread_create : erreur");exit(-1);}
40         }
41
42         pthread_mutex_lock (&verrou23fin);
43         printf("Resultat : %d\n",M33);
44 }
45
46
47 void *T11(void* arg) {  /*T11*/
48
49         printf("T11: M11, M21, M31 connues (%d,%d,%d)\n",M11,M21,M31);
50
51         printf("T11: calcul de M21 et M31\n");
52
53         printf("T11: envoi de M21, M31 vers T12\n");
54         pthread_mutex_unlock(&verrou1112);
55
56         printf("T11: envoi de M21, M31 vers T13\n");
57         pthread_mutex_unlock(&verrou1113);
58
59         pthread_exit(0);
60 }
61
62 void *T12(void* arg) {  /*T12*/
63
```

```
64          printf("T12: M11, M12, M22, M32 connues (%d,%d,%d,%d)\n",M11, M12, M22, M32);
65          pthread_mutex_lock(&verrou1112);
66          printf("T12: M21, M31 de T11 (%d,%d)\n",M21, M31);
67
68          printf("T12: calcul de M22 et M32\n");
69
70          printf("T12: envoi de M22, M32 vers T22\n");
71          pthread_mutex_unlock(&verrou1222);
72
73          pthread_exit(0);
74  }
75
76  void *T13(void* arg) {   /*T13*/
77
78          printf("T13: M11, M13, M23, M33 connues (%d,%d,%d,%d)\n",M11, M13, M23, M33);
79          pthread_mutex_lock(&verrou1113);
80          printf("T13: M21, M31 de T11 (%d,%d)\n",M21, M31);
81
82          printf("T13: calcul de M23 et M33\n");
83
84          printf("T13: envoi de M23, M33 vers T23\n");
85          pthread_mutex_unlock(&verrou1323);
86
87          pthread_exit(0);
88  }
89
90  void *T22(void* arg) {   /*T22*/
91
92          pthread_mutex_lock(&verrou1222);
93          printf("T22: M22, M32 de T12 (%d,%d)\n",M22, M32);
94
95          printf("T22: calcul de M32\n");
96
97          printf("T22: envoi de M22, M32 vers T23\n");
98          pthread_mutex_unlock (&verrou2223);
99
100         pthread_exit(0);
101 }
102
103 void *T23(void* arg) {   /*T23*/
104
105         pthread_mutex_lock(&verrou2223);
106         printf("T23: M22, M32 de T22 (%d,%d)\n",M22, M32);
107         pthread_mutex_lock(&verrou1323);
108         printf("T23: M23, M33 de T13 (%d,%d)\n",M23, M33);
109
110         printf("T23: calcul de M33\n");
111
112         printf("T23: envoi de M33 vers fin\n");
113         pthread_mutex_unlock (&verrou23fin);
114
115         pthread_exit(0);
116 }
```

¶