LI324- Système - Durée : 3 heures

Mai 2007 - Toute documentation autorisée

Examen - Barème indicatif

Luciana Arantes, Gael Thomas et Maria Gradinariu

1. SYNCHRONISATION

Soit un processus coordinateur ***Z*** et N processus ***X*** de calcul (X1, …, XN). Chaque processus ***Xi*** attend d'être réveillé par le processus *Z*. Les processus X appellent la fonction *calcul_1( )* puis la fonction *calcul_2( )*. Les deux fonctions peuvent êtres exécutées concurremment par les processus *Xi*. Le squelette d'un processus *Xi* (1 <= i <= N) est :

Processus Xi ( ){

```
  P(S1);
  calcul_1 ( ) ;
   . . .
  calcul_2 ( ) ;
   . . .
}
```

Le sémaphore S1 est initialisé à 0 : S1 = CS (0) ;

Observations pour toutes les questions qui suivent:

1. Vous **ne pouvez pas** utiliser d'attente active. Utilisez des sémaphores pour bloquer des processus.

2. Tous les processus *Xi* doivent exécuter une fois la fonction *calcul_1( )* et une fois la fonction *calcul_2( ).*

3. Si vous ajoutez des sémaphores et/ou variables partagées à la solution d'une question, vous devez indiquer leur initialisation.

1.1.

Nous souhaitons dans un premier temps que la fonction *calcul_2 ( )* ne puisse être appelée que si **tous** les *Xi* ont déjà exécuté *calcul_1 ( )*. De plus, il faut que le processus ***Z*** coordonne le démarrage de l'exécution de la fonction *calcul_1 ( )* ainsi que le démarrage de l'exécution de la fonction *calcul_2 ( )*.

Donnez le code du processus ***Z*** et compléter le code du processus ***Xi***.

1.2.

Nous souhaitons encore que la fonction *calcul_2 ( )* ne puisse être appelée que si tous les ***Xi*** ont déjà exécuté *calcul_1 ( )*. Cependant, le processus ***Z*** n'autorise qu'une exécution à la fois de la fonction *calcul_1()*. *calcul_1 ( )* ne peut donc être exécutée que par un processus ***Xi*** à la fois. En revanche, comme précédemment la fonction *calcul_2 ( )* peut être exécutée concurremment par les processus ***Xi***. Observez que le processus coordinateur ***Z*** est toujours

le coordinateur du démarrage de l'exécution de la fonction *calcul_1 ( )* et de la fonction *calcul_2 ( )* par un processus *Xi*.

Donnez le code du processus ***Z*** et des processus ***Xi***.

Nous considérons maintenant que le processus ***Z*** n'a plus le rôle de coordinateur et ne fait que démarrer les N processus ***Xi***. Son code est :

Processus Z :

```
for (i = 0; i < N; i++)
   V(S1) ;
```

Le code des processus *Xi* est :

Processus Xi ( ){

```
P(S1);
calcul_1 ( ) ;
calcul_2 ( ) ;
```

}

Nous ajoutons au programme la variable **partagée *cpt*** initialisée au début du programme à la valeur *K (K < N)*.

Nous nous intéressons maintenant au code des fonctions *calcul_1 ( )* et *calcul_2 ( )*.

1.3.

Nous considérons que la fonction *calcul_1 ( )* décrémente d'une unité la valeur de *cpt* tandis que la fonction *calcul_2 ( )* l'incrémente d'une unité. Cependant, la variable *cpt* ne peut être décrémentée que si elle est supérieure à 0. Si n'est pas le cas, la fonction *calcul_1 ( )* doit attendre que la valeur de *cpt* redevienne supérieur à *0* pour la décrémenter. Notez que tous les processus *Xi* doivent décrémenter et incrémenter la valeur de la variable une fois et lorsque tous les processus *Xi* ont terminé leur exécution la valeur de ***cpt*** sera égale à K.

Observation : Un même processus *Xi* **ne doit pas garder** l'accès exclusif à ***cpt*** entre *calcul_1 ( )* et *calcul_2 ( )*.

Donnez le code de *calcul_1 ( )* et *calcul_2 ( )*.

2. PAGINATION

Soient deux processus P1 et P2. Le code de P1 se trouve en mémoire à l'adresse E1 et celui de P2 à l'adresse E2. Chaque instruction est codée sur 1 mot (octet) mémoire.

Les variables M, N et K sont des variables entières **partagées** par P1 et P2, et %r1 %r2 sont des registres du processeur.

Le code de P1 en assembleur est :

E1	load_mem %r1, M	(a)	lit l'octet à l'adresse de M dans le registre %r1
E1+1	sub %r1, 1	(b)	%r1 := %r1 - 1
E1+2	store_mem M, %r1	(c)	écrit %r1 à l'adresse de M
E1+3	store_mem K, %r1	(d)	écrit %r1 à l'adresse de K
E1+4	jmp_if_not_zero E1	(e)	saute en E1 si %r1 est différent de 0

Ce qui est équivalent à la séquence C suivante :

```
for( ; M !=0 ; M--) {
  K = M ;
}
```

Le code de P2 en assembleur est :

E2	load_mem %r1, M	(f)	lit l'octet à l'adresse de M dans %r1
E2+1	load_mem %r2, N	(g)	lit l'octet à l'adresse de N dans %r2
E2+2	add %r2, %r1	(h)	%r2 := %r2 + %r1
E2+3	store_mem N, %r2	(i)	écrit %r2 à l'adresse de N

Ce qui est équivalent à la séquence C suivante :

```
N = N + M ;
```

Chaque instruction est exécutée de façon atomique, c'est-à-dire qu'il ne peut pas y avoir commutation pendant une instruction assembleur.

On suppose une mémoire simplement paginée et que **les pages et les cases mémoire font 512 mots**. E1 est à l'adresse 0, E2 est à l'adresse 512, M est à l'adresse 1024 et contient initialement 2, N est à l'adresse 1536 et contient initialement -1, K est à l'adresse 2048 et contient initialement 0.

2.1

P1 et P2 s'exécutant de manière concurrente, donnez les différentes valeurs de N à la fin de l'exécution en donnant, pour chaque valeur, une séquence d'exécution.

2.2

Donnez, dans l'ordre, les numéros de pages accédées par P1. Vous noterez aussi les pages accédées en écriture. Même question pour P2. (Conseil : pensez à compter les accès au code).

2.3

P1 et P2 s'exécutent sur un système multi-tâches à temps partagé. Un quantum de temps correspond exactement à l'exécution de 3 instructions. Initialement, P1 s'exécute. Donnez la séquence d'exécution et la séquence d'utilisation des pages.

Dans la suite de l'exercice, on considère la séquence d'accès mémoire suivante

0 2 0 2 0 4 1 2 1 3 1 2 0 2 0 2 0 4 0

qui correspond à la séquence d'accès mémoire exécutée avec un quantum de temps de 4 instructions, dans laquelle les doublons ont été supprimés (par exemple 0 – 0 est simplifié en 0).

Le système dispose de deux cases de mémoire libre, numérotée 10 et 20.

2.4

En utilisant un algorithme de remplacement de page FIFO, donnez l'évolution de la table des pages sur la séquence d'accès mémoire. Combien de défauts de pages sont engendrés ? Vous indiquerez aussi l'état des tables des pages des deux processus à la fin de l'exécution.

2.5

Combien de défauts de page, au minimum, cette séquence peut-elle engendrer ? Décrivez votre politique de remplacement de page et donnez l'évolution de la table des pages.

On suppose qu'un compteur est associé à chaque page. Ce compteur est stocké sur 2 bits **:** à chaque accès à la page, ce compteur est incrémenté. Une fois que le compteur vaut 3 (i.e. les deux bits valent 1), le compteur n'est plus incrémenté.

Le système gère une liste FIFO des pages : lorsqu'une page est chargée, elle est placée en queue du FIFO avec un compteur égal à 1. Lors d'un défaut de page, la page en tête de liste est examinée. Si son compteur vaut 0, elle est déchargée et la case libérée est affectée à la page qui a engendré le défaut de page. Sinon, la page est remise en queue de liste, son compteur est décrémenté de 1 et la nouvelle page en tête de liste est examinée.

2.6

Donnez l'évolution de la table des pages et le nombre de défauts de page engendrés.

2.7

Comment peut-on réaliser l'algorithme optimal vu en TD pour un processus ?

3. PROCESSUS

Soit le programme suivant :

```
1.int main(int argc,char *argv[])
2. {
3.  pid_t pid;
4. int var,i,j;


5.  pid = fork();
6.  var=une_fonction_qui_calcule_quelque_chose();

7.  switch (pid) {
8.  case -1:
9.    fprintf(stderr,"%s: fork impossible\n",argv[0]);
10.    exit(EXIT_FAILURE);
11. case 0:
12.    printf("Var fils %d\n", var+=13);
13.    for (i=1; i<3 && fork()==0; i++)
14.      printf("i=%d a var=%d\n",i, var+=i);
15.    j=i;
16.    while(--j >0 && fork()==0) printf("i=%d et j=%d\n", i,j);
17.    exit(EXIT_SUCCESS);
```

```
18.  default:
19.    printf("Pere %d\n",var);
20.    wait(NULL);
21.    exit(EXIT_SUCCESS);
22.  }
23. }
```

La fonction une_fonction_qui_calcule_quelque_chose() renvoie toujours la valeur 13.

Remarque : `--j < 0` ligne 16 signifie dans la décrémentation de j a lieu **avant** le test <0.

3 .1.

Combien de processus sont créés par cette application ? Afficher l'arbre des dépendances.

3.2.

Quels sont les affichages produits à l'ecran ?

3.3.

Supposons que la ligne (6) est remplacée par

6. sprintf(buf,"%d",var=une_fonction_qui_calcule_quelque_chose()))

6'. execl(''/bin/echo'',''echo'',buf,NULL).

Combien de processus sont créés par la nouvelle application ?

2 processus (en comptant le main)