Partiel 2004-2005

Documents et calculatrice interdits. Durée : 2 h 00.

{Nicolas.Ollinger,Emmanuel.Godard,Yann.Esposito}@lif.univ-mrs.fr

3 novembre 2004

1 Questions de cours

Soyez concis ! Répondez en au plus *20 lignes* pour chaque question.

1.1. Décrivez les grandes étapes de l'exécution d'un système d'exploitation (du boot à l'arrêt). N'oubliez pas de traiter `init`. Précisez les étapes de vie et de mort des processus.

1.2. Expliquez ce qu'est la synchronisation de processus. Quelle est son utilité ? Donnez un exemple. Définissez les concepts : interblocage, sémaphore et mutex.

1.3. Expliquez par un petit texte et des schémas les interactions d'un processus avec le système d'exploitation ; en particulier, précisez le rôle de : l'ordonnanceur, la gestion d'interruption et la gestion d'appel système.

2 Synchronisation de processus

2.4. Synchronisation de deux processus.

(a) Donnez un scénario dans lequel, les deux processus exécutent leur section critique simultanément.

```
   Processus 1                    Processus 2
...                          |   ...
while (incs);                |   while (incs);
incs=1;                      |   incs=1;
                             |
  CS                         |     CS
                             |
incs=0;                      |   incs=0;
...                          |   ...
```

(b) Voici une amélioration. Est-ce que les deux processus peuvent entrer en section critique en même temps ? Si oui donner un parcours d'exécution qui permet d'exécuter les deux section critiques en même temps. Si non donnez une preuve (par exemple en testant tous les cas possibles).

```
   Processus 1                    Processus 2
...                          |   ...
interested[1]=1              |   interested[2]=1
while (interested[2]         |   while (interested[1]
       && incs);             |         && incs) ;
incs=1                       |   incs=1
                             |
  CS                         |     CS
                             |
incs=0                       |   incs=0
...                          |   ...
```

2.5. On veut créer deux processus avec le comportement suivant : le premier pose des éléments sur une pile de taille `N`, le second dépile les éléments. Il ne faut jamais que la pile soit dépilée alors qu'elle est vide. Il ne faut jamais remplir la pile si la taille maximale est dépassée.

On propose les deux algorithmes suivants :

```
Initialisation :
Semaphore mutex = 1 ;
Semaphore plein = N ;

Algorithme 1     |  Algorithme 2
...              |  ...
down(plein) ;    |
down(mutex) ;    | down(mutex) ;
empile()  ;      | depile() ;
up(mutex) ;      | up(mutex) ;
                 | up(plein) ;
...              | ...
```

(a) Expliquez pourquoi cette solution n'est pas souhaitable.

(b) Donnez une solution avec des sémaphores et des mutex.

3 Ordonnancement

Considérons un système dans lequel les processus suivants sont soumis à l'ordonnanceur tous en même temps.

3.6. Comparez les performances du système en suivant des politiques d'ordonnancement de type "Premier arrivé, premier servi", "le plus court d'abord", "tourniquet" avec échantillon de temps égal à 4.

Proc.	durée
0	11
1	6
2	7
3	14
4	8

Qu'apporte l'introduction d'un processus "bidon" P_{-1} qui s'exécute pendant deux unités de temps et une fois terminé est recréé et remis dans le système. Comment changent les performances du système par rapport aux algorithmes d'ordonnancements précédents ?

3.7. On suppose maintenant les temps d'arrivée suivants pour chacun des processus.

Proc.	arrivée
0	2
1	3
2	0
3	1
4	5

Comment se comparent les performances par rapport à l'exercice précédent.

4 Gestion de la mémoire

4.8. Soit la suite de références de pages suivantes :

1,2,3,4,2,1,5,2,6,1,2,3,7,5,3,2,1,2,3,4

Combien se produit-il de fautes de pages pour les algorithmes de remplacement LRU, FIFO, optimal, en supposant qu'il y a trois, puis cinq pages physiques.

4.9. Une machine a un espace d'adressage de 32 bits et des pages de 8Ko. La table des pages est entièrement gérée par le matériel. Lorsqu'un processus est exécuté, la table des pages est recopiée dans le matériel à partir de la mémoire, un mot toutes les 200ns. Si chaque processus s'exécute pendant 50ms (y compris le temps de chargement de la table des pages), quelle fraction du temps de la CPU est-elle consacrée au chargement des tables des pages ? (rappel $1\mathrm{ns} = 10^6\mathrm{ms}$).

4.10. Considérons un système avec un dispositif de type TLB. Le temps d'accès mémoire est de 60ns. Une requête au TLB coûte 5ns. La mémoire est subdivisée en pages de taille 4Ko. La table des pages est elle-même subdivisée en page de 4Ko. Quel doit-être le taux de succès pour que l'ajout du TLB ne ralentisse pas le système de plus de 20% ?

5 Du code source au code binaire (question bonus)

5.11. Voici un programme qui calcule x^n.

```
1  int puissance(int x, int n) {
2    if (n<1)
3      return 1;
4    return x*puissance(x,n-1);
5  }
6
7  void main(void) {
8    puissance(3);
9  }
```

Voici le code assembleur correspondant sur une plateforme ant32.

```
1  :puissance
2    entry 0            // allocation mémoire des variables locales
3    ld4 g1,fp,12       // récupération du premier argument (g1 = x)
4    ld4 g2,fp,8        // récupération du second argument  (g2 = n)
5
6    gts g2,1,g1        // g2 vaut 1 si g1<1 et 0 sinon
7    jnz r0,g2,$casbase // saute à casbase si g2 est différent de 0
8
9    push g1            // met g1 sur la pile (met x sur la pile)
10
11   sub g2,g2,1        // g1 = g1 - 1  (g1 = n-1)
12   push g1            // empile g1    (x)
13   push g2            // empile g2    (n-1)
14   call $puissance     // appelle puissance (g0 vaut puissance (x,n-1) )
15
```

```
16      pop g1              // depile g1 (qui vaut x)
17      mul g3,g1,g0        // g3 = g1*g0 (c'est-à-dire x*puissance(x,n-1))
18      return g3           // retour du résultat
19
20    :casbase
21      return 1
22
23    :main
24      push 2            // Le premier argument est 2
25      push 3            // Le second argument est 3
26      call $fonction    // on appelle la fonction fonction
```

(a) Expliquez pourquoi est-ce qu'on empile g1 à la ligne 9.

(b) Expliquez pourquoi est-ce qu'on empile g1 à la ligne 12.

(c) Peut-on se passer d'empiler g1 à la ligne 9 ? Si oui expliquez comment faire sinon expliquez pourquoi.

En vous inspirant du code du programme précédent, donnez le code assembleur du programme suivant (rappel `add g1,g2,g3` revient ‡ faire g1=g2+g3) :

```
1   int fonction(int n) {
2     if (n<1)
3       return 1;
4     return n+fonction(n-1) ;
5   }
6
```