Institut Galilée
Année 2014-2015

Systèmes d'exploitation, Réseaux
Licence 2

Contrôle Systèmes d'exploitation, Réseaux

Mercredi 11 Mars 2015 — 9h - 12h
Aucun document n'est autorisé

Exercice 1 (3 = 1,5 + 1,5)

1. Les processus suivants doivent être exécutés sur un ordinateur ayant un seul CPU :

Processus	Date de début d'exécution	Durée supposée d'exécution
A	0	2
B	1	7
C	3	3
D	5	2
E	6	2

   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).

2. Même question avec la méthode dite tourniquet (Round Robin). On prendra comme quantum $q = 2$.

Exercice 2 (4,5 = 2 + 1 + 1,5)

L'ordonnancement EDF (Earliest Deadline First) est un algorithme d'ordonnancement temps réel de processus. A chaque fois qu'un processus demande du temps CPU, il doit préciser une *échéance* (une date limite > 0 de début d'exécution). Le processus doit obtenir du temps CPU avant d'atteindre l'échéance. L'ordonnanceur vise à satisfaire les demandes avant leurs échéances. Pour cela, il gère une liste des processus prêts, classés par ordre croissant des échéances. L'algorithme exécute le premier processus de la liste qui correspond à celui dont l'échéance est la plus proche. Lorsqu'un processus devient prêt ou est créé, le système vérifie si son échéance est strictement inférieure à celle du processus en cours d'exécution. Si tel est le cas, le nouveau processus préempte le processus en cours. Chaque processus a normalement le comportement cyclique suivant: il est d'abord prêt, puis en exécution, puis en attente (donc pas prêt), puis redevient prêt, puis en exécution, puis en attente, etc.

1. Considérez trois processus A, B et C suivants :

   - A devient prêt toutes les 30ms et son temps d'exécution à chaque fois est de 10 ms.
   - B devient prêt toutes les 40ms et son temps d'exécution à chaque fois est de 15 ms.
   - C devient prêt toutes les 50ms et son temps d'exécution à chaque fois est de 5 ms.

   Supposez qu'à la date t_0, les processus sont prêts et que l'échéance de chaque demande est à t_0 la date de la prochaine demande (donc au départ l'échéance est à $t_0 + 30$ pour A, $t_0 + 40$ pour B, $t_0 + 50$ pour C, puis ensuite $t_0 + 60$ pour A, etc.). Donnez le diagramme de Gantt de t_0 à $t_0 + 150$ms.

2. Calculez le temps moyen de traitement entre t_0 et $t_0 + 150$ms.

3. Peut-on avoir des cas de non respect d'échéances ? Justifiez votre réponse.

Exercice 3 (4 = 2+1+1)

Soit les fonctions système suivantes:

- `WIFEXITED (status)` renvoie vrai si le statut provient d'un processus fils qui s'est terminé normalement;
- `WEXITSTATUS (status)` (si `WIFEXITED (status)` renvoie vrai) renvoie le code de retour du processus fils passé à `exit()` ou la valeur retournée par la fonction `main()`.

On considère le programme suivant:

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <wait.h>

char n;

int main(void) {
        pid_t pid;
        int status;
        if( (pid = fork()) == -1 ) exit(EXIT_FAILURE);
        if(pid != 0){
                n++;
                wait(&status);
                if(WIFEXITED(status)) n+=WEXITSTATUS(status);
        } else n--;
        printf("[%d] : valeur de n est %d\n",getpid(),-n);
        exit(n);
}
```

1. Détailler (ligne par ligne) le comportement de ce programme.
2. On suppose que : le PID du père est 2000, le processus fils a comme PID 2001 et se termine de façon normale. Que va-t-il être affiché à l'écran?
3. Sous les mêmes hypothèses, dire si le programme compile, et si oui ce qui est affiché à l'écran si on déplace la ligne 6 respectivement à la ligne 12 et à la ligne 15. Justifiez vos réponses.

Exercice 4 (3)

On considère le programme suivant :

```c
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int main(void) {
        if(fork()) wait(NULL);
        printf("[%d] fini \n",getpid());
        return 0;
}
```

En sachant qu'à la terminaison d'un fils le signal `SIGCHLD` est envoyé au père, réécrire ce programme sans utiliser l'appel système `wait`. Par exemple, en utilisant l'appel système `int pause(void)` (`pause` endort le processus jusqu'à ce qu'un signal soit reçu), on pourra écrire une fonction `void wait_simple(int)` qui met le processus dans l'état endormi jusqu'à ce qu'un signal `SIGCHLD` soit reçu.

Exercice 5 (5,5 = 1 + 1 + 0,5 + 1 + 2)

1. Donner un exemple simple de système de tâche ne pouvant pas être décrit avec les primitives parbegin/parend.

   On considère pour la suite de cet exercice le programme utilisant les primitives parbegin/parend donné ci-dessous.
2. Donner le graphe de flot et le graphe de précédence en précisant les tâches que vous considérez (une tâche est une instruction de calcul).
3. Indiquer les domaines de lecture et écriture des différentes tâches.
4. Donner le programme correspondant en utilisant les primitives de Conway fork/join/quit.
5. Donner le programme correspondant en C en utilisant des threads.

```
begin
  parbegin
    lire(a) |
    lire(b)
```

```
  parend ;
  parbegin
    c := a*b |
    begin
      d := a*a ;
      e := d*a
    end
  parend ;
  e := c + d + e
end
```

(Annexe) Fonctions et en-tête utiles:

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>
#include <pthread.h>

pid_t getpid(void);
pid_t getppid(void);
pid_t fork(void);

int pthread_create(pthread_t *thread,
                   pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
void pthread_exit(void *value_ptr);
int pthread_join(pthread_t thread, void **value_ptr);

void (*signal(numsig, void (*fonction) (int))) (int);
int kill(pid_t pid, int sig);

pid_t wait(int *status);
```