Lian Paul ft

Examen Intra

IFT 2245

18 février 2025

Directives

- Vous avez droit à une page de notes (recto verso) écrite **à la main** (remettez votre page avec votre examen).
- Calculatrice est autorisée.
- Répondez dans le cahier fourni **à l'exception de Question 1** que vous pouvez répondre sur cette feuille.
- Le pointage pour chaque question est entre parenthèses (total = 20).
- Les traductions en anglais sont en *italics*.
- Vous pouvez répondre en anglais ou en français.
- Notez clairement toutes les suppositions que vous faites.

Question 0. *Nom et matricule (1 point de bonus)*

Écrivez votre nom et votre matricule en haut de cette feuille et sur votre cahier d'examen.

Question 1. *Questions à choix multiples ou à réponses courtes (5 points)*

(a) (1 point) Considérez cette solution simple au problème de la section critique :

```
do {
   while (compare_and_swap(&lock, 0, 1) != 0)
     do nothing;
       /* critical section */
     lock = 0;
       /* remainder section */
} while (true);
```

Quelles propriétés d'une solution au problème de la section critique **sont satisfaites** (écrivez-les ici) ?

Exclusion mutuelle, Progrès

Quelles propriétés d'une solution au problème de la section critique **ne sont pas satisfaites** (écrivez-les ici) ?

Attente limitée

(b) (0.5 points) L'algorithme d'ordonnancement *Shortest Job First* est optimal en termes de minimisation du temps de réponse moyen.

A. Vrai
B. Faux

(c) (0.5 points) Dans l'algorithme d'ordonnancement tourniquet (*round robin*) :

A. L'augmentation du quantum **augmentera toujours** le délai d'exécution moyen (*average turnaround time*)
B. L'augmentation du quantum **diminuera toujours** le délai d'exécution moyen (*average turnaround time*)
C. L'augmentation du quantum **augmentera parfois et diminuera parfois** le délai d'exécution moyen (*average turnaround time*)

(d) (0.5 points) Lesquels des éléments suivants sont stockés dans le bloc de contrôle de processus (*Process Control Block (PCB)*) (cochez tout ce qui s'applique) :

- [x] Contenu des registres
- [x] L'etat du processus
- [x] Le compteur de programme *program counter*
- [x] Liste des fichiers ouverts

(e) (0.5 points) Quel type de communication interprocessus est presque exclusivement utilisé pour la communication entre un processus parent et son processus enfant :

A. Sockets
B. Appels de procédure à distance (*Remote procedure calls*)
C. Tuyaux ordinaires (*ordinary pipes*)
D. Tuyaux nommés (*named pipes*)

(f) (0.5 points) Pourquoi le temps nécessaire pour créer un nouveau thread dans un processus est-il inférieur au temps requis pour créer un nouveau processus (cochez tout ce qui s'applique) ?

- [x] Parce que nous n'avons pas besoin de créer un nouvel espace d'adressage
- [ ] En raison de la parallélisation
- [ ] Parce que nous n'avons pas à créer une nouvelle pile
- [x] Parce que nous n'avons pas à créer un nouveau PCB

(g) (0.5 points) Un système avec un seul core peut exécuter des threads :

A. En parallèle et concurremment
B. En parallèle mais pas concurremment
C. Concurremment mais pas en parallèle
D. Ni concurremment ni en parallèle

(h) (1 point) De quelles manières un processus peut-il arriver dans la "queue prête" (*ready queue*) ? (cochez tout ce qui s'applique) ?

- [x] Être créé par un "fork()"
- [x] Être interrompu
- [ ] Être bloqué par un E/S (I/O)
- [ ] Terminer
- [x] Réalisation d'un E/S (I/O)
- [ ] Être choisi par l'ordonnanceur court-terme
- [ ] Être "swapped oui"
- [x] Être "swapped in"

Question 2. *Synchronization (5 points)*

Dans le problème des "dining philosophes", il y a N philosophes assis autour d'une table avec N baguettes. Chaque philosophe i a une baguette à sa droite (baguette i) et une baguette à sa gauche (baguette $(i+1)\%N$). Il y a un bol de riz au milieu. Un philosophe passe son temps à manger et à réfléchir. Pour manger, il doit tenir deux baguettes (celle de droite et celle de gauche).

La structure du philosophe i est :

```
while(true){
  THINK(i);
  PICK_UP(i);
  EAT(i);
  PUT_DOWN(i);
```

Vous concevrez une solution au problème des "dining philosophes" qui évite l'impasse (*deadlock*) (mais pas nécessairement la famine) en utilisant **des sémaphores**, où il y a un sémaphore pour chaque baguette (initialisé à 1 puisque toutes les baguettes sont initialement disponibles).

```
sem chopstick[N] \\ all initialised to 1
```

Rappelons qu'il existe deux opérations que l'on peut effectuer sur un sémaphore : `signal(S)` et `wait(S)`. Pour un sémaphore S, elles sont définies comme suit :

```
wait (S) {
  while (S <= 0); // blocked
  S--;
}

signal (S) {
  S++;
}
```

Dans votre solution, vous devez éviter l'interblocage (*deadlock*) en imposant que si un philosophe essaie de prendre une deuxième baguette, mais qu'elle n'est pas disponible, il doit remettre en place la baguette qu'il tient. Pour ce faire, vous pouvez utiliser une troisième opération, `try_wait(S)`, qui fonctionne exactement comme `wait(S)` sauf qu'elle ne bloque pas, mais renvoie `false` dans le cas où le sémaphore n'est pas disponible, et `true` dans le cas où il l'est :

```
bool try_wait(S) {
  if (S <= 0) return false;
  else {
    S--;
    return true
  }
}
```

Implémenter (écrire le pseudo-code) pour les fonctions `PICK_UP(i)` et `PUT_DOWN(i)`.

Question 3. *Ordonnancement (6 points)*

Considérez un système informatique avec une CPU et un périphérique (I/O) et trois processus, P1, P2 et P3. Dans cette question, nous ferons l'ordonnancement pour le CPU et le I/O ensemble. Notez que :

- un processus ne peut pas s'exécuter dans le CPU et faire du I/O en même temps
- un seul processus peut s'exécuter dans le CPU à la fois, et un seul processus peut faire du I/O à la fois
- il est possible qu'un processus s'exécute dans le CPU en même temps qu'un autre processus fait du I/O

Les séquences (l'ordre est important !) des "CPU bursts" et "I/O burst" pour les processus sont les suivantes (en commençant en même temps pour tous les processus) :

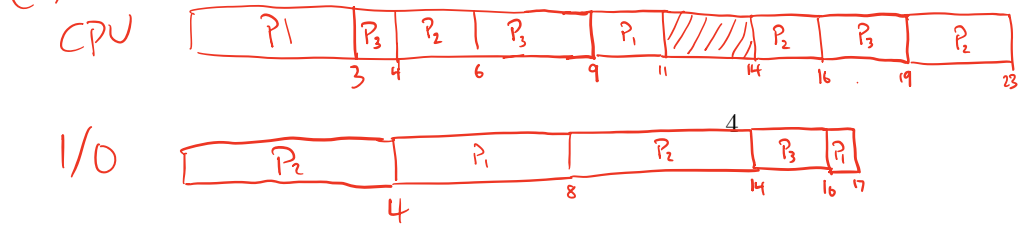
- P1 : CPU burst de 3ms, I/O burst de 4ms, CPU burst de 2ms, I/O burst de 1ms
- P2 : I/O burst de 4ms, CPU burst de 2ms, I/O burst de 6ms, CPU burst de 6ms
- P3 : CPU burst de 5ms, I/O burst de 2ms, CPU burst de 3ms

Le CPU utilise l'algorithme "temps restant le plus court en premier (avec préemption)" (*shortest remaining time first with preemption)*, et le I/O utilise l'algorithme "premier arrivé premier servi" (*first come first served*).

(a) (4 points) Dessinez les diagrammes de Gantt pour le CPU et le I/O

(b) (1 point) Calculez l'utilisation du processeur (*CPU utilization*) entre le moment où le premier processus démarre et le moment où le dernier processus termine toutes ses CPU et I/O bursts.

(c) (1 point) Quel est le temps d'attente (*waiting time*) total pour le processus P3 ?



(b) 20/23

(c) 3 + 2 + 5 = 10

Question 4. *Processus et Threads (4 points)*

(a) (2 points) Qu'est qui est imprimé quand ce code est exécuté :

```
int value = 5;
void *runner(void *param);
int main(int argc, char *argv[])
{
    pid_t pid;
    pthread_t tid;
    pthread_attr_t attr;
    pid = fork();
    if (pid == 0) {
        pid_t pid2;
        pid2 = fork();
        if (pid2 == 0) {
            pthread_attr_init(&attr);
            pthread_create(&tid, &attr, runner, NULL);
            pthread_join(tid, NULL);
        }
        else {
            waitpid(pid2, NULL, 0);
        }
        printf("LINE A, value=%d\n", value);
    }
    else {
        waitpid(pid, NULL, 0);
    }
    printf("LINE B, value=%d\n", value);
}
void *runner(void *param) {
    value += 5;
    pthread_exit(0);
}
```

LINE A, value = 10
LINE B, value = 10
LINE A, value = 5
LINE B, value = 5
LINE B, value = 5

(voir (b) sur la page suivante)

(b) (2 points) Qu'est qui est imprimé quand ce code est exécuté :

```
int value = 5;
void *runner(void *param);
int main(int argc, char *argv[])
{
  pid_t pid;
  pthread_t tid;
  pthread_attr_t attr;
  pid = fork();
  if (pid == 0) {
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,NULL);
    pthread_join(tid,NULL);
    printf("LINE A value = %d\n",value);
  }
  else if (pid > 0) {
    wait(NULL);
    printf("LINE B value = %d\n", value);

  }
  printf("LINE C value = %d\n", value);

}

void *runner(void *param) {
  value += 5;
  char buffer[100];
  sprintf(buffer, "LINE D value = %d",value);
  execlp("echo","echo",buffer,NULL);
  pthread_exit(0);
}
```

LINE D value = 10

LINE B value = 5

LINE C value = 5