

# Examen Intra

IFT 2245

9 mars 2022

## Directives

- Vous avez droit à une page de notes (recto) écrite à la main.
- 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 aux questions en anglais ou en français.
- Notez clairement toutes les suppositions que vous faites.

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

Écrivez votre nom et prénom et votre matricule en haut et sur la page couverture du cahier réponse.

### Question 1. *Questions à choix multiple (3 points)*

(a) (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)

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

(b) (0.5 points) De quelles manières un processus peut-il arriver dans la "queue prête" (*ready queue*)? (cochez tout ce qui s'applique)?

- Être créé par un "fork()"
- Être interrompu
- Être bloqué par un E/S (*I/O*)
- Terminer
- Réalisation d'un E/S (*I/O*)
- Être choisi par l'ordonnanceur court-terme

(c) (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

(d) (0.5 points) La relation entre un programme et un processus est :

- A. un-à-un
- B. un-à-plusieurs
- C. plusieurs-à-un
- D. plusieurs-à-plusieurs

(e) (0.5 points) Étant donné qu'un changement de contexte prend temps = T, quelle est la durée maximale pendant laquelle un processus peut être bloqué pour qu'il soit plus efficace d'utiliser un spinlock au lieu d'un mutex ?

- A. Il est toujours plus efficace d'utiliser un spinlock
- B.  $0.5 * T$
- C.  $2 * T$
- D. T
- E. Il n'est jamais plus efficace d'utiliser un spinlock

(f) (0.5 points) Spinlocks ne devraient pas être utilisés dans les systèmes à processeur unique.

- A. Vrai
- B. Faux

## Question 2. Synchronization (4 points)

Implémentez une solution au problème "bounded-buffer" avec un moniteur. Votre buffer doit stocker des variable "int". La fonction `void produce(int v)` prend un "int" et le place dans le buffer (tant qu'il y a de l'espace) et la fonction `int consume()` renvoie un "int" (tant qu'il y en a un). Vous pouvez supposer que la taille maximale du "buffer" est définie par la variable globale `MAX_SIZE`. Veuillez définir les structures de données nécessaires dans le moniteur (y compris le buffer) ainsi que les fonctions de production et de consommation.

Utilisez la structure ci-dessous :

```
monitor bounded buffer {
    int buffer[MAX_SIZE];
    // À faire

    void produce(int v) {
        // À faire
    }
    int consume() {
        int retVal;
        // À faire
        return retVal;
    }
}
```

**Question 3. Ordonnancement (5 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 bien sûr qu’un processus ne peut pas s’exécuter dans le CPU et faire du I/O en même temps. Et bien sûr, un seul processus peut s’exécuter dans le CPU à la fois, et un seul processus peut faire du I/O à la fois (mais il est possible qu’un processus s’exécute dans le CPU en même temps qu’un autre processus fait du I/O).

Les séquences 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 3ms
- P2 : I/O burst de 6ms, CPU burst de 2ms, I/O burst de 8ms, CPU burst de 2ms, I/O burst de 1ms, CPU burst de 3ms
- P3 : CPU burst de 6ms, I/O burst de 2ms, CPU burst de 5ms

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) (3 points) Dessinez les diagrammes de Gantt pour le CPU et le I/O
- (b) (1 point) Calculez l’utilisation du processeur (*CPU utilization*)
- (c) (1 point) Calculez le délai d’exécution moyen (*average turnaround time*)

**Question 4. Processus (2 points)**

Considérez le code suivant et supposez que le vrai PID de l’enfant est 1234 et le vrai PID du parent est 5678 (la fonction `getpid` renvoie le PID du processus qui l’appelle) :

```
int main()
{
    pid_t pid, pid1, pid2;
    pid1 = getpid();
    pid = fork();
    pid2 = getpid();
    if(pid == 0){
        printf("child: pid = %d \n", pid);
        printf("child: pid1 = %d \n", pid1);
        printf("child: pid2 = %d \n", pid2);
    }
    else {
        wait(NULL);
        printf("parent: pid = %d \n", pid);
        printf("parent: pid1 = %d \n", pid1);
        printf("parent: pid2 = %d \n", pid2);
    }
    return 0;
}
```

Qu’est-ce qui est imprimé ?

**Question 5. Threads (3 points)**

Considérez l'extrait de code suivant (supposons que toutes les structures de données sont correctement initialisées) :

```
int value;
void *runner(void *param);
int main(int argc, char *argv[])
{
    value = J;
    int N = N;

    for (int i = 0; i < N; i++){
        pthread_create(&tid[i], &attr, runner, NULL);
    }
    for (int i = 0; i < N; i++){
        pthread_join(tid[i], NULL);
    }

    printf("value = %d\n", value);
    return 0;
}

void *runner(void *param)
{
    value = value + value
    pthread_exit(0);
}
```

- (a) (1 point) Quelles modifications apporteriez-vous pour éviter les conditions de course ?
- (b) (2 points) Une fois les conditions de courses sont éliminées, quelle valeur est imprimée (en fonction de J et N) ?

**Question 6. Interblocage (3 points)**

Considérez un système d'exploitation qui gère les interblocages en garantissant qu'il ne s'en produise pas (évitement). Il n'y a qu'une seule ressource  $R$  avec 15 instances, et les allocations maximum et actuelle de la ressource entre 3 processus sont les suivantes :

Processus	Maximum	Actuel
$P_0$	14	6
$P_1$	7	2
$P_2$	9	2

- (a) (1 point) le système est-il dans un état sûr ? pourquoi ou pourquoi pas ?
- (b) (1 point)  $P_1$  fait une demande de 4 ressources, doit-elle être donnée ? pourquoi ou pourquoi pas ?
- (c) (1 point)  $P_2$  fait une demande pour 1 ressource, doit-elle être donnée ? pourquoi ou pourquoi pas ?