Examen Intra

IFT 2245

10 mars 2021

Directives

— Tout documentation autorisée.

— Test a été administré sur Studium et certains des nombres ont été randomisés

Question 1.

Pourquoi le temps nécessaire pour créer un nouveau thread dans un processus est-il inférieur au temps requis pour créer un nouveau processus ?

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

Question 2.

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

- ☒ Etre créé par un "fork()"
- ☒ Etre interrompu
- ☐ Bloqué par un E / S
- ☐ La terminaison du processus
- ☒ Réalisation d'un E / S
- ☐ Etre ~~choisi par l'ordonnanceur~~

Question 3.

Un système avec un seul core peut exécuter des threads :

1. en parallèle et concurremment
2. en parallèle mais pas concurremment
3. concurremment mais pas en parallèle
4. ni concurremment ni en parallèle

Question 4.

La relation entre un programme et un processus est :

1. un-à-un
2. un-à-plusieurs
3. plusieurs-à-un
4. plusieurs-à-plusieurs

Question 5.

La relation entre un programme et un processus est :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 ?

1. Il est toujours plus efficace d'utiliser un spinlock
2. 0.5*T
3. 2*T
4. Il n'est jamais plus efficace d'utiliser un spinlock
5. T

Question 6.

Spinlocks ne devraient pas être utilisé dans les systèmes à processeur unique.

1. vrai
2. faux

Question 7.

Implémentez une solution au problème "bounded-buffer" avec un moniteur. Votre buffer doit stocker des ints. 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;
  }
}
```

```
monitor bounded buffer
{
        int items[MAX ITEMS];
        int numItems = 0;
        condition full, empty;

        void produce(int v) {
                while (numItems == MAX ITEMS) full.wait();
                items[numItems++] = v;
                empty.signal();
        }

        int consume() {
                int retVal;
                while (numItems == 0) empty.wait();
                retVal = items[--numItems];
                full.signal();
                return retVal;
        }
}
```

Question 8.

Considérez un système informatique avec une CPU et un périphérique (I/O) et trois processus, P1, P2 et P3. Les séquence 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

Nous allons faire l'ordonnancement pour le CPU et le périphérique (ils sont interdépendants). Le CPU utilise l'algorithme "shortest remaining time first" (avec préemption), et le périphérique utilise l'algorithme "first come first served" (FCFS).

(a) (3 points) Dessinez les diagrammes de Gantt pour le CPU et le périphérique

(b) (1 point) Calculez l'utilisation du processeur (CPU utilization)

(c) (1 point) Calculez le temps d'attente moyen (average waiting time) (N.B. : temps d'attente = temps dans la queue prête)

(d) (1 point) Calculez le délai d'exécution moyen (average turnaround time)

P1	CPU 3	IO 4	CPU 3			
P2	IO 6	CPU 2	IO 8	CPU 2	IO 1	CPU 3
P3	CPU 6	IO 2	CPU 5			

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
CPU	P1			P3			P2			P3			P1						P2		P3		P2				P3		
IO			P2					P1						P2					P3		P2								

CPU alg

SRTF

I/O

FCFS

CPU ut	24/28		waiting time P1	1	turnaround time P1	14
avg waiting time	3		waiting time P2	0	turnaround time P2	24
avg turnaround time	22		waiting time P3	8	turnaround time P3	28

Question 9.

Considérez le code suivant et supposez que le vrai PID de l'enfant est 1234 et le vrai PID du parent est 5678 :

```
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 ?

child: pid = 0
child: pid1 = 5678
child: pid2 = 1234
parent: pid = 1234
parent: pid1 = 5678
parent: pid2 = 5678

Question 10.

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) un mutex pour protéger l'accès à cette ligne

(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) ?

(b) J*2^N