Systèmes d'exploitation
Test n°2

Partie A : Vrai/Faux et Choix multiples

1. Vrai/Faux :

— **a.** *Vrai* : Les threads partagent le même espace mémoire que le processus parent, ce qui permet un accès rapide aux ressources partagées.
— **b.** *Vrai* : Les sémaphores sont utilisés pour contrôler l'accès concurrentiel et éviter les conditions de concurrence.

2. Choix multiples :

— **a.** Laquelle des affirmations suivantes est vraie à propos des threads ?
Réponse : *i) Les threads peuvent s'exécuter en parallèle sur des processeurs multi-cœurs.*
— **b.** Que fait un sémaphore ?
Réponse : *i) Limite le nombre de threads pouvant accéder à une ressource.*

Partie B : Problèmes pratiques

1. Synchronisation avec les threads

Code en C :

```c
#include <pthread.h>
#include <stdio.h>

int shared_counter = 0;
pthread_mutex_t lock;

void* increment_counter(void* arg) {
```

```c
    for (int i = 0; i < 50; i++) {
        pthread_mutex_lock(&lock);
        shared_counter++;
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    pthread_mutex_init(&lock, NULL);

    pthread_create(&thread1, NULL, increment_counter, NULL);
    pthread_create(&thread2, NULL, increment_counter, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    pthread_mutex_destroy(&lock);

    printf("Final counter value: %d\n", shared_counter);
    return 0;
}
```

2. Exemple avec sémaphore

Code en C :

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>

sem_t printer_semaphore;

void* use_printer(void* arg) {
    int id = *(int*)arg;

    sem_wait(&printer_semaphore);
    printf("Student %d is using a printer.\n", id);
    sleep(2); // Simulate printing time
    printf("Student %d is done printing.\n", id);
    sem_post(&printer_semaphore);
```

```
16
17      return NULL;
18  }
19
20  int main() {
21      pthread_t students[5];
22      int ids[5];
23
24      sem_init(&printer_semaphore, 0, 2);
25
26      for (int i = 0; i < 5; i++) {
27          ids[i] = i + 1;
28          pthread_create(&students[i], NULL, use_printer, &ids[i]);
29      }
30
31      for (int i = 0; i < 5; i++) {
32          pthread_join(students[i], NULL);
33      }
34
35      sem_destroy(&printer_semaphore);
36      return 0;
37  }
```