Systèmes d'exploitation

Test n°2

Partie A : Questions de base

1. Définir ce qui suit :

- **a. Fil :** Un fil (thread) est une unité d'exécution d'un programme, souvent appelée "fil d'exécution". Il partage l'espace mémoire et les ressources avec d'autres fils du même processus.
- **b. Sémaphore :** Un sémaphore est une variable ou un type de signal utilisé pour contrôler l'accès à des ressources partagées dans un environnement multithread. Il est principalement utilisé pour synchroniser l'exécution.

2. Quel est l'objectif de l'utilisation des sémaphores dans les programmes multithread ?

Les sémaphores sont utilisés pour éviter les conditions de concurrence en contrôlant l'accès simultané aux ressources partagées, garantissant ainsi une exécution correcte et cohérente.

3. Expliquer ce qui se passe lorsque deux threads accèdent à la même ressource simultanément sans synchronisation appropriée.

Lorsque deux threads accèdent à la même ressource sans synchronisation, des comportements indéterminés peuvent se produire, comme des incohérences dans les données, des corruptions, ou des conditions de concurrence qui rendent le programme imprévisible.

4. Quel est le rôle des opérations `wait()` et `post()` dans les sémaphores ?

— `wait()` : Diminue la valeur du sémaphore. Si le résultat est négatif, le thread est mis en attente jusqu'à ce qu'une ressource soit disponible.
— `post()` : Augmente la valeur du sémaphore. Si un thread était en attente, il est réveillé pour continuer l'exécution.

Partie B : Problèmes simples

1. Création de fil

Code en C :

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

void* print_bonjour(void* arg) {
    for (int i = 0; i < 5; i++) {
        printf("Bonjour\n");
    }
    return NULL;
}

void* print_monde(void* arg) {
    for (int i = 0; i < 5; i++) {
        printf("Monde\n");
    }
    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    pthread_create(&thread1, NULL, print_bonjour, NULL);
    pthread_create(&thread2, NULL, print_monde, NULL);

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

    return 0;
}
```

2. Mise en œuvre du sémaphore

Code en C :

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

#define PARKING_CAPACITY 10

sem_t parking_spots;

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

    printf("Car %d is trying to park.\n", id);
    sem_wait(&parking_spots); // Wait for a parking spot
    printf("Car %d has parked.\n", id);
    sleep(2); // Simulate parking duration
    printf("Car %d is leaving the parking.\n", id);
    sem_post(&parking_spots); // Release the parking spot

    return NULL;
}

int main() {
    pthread_t cars[15];
    int ids[15];

    sem_init(&parking_spots, 0, PARKING_CAPACITY);

    for (int i = 0; i < 15; i++) {
        ids[i] = i + 1;
        pthread_create(&cars[i], NULL, car_arrives, &ids[i]);
    }

    for (int i = 0; i < 15; i++) {
        pthread_join(cars[i], NULL);
    }

    sem_destroy(&parking_spots);
    return 0;
}
```