Solution for the Test

Partie A : Vrai/Faux et Choix multiples

1. Vrai/Faux :

a- **Vrai** : Les threads partagent le même espace mémoire que le processus qui les a créés. Cela signifie qu'ils peuvent accéder aux mêmes variables et données.

b- **Vrai** : Un sémaphore peut être utilisé pour empêcher les conditions de course. Les sémaphores sont des mécanismes de synchronisation qui permettent de contrôler l'accès à une ressource partagée par plusieurs threads.

2. Choix multiple :

a- **i) Les threads peuvent s'exécuter en parallèle sur des processeurs multicœurs.** Les threads peuvent s'exécuter en parallèle sur des processeurs multicœurs, ce qui permet une exécution concurrente. Les threads partagent également des ressources comme la mémoire, et ils peuvent être synchronisés.

b- **i) Limite le nombre de threads pouvant accéder à une ressource.** Un sémaphore est utilisé pour limiter le nombre de threads pouvant accéder à une ressource partagée, ce qui permet d'éviter les conflits et les conditions de course.

Partie B : Scénarios simples

1. Création de thread

Voici un exemple de pseudo-code pour créer deux threads. Le premier thread imprime les numéros de 1 à 5, et le second thread imprime les numéros de 6 à 10.

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

// Fonction pour le premier thread
void* thread1(void* arg) {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return NULL;
}

// Fonction pour le second thread
void* thread2(void* arg) {
```

```
    for (int i = 6; i <= 10; i++) {
        printf("%d\n", i);
    }
    return NULL;
}

int main() {
    pthread_t t1, t2;

    // Création des threads
    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);

    // Attente de la fin des threads
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;
}
```

2. Mise en œuvre du sémaphore

Voici un exemple de pseudo-code pour résoudre le problème des philosophes à l'aide de sémaphores. Chaque philosophe a besoin de deux fourchettes pour manger, mais il n'y a que cinq fourchettes sur la table.

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

#define N 5 // Nombre de philosophes et de fourchettes

sem_t forks[N]; // Sémaphores pour les fourchettes

// Fonction pour un philosophe
void* philosopher(void* arg) {
    int id = *(int*)arg;

    while (1) {
        // Penser
        printf("Philosophe %d pense\n", id);

        // Prendre les fourchettes
        sem_wait(&forks[id]); // Prend la fourchette à gauche
        sem_wait(&forks[(id + 1) % N]); // Prend la fourchette à droite
```

```c
        // Manger
        printf("Philosophe %d mange\n", id);

        // Relâcher les fourchettes
        sem_post(&forks[id]); // Relâche la fourchette à gauche
        sem_post(&forks[(id + 1) % N]); // Relâche la fourchette à droite
    }
    return NULL;
}

int main() {
    pthread_t philosophers[N];
    int ids[N];

    // Initialisation des sémaphores
    for (int i = 0; i < N; i++) {
        sem_init(&forks[i], 0, 1); // Chaque fourchette est initialisée à 1
    }

    // Création des threads pour chaque philosophe
    for (int i = 0; i < N; i++) {
        ids[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher, &ids[i]);
    }

    // Attente de la fin des threads (jamais atteint dans cet exemple)
    for (int i = 0; i < N; i++) {
        pthread_join(philosophers[i], NULL);
    }

    return 0;
}
```