CS 347 (M)
TAKE HOME QUIZ - 3

Mantri Krishna Sri Ipsit
180070032

PROBLEM STATEMENT

- Consider two threads A and B that perform two operations each. Let the operations of thread A be A1 and A2; let the operations of thread B be B1 and B2. We require that threads A and B each perform their first operation before either can proceed to the second operation. That is, we require that A1 be run before B2 and B1 before A2

Approaches

- Semaphores
  - A1Done
  - B1Done
- Cond. Variables
  - A1Done
  - B1Done

Variables Used

- Boolean Variables
  - a1
  - a1
- Locks
  - mutex

Using Semaphores

Incorrect Solution

- sem A1Done = 0
- // Thread A
- A1
- down(B1Done)
- up(A1Done)
- A2

- Sem B1Done = 0
- // Thread B
- B1
- down(A1Done)
- up(B1Done)
- B2

Deadlock Condition!

Using Semaphores

- sem A1Done = 0
- // Thread A
- A1
- down(B1Done)
- up(A1Done)
- A2

- Sem B1Done = 0
- // Thread B
- B1
- up(B1Done)
- down(A1Done)
- B2

Correct Solution

Using Conditional Variables

Incorrect Solution

- // Thread A
- A1
- acquire(mutex)
- while(b1 == 0)
  - wait(B1Done, mutex)
- a1 = 1
- signal(A1Done)
- release(mutex)
- A2

- // Thread B
- B1
- acquire(mutex)
- while(a1 == 0)
  - wait(A1Done, mutex)
- b1 = 1
- signal(B1Done)
- release(mutex)
- B2

Deadlock Condition!

Using Conditional Variables

Correct Solution

- // Thread A
- A1
- acquire(mutex)
- while(b1 == 0)
  - wait(B1Done, mutex)
- a1 = 1
- signal(A1Done)
- release(mutex)
- A2

- // Thread B
- B1
- acquire(mutex)
- b1 = 1
- signal(B1Done)
- while(a1 == 0)
  - wait(A1Done, mutex)
- release(mutex)
- B2

Code

Semaphores

```c
#include <stdio.h>
#include "zemaphore.h"

zem_t A1Done, B1Done;

void *threadA(void *args)
{
    printf("A1\n");
    zem_down(&B1Done);
    zem_up(&A1Done);
    printf("A2\n");
    return 0;
}

void *threadB(void *args)
{
    printf("B1\n");
    zem_up(&B1Done); // correct
    zem_down(&A1Done);
    // zem_up(&B1Done); // incorrect
    printf("B2\n");
    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    zem_init(&A1Done, 0);
    zem_init(&B1Done, 0);
    pthread_t A;
    pthread_t B;
    pthread_create(&A, NULL, threadA, NULL);
    pthread_create(&B, NULL, threadB, NULL);

    pthread_join(A, NULL);
    pthread_join(B, NULL);

    return 0;
}
```

Code

Conditional Variables

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

pthread_mutex_t lock;
pthread_cond_t A1Done, B1Done;
int A1, B1;

void *threadA(void *args)
{
    printf("A1\n");
    pthread_mutex_lock(&lock);
    while (B1 == 0)
    {
        pthread_cond_wait(&B1Done, &lock);
    }
    A1 = 1;
    pthread_cond_signal(&A1Done);
    pthread_mutex_unlock(&lock);
    printf("A2\n");
    return 0;
}
```

```c
void *threadB(void *args)
{
    printf("B1\n");
    pthread_mutex_lock(&lock);
    B1 = 1;
    pthread_cond_signal(&B1Done);
    while (A1 == 0)
    {
        pthread_cond_wait(&A1Done, &lock);
    }
    // B1 = 1;      // incorrect
    // pthread_cond_signal(&B1Done); // incorrect
    printf("B2\n");
    return 0;
}
int main(int argc, char *argv[])
{
    pthread_t A;
    pthread_t B;
    pthread_create(&A, NULL, threadA, NULL);
    pthread_create(&B, NULL, threadB, NULL);

    pthread_join(A, NULL);
    pthread_join(B, NULL);

    return 0;
}
```