Operating Systems – Lab 6

Topic: Thread Synchronization using Mutex & Condition Variables

Threads and Race Conditions: Threads share the same memory. When multiple threads modify shared data concurrently, race conditions occur leading to inconsistent results.

Mutex (Mutual Exclusion): Mutex ensures that only one thread accesses a shared resource at a time using lock and unlock operations.

Condition Variables: Used to make threads wait for specific conditions to occur, enabling synchronized execution without busy waiting.

Q1. Race Condition and Mutex Lock

Aim: Demonstrate race condition and fix using mutex.

Without Lock:

```
#include <stdio.h>
#include <pthread.h>
#define THREADS 10
#define INCREMENTS 1000
int counter = 0;
void* increment(void* arg) {
    for(int i = 0; i < INCREMENTS; i++) counter++;
    return NULL;
}
int main() {
    pthread_t tid[THREADS];
    for(int i = 0; i < THREADS; i++) pthread_create(&tid[i], NULL, increment, NULL);
    for(int i = 0; i < THREADS; i++) pthread_join(tid[i], NULL);
    printf("Final counter (without lock): %d\n", counter);
    return 0;
}
```

With Lock:

```
#include <stdio.h>
#include <pthread.h>
#define THREADS 10
#define INCREMENTS 1000
int counter = 0;
pthread_mutex_t lock;
void* increment(void* arg) {
    for(int i = 0; i < INCREMENTS; i++) {
        pthread_mutex_lock(&lock);
        counter++;
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}
int main() {
    pthread_t tid[THREADS];
    pthread_mutex_init(&lock, NULL);
    for(int i = 0; i < THREADS; i++) pthread_create(&tid[i], NULL, increment, NULL);
```

```
    for(int i = 0; i < THREADS; i++) pthread_join(tid[i], NULL);
    printf("Final counter (with lock): %d\n", counter);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Expected Output:

Final counter (without lock): less than 10000

Final counter (with lock): 10000

Q2. Thread Identification and Synchronization

```
#include <stdio.h>
#include <pthread.h>
#define N 10
void* print_thread(void* arg) {
    int id = *(int*)arg;
    printf("I am thread %d\n", id);
    return NULL;
}
int main() {
    pthread_t threads[N];
    int ids[N];
    for(int i = 0; i < N; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, print_thread, &ids[i]);
    }
    for(int i = 0; i < N; i++) pthread_join(threads[i], NULL);
    printf("I am the main thread\n");
    return 0;
}
```

Expected Output:

I am thread 0

I am thread 1

...

I am the main thread

Q3. Sequential Thread Printing using Condition Variables

```
#include <stdio.h>
#include <pthread.h>
#define N 3
pthread_mutex_t lock;
pthread_cond_t cond;
int turn = 0;
void* print_thread(void* arg) {
    int id = *(int*)arg;
    while(1) {
        pthread_mutex_lock(&lock);
        while(turn != id) pthread_cond_wait(&cond, &lock);
        printf("I am thread %d\n", id);
        turn = (turn + 1) % N;
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&lock);
    }
```

```
}
int main() {
    pthread_t threads[N];
    int ids[N];
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&cond, NULL);
    for(int i = 0; i < N; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, print_thread, &ids[i]);
    }
    for(int i = 0; i < N; i++) pthread_join(threads[i], NULL);
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&cond);
    return 0;
}
```

Expected Output:

I am thread 0

I am thread 1

I am thread 2

Repeated in sequence