

To get familiar with Multithreading with Pthread Library:

1. Write a simple program to create a thread.

```
1 #include <stdio.h>
2 #include <pthread.h>
3
4 // Function to be executed in the thread
5 void *thread_function(void *arg) {
6     printf("This is running in a thread.\n");
7     pthread_exit(NULL);
8 }
9
10 int main() {
11     pthread_t thread;
12     int result;
13     // Create a thread
14     result = pthread_create(&thread, NULL, thread_function, NULL);
15     if (result != 0) {
16         printf("Failed to create thread.\n");
17         return 1;
18     }
19     // Main program continues to execute while the thread is running
20     printf("This is the main program.\n");
21     // Wait for the thread to finish
22     pthread_join(thread, NULL);
23     // Execution continues after the thread has finished
24     printf("Thread execution has completed.\n");
25     return 0;
26 }
```

```
arun@arun-ubuntu:~$ gedit question1.c
arun@arun-ubuntu:~$ gcc question1.c -o question1
arun@arun-ubuntu:~$ ./question1
This is the main program.
This is running in a thread.
Thread execution has completed.
```

2. Write a program to create two threads such that one thread will be calculating sum of the element of an array and other thread will be performing multiplication of the element of array.

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #define SIZE 5
4 int array[SIZE] = {1, 2, 3, 4, 5};
5 int sum_result = 0;
6 int mul_result = 1;
7 // Function to calculate the sum of array elements
8 void *sum_thread(void *arg) {
9     int i;
10    for (i = 0; i < SIZE; i++) {
11        sum_result += array[i];
12    }
13    pthread_exit(NULL);
14 }
15 // Function to calculate the multiplication of array elements
16 void *mul_thread(void *arg) {
17     int i;
18    for (i = 0; i < SIZE; i++) {
19        mul_result *= array[i];
20    }
21    pthread_exit(NULL);
22 }
23 int main() {
24    pthread_t sum_thread_id, mul_thread_id;
25    // Create the sum thread
26    if (pthread_create(&sum_thread_id, NULL, sum_thread, NULL) != 0) {
27        printf("Failed to create sum thread.\n");
28        return 1;
29    }
30    pthread_create(&mul_thread_id, NULL, mul_thread, NULL);
31    pthread_join(sum_thread_id, NULL);
32    pthread_join(mul_thread_id, NULL);
33    printf("Sum: %d\n", sum_result);
34    printf("Multiplication: %d\n", mul_result);
35    return 0;
36 }
```

```
arun@arun-ubuntu:~$ gcc question2.c -o question2
arun@arun-ubuntu:~$ ./question2
Sum: 15
Multiplication: 120
```

3. Write a program to implement producer consumer problem using semaphore and multithreading.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5 #define BUFFER_SIZE 5
6 #define NUM_PRODUCERS 2
7 #define NUM_CONSUMERS 2
8 #define NUM_ITEMS 10
9 int buffer[BUFFER_SIZE];
10 int in = 0;
11 int out = 0;
12 int counter = 0;
13 sem_t mutex, empty, full;
14 void *producer(void *arg) {
15     int producer_id = *(int *)arg;
16     int item;
17     for (int i = 0; i < NUM_ITEMS; i++) {
18         item = rand() % 100; // Generate a random item
19         sem_wait(&empty); // Wait for an empty slot in the buffer
20         sem_wait(&mutex); // Obtain exclusive access to the buffer
21         // Produce item and add it to the buffer
22         buffer[in] = item;
23         printf("Producer %d produced item: %d\n", producer_id, item);
24         in = (in + 1) % BUFFER_SIZE;
25         counter++;
26         sem_post(&mutex); // Release exclusive access to the buffer
27         sem_post(&full); // Signal that there is an item in the buffer
28     }
29 }
30 void *consumer(void *arg) {
31     int consumer_id = *(int *)arg;
32     int item;
33     for (int i = 0; i < NUM_ITEMS; i++) {
34         sem_wait(&full); // Wait for an item in the buffer
35         sem_wait(&mutex); // Obtain exclusive access to the buffer
36         // Consume item from the buffer
37         item = buffer[out];
38         printf("Consumer %d consumed item: %d\n", consumer_id, item);
39         out = (out + 1) % BUFFER_SIZE;
40         counter--;
41         sem_post(&mutex); // Release exclusive access to the buffer
42         sem_post(&empty); // Signal that there is an empty slot in the buffer
43     }
44     pthread_exit(NULL);
45 }
46 int main() {
47     pthread_t producers[NUM_PRODUCERS];
48     pthread_t consumers[NUM_CONSUMERS];
49     int producer_ids[NUM_PRODUCERS];
50     int consumer_ids[NUM_CONSUMERS];
51     sem_init(&mutex, 0, 1);
52     sem_init(&empty, 0, BUFFER_SIZE);
53     sem_init(&full, 0, 0);
54     for (int i = 0; i < NUM_PRODUCERS; i++) {
55         producer_ids[i] = i + 1;
56         if (pthread_create(&producers[i], NULL, producer, &producer_ids[i]) != 0)
57         {
58             printf("Failed to create producer thread %d.\n", i);
59             return 1;
60         }
61     }
62     for (int i = 0; i < NUM_CONSUMERS; i++) {
63         consumer_ids[i] = i + 1;
64         if (pthread_create(&consumers[i], NULL, consumer, &consumer_ids[i]) != 0)
65         {
66             printf("Failed to create consumer thread %d.\n", i);
67             return 1;
68         }
69     }
70     for (int i = 0; i < NUM_ITEMS; i++) {
71         printf("Item %d produced\n", i);
72     }
73     return 0;
74 }
```

```
arun@arun-ubuntu:~$ gcc question3.c -o question3
arun@arun-ubuntu:~$ ./question3
Producer 1 produced item: 83
Producer 1 produced item: 77
Producer 1 produced item: 15
Producer 1 produced item: 93
Producer 2 produced item: 86
Consumer 2 consumed item: 83
Consumer 2 consumed item: 77
Consumer 2 consumed item: 15
Consumer 2 consumed item: 93
Producer 2 produced item: 86
Producer 2 produced item: 92
Producer 2 produced item: 49
Consumer 1 consumed item: 86
Consumer 1 consumed item: 86
Consumer 1 consumed item: 92
Consumer 2 consumed item: 49
Producer 2 produced item: 21
Producer 2 produced item: 62
Producer 2 produced item: 27
Producer 2 produced item: 90
```