To get familiar with Multithreading with Pthread Library:

1. Write a simple program to create a thread.

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

// Function to be executed in the thread
void *thread_function(void *arg) {
    printf("This is running in a thread.\n");
    pthread_exit(NULL);
}

int main() {
    pthread_t thread;
    int result;
    // Create a thread
    result = pthread_create(&thread, NULL, thread_function, NULL);
    if (result != 0) {
        printf("Failed to create thread.\n");
        return 1;
    }
    // Main program continues to execute while the thread is running
    printf("This is the main program.\n");
    // Wait for the thread to finish
    pthread_join(thread, NULL);
    // Execution continues after the thread has finished
    printf("Thread execution has completed.\n");
    return 0;
}
```

```
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.

```c
#include <stdio.h>
#include <pthread.h>
#define SIZE 5
int array[SIZE] = {1, 2, 3, 4, 5};
int sum_result = 0;
int mul_result = 1;
// Function to calculate the sum of array elements
void *sum_thread(void *arg) {
    int i;
    for (i = 0; i < SIZE; i++) {
        sum_result += array[i];
    }
    pthread_exit(NULL);
}
// Function to calculate the multiplication of array elements
void *mul_thread(void *arg) {
    int i;
    for (i = 0; i < SIZE; i++) {
        mul_result *= array[i];
    }
    pthread_exit(NULL);
}
int main() {
    pthread_t sum_thread_id, mul_thread_id;
    // Create the sum thread
    if (pthread_create(&sum_thread_id, NULL, sum_thread, NULL) != 0) {
        printf("Failed to create sum thread.\n");
        return 1;
```

```
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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define BUFFER_SIZE 5
#define NUM_PRODUCERS 2
#define NUM_CONSUMERS 2
#define NUM_ITEMS 10
int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0;
sem_t mutex, empty, full;
void *producer(void *arg) {
    int producer_id = *(int *)arg;
    int item;
    for (int i = 0; i < NUM_ITEMS; i++) {
        item = rand() % 100; // Generate a random item
        sem_wait(&empty); // Wait for an empty slot in the buffer
        sem_wait(&mutex); // Obtain exclusive access to the buffer
        // Produce item and add it to the buffer
        buffer[in] = item;
        printf("Producer %d produced item: %d\n", producer_id, item);
        in = (in + 1) % BUFFER_SIZE;
        counter++;
        sem_post(&mutex); // Release exclusive access to the buffer
        sem_post(&full); // Signal that there is an item in the buffer
    }

    pthread_exit(NULL);
void *consumer(void *arg) {
    int consumer_id = *(int *)arg;
    int item;
    for (int i = 0; i < NUM_ITEMS; i++) {
        sem_wait(&full); // Wait for an item in the buffer
        sem_wait(&mutex); // Obtain exclusive access to the buffer
        // Consume item from the buffer
        item = buffer[out];
        printf("Consumer %d consumed item: %d\n", consumer_id, item);
        out = (out + 1) % BUFFER_SIZE;
        counter--;
        sem_post(&mutex); // Release exclusive access to the buffer
        sem_post(&empty); // Signal that there is an empty slot in the buffer
    }
    pthread_exit(NULL);
}
int main() {
    pthread_t producers[NUM_PRODUCERS];
    pthread_t consumers[NUM_CONSUMERS];
    int producer_ids[NUM_PRODUCERS];
    int consumer_ids[NUM_CONSUMERS];
    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    for (int i = 0; i < NUM_PRODUCERS; i++) {
        producer_ids[i] = i + 1;
        if (pthread_create(&producers[i], NULL, producer, &producer_ids[i]) != 0)
{
            printf("Failed to create producer thread %d.\n", i);
            return 1;
        }
```

```
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
```