Report: Multithreading in C using POSIX Threads

Name: Abrar Khatib Lajim

ID:2022-3-60-043

Course:CSE325 (1)

Program 01: Simple Hello World Using Threads (Problem-1)

Objective:
Create a basic thread and display a message from it.

Learnings:

➔ How to define and start a thread using pthread_create.

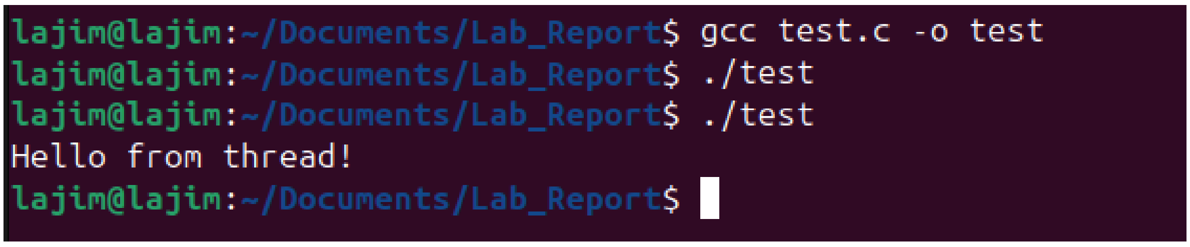
➔ Basic structure of a multithreaded C program.

➔ Introduction to using void* as a thread function argument.

Code:

```
// File: thread_hello.c

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

// Thread function
void* hello(void* arg) {
    printf("Hello from thread!\n");
    return NULL;
}

int main() {
    pthread_t thread;

    // Create a new thread
    if (pthread_create(&thread, NULL, hello, NULL) != 0) {
        perror("Failed to create thread");
        return 1;}
```

```c
    // Wait for the thread to finish
    pthread_join(thread, NULL);
    return 0;}
```



```
lajim@lajim:~/Documents/Lab_Report$ gcc test.c -o test
lajim@lajim:~/Documents/Lab_Report$ ./test
lajim@lajim:~/Documents/Lab_Report$ ./test
Hello from thread!
lajim@lajim:~/Documents/Lab_Report$
```

Program 02: Multiple Threads with Messages (Problem-2)

Objective:

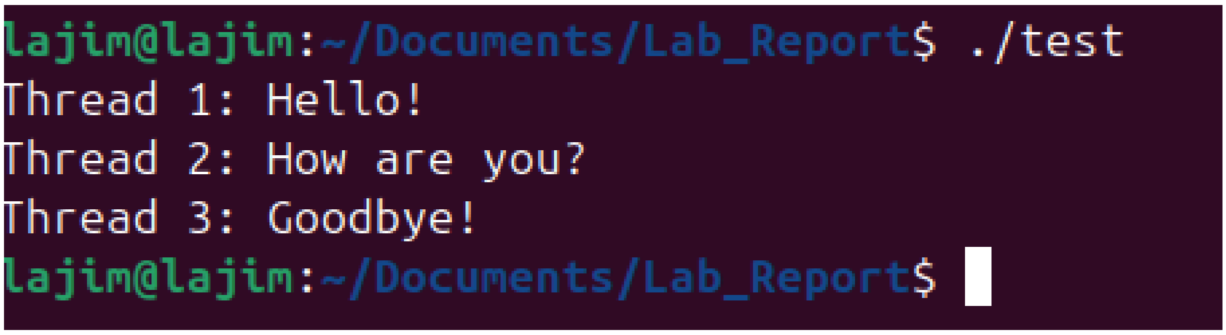
Start three threads, each printing a different message, and ensure the main thread waits for all of them.

Learnings:

- ➔ How to pass a string to each thread.
- ➔ Use of pthread_join() to wait for thread completion.
- ➔ Importance of synchronization to avoid race conditions or early exits.

Code:

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

void* printMessage(void* arg) {
    char* msg = (char*)arg;
    printf("%s\n", msg);
    return NULL;
}

int main() {
    pthread_t t1, t2, t3;

    char m1[] = "Thread 1: Hello!";
    char m2[] = "Thread 2: How are you?";
    char m3[] = "Thread 3: Goodbye!";

    pthread_create(&t1, NULL, printMessage, m1);
    pthread_create(&t2, NULL, printMessage, m2);
    pthread_create(&t3, NULL, printMessage, m3);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    return 0;}
```



```
lajim@lajim:~/Documents/Lab_Report$ ./test
Thread 1: Hello!
Thread 2: How are you?
Thread 3: Goodbye!
lajim@lajim:~/Documents/Lab_Report$ 
```

Program 03: Producer-Consumer with Shared Buffer (Condition Variables)

(Problem-3)

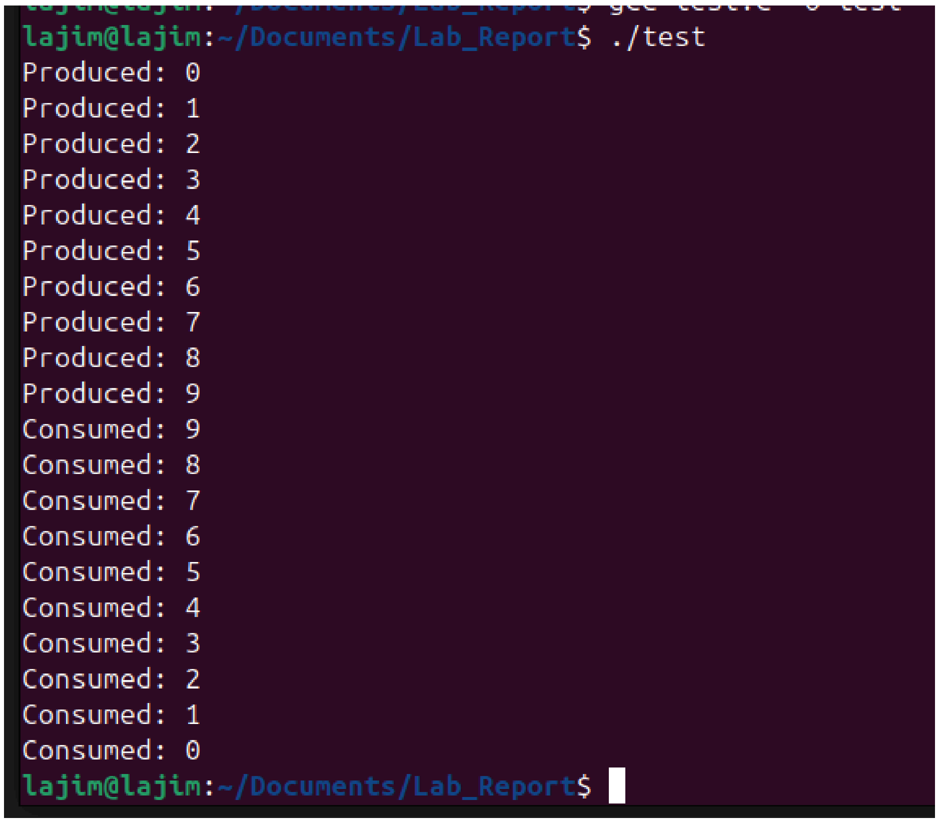
Objective:
 Implement the classic producer-consumer problem using threads, a shared buffer, mutex, and condition variables.

Learnings:

- Understanding the producer-consumer problem.
- Use of pthread_mutex_t and pthread_cond_t to prevent race conditions.
- Buffer synchronization to avoid overproduction or underconsumption.
- How threads signal each other using pthread_cond_signal() and pthread_cond_wait().

Code:

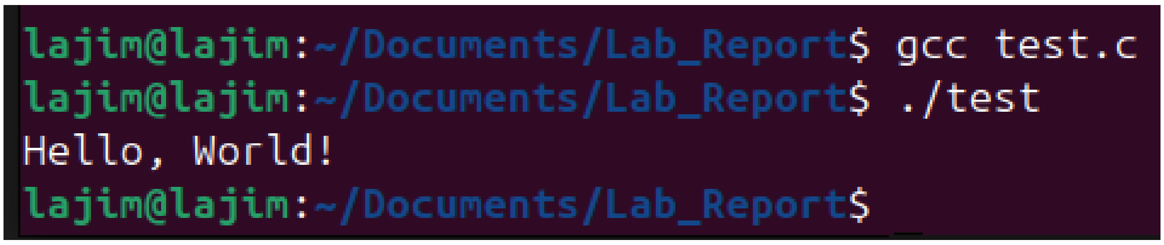
```
#include <pthread.h>
#include <stdio.h>

#define SIZE 10
int buffer[SIZE], count = 0;

pthread_mutex_t mutex;
pthread_cond_t cond_prod, cond_cons;

void* producer(void* arg) {
    for (int i = 0; i < 10; i++) {
        pthread_mutex_lock(&mutex);
        while (count == SIZE)
            pthread_cond_wait(&cond_prod, &mutex);
        buffer[count++] = i;
        printf("Produced: %d\n", i);
        pthread_cond_signal(&cond_cons);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

void* consumer(void* arg) {
    for (int i = 0; i < 10; i++) {
        pthread_mutex_lock(&mutex);
        while (count == 0)
            pthread_cond_wait(&cond_cons, &mutex);
        int item = buffer[--count];
        printf("Consumed: %d\n", item);
        pthread_cond_signal(&cond_prod);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;}
int main() {
    pthread_t prod, cons;
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond_prod, NULL);
    pthread_cond_init(&cond_cons, NULL);
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_prod);
    pthread_cond_destroy(&cond_cons);
    return 0;}
```



```
lajim@lajim:~/Documents/Lab_Report$ ./test
Produced: 0
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Produced: 5
Produced: 6
Produced: 7
Produced: 8
Produced: 9
Consumed: 9
Consumed: 8
Consumed: 7
Consumed: 6
Consumed: 5
Consumed: 4
Consumed: 3
Consumed: 2
Consumed: 1
Consumed: 0
lajim@lajim:~/Documents/Lab_Report$
```

Program-4 (Try-1) Print "Hello, World!" from a thread

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

void* helloThread(void* arg) {
    printf("Hello, World!\n");
    return NULL;
}

int main() {
    pthread_t thread_id;
    pthread_create(&thread_id, NULL, helloThread, NULL);
    pthread_join(thread_id, NULL);
    return 0;
}
```

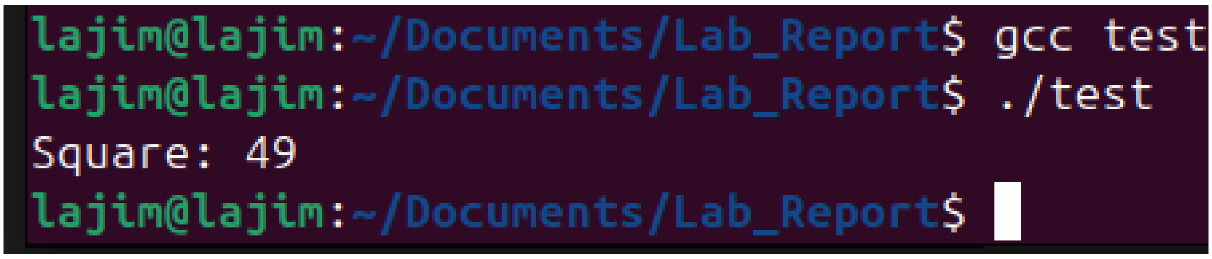


```
lajim@lajim:~/Documents/Lab_Report$ gcc test.c
lajim@lajim:~/Documents/Lab_Report$ ./test
Hello, World!
lajim@lajim:~/Documents/Lab_Report$ _
```

Program-5 (Try-2) Pass an integer and print its square

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

void* squareThread(void* arg) {
    int num = *((int*)arg);
    printf("Square: %d\n", num * num);
    return NULL;
}
```

```c

int main() {
    pthread_t thread_id;
    int value = 7; // example integer to pass
    pthread_create(&thread_id, NULL, squareThread, &value);
    pthread_join(thread_id, NULL);
    return 0;
}
```



```
lajim@lajim:~/Documents/Lab_Report$ gcc test
lajim@lajim:~/Documents/Lab_Report$ ./test
Square: 49
lajim@lajim:~/Documents/Lab_Report$
```

Program 06: Modify Producer-Consumer to Include Integer Argument (Try 03)

Objective:

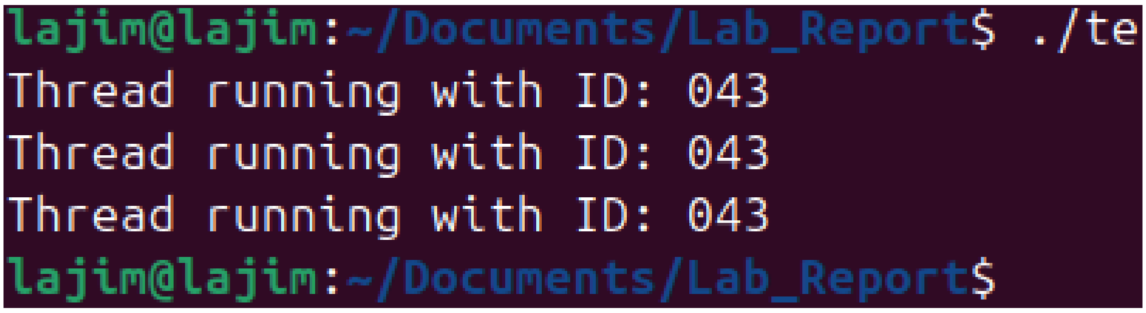
Pass an integer argument (e.g., your ID digits like 043) to each thread and print it.

Learnings:

- How to pass integers to threads.
- Typecasting void* to int* and back.
- Using thread arguments for unique identifiers or custom logic.

Code:

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

void* printID(void* arg) {
    int id = *((int*)arg);
    printf("Thread running with ID: %03d\n", id);
    return NULL;
}

int main() {
    pthread_t t1, t2, t3;
    int id = 43;

    pthread_create(&t1, NULL, printID, &id);
    pthread_create(&t2, NULL, printID, &id);
    pthread_create(&t3, NULL, printID, &id);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    return 0;
}
```



```
lajim@lajim:~/Documents/Lab_Report$ ./te
Thread running with ID: 043
Thread running with ID: 043
Thread running with ID: 043
lajim@lajim:~/Documents/Lab_Report$
```

Program 07: Pass a Struct to a Thread (Person's Name and Age)

(Problem 15)

Objective:
Pass a structure containing name and age to a thread, and print these values inside the thread.

Learnings:

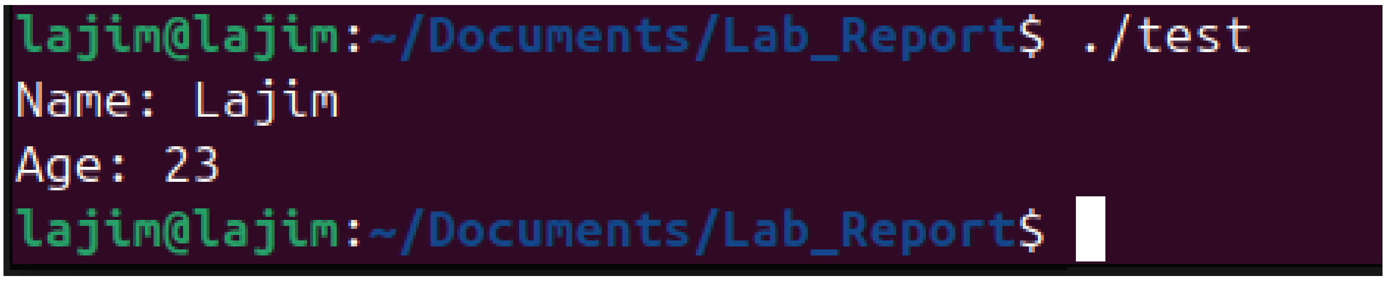
- ➔ Defining and using C struct.
- ➔ Passing complex data types to threads using pointers.
- ➔ Typecasting from void* to a custom struct.

➔ Using structs in multithreaded programs.

➔ Struct pointer casting in threads.

Code:

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

typedef struct {
    char name[50];
    int age;
} Person;


void* printPersonInfo(void* arg) {
    Person* p = (Person*) arg;
    printf("Name: %s\n", p->name);
    printf("Age: %d\n", p->age);
    return NULL;
}

int main() {
    pthread_t thread;
    Person person = {"Lajim", 23};


    if (pthread_create(&thread, NULL, printPersonInfo, &person)) {
        fprintf(stderr, "Error creating thread\n");
        return 1;
    }


    if (pthread_join(thread, NULL)) {
        fprintf(stderr, "Error joining thread\n");
        return 2;
    }

    return 0;
}
```



```
lajim@lajim:~/Documents/Lab_Report$ ./test
Name: Lajim
Age: 23
lajim@lajim:~/Documents/Lab_Report$ 
```

Overall Concepts Covered:

Topic	Description
pthread_create()	Creating threads.
pthread_join()	Waiting for threads to finish.
pthread_mutex_t	Locking shared resources.
pthread_cond_t	Signaling/waiting using condition variables.
Argument Passing	Strings, integers, and structs.
Shared Resource Handling	Synchronizing access to buffers.
Real-world Problems	Implementing producer-consumer pattern.