# NATIONAL TEXTILE

# UNIVERSITY

## DEPARTMENT OF COMPUTER SCIENCE

## Subject

Operating System

## SUBMITTED BY:

Fatima Waseem: 23-NTU-CS1155

**SECTION SE: 5th (A)**

## SUBMITTED TO:

Sir Nasir

# Lab 5

**Program 1: Creating a Simple Thread**

**Objective:** Create a thread and print messages from both main thread and new thread.

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
// Thread function - this will run in the new thread
void* thread_function(void* arg) {
printf("Hello from the new thread!\n");
printf("Thread ID: %lu\n", pthread_self());
return NULL;
}
int main() {
pthread_t thread_id;
printf("Main thread starting...\n");
printf("Main Thread ID: %lu\n", pthread_self());
// Create a new thread
pthread_create(&thread_id, NULL, thread_function, NULL);

// Wait for the thread to finish
pthread_join(thread_id, NULL);
printf("Main thread exiting...\n");
return 0;
}
```

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                bash - Lab5

fatima@DESKTOP-3BA3T21:~/Lab5$ gcc task1.c -o output
fatima@DESKTOP-3BA3T21:~/Lab5$ ./output
Main thread starting...
Main Thread ID: 123276680759104
Hello from the new thread!
Thread ID: 123276678395584
Main thread exiting...
```

**Program 2: Passing Arguments to Threads**

**Objective:** Pass data to a thread function.

Code:

```
#include <stdio.h>
 #include <pthread.h>
 void* print_number(void* arg) {
// We know that we've passed an integer pointer
 float num = *(float*)arg; // Cast void* back to int*
 printf("Thread received number: %f\n", num);
 printf("CGPA: %f\n", num * 2);
 return NULL;
 }
 int main() {
 pthread_t thread_id;
 float number = 3.6;
 printf("Creating thread with argument: %f\n", number);
 // Pass address of 'number' to thread
 pthread_create(&thread_id, NULL, print_number, &number);
 pthread_join(thread_id, NULL);
 printf("Main thread done.\n");
 return 0;
 }
```

Output:

```
• fatima@DESKTOP-3BA3T21:~/Lab5$ gcc task2.c
• fatima@DESKTOP-3BA3T21:~/Lab5$ ./a.out
  Creating thread with argument: 3.600000
  Thread received number: 3.600000
  Square: 7.200000
  Main thread done.
```

**Program 3: Passing Multiple Data**

```
#include <stdio.h>
#include <pthread.h>
typedef struct {
float id;
char* message;
} ThreadData;
void* printData(void* arg) {
ThreadData* data = (ThreadData*)arg;
printf("CGPA %f Name: %s\n", data->id, data->message);
return NULL;
}
int main() {
pthread_t t1;
ThreadData data1 = {3.61, "Fatima"};

pthread_create(&t1, NULL, printData, &data1);
//pthread_create(&t2, NULL, printData, &data2);
pthread_join(t1, NULL);
//pthread_join(t2, NULL);
printf("All threads done.\n");
return 0;
}
```

**Output:**

```
fatima@DESKTOP-3BA3T21:~/Lab5$ ./a.out
CGPA 3.610000 Name: Fatima
All threads done.
```

**Program 4: Thread Return Values**

**Objective:** Get return values from threads.

Code:

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

```c
#include <stdlib.h>
void* calculate_sum(void* arg) {
int n = *(int*)arg;
int* result = malloc(sizeof(int)); // Allocate memory for
result
*result = 0;
for (int i = 1; i <= n; i++) {
*result += i;
}
printf("Thread calculated sum of 1 to %d = %d\n", n, *result);
return (void*)result; // Return the result
}
int main() {
pthread_t thread_id;
int n = 100;
void* sum;
pthread_create(&thread_id, NULL, calculate_sum, &n);
// Get the return value from thread
pthread_join(thread_id, &sum);
printf("Main received result: %d\n", *(int*)sum);
free(sum); // Don't forget to free allocated memory
return 0;
}
```

Output:

```
• fatima@DESKTOP-3BA3T21:~/Lab5$ gcc task4.c
• fatima@DESKTOP-3BA3T21:~/Lab5$ ./a.out
  Thread calculated sum of 1 to 100 = 5050
  Main received result: 5050
```

**Program 5: Creating and Running Multiple Threads**

**Objective:**

Create multiple threads that execute independently and print messages concurrently.

Code:

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

```c
#include <pthread.h>
#include <unistd.h>
void* worker(void* arg) {
int thread_num = *(int*)arg;
printf("Thread %d: Starting task...\n", thread_num);
sleep(1); // Simulate some work
printf("Thread %d: Task completed!\n", thread_num);
return NULL;
}
int main() {
pthread_t threads[3];
int thread_ids[3];
for (int i = 0; i < 3; i++) {
thread_ids[i] = i + 1;
pthread_create(&threads[i], NULL, worker, &thread_ids[i]);
}
for (int i = 0; i < 3; i++) {
    pthread_join(threads[i], NULL);
}
printf("Main thread: All threads have finished.\n");
return 0;
}
```

Output:

```
• fatima@DESKTOP-3BA3T21:~/Lab5$ gcc task5.c
• fatima@DESKTOP-3BA3T21:~/Lab5$ ./a.out
  Thread 1: Starting task...
  Thread 2: Starting task...
  Thread 3: Starting task...
  Thread 1: Task completed!
  Thread 3: Task completed!
  Thread 2: Task completed!
  Main thread: All threads have finished.
```

**Program 2: Demonstrating a Race Condition**

**Objective:** What happens when multiple threads modify a shared variable **without synchronization**.

Code:

```
#include <stdio.h>
#include <pthread.h>
int counter = 0; // Shared variable
void* increment(void* arg) {
for (int i = 0; i < 100000; i++) {
counter++; // Not thread-safe
}
return NULL;
}
int main() {
pthread_t t1, t2;
pthread_create(&t1, NULL, increment, NULL);
pthread_create(&t2, NULL, increment, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Expected counter value: 200000\n");
printf("Actual counter value: %d\n", counter);
return 0;
}
```

Output:

```
fatima@DESKTOP-3BA3T21:~/Lab5$ gcc task6.c
fatima@DESKTOP-3BA3T21:~/Lab5$ ./a.out
Expected counter value: 200000
Actual counter value: 200000
```