



## Laboratory Manual

*for*

### Operating Systems Lab

(CL-2006)

Course Instructor	Ms. Rubab Anam
Lab Instructor	Muhammad Faheem
Section	BCS-4F
Semester	Spring 2025

Department of Computer Science

FAST-NU, Lahore, Pakistan

## Threads

### Pthreads:

```
pthread_create(pthread_t*, NULL, void*, void*)
```

- First Parameter is pointer of thread ID it should be different for all threads.
- Second Parameter is used to change stack size of thread. Null means use default size.
- Third parameter is address of function which we are going to use as thread.
- Forth parameter is argument to function.

`pthread_join( pthread_t , void** )`

- Pthread join is used in main program to wait for the end of a particular thread.
- First parameter is Thread ID of particular thread.
- Second Parameter is used to catch return value from thread.

### **Thread Library:**

- POSIX Pthreads
- Two general strategies for creating multiple threads.

#### a) Asynchronous threading:

- Parent and child threads run independently of each other
- Typically little data sharing between threads

#### b) Synchronous threading:

- Parent thread waits for all of its children to terminate
- Children threads run concurrently
- Significant data sharing **Getting Thread's ID in itself:**

`pthread_self()`

### **Pthreads Header File:**

- **pthread.h**
  - Each thread has a set of attributes, including stack size and scheduling information
  - In a Pthreads program, separate threads begin execution in a specified function //runner()
  - When a program begins

A single thread of control begins in main()

main() creates a second thread that begins control in the runner() function  Both threads share the global data

### **Compiling multithreads C program:**

```
gcc -lpthread main.c -o main
```

Example:

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

int var = 0;

void *worker(void *param)
{
    var = param;
    pthread_exit(0);
}

int main()
{
    printf("[Before Thread Execution] Var = %d\n", var);

    int value = 10;
    pthread_t threadID;
    pthread_create(&threadID, NULL, worker, (void *)value);

    pthread_join(threadID, NULL);
    printf("[After Thread Execution] Var = %d\n", var);
    return 0;
}
```

Example: Sending and Receiving an array

```

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

void *worker(void *params)
{
    int *numbers = (int *)params;
    numbers[0] += 10;
    numbers[1] += 10;
    printf("New Thread -\t Value[0] = %d Value[1] = %d\n", numbers[0], numbers[1]);
    pthread_exit(numbers);
}

int main()
{
    int values[] = {10, 20};
    pthread_t threadID;

    pthread_create(&threadID, NULL, worker, (void *)values);

    int *updated_values;
    pthread_join(threadID, (void **)&updated_values);

    printf("Main Thread -\t Value[0] = %d Value[1] = %d\n", updated_values[0], updated_values[1]);
    return 0;
}

```

### In Lab Task 1:

- Write a program that creates N number of threads using a loop.
- Print thread ID and process ID in each thread worker.

### In Lab Task 2:

Design a multi-threaded program that performs the summation of square of an integer in a separate thread using the summation function.

- For example, if N were 3, this summation function would represent the summation of squares of integers from 0 to 3, which is 14.

### In Lab Task 3:

Design a **synchronous** multi-threaded application for the following tasks to be executed in an order.

- Write a program which takes some positive integer (let's say **N**) as command line argument
  - It creates a thread, and send N to it as parameter.
  - This thread is a **fibonacciGenerator**. The function returns the generated Fibonacci series until N to the main thread, which prints it on the screen along with the thread ID.
  - This series is to be passed to another thread by the main thread.
  - This new worker has to count the number of even numbers and return the count to the main thread, which prints it along with the 2<sup>nd</sup> thread's id.
  - Then the series is to be passed to a third thread by the main thread.
  - This new worker has to count the number of odd numbers and return the count to the main thread, which prints it along with the 3<sup>rd</sup> thread's id.
  - Finally the sum of this series is calculated by a fourth thread. This thread also returns the sum value to the main thread where it is printed along with the thread ID.

**For Example:**

N = 8

**Output:**

ID = 40, Series: 0 1 1 2 3 5 8 13

ID = 41, Even Numbers: 3

ID = 42, Odd Numbers: 5

ID = 43, Sum: 228

**Evaluations will be in the Lab but you need to submit code within time.**