National Textile University

Department of Computer Science

Subject

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Kashmir Jamshaid

Registration Number

23-NTU-CS-1167

Lab No.

04

Semester

5th

Operating Systems – COC 3071L

SE 5th A – Fall 2025

Lab 4: Introduction to Threads

1. Introduction to Threads

1.1 What is a Thread?

A **thread** is the smallest unit of execution within a process.

- A **process** can have multiple threads running concurrently All
- threads within a process share:
  - Memory space (code, data, heap)
  - File descriptors
  - Process ID
- Each thread has its own:
  - Thread ID (TID)
  - Stack
  - Program counter
  - Register set

Real-world analogy:

- **Process** = A restaurant kitchen
  Threads = Multiple cooks working together in the same kitchen, sharing ingredients and equipment

1.2 Threads vs Processes – Quick Comparison

Feature	Process	Thread
Memory	Separate memory space	Shared memory space
Creation	Expensive (fork)	Lightweight (pthread_create)
Communication	IPC needed (pipes, etc.)	Direct (shared variables)
Context Switch	Slower	Faster
Independence	Fully independent	Dependent on parent process

When to use threads?

- When tasks need to share data frequently
- For parallel execution within the same application
- When you need lightweight concurrency

---

2. POSIX Threads (pthreads) Library

In Linux, we use the **POSIX threads (pthreads)** library for thread programming.

2.1 Compilation Requirements

When compiling programs with threads, you **must** link the pthread library:

```
gcc program c -o program -lpthread
```

The `-lpthread` flag links the pthread library.

---

3. C Programs with Threads

Program 1: Creating a Simple Thread

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

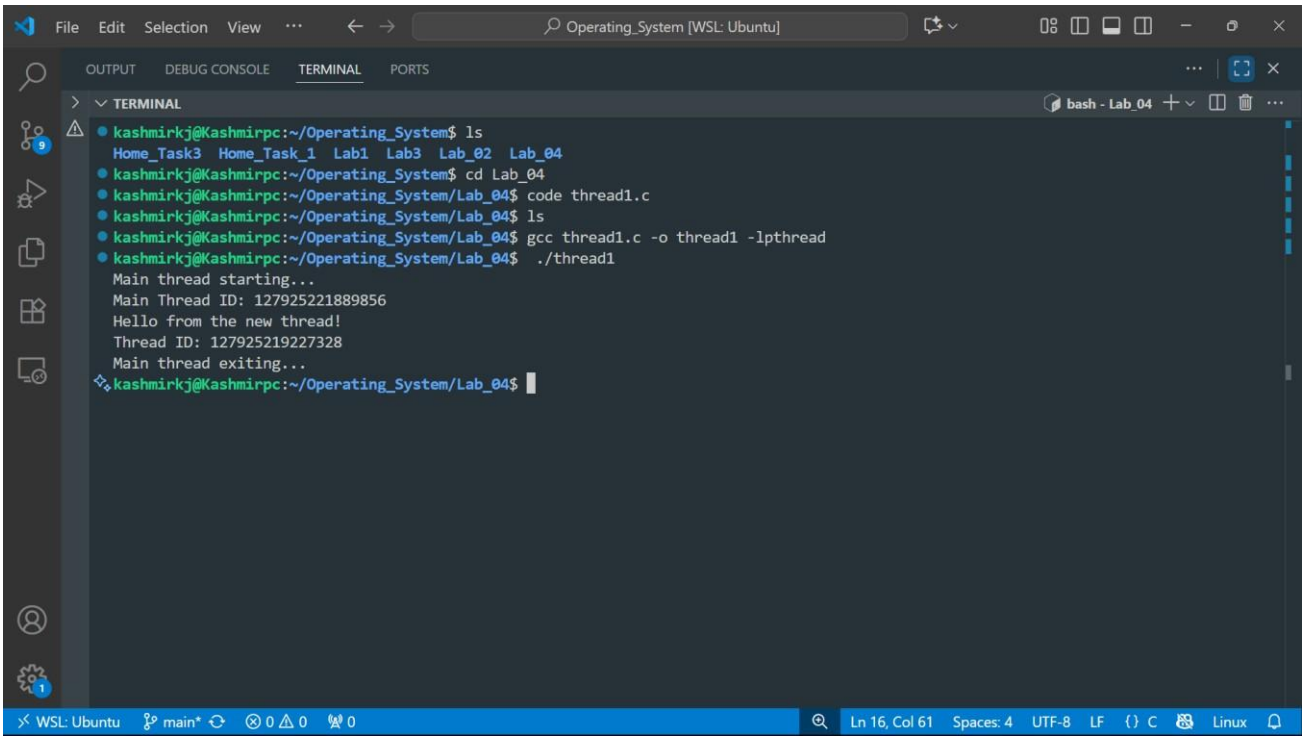
```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

// 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;
}
```

Compile and run:

```
gcc thread1 c -o thread1 -lpthread
./thread1
```

Commands:

1. mkdir Lab_04
2. cd Lab_04
3. code thread1.c (ctrl+s to save the file)
4. gcc thread1.c -o thread1 -lpthread
5. ./thread1



Explanation:

`pthread_t thread_id`

This creates a **variable** to hold the thread's ID (like a file descriptor or process ID).
It's just a handle the OS uses to manage the thread.

pthread_create(&thread_id, NULL, thread_function, NULL)`

Let's decode the four parameters:

Parameter	Type	Meaning
&thread	pthread_t*	Where the new thread ID will be stored
NULL	pthread_attr_t*	Thread attributes (priority, stack size, etc.) — NULL means default

myThread	void* (*start_routine) (void*)	Function to run in the new thread
NULL	void*	Pointer passed to the function for data

- `pthread_join()` → Waits for thread to finish (like `wait()` for processes)
- `pthread_self()` → Returns the thread ID of calling thread

Program 2: Passing Arguments to Threads

Objective: Pass data to a thread function.

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

void* print number(void* arg) {
    // We know that we've passed an integer pointer
    int num = *(int*)arg;  // Cast void* back to int*
    printf("Thread received number: %d\n", num);
    printf("Square: %d\n", num * num);
    return NULL;
}

int main() {
    pthread_t thread_id;
    int number = 42;

    printf("Creating thread with argument: %d\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;
}
```

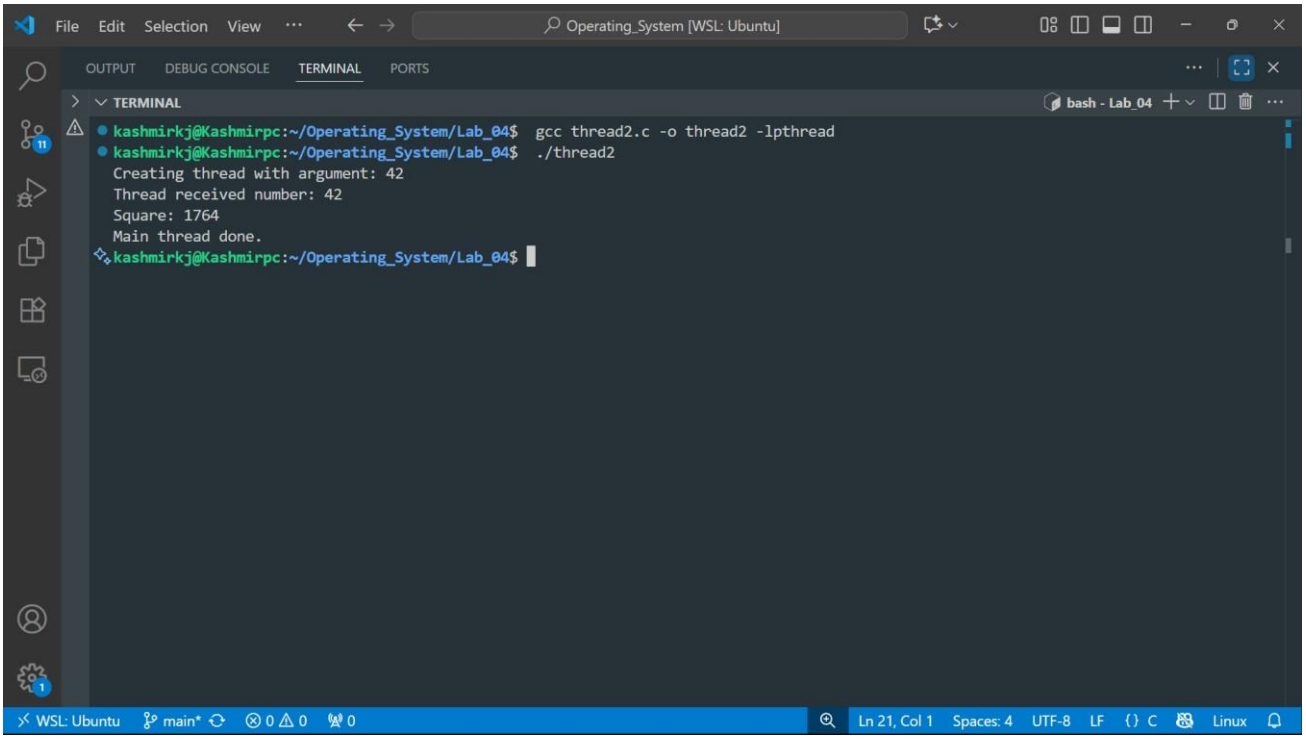
Compile and run:

```
gcc thread2 c -o thread2 -lpthread
./thread2
```

Commands:

6. mkdir Lab_04

7. cd Lab_04
8. code thread2.c (ctrl+s to save the file)
9. gcc thread2.c -o thread1 -lpthread
10. ./thread2



Important Notes:

- The 4th argument of `pthread_create()` is passed to the thread function
- It's a `void*` pointer, so you can pass any data type
- Remember to cast it properly inside the thread function

Here's what happens step by step:

```
int value = *(int*)arg;
```

1. `(int*)arg` — cast `void*` back to `int*`.
2. `*(int*)arg` — dereference the pointer to get the integer value it points to.

Why use `void*`

The thread function must have the **standard signature**:

```
void* function name(void* arg)
```

That's because threads can accept *any* data type — integers, structs, arrays, etc.
`void*` acts like a universal pointer type.
If you need to pass multiple variables, you wrap them in a `struct` and pass a pointer to it.
