**LAB REPORT**

**Lab Session 4: Introduction to Threads**

**1. Objective**

* To create POSIX threads in a C program using the pthread.h library.
* To understand and demonstrate the use of pthread\_create() for thread creation.
* To explore the functionality of pthread\_join() for synchronizing threads.
* To examine the behavior of fork() in multithreaded programs.

**2. Theory**

In modern computing, **threads** allow concurrent execution within a single process. Unlike processes, threads share the same memory space and resources, making them lightweight and more efficient for multitasking within a program.

* **POSIX Threads (pthreads)**: A standard for thread programming in Unix-based systems using the pthread.h library.
* **pthread\_create()**: Used to create a new thread. It takes arguments such as thread ID, thread attributes, function pointer, and arguments for the thread function.
* **pthread\_join()**: Allows the main thread to wait for a specific thread to finish execution before proceeding. It ensures synchronization and prevents the premature termination of threads.
* **fork() with threads**: When a multithreaded program calls fork(), only the thread that invoked fork() is duplicated in the child process. This can lead to issues if other threads were managing shared resources, and hence requires careful use.

**3. Tools and Commands Used**

* **gcc** – GNU Compiler for compiling multithreaded C programs.
* **pthread.h** – POSIX thread library header file.
* **Linux Terminal** – For executing compiled binaries.
* **System Functions** – pthread\_create(), pthread\_join(), fork().

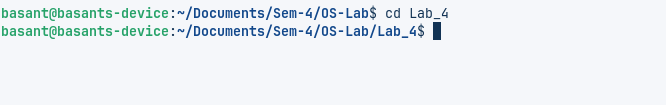
**4. Procedure**

1. **Creating POSIX Threads**
   * Write a C program including pthread.h.
   * Use pthread\_create() to launch multiple threads executing a custom function.
   * Compile with gcc -pthread filename.c -o outputname.
   * Run the program and observe concurrent thread execution.
2. **Using pthread\_join()**
   * Modify the program to use pthread\_join() after creating each thread.
   * Ensure that the main thread waits for all created threads to finish.
   * Print messages before and after pthread\_join() to observe execution flow.
3. **Exploring fork() with Threads**
   * Add a fork() call inside a multithreaded program.
   * Observe behavior in both the parent and child process.
   * Print the process ID and thread ID to analyze the result.
   * Note how only the calling thread is replicated in the child, while others are not.

**5. Programs**

1. Make use of pthread.h library to create POSIX threads using a C program. Demonstrate the use of pthread\_create to create threads in your program.

**a) Initiate the directory.**



**b) Create the C-Language file.**



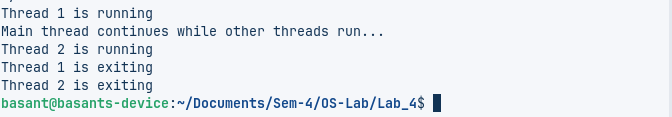
**c) Write the Program having POSIX\_Library and POSIX threads.**



**d) Compile the C Program File.**



**e) Execute the C File.**

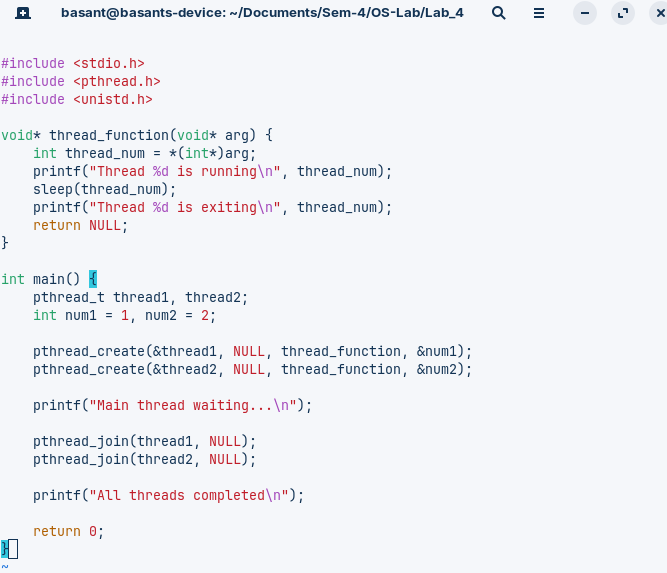


2. Demonstrate and explain the use of pthread\_join in your C program

a) Create the C-Language file.



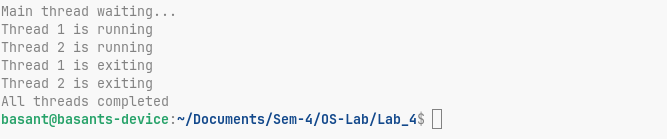
b) Write the C Program



c) Compile the File.



d) Execute the File

   
 **Use of pthread\_join:-**

* To **synchronize** threads with the main program.
* To **get return values** from threads.
* To ensure **resources are cleaned up** properly.
* Without it, the main thread might exit before others complete.

3. Explore the behavior of fork while using threads.  
 a) Create a C-Language File.

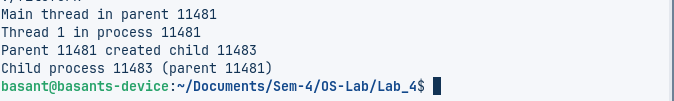


b) Write the C Program of fork using threads.  
 

c) Compile the File



d) Execute the File



**Key Observations:**

* When fork() is called in a multithreaded program:
  + **Only the calling thread is duplicated** in the child process.
  + **Other threads are not copied** to the child.
  + The **entire memory space is copied** (copy-on-write), but **only one thread exists** in the child.

**Potential Issues in Child Process:**

* **Deadlocks**: If the child process accesses a **mutex or resource that was locked** by another thread in the parent (which wasn’t copied), it may cause the child to hang.
* **Inconsistent state**: Since shared data structures or synchronization primitives (e.g., condition variables, semaphores) may now be in an **incomplete or locked** state, behavior in the child can be unpredictable.
* **Undefined behavior**: Using fork() without care in a multithreaded environment can lead to crashes, memory corruption, or unexpected behavior.

**6. Conclusion**

This lab introduced the concept of threads and demonstrated how to implement them using the POSIX pthread library in C. We practiced creating multiple threads and synchronizing them using pthread\_join(). Additionally, we explored the behavior of fork() in a multithreaded context, highlighting the implications of process creation from within a threaded environment. Understanding threads is essential for building efficient, concurrent programs in Linux.