Operating Systems Lab Fall 2024

Lab Task 14:

**Threads (Part 2) Thread Programming**



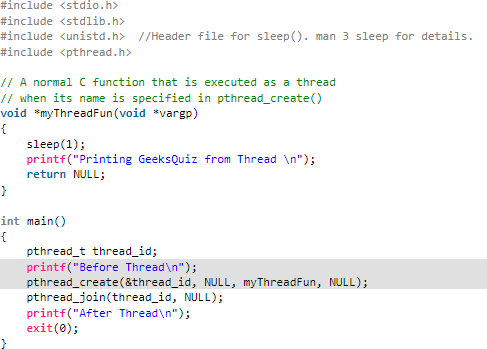
**Lab Instructor: Kausar Nasreen**

**Submitted By: Faareha Raza**

**Sap Id: 47431**

**Exercise 1:**

Type the following and execute it. **01**



stdio.h: Provides input/output functionality.

stdlib.h: Provides standard library functions (e.g., exit()). unistd.h: Contains the sleep() function to introduce delays.

pthread.h: Contains functions for thread creation and management. Calls sleep(1) to pause the execution for 1 second.

Prints "Printing GeeksQuiz from Thread".

Returns NULL since threads in this example do not return any meaningful value. pthread\_t thread\_id;

Declares a variable thread\_id of type pthread\_t to hold the thread identifier. printf("Before Thread\n");

Prints "Before Thread" before creating the thread. pthread\_create(&thread\_id, NULL, myThreadFun, NULL);

Creates a new thread:

&thread\_id: Pointer to store the thread ID.

NULL: Attributes for the thread (default attributes used here). myThreadFun: The function that the thread will execute.

NULL: Argument to the thread function (none used here). pthread\_join(thread\_id, NULL);

Waits for the created thread to finish execution. Ensures the main program does not terminate before the thread completes its task.

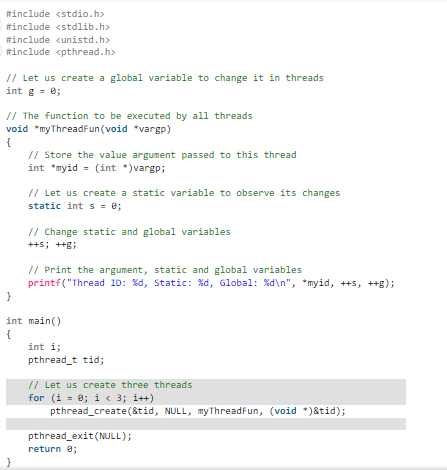
printf("After Thread\n");

Prints "After Thread" after the thread has finished executing. exit(0);

Terminates the program successfully.

**Exercise # 2:**

Try to execute following code: **02**



This program demonstrates multithreading in C, focusing on the use of global and static variables:

Global Variable g: Shared by all threads.

Changes made by one thread are visible to all others. Static Variable s:

Retains its value across function calls.

Shared among threads but not reset for each thread. Thread Creation:

A for loop creates three threads using pthread\_create.

The same tid is passed to all threads (potential issue: threads overwrite each other's IDs).

Thread Function:

Each thread increments:

g (global variable, causing potential race conditions). s (static variable, also prone to race conditions).

Outputs the thread ID, s, and g. Concurrency Issues:

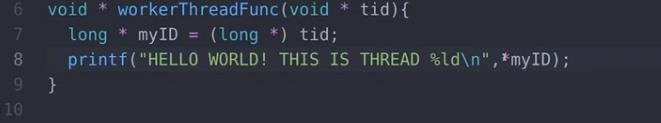
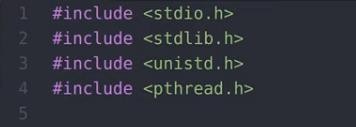
No synchronization is used, leading to race conditions with g and s.

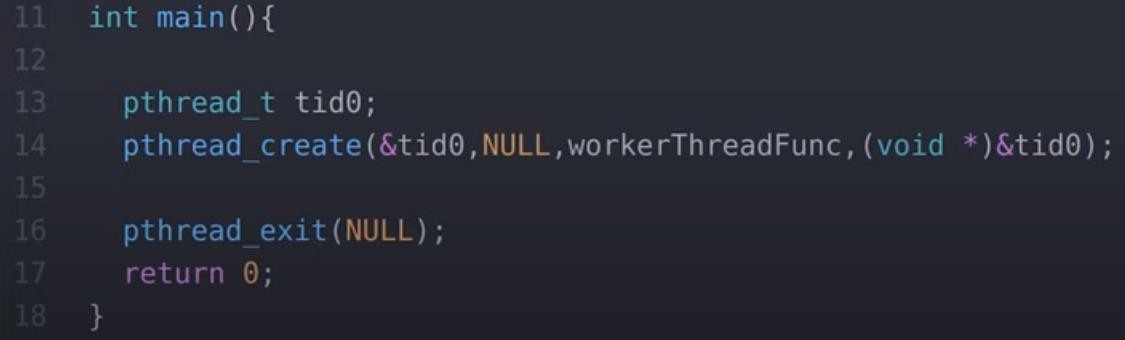
Key Takeaway: Global and static variables can lead to unpredictable behavior in multithreading without proper synchronization mechanisms.

**Show output with explanation**

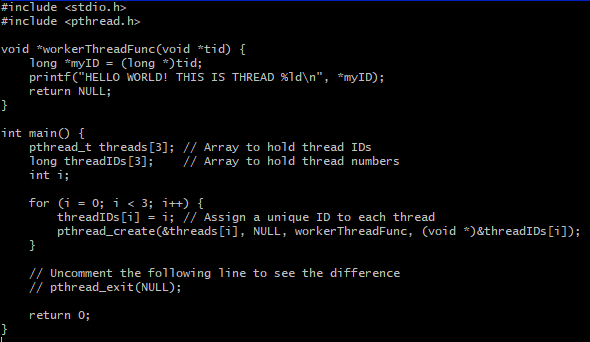
**Exercise # 3:**

Try to execute following code: **04**





Modify **pthread\_create** and convert into for loop and create three threads and show output. Also removes pthread\_exit and see what happens.



Explanation of Changes

For Loop: The pthread\_create function is called inside a for loop to create three threads. Each thread gets a unique threadID from the threadIDs array.

Thread IDs: We use the threadIDs array to pass a unique value to each thread, which is cast to (void \*) during the function call.

Removed pthread\_exit: The pthread\_exit(NULL) line has been commented out. Expected Output with pthread\_exit

When pthread\_exit is included, the main thread will wait for all created threads to finish their execution before terminating the program.

HELLO WORLD! THIS IS THREAD 0 HELLO WORLD! THIS IS THREAD 1

HELLO WORLD! THIS IS THREAD 2

Expected Output without pthread\_exit

If pthread\_exit is removed, the main thread may terminate before the worker threads complete their execution. This might result in incomplete or no output from the threads, depending on the scheduler's timing. You may see no output at all or partial output like this:

vbnet Copy code

HELLO WORLD! THIS IS THREAD 0

This happens because the program exits when the main function finishes, and any running threads are abruptly terminated.

Exercise 4:

Define posix thread and its working in your own words. **03**

A POSIX thread (Pthread) is a lightweight, independent execution unit within a process, defined by the POSIX standard for thread creation and management.

Threads share the same memory space and resources of the parent process, making them efficient for concurrent execution.

How They Work:

Creation: Threads are created using pthread\_create(), executing a specified function.

Shared Resources: Threads share memory but require synchronization (e.g., mutexes) to prevent data conflicts.

Lifecycle: Threads run independently and terminate via pthread\_exit() or by returning from their function.

Coordination: Tools like mutexes and condition variables handle synchronization. Benefits:

Concurrency: Enables multitasking.

Efficiency: Lightweight compared to processes. Scalability: Ideal for leveraging multi-core systems. Use Cases:

Web servers, real-time systems, and computational tasks to improve speed and responsiveness.