Lab No. 08
Creating Multithreaded Applications

Objective
This lab is designed to familiarize the students with the implementation of multithreading.

Activity Outcomes:
On completion of this lab students will be able to

- Implement simple multithreaded applications

Instructor Notes
As pre-lab activity, read the content from the following (or some other) internet source:
https://www.geeksforgeeks.org/multithreading-in-cpp/

1) Useful Concepts

Multithreading

A thread is a path of execution within a process. A process can contain multiple threads. A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. Parallelism is the feature that allows your computer to run two or more programs from same application simultaneously. As a result, our applications are executed in less time and become more interactive. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

We can implement parallel applications in two ways: process-based and thread-based. Process-based parallelism is achieved through the simultaneous execution of more than one processes from the same application while thread-based parallelism deals with the simultaneous execution of pieces of the same processes. Generally, thread-based parallelism is considered efficient than process-based parallelisms.

Pthread Library

Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications. In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995). Implementations which adhere to this standard are referred to as POSIX threads, or Pthreads. Most hardware vendors now offer Pthreads in addition to their proprietary API's. Pthreads are defined as a set of C language programming types and procedure calls. Vendors usually provide a Pthreads implementation in the form of a header/include file and a library, which you link with your program.

In this lab we are going to write multi-threaded C++ program using POSIX. POSIX Threads, or Pthreads provides API which are available on many Unix-like POSIX systems such as FreeBSD, NetBSD, GNU/Linux, Mac OS X and Solaris.

Creating Threads

We can use the following routine to create a POSIX thread

```
pthread_create (thread, attr, start_routine, arg)
```

Here, pthread_create creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code. Here is the description of the parameters is as follows:

Parameter	Description
Thread	An unique identifier for the new thread returned by the subroutine.
Attr	An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
start_routine	The C++ routine that the thread will execute once it is created.
Arg	A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

It is to be noted that global variables can be accessed from all threads while variables defined within a thread are not accessible to other threads.

Displaying Thread ID

We can use the pthread_self( ) routine to display the id of the current thread.

Terminating Threads

Following routine terminates a POSIX thread

```
pthread_exit (status)
```

Here **pthread_exit** is used to explicitly exit a thread. Typically, the pthread_exit routine is called after a thread has completed its work and is no longer required to exist. If main finishes before the threads it has created, and exits with pthread_exit, the other threads will continue to execute. Otherwise, they will be automatically terminated when main finishes.

Joining and Detaching Threads

Following routines are used for joining threads

```
pthread_join (threadid, status)
```

The pthread_join subroutine blocks the c

alling thread until the specified threadid thread terminates.

Note: To compile a program that uses pthread library, we need to use the –pthread option. For example, to compile a program test.cpp we can write the following command

```
g++ test.cpp –o test -pthread
```

2) Solved Lab Activities

Activity 1:

The following example code creates 5 threads with the pthread_create routine. Each thread prints a "Hello World!" message, and then terminates with a call to pthread_exit.

Solution:

Code

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <cstdint>

using namespace std;

#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)(intptr_t)threadid;

   cout << "Hello World! Thread ID, " << tid << endl;

   pthread_exit(NULL);
}

int main()
{
   pthread_t threads[NUM_THREADS];
   int rc;
   int i;

   for (i = 0; i < NUM_THREADS; i++)
   {
      cout << "main() : creating thread, " << i << endl;

      rc = pthread_create(&threads[i], NULL, PrintHello, (void*)(intptr_t)i);

      if (rc)
      {
         cout << "Error: unable to create thread, " << rc << endl;
         exit(-1);
      }
   }

   pthread_exit(NULL);
   return 0;
}
```

```
main() : creating thread, 0
main() : creating thread, 1
main() : creating thread, 2
main() : creating thread, 3
main() : creating thread, 4
Hello World! Thread ID, 0
Hello World! Thread ID, 1
Hello World! Thread ID, 2
Hello World! Thread ID, 3
Hello World! Thread ID, 4
```

Activity 2:

This example demonstrates how to wait for thread completions by using the Pthread join routine.

Solution:

Code

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <unistd.h>

using namespace std;

#define NUM_THREADS 5

void *wait(void *t)
{
   long tid = (long)t;

   sleep(1);

   cout << "Sleeping in thread " << endl;
   cout << "Thread with id : " << tid << " ...exiting " << endl;

   pthread_exit(NULL);
}

int main()
{
   int rc;
   int i;

   pthread_t threads[NUM_THREADS];
   pthread_attr_t attr;
   void *status;

   // Initialize and set thread joinable
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

   // Create threads
   for (i = 0; i < NUM_THREADS; i++)
   {
      cout << "main() : creating thread, " << i << endl;

      rc = pthread_create(&threads[i], &attr, wait, (void *)(long)i);

      if (rc)
      {
         cout << "Error: unable to create thread, " << rc << endl;
         exit(-1);
      }
   }
```

```
    // Destroy attribute
    pthread_attr_destroy(&attr);

    // Join threads
    for (i = 0; i < NUM_THREADS; i++)
    {
       rc = pthread_join(threads[i], &status);

       if (rc)
       {
          cout << "Error: unable to join, " << rc << endl;
          exit(-1);
       }

       cout << "Main: completed thread id : " << i;
       cout << " exiting with status : " << status << endl;
    }

    cout << "Main: program exiting." << endl;

    pthread_exit(NULL);
}
```

Out-put

```
main() : creating thread, 0
main() : creating thread, 1
main() : creating thread, 2
main() : creating thread, 3
main() : creating thread, 4
Sleeping in thread
Thread with id : 0..... exiting
Sleeping in thread
Thread with id : 1..... exiting
Sleeping in thread
Thread with id : 2..... exiting
Sleeping in thread
Thread with id : 3..... exiting
Sleeping in thread
Thread with id : 4..... exiting
Main: completed thread id :0  exiting with status :0
Main: completed thread id :1  exiting with status :0
Main: completed thread id :2  exiting with status :0
Main: completed thread id :3  exiting with status :0
Main: completed thread id :4  exiting with status :0
Main: program exiting.
```

Task 1:

Write a C++ program using pthreads that creates **two threads**. The first thread should print numbers from 1 to 10, and the second thread should print numbers from 11 to 20. The main program should wait for both threads to complete before exiting.

Task 2:

Write a C++ program using pthreads that creates three threads. Each thread should print its own thread ID along with a message "Thread is running". Observe the output and explain why the execution order is not fixed.

Task 3:

Write a C++ program using pthreads where two threads are created. One thread should calculate the sum of numbers from 1 to 50, and the second thread should calculate the sum of numbers from 51 to 100. The main program should wait for both threads and display the results.

Task 4:

Write a C++ program using pthreads that creates **five threads in a loop**. Each thread should print its thread number and then exit. Ensure the main program waits for all threads using pthread_join.

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation

LABORATORY SKILLS ASSESSMENT (Psychomotor)
Total Marks: 100

(Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< G0%	Level 4 G0%≤ S ≤100%	Score (S)
Procedural Awareness (20)	Selects inappropriate Linux commands, shell scripting techniques, or process management methods.	Selects and applies partially appropriate Linux commands and techniques	Selects and applies considerably appropriate Linux commands and techniques.	Selects and applies completely appropriate Linux commands and techniques	
Practical Implementation (20)	Makes major critical errors in executing Linux commands, scripting, and system processes.	Makes numerous critical errors in executing commands and process management.	Makes minor non-critical errors in executing Linux commands and system operations.	Executes Linux commands and manages processes correctly with no errors.	
Process Management and Shell Scripting (20)	Program logic contains major errors with incorrect or contradictory script flow.	Program logic has some errors with occasional contradictions in process execution.	Program logic is mostly correct but may contain occasional redundancy or minor errors.	Program logic is completely correct with no contradictions or redundant processes.	
Syntax Correctness and Results (20)	Program does not follow proper syntax for Linux commands and shell scripting, leading to incorrect outputs	Program partially follows proper syntax, producing correct results for few inputs.	Program adequately follows proper syntax, producing correct results for most inputs.	Program fully follows proper syntax, producing accurate results for all inputs.	
Use of OS Tools (10)	Uses OS tools (like terminal, process manager) with limited competence.	Uses OS tools with some competence.	Uses OS tools with considerable competence.	Uses OS tools proficiently with a high degree of competence.	
Safety (10)	Requires constant reminders to follow system safety procedures (e.g., file permissions, process handling).	Requires some reminders to follow system safety procedures.	Follows system safety procedures with minimal reminders.	Routinely follows system safety procedures.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Affective)
Total Marks: 40

Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < G0%	Level 4 G0% ≤ S ≤ 100%	Score (S)
Attitude s Engagement (5)	Shows little interest in lab activities; does not participate actively.	Participates occasionally but lacks enthusiasm and consistency.	Engages actively in most lab activities with interest.	Highly motivated, participates enthusiastically, and shows a proactive approach	
Responsibility s Punctuality (5)	Frequently misses deadlines and is often late to lab sessions.	Occasionally late or misses deadlines but tries to catch up.	Submits work on time and attends lab sessions regularly.	Always punctual, meets deadlines, and takes full responsibility for assigned tasks.	
Collaboration s Teamwork (10)	Rarely collaborates, struggles to work in a team, and does not contribute effectively.	Works with team members occasionally but struggles with communication.	Cooperates well, contributes effectively, and maintains professional interactions.	Actively engages in teamwork, supports peers, and demonstrates excellent collaboration.	
Communicatio n s Presentation Skills (10)	Struggles to explain concepts, unclear verbal/written communication.	Communicates ideas with some clarity but lacks confidence or coherence.	Presents ideas effectively with minor issues in clarity or structure.	Communicates clearly, confidently, and effectively in all aspects of lab work.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	