National University of Computer and Emerging Sciences



Laboratory Manual

for

Operating Systems Lab

(CL-220)

|  |  |
| --- | --- |
| Course Instructor | Rubab Anum |
| Lab Instructor(s) | Mr. Affan Malik  Ms. Khulda Zubair |
| Section | SE- 5B |
| Semester | Fall 2023 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives:

In this lab, students will practice multithreading

Differences between Process and Thread

|  |  |
| --- | --- |
| **Process** | **Thread** |
| Process is heavy weight or resource intensive. | Thread is light weight, taking lesser resources than a process. |
| Process switching needs interaction with operating system. | Thread switching does not need to interact with operating system. |
| In multiple processing environments, each process executes the same code but has its own memory and file resources. | All threads can share same set of open files, child processes. |
| If one process is blocked, then no other process can execute until the first process is unblocked. | While one thread is blocked and waiting, a second thread in the same task can run. |
| Multiple processes without using threads use more resources. | Multiple threaded processes use fewer resources. |
| In multiple processes each process operates independently of the others. | One thread can read, write or change another thread's data. |

## Advantages of Thread

* Threads minimize the context switching time.
* Use of threads provides concurrency within a process.
* Efficient communication.
* It is more economical to create and context switch threads.
* Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

## Types of Thread

Threads are implemented in following two ways −

* **User Level Threads** − User managed threads.
* **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

1) **int pthread\_create(pthread\_t \****thread***, const pthread\_attr\_t \****attr***, void \*(\****start\_routine***) (void \*), void \****arg***);**

The **pthread\_create**() function starts a new thread in the calling process. The new thread starts execution by invoking *start\_routine*(); *arg* is passed as the sole argument of *start\_routine*().

On success, **pthread\_create**() returns 0; on error, it returns an error number, and the contents of *\*thread* are undefined.

2)**int pthread\_join(pthread\_t** *thread***, void \*\****retval***);**

The **pthread\_join**() function waits for the thread specified by *thread* to terminate. If that thread has already terminated, then **pthread\_join**() returns immediately. The thread specified by *thread* must be joinable.

3) **void pthread\_exit(void \****retval***);**

The **pthread\_exit**() function terminates the calling thread and returnsa value via *retval* that (if the thread is joinable) is available toanother thread in the same process that calls [pthread\_join](http://man7.org/linux/man-pages/man3/pthread_join.3.html).

This call always succeeds and does not return anything in calling process.

Every thread has set of attributes.

Default set of attributes: Table 3-1 Default Attribute Values for *tattr*

| **Attribute** | **Value** | **Result** |
| --- | --- | --- |
| *scope* | PTHREAD\_SCOPE\_PROCESS | New thread is unbound - not permanently attached to LWP. |
| *detachstate* | PTHREAD\_CREATE\_JOINABLE | Exit status and thread are preserved after the thread terminates. |
| *stackaddr* | NULL | New thread has system-allocated stack address. |
| *stacksize* | 1 megabyte | New thread has system-defined stack size. |
| *priority* |  | New thread inherits parent thread priority. |
| *inheritsched* | PTHREAD\_INHERIT\_SCHED | New thread inherits parent thread scheduling priority. |
| *schedpolicy* | SCHED\_OTHER | New thread uses Solaris-defined fixed priority scheduling; threads run until preempted by a higher-priority thread or until they block or yield. |

**Check Online:**

1. See Joinable vs Detached thread online
2. if a thread executes fork system call
3. if a thread executes exec system call

**Question 1.**

Write a program which takes some positive integers (let’s say **N** number ofpositive integers) as

command line parameters, creates **N** synchronous threads, and send s the corresponding integer as parameter to the thread function fibonacciGenerator. The function returns the generated series to the main thread. The main thread will then print the thread number and the series generated by that thread. The output will be like:

Thread 1: 0 1 1 2 3 5 8 13

**Example:**

If you pass as command line argument the following numbers: 3 13 34 89

Then the program will create 4 threads. The first thread will find Fibonacci terms until 3 is generated, the second Fibonacci term will find Fibonacci terms until the term generated is 13 , so on and so forth. All generated terms will be output on the screen by the main thread as follows:

Thread 0: 0, 1, 1, 2, 3

Thread 1: 0, 1, 1, 2, 3, 5, 8, 13

Thread 2: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Thread 3: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

It is possible that the number passed to the thread is not a Fibonacci number. In this case the thread will generate numbers until the term generated is greater than the passed number. For example if 7 is passed as parameter to a thread, then the thread will return the following series:

0, 1, 1, 2, 3, 5, 8