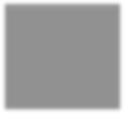
AIR UNIVERSITY, ISLAMABAD



Department of Computer Science

FACULTYOF COMPUTING AND ARTIFICIAL INTELLIGENCE

Operating Systems Lab-CS225L Lab: 07

Topic: Threads

Lab Instructor:

**FACULTY OF COMPUTING & ARTIFICAL**

**INTELLIGENCE**

**Department of Computer Science**

## Instructions:

**Plagiarism:** Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in this assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

**Deadline:** Late submission with marks deduction will be accepted according to the course policy shared earlier. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

**Tip:** For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

1. Note: Follow the given instructions to the letter, failing to do so will result in a zero.

## Objectives:

In this lab, you will learn:

* How to create threads
* How to pass arguments to thread
* Distinguish b/w Parent & Child Process

**Threads**

Threads, like processes, are a mechanism to allow a program to do more than one thing at a time.Conceptually, a thread exists within a process.When you invoke a program, Linux creates a new process and in that process creates a single thread, which **runs the program sequentially**.Thread can create additional threads.All these threads run the same program in the same process.But **each thread may be executing a different part of the program** at any given time**.**

## How fork works:

We’ve seen how a program can fork() a child process is created.

* + The child process is initially running its parent’s program.
  + With its parent’s virtual memory, file descriptors, and so..
  + The child process can modify its memory, close file descriptors, without affecting its parent, and vice versa.

## How Thread works:

* + When a program creates another thread, though, nothing is copied With its parent’s

virtual memory, file descriptors, and so..

* + The creating and the created thread share the same memory space, file descriptors, and other system resources as the original
  + If one thread changes the value of a variable, for instance, the other thread subsequently will see the modified value
  + Similarly, if one thread closes a file descriptor, other threads may not read from or write to that file descriptor. Because a process and all its threads can be executing only one program at a time.

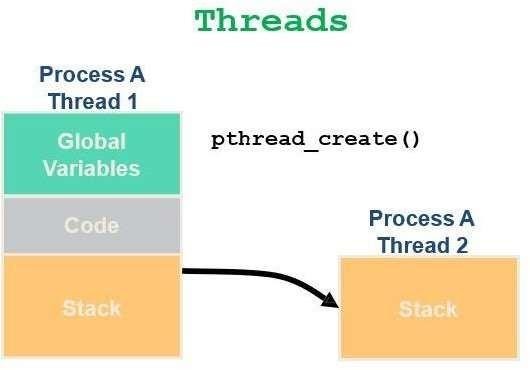
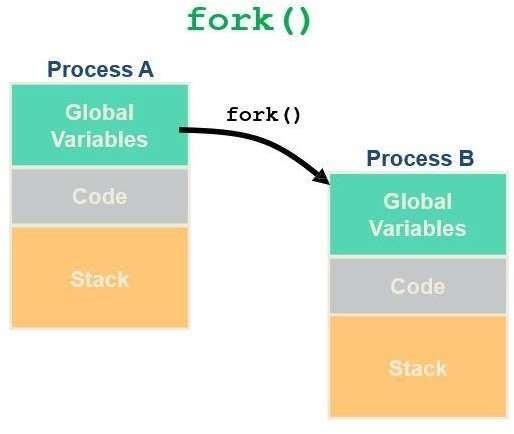
## Multiple Threads:

Each process can include many threads. All threads of a process share:

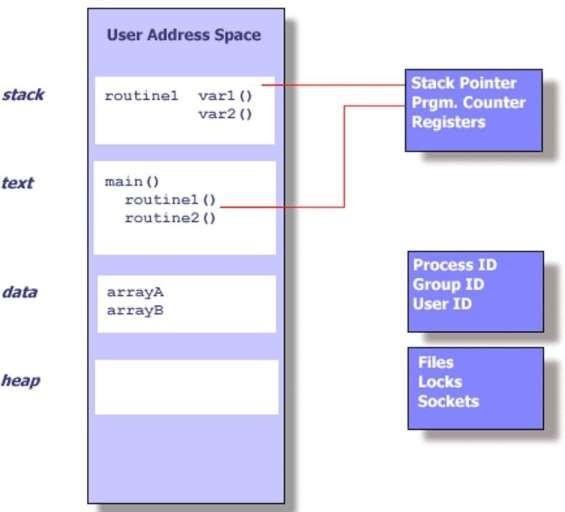
* + memory (program code and global data)
  + open file/socket descriptors
  + signal handlers
  + working environment (current directory, user ID, etc.)

Each thread has it’s own:

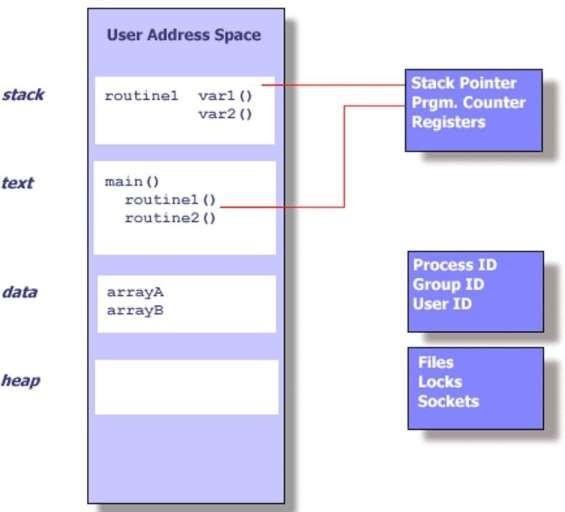
* + Thread ID (integer)
  + Stack, Registers, Program Counter
  + Threads within the same process can communicate using shared memory.



# UNIX Process



**Threads within a UNIX Process**



## Thread Creation:

* + Each thread in a process is identified by a ***thread ID***
  + When referring to thread IDs in C or C++ programs, use the type **pthread\_t.**
  + Upon creation, each thread executes a *thread function*
  + This is just an ordinary function and contains the code that the thread should run.
  + When the function returns, the thread exits.
  + If successful, pthread\_create returns 0.
  + If unsuccessful, pthread\_create returns a nonzero error code

## Posix Threads:

We will focus on *Posix Threads*- most widely supported threads programming API.

### #include <pthread.h>

The pthread\_create function creates a new thread

## How to Compile the thread files:

Linux - you need to link with “-lpthread”

**i.e g++ -o hello hello.cpp -lpthread**

# Syntax:

## pthread\_Func:

### pthread\_create( pthread\_t \*thread,

**const pthread\_attr\_t \*attr, void \*(\*func)(*void \*), void \*arg);***

* + pthread\_create is the function to be called.
  + When func() returns the thread is terminated.

## Arguments 1:

### pthread\_t \*thread

* + The first argument is a pointer to pthread\_t.
  + An identifier is written to the memory location to which this variablepoints
  + This identifier enables you to refer to the thread.

## Arguments 2:

### const pthread\_attr\_t \*attr

* + A pointer to a *thread attribute* object.
  + This object controls details of how the thread interacts with the rest of theprogram.
  + If you pass NULL as the thread attribute, a thread will be created with the default thread attributes.

## Arguments 3:

**void \*(\*func)(*void \*)***

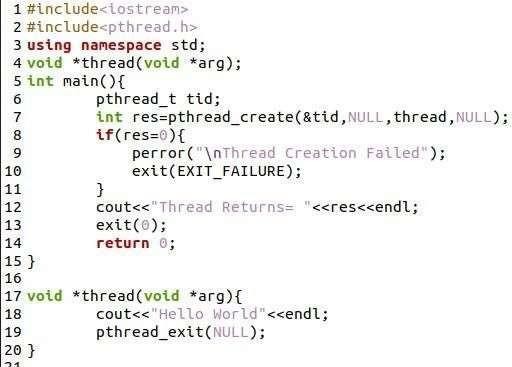
* you must pass the address of a function.
* Taking a pointer to void as a parameter.
* Function will return a pointer to void.
* Pass any type of single argument and return a pointer to any type.
  + The return value is 0 for success
  + An error number if anything goes wrong

## Arguments 4:

### void \*arg

* \*arg is a pointer to argument of function.
* To pass multiple arguments, send a pointer to a structure.

## Example 1:



**pthread\_exit(\*status):**

* Used to exit the process
* Status argument is used to pass any value to the original process on return.

## Task 1:

Write a program which print the following Using threads:

First Name Last Name Class , Section

Air University Islamabad

## pthread\_join :

### pthread\_join (pthread\_t thread , pthread\_addr\_t \*status);

* Pthread\_join : join with a terminated thread
* The pthread\_join() function waits for the thread specified by *thread* to terminate.
* If that thread has already terminated, then pthread\_join() returns immediately.
* It is **similar to wait()** call while using fork system call.

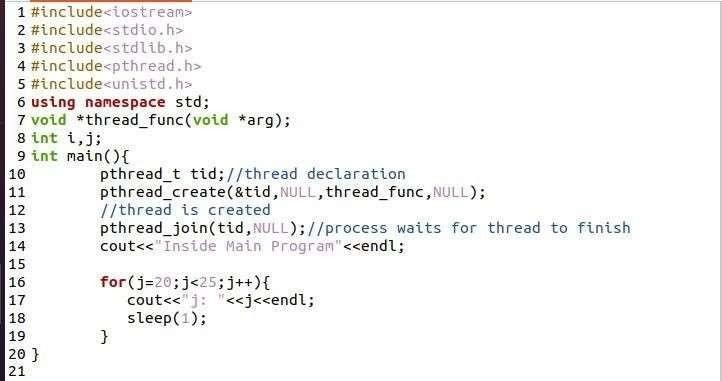
pthread\_join() does not kill the thread but waits for the thread to complete. pthread\_join will wait for completion of specified thread.so if function call is returned it means that thread is terminated.

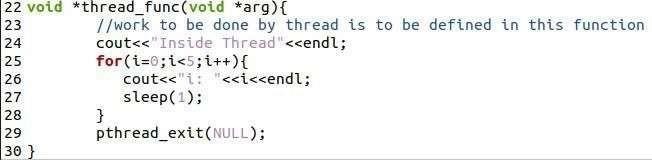
### Parameters

**thread :**The first parameter is the thread for which to wait.

**status :**Status value of the terminating thread when that thread returns or call pthread\_exit**. Return Value:** On success, pthread\_join() returns 0; on error, it returns an error number.

## Example 2:

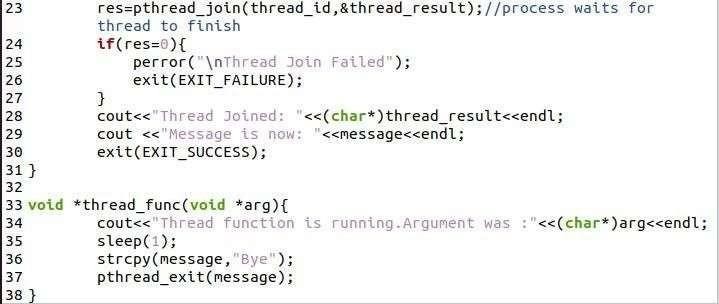
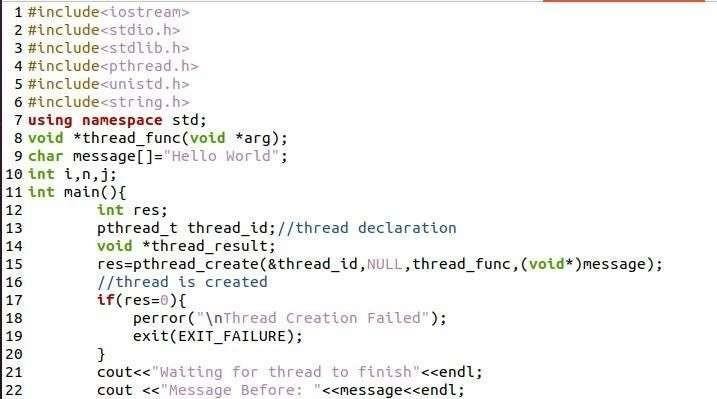




**Passing Value to newly created thread : Example 3:**



## Example 4:



**Task 2:**

Write a program which print the following series using threads

