

**Lab Manual for Operating System**

Lab No. 8

**Exploring Threads in Linux**

*Objectives*

*To understand basic concept, working and usage of threads and multithreading.*

**Threads**

**LAB # 08**

**Exploring Threads in Linux**

Threads, in Linux, are Light-Weight Processes (LWP). Threads are the core element of a multi-tasking programming environment. By definition, a thread is an execution context in a process; hence, every process has at least one thread. Multi-threading implies the existence of multiple, concurrent (on multi- processor systems), and often synchronized execution contexts in a process.

The creation and destruction of threads is quick. Unlike fork(), there is no new copy of the parent process, but it uses the same address space and shares resources. A multi-threaded application uses resources optimally, and is highly efficient. In such an application, threads are loaded with different categories of work, in such a manner that the system is optimally used. This improves system utilization, and hence, throughput.

In Linux, threads (also called Lightweight Processes (LWP)) created within a program will have the same "thread group ID" as the program's PID. Each thread will then have its own thread ID (TID). There are several ways to show threads for a process on Linux.

**ps command:**

It is used to display information about current active processes. In order to view threads associated with a process, use **-T** option

**Syntax:** $ps -option <argument>

**Examples:**

$ ps -T lists threads of all processes

$ ps -T <pid> lists threads of a particular process with ID specified by pid

For more information on threads, you may execute the command with the following option

$ ps -eLf

**top command:**

It is used to show a real-time view of individual threads.

**Syntax:** $top -option <argument>

**Examples:**

$ top gives the total number of threads

![Text

Description automatically generated]()

Press q to quit.

A more user-friendly and interactive way to view threads in a process is to use htop. It is an application not installed on the operating system. You first have to install the package by executing the following.

$ sudo apt install htop

This will prompt you for the user password. Once you type it and press Enter, the package will install automatically. After the installation is complete, you may execute htop as follows.

$ htop

This program allows you to monitor individual threads in tree views. To enable thread views in htop, launch htop, and press <F2> to enter htop setup menu. Choose "Display option" under "Setup" column, and toggle on "Tree view" and "Show custom thread names" options. Press <F10> to exit the setup. You can explore other options as well.

![A picture containing graphical user interface

Description automatically generated]()

Threads in Linux

Linux supports the development and execution of multi-threaded applications. User-level threads in

Linux follow the open POSIX (Portable Operating System Interface for uniX) standard, designated as IEEE 1003. The user-level library (on Ubuntu, glibc.so) has an implementation of the POSIX API for threads.

Threads exist in two separate execution spaces in Linux — in user space and the kernel. User-space

threads are created with the pthread library API (POSIX compliant). These user-space threads are mapped to kernel threads. In Linux, kernel threads are regarded as “light-weight processes”. An LWP is the unit of a basic execution context. Unlike other UNIX variants, including HP-UX and SunOS, there is no special treatment for threads. A process or a thread in Linux is treated as a “task”, and shares the

same structure representation.

# POSIX Library

The POSIX thread libraries are a standards-based thread API for C/C++. It allows one to spawn a new concurrent process flow. It is most effective on multi-processor or multi-core systems where the process flow can be scheduled to run on another processor thus gaining speed through parallel or distributed processing. Threads require less overhead than "forking" or spawning a new process because the system does not initialize a new system virtual memory space and environment for the process. While most effective on a multiprocessor system, gains are also found on uniprocessor systems which exploit latency in I/O and other system functions which may halt process execution. (One thread may execute while another is waiting for I/O or some other system latency.) Parallel programming technologies such as MPI and PVM are used in a distributed computing environment while threads are limited to a single computer system. All threads within a process share the same address space. A thread is spawned by defining a function and its arguments which will be processed in the thread. The purpose of using the POSIX thread library in your software is to execute software faster.

Thread operations include thread creation, termination, synchronization (joins, blocking), scheduling, data management and process interaction. All threads within a process share the same address space, as well as the process instructions, most data, open files, signals and signal handlers, current working directory, and user and group id. Each thread has a unique thread ID, set of registers, stack pointer, stack for local variables, return addresses, signal mask, and priority.

## Example 1: Two Threads displaying two strings “Hello” and “How are you?” independent of each other

* Create a new file thread.c with .c extension using any editor
* Type the following code.

#include<stdio.h> #include<pthread.h> #include <stdlib.h> void \* thread1()

{

while(1){ printf("Hello!!\n";

}

}

void \* thread2()

{

while(1){

printf("How are you?\n");

}

}

int main()

{

int status; pthread\_t tid1,tid2;

pthread\_create(&tid1,NULL,thread1,NULL); pthread\_create(&tid2,NULL,thread2,NULL; pthread\_join(tid1,NULL); pthread\_join(tid2,NULL);

return 0;}

* Save and exit.
* To compile it type the following command on terminal.

gcc –o thread thread.c -lpthread

* Run it by using following command.

./thread

The –lpthread at the end to link the pthread library.

## Example 2: Create a function message() that takes thread id as argument and prints the message with thread id. There should be atleast four independent threads

* Create a new file msgthreads.c with .c extension using any editor
* Type the following code.

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#define NUM\_THREADS 4

#define MSG “Hello message from”

void \*message(void \*threadid) {

long tid;

tid= (long)threaded;

printf(“msgthreads [INFO] Message: %s \t Thread ID: %ld \n”, MSG, tid);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int rc;

long t;

for(t=0;t<NUM\_THREADS;t++)

{

printf ("IN:main creating thread %ld\n", t);

rc = pthread\_create(&threads[t],0, message,(void \*)t);

}

pthread\_join(threads[0],0);

pthread\_join(threads[1],0);

pthread\_join(threads[2],0);

pthread\_join(threads[3],0);

return 0;

}

* Save and exit.
* To compile it type the following command on terminal.

gcc –o msgthreads msgthreads.c -lpthread

* Run it by using following command.

./msgthreads

![Text

Description automatically generated]()

# Lab Tasks

1. Explore HTOP, including its options. Attach outputs for the same.
2. Write a multithreaded C program for performing summation of numbers.
3. Write a program which make 4 threads. Each thread will print one table out of [5678] up to 1000.