

**Usman Institute of Technology Department of Computer Science Fall 2023**

**CSC203–Operating Systems**

**Lab #7**

**Objective:**

**Threads in Operating Systems**

Understanding Threads, Threads Creation, Multithreaded Programming using Python

| Name of Student | MAHA REHAN |
| --- | --- |
| Student ID | 22SP-018-SE |
| Date of Lab Conducted |  |
| Marks Obtained |  |
| Remarks |  |
| Signature |  |

**Objective:** Understanding Threads, Threads Creation, Multithreaded Programming using Python & C.

**THEORY**

Thread is the smallest unit of processing. It is scheduled by an OS. In general, it is contained in a process. So, multiple threads can exist within the same process. It shares the resources with the process. Each thread belongs to exactly one process and no thread can exist outside a process.

A thread uses the same address space of a process. A process can have multiple threads. A key difference between processes and threads is that multiple threads share parts of their state. Typically, multiple threads can read from and write to the same memory (no process can directly access the memory of another process). However, each thread still has its own stack of activation records and its own copy of CPU registers, including the stack pointer and the program counter, which together describe the state of the thread's execution. A thread is a particular execution path of a process. When one thread modifies a process resource, the change is immediately visible to sibling threads. Processes are independent while thread is within a process. Processes have separate address spaces while threads share their address spaces. Multithreading has some advantages over multiple processes. Threads require less overhead to manage than processes, and intra-process thread communication is less expensive than inter-process communication.

**Types of Threads:**

**User Level thread (ULT) –**

User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.

**Kernel Level Thread (KLT) –**

Kernel threads are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

**Viewing threads of a process/processes on Linux**

**Ps Command**

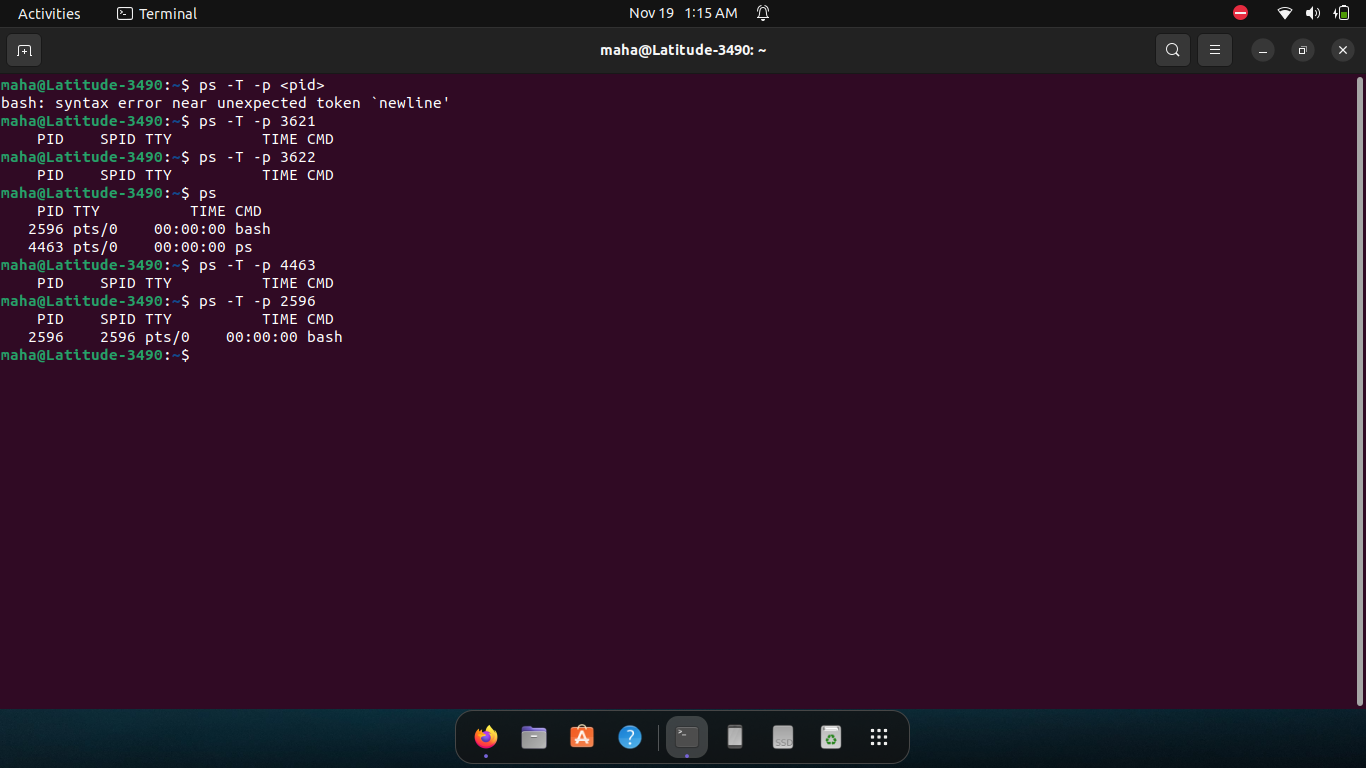
In ps command, "-T" option enables thread views. The following command list all threads created by a process with

<pid>.

ps -T -p <pid>

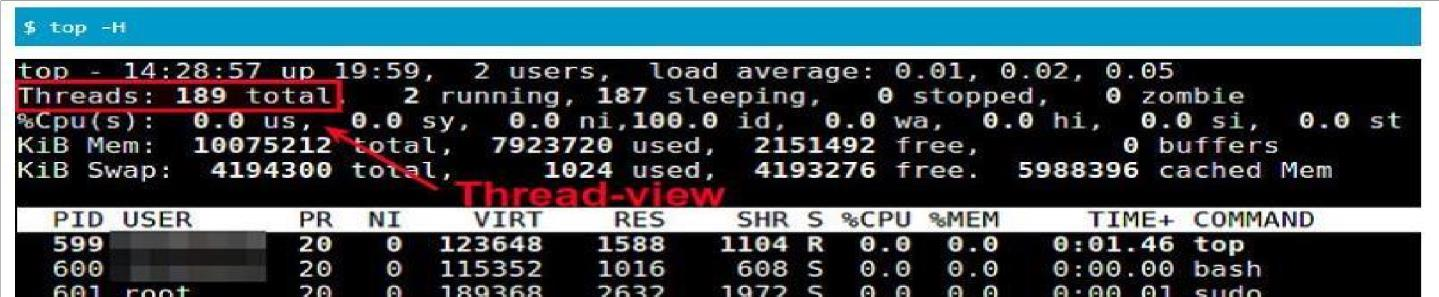
The "SID" column represents thread IDs, and "CMD" column shows thread names.

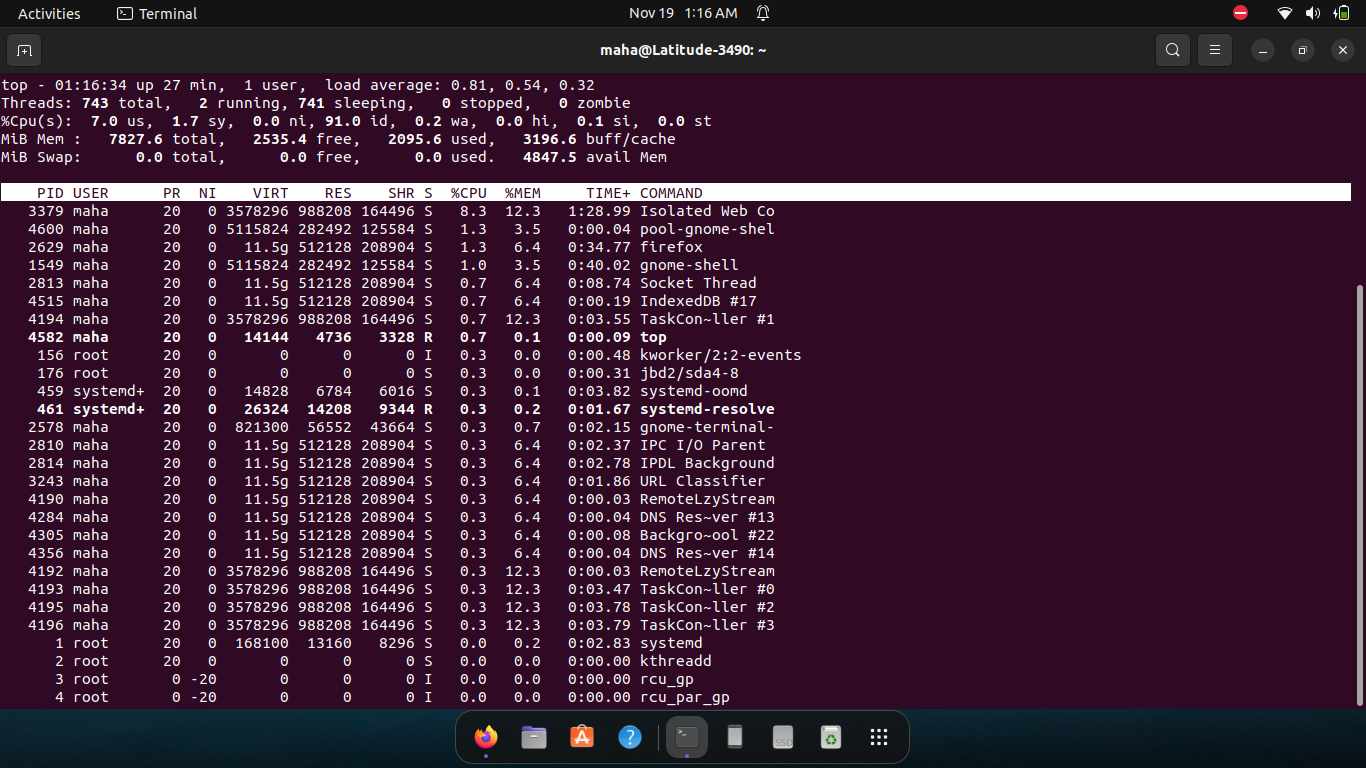




**Top Command**

The top command can show a real-time view of individual threads. To enable thread views in the top output, invoke top with "-H" option. This will list all Linux threads. You can also toggle on or off thread view mode while top is running, by pressing 'H' key.





***\*Command-line tools such as ps or top, which display process-level information by default, can be instructed in many ways to display thread-level information.***

**THREAD CREATION in Python- (Multithreaded Programming):**

There are two modules which support the usage of threads in Python3 − \_thread , Threading

**The Threading Module**

The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module

the threading module has the Thread class that implements threading. The methods provided by the Thread class are as follows –

• run() − The run() method is the entry point for a thread.

• start() − The start() method starts a thread by calling the run method.

• join([time]) − The join() waits for threads to terminate.

• isAlive() − The isAlive() method checks whether a thread is still executing.

• getName() − The getName() method returns the name of a thread.

• setName() − The setName() method sets the name of a thread.

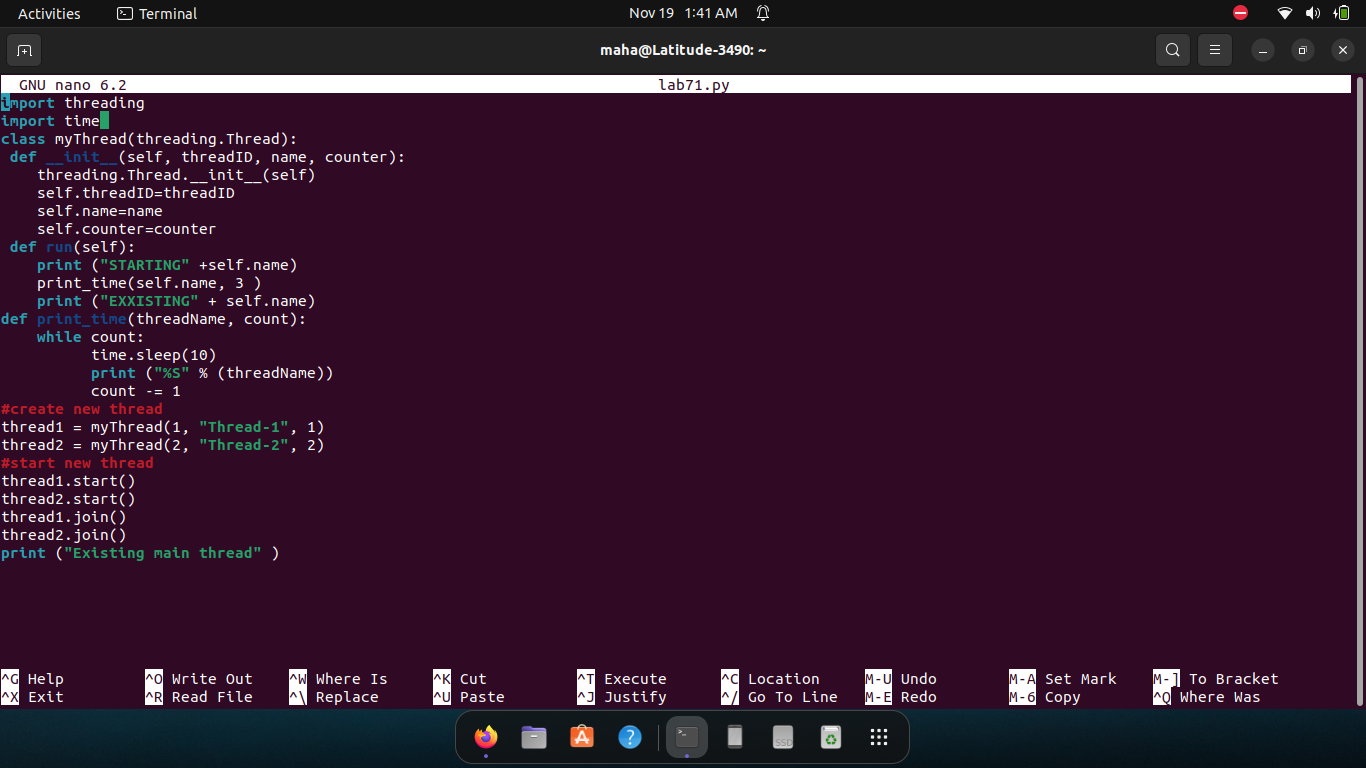
To implement a new thread using the threading module, you have to do the following − Define a new subclass

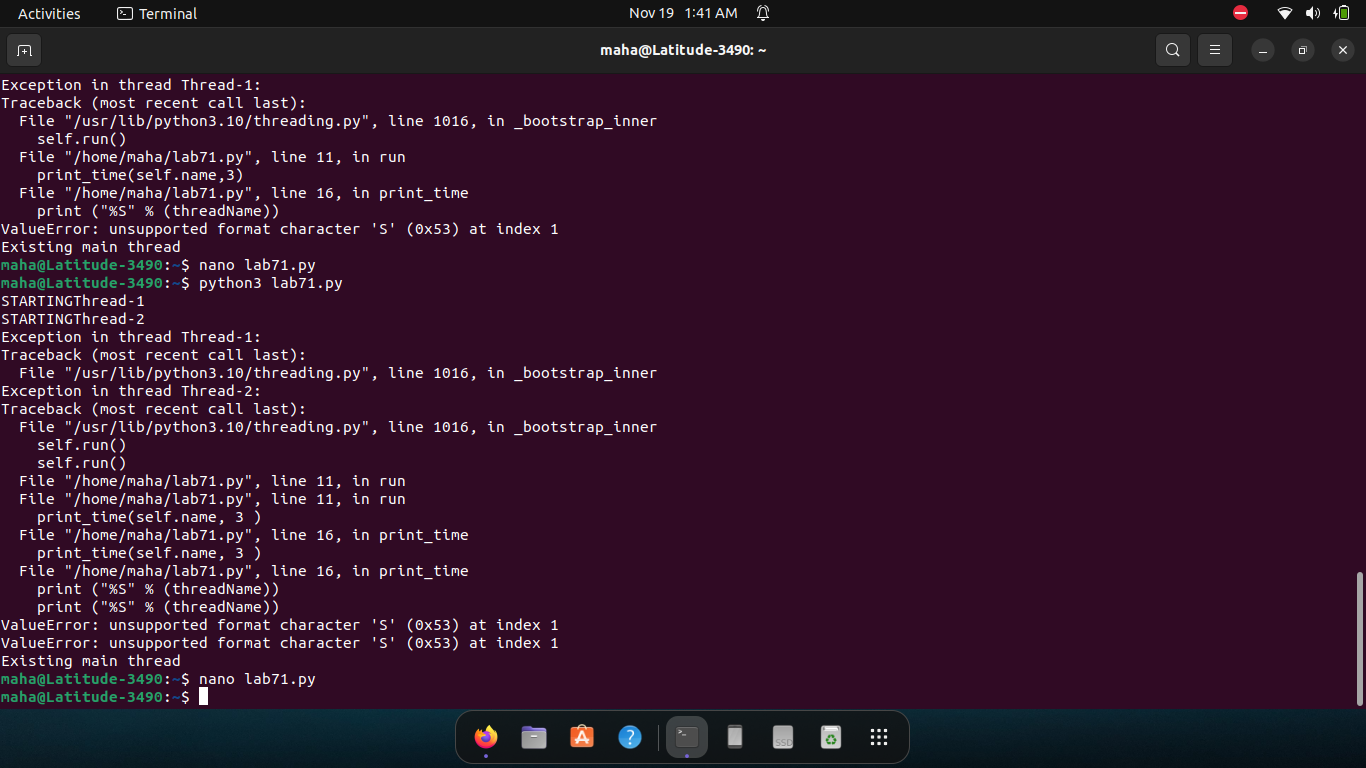
of the Thread class.

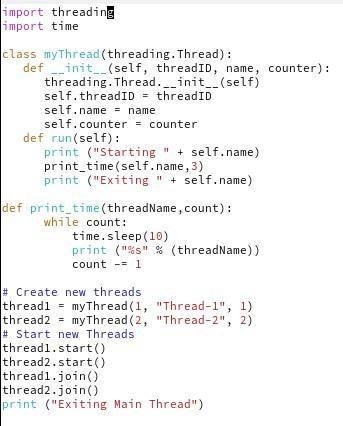
Override the init\_\_(self [,args]) method to add additional arguments.

Then, override the run(self [,args]) method to implement what the thread should do when started. Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls the run() method.

**Example Program:**







**Limitations and difficulty in using Python for Multithreading**

GIL (global interpreter lock), the mechanism used by the CPython interpreter to assure that only one thread executes Python bytecode at a time. Due to the GIL only one thread can be executed at a time. Therefore, the above code is concurrent but not parallel. Multi-Threading in python can be considered as an appropriate model only if you want to run multiple I/O-bound tasks simultaneously.

Using the threading module in Python or any other interpreted language with a GIL can actually result in reduced performance. If your code is performing a CPU bound task, using the threading module will result in a slower execution time. For CPU bound tasks and truly parallel execution, we can use the multiprocessing module instead of multithreading in Python.

**Thread Creation in C**

Normally when a program starts up and becomes a process, it starts with a default thread. So, we can say that every process has at least one thread of control.

C does not contain any built-in support for multithreaded applications, so POSIX (pthreads) can be used to write multi-threaded C program. 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.

The following routine is used to create a POSIX thread –

#include <pthread.h>

pthread\_create (thread, attr, start\_routine, arg)

| **Parameter** | **Description** |
| --- | --- |
| thread | An opaque, unique identifier (pthread\_t type address) for the new thread returned by the subroutine. So the first argument will hold the thread ID of the newly created thread. |
| 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 third argument is a function pointer. The C routine that the thread will execute once it is created. This is something to keep in mind that each thread starts with a function and that functions address is passed here as the third argument so that the kernel knows which function to start the thread f rom. |
| arg | A single argument that may be passed to start\_routine. As the function (whose address is passed in the third argument above) may accept some arguments also so we can pass these arguments in form of a pointer to a void type. Now, why a void type was chosen? This was because if a function accepts more than one argument then this pointer could be a pointer to a structure that may contain these arguments. |

**Example C program:**

#include <stdio.h>

#include <pthread.h> #include <stdlib.h>

void \*print\_message\_function( void \*ptr ); int main()

{

int status;

char \*msg1 = "Thread 1";

char \*msg2 = "Thread 2";

pthread\_t tid1,tid2; pthread\_create(&tid1,NULL,myfunc,(void\*)msg1); pthread\_create(&tid2,NULL,myfunc,(void\*)msg2); pthread\_join(tid1,NULL); pthread\_join(tid2,NULL); return 0;

}

void \*myfunc ( void \*ptr )

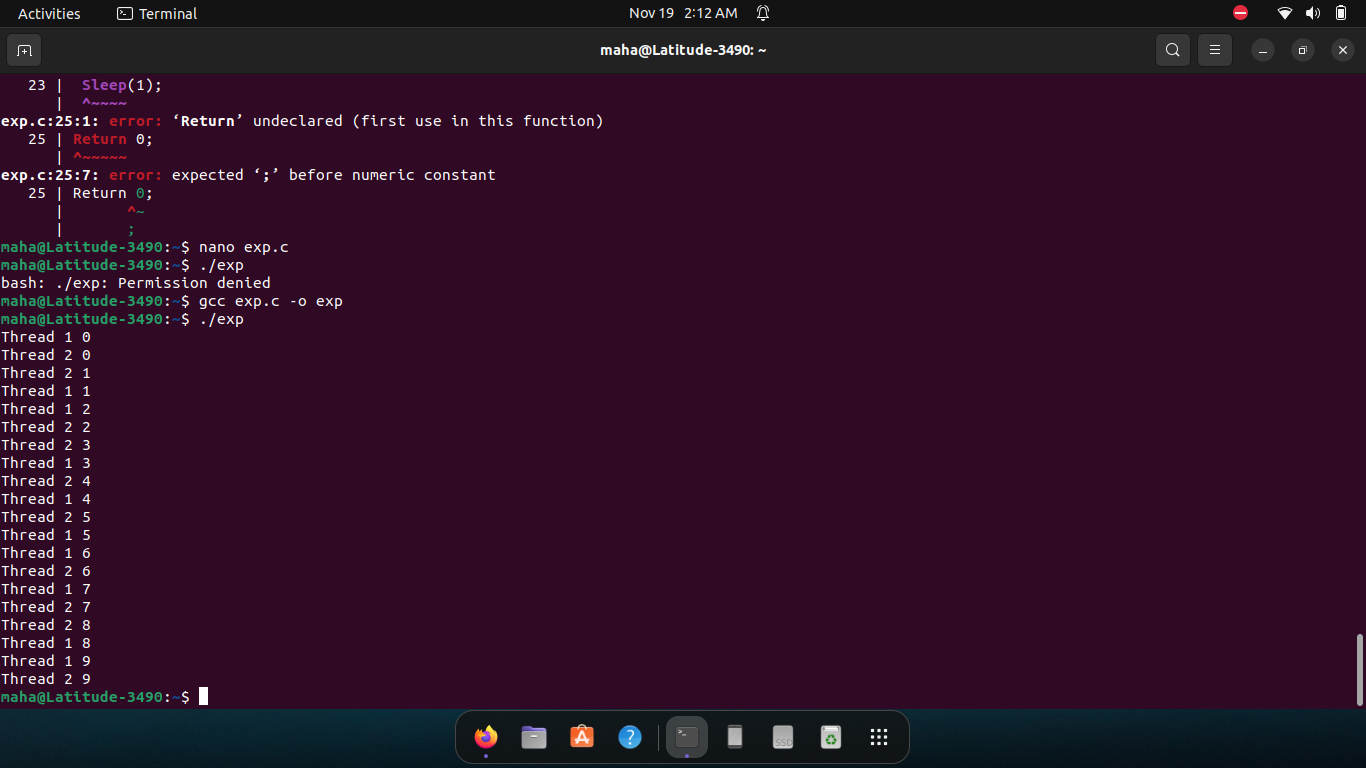
{

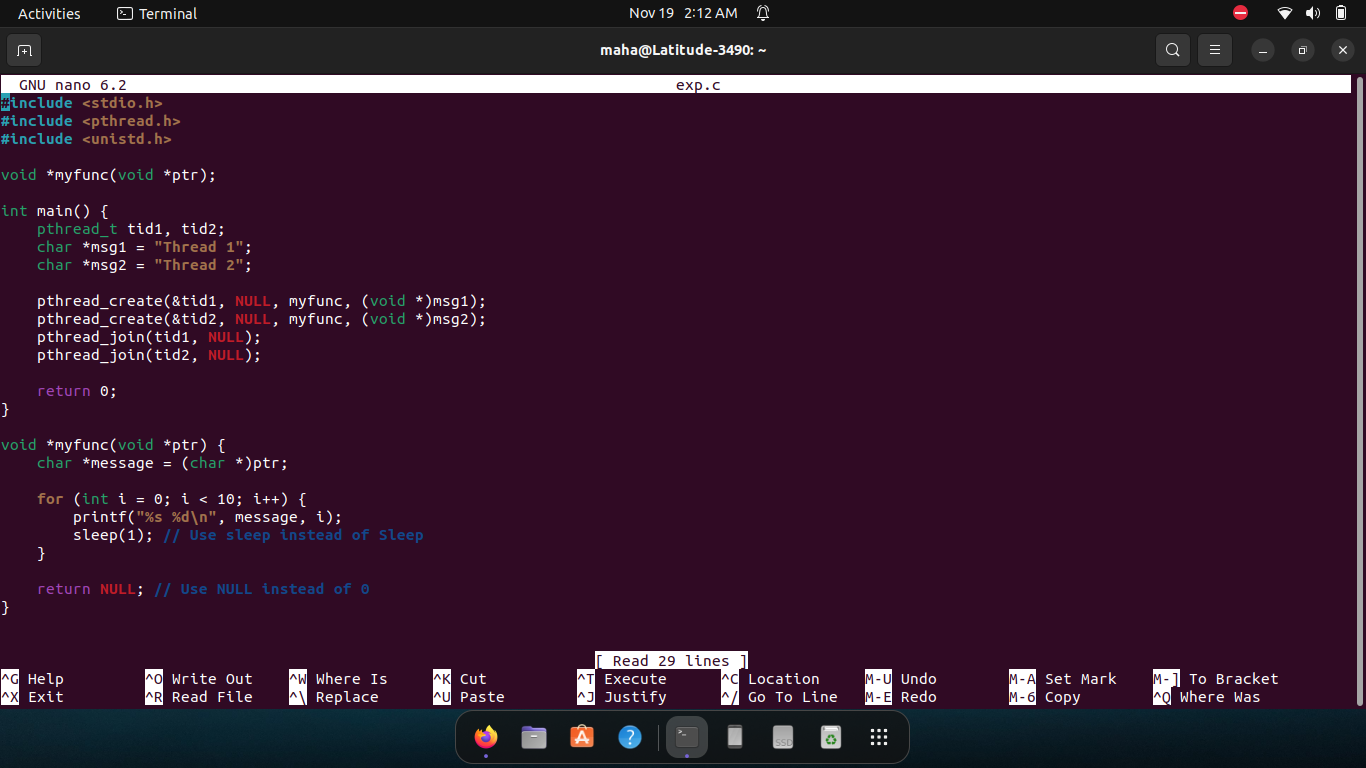
char \*message; message = (char \*) ptr; for

(int i=0; i<10;i++)

{ printf(“%s %d\n”, message,i);

Sleep(1); } Return 0;

}



**Compile C program.**

To compile a multithreaded program using gcc, we need to link it with the pthreads library. Following is the command used to compile the program.

**gcc -o hello hello.c -lpthread**

This command will invoke the GNU C compiler to compile the file hello.c and output (-o) the result to an executable called hello **Execute the program.**

Type the command

./hello

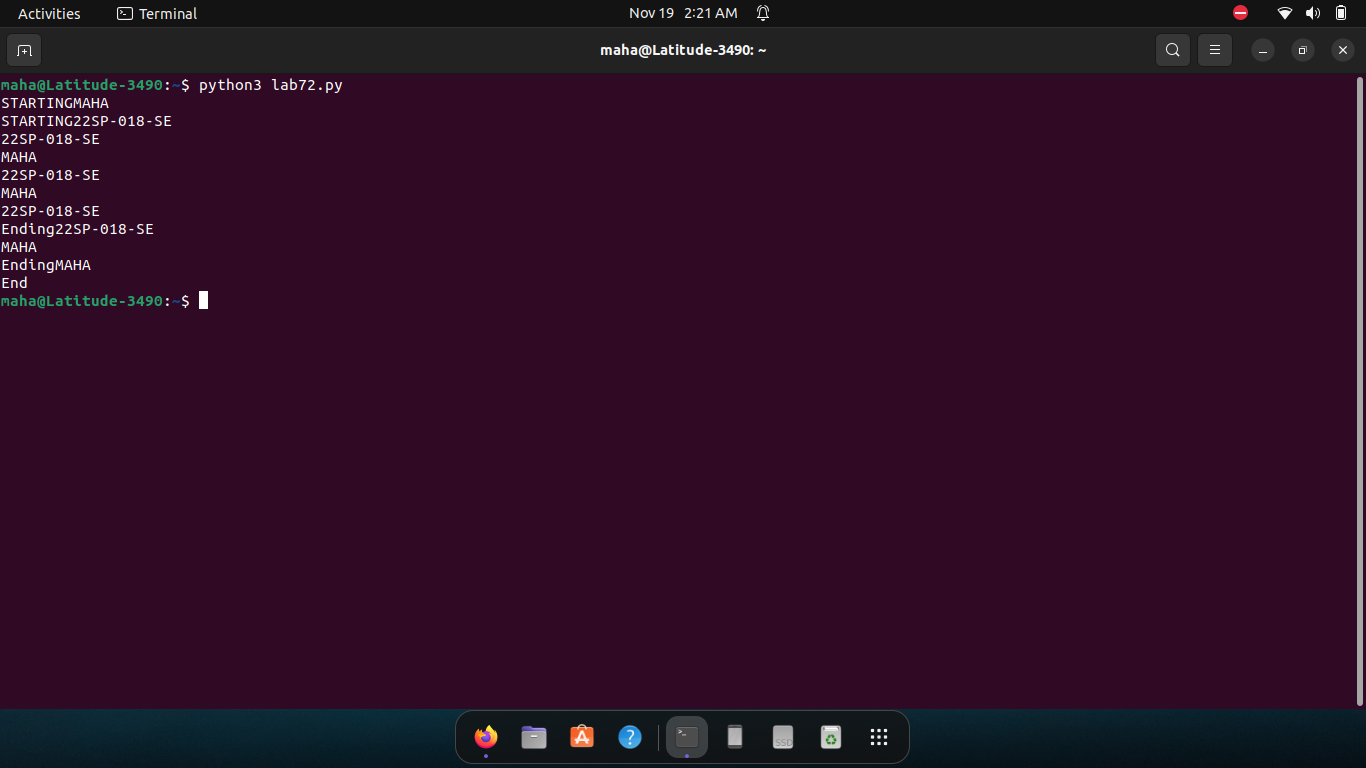
This should result in the output

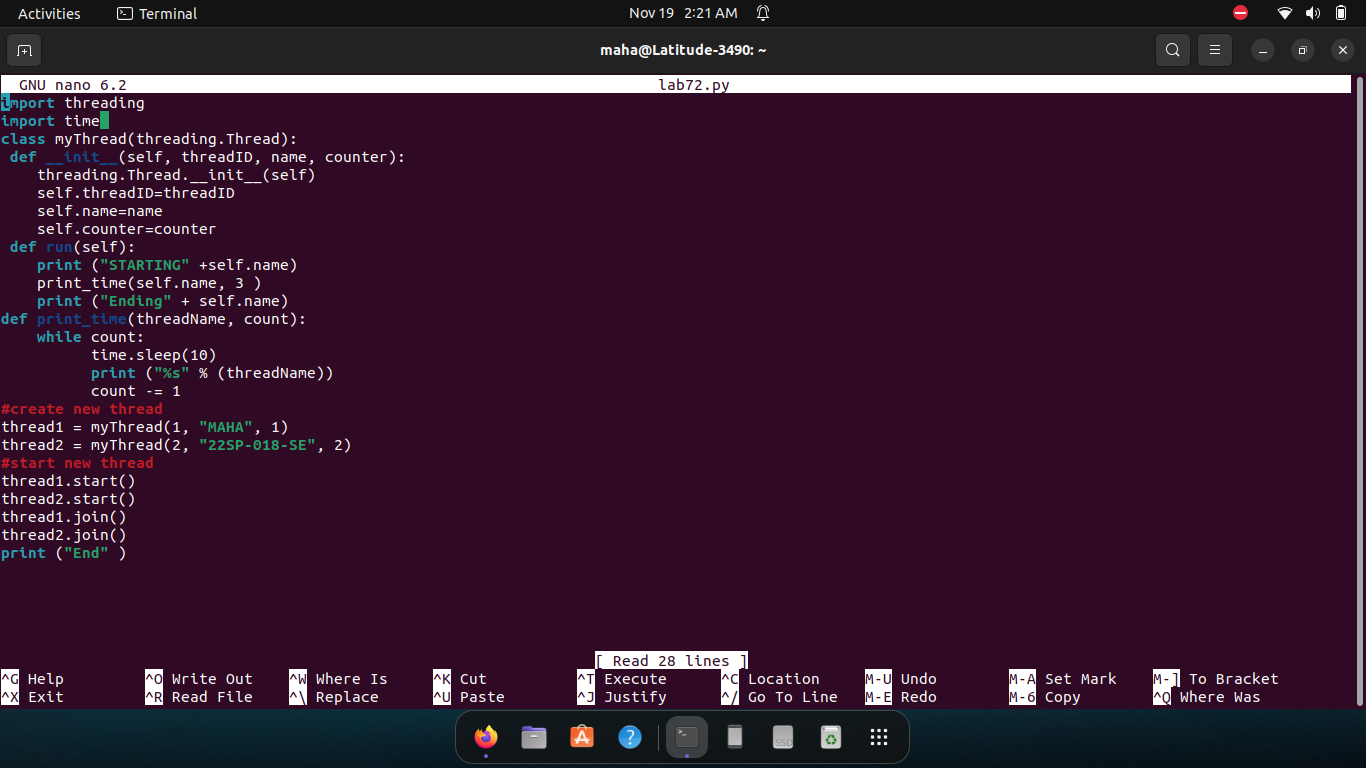
Hello World

**Lab Exercise(s):**

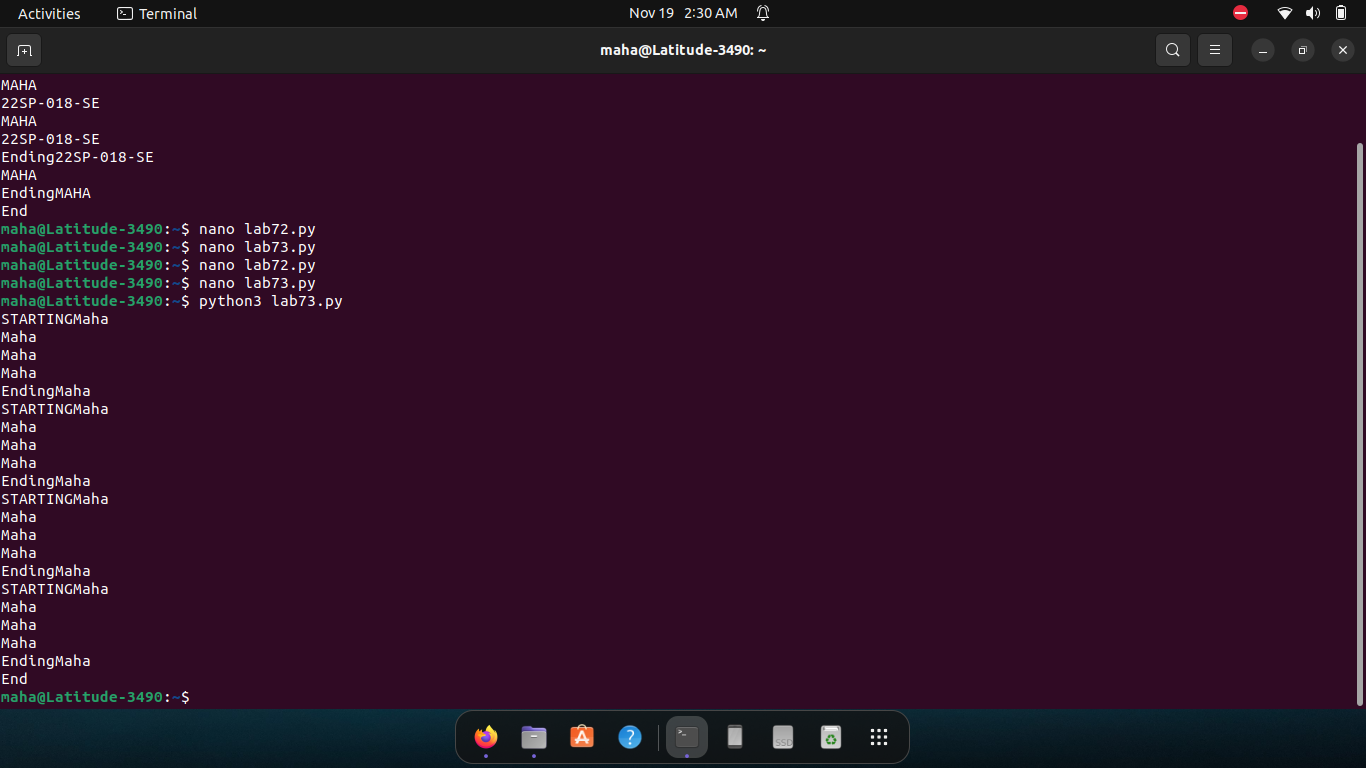
1. Modify Example 1 to display strings via two independent threads:

thread1: “Hello ! StudentName ”, thread 2: “Student roll no is : \_”





2. Create threads message as many times as user wants to create threads by using array of threads and loop.

Threads should display message that is passed through argument.

