

# Lab # 5

## Threading

A thread is a sequence of such instructions within a program that can be executed independently of other code.

### A. Identifying a thread

Each thread identified by an ID, which is known as Thread ID. Thread ID is quite different from Process ID. A Thread ID is unique in the current process, while a Process ID is unique across the system.

Thread ID is represented by type `pthread_t`

#### a) Header file(s)

The header file which needs to be included to access thread functions

**#include<pthread.h>**

Then, how to compile C program with pthread.h library?

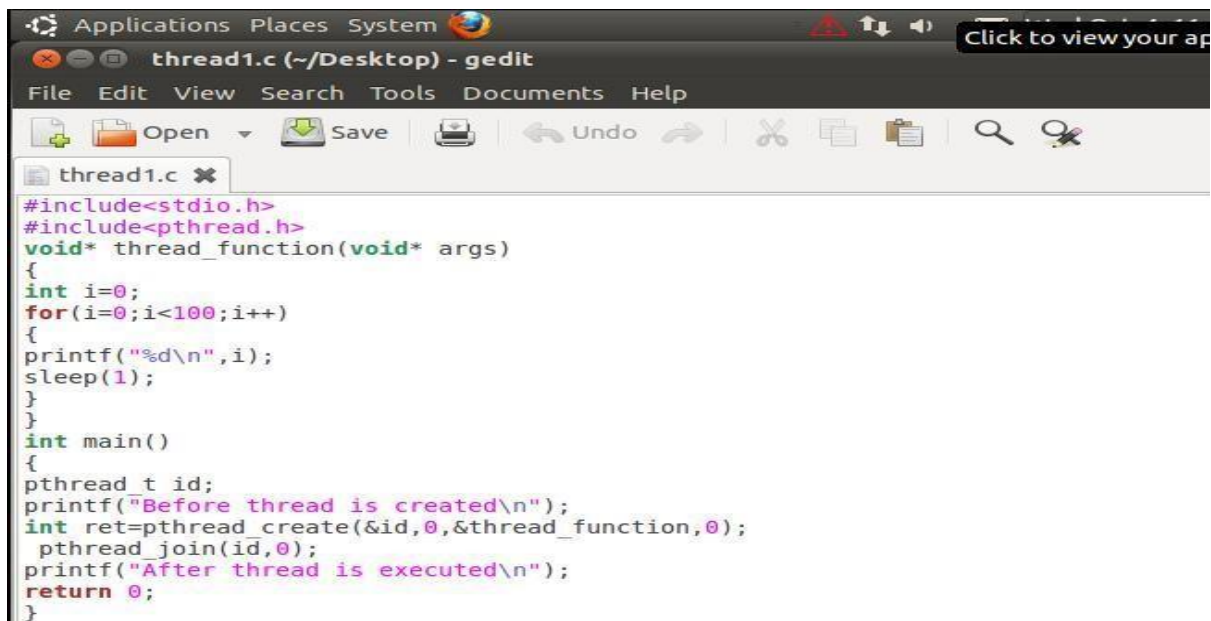
To compile C program with pthread.h library, you have to put `-lpthread` just after the compile command `gcc thread.c -o thread`, this command will tell to the compiler to execute program with pthread.h library.

#### b) The command is:

`gcc thread.c -o thread -lpthread`

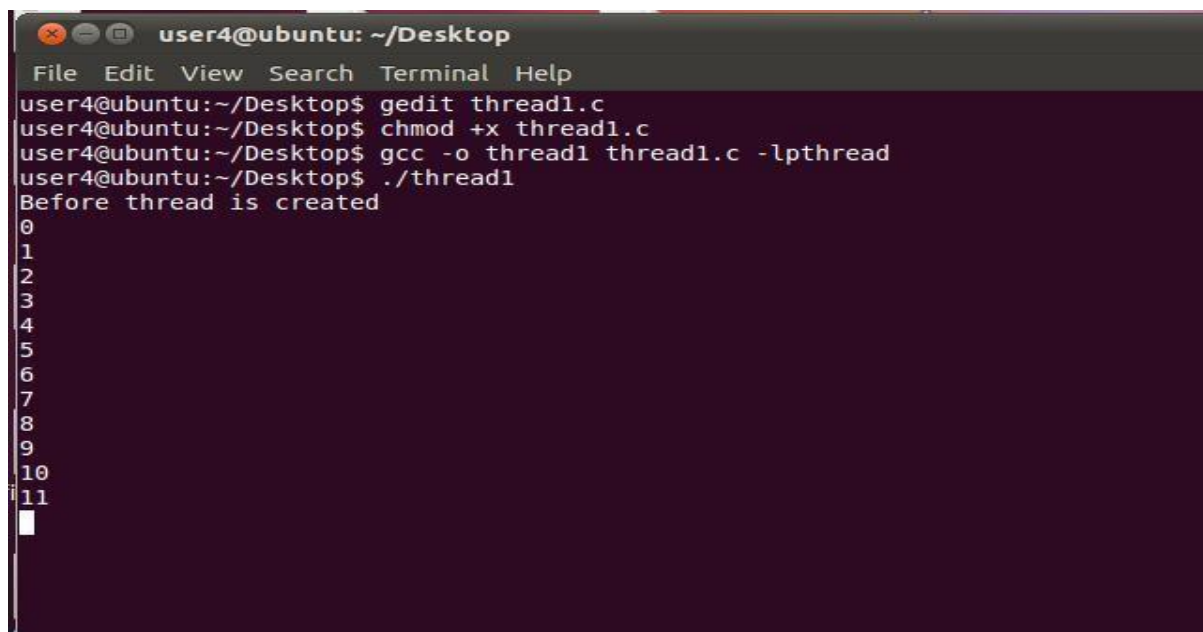
- **gcc** is the compiler command.
- **thread.c** is the name of c program source file.
- **-o** is option to make object file (o/p file or executable file).
- **thread** is the name of object file.
- **-lpthread** is option to execute pthread.h library file.

- 1) Create a thread and display numbers from 1 to 100;



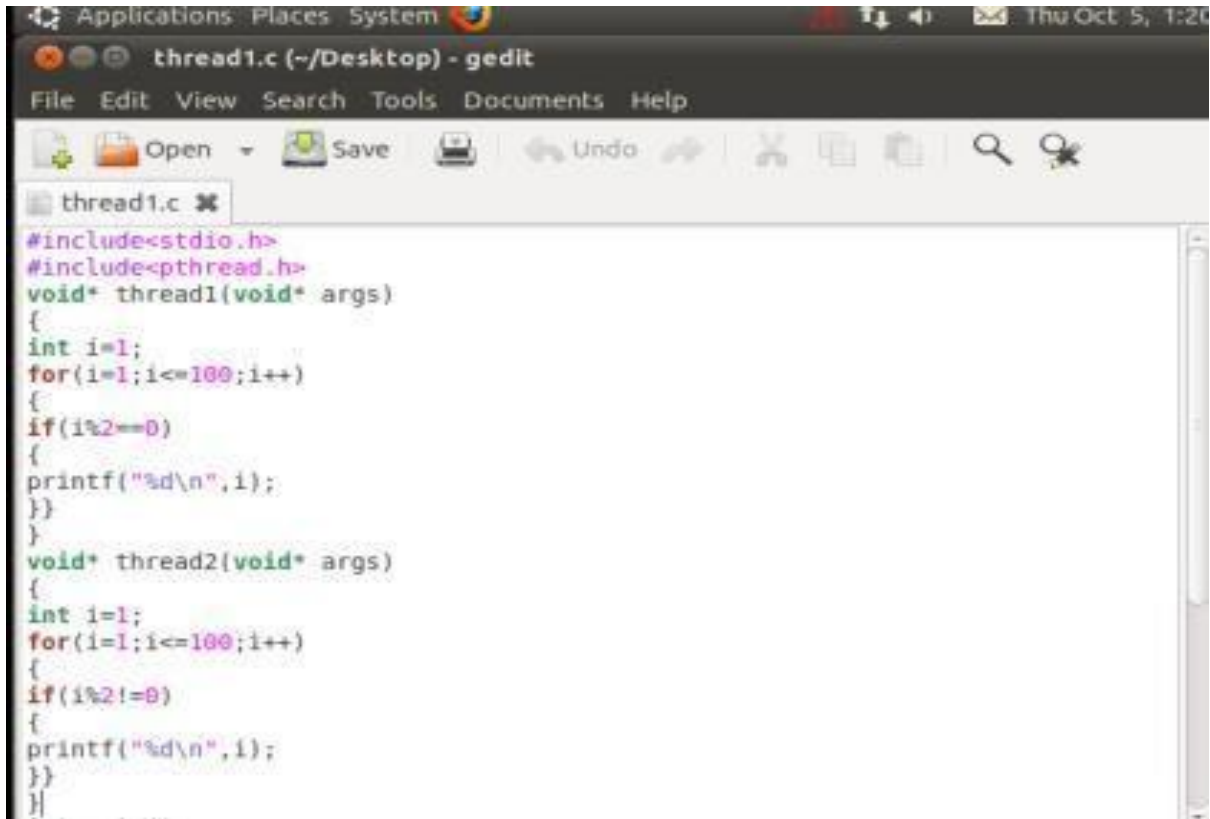
```
#include<stdio.h>
#include<pthread.h>
void* thread_function(void* args)
{
    int i=0;
    for(i=0;i<100;i++)
    {
        printf("%d\n",i);
        sleep(1);
    }
}
int main()
{
    pthread_t id;
    printf("Before thread is created\n");
    int ret=pthread_create(&id,0,&thread_function,0);
    pthread_join(id,0);
    printf("After thread is executed\n");
    return 0;
}
```

## OUTPUT



```
user4@ubuntu: ~/Desktop
File Edit View Search Terminal Help
user4@ubuntu:~/Desktop$ gedit thread1.c
user4@ubuntu:~/Desktop$ chmod +x thread1.c
user4@ubuntu:~/Desktop$ gcc -o thread1 thread1.c -lpthread
user4@ubuntu:~/Desktop$ ./thread1
Before thread is created
0
1
2
3
4
5
6
7
8
9
10
11
```

- 2) Create two threads Thread1 and Thread2. Display even number in Thread1 and odd numbers in Thread2 from 1 to 100. Debug them and display their outputs by single stepping (running individual threads).



```
#include<stdio.h>
#include<pthread.h>
void* thread1(void* args)
{
    int i=1;
    for(i=1;i<=100;i++)
    {
        if(i%2==0)
        {
            printf("%d\n",i);
        }
    }
}
void* thread2(void* args)
{
    int i=1;
    for(i=1;i<=100;i++)
    {
        if(i%2!=0)
        {
            printf("%d\n",i);
        }
    }
}
```

## **B. Debugging Individual Threads**

GDB has the ability to debug individual threads, and to manipulate and examine them independently. This functionality is not enabled by default. To do so use set non-stop on and set target-async on. These can be added to .gdbinit. Once that functionality is turned on, GDB is ready to conduct thread debugging.

For example, the following program creates two threads.

## OUTPUT

```
user4@ubuntu: ~/Desktop
File Edit View Search Terminal Help
user4@ubuntu:~/Desktop$ gedit thread1.c
user4@ubuntu:~/Desktop$ chmod +x thread1.c
user4@ubuntu:~/Desktop$ gcc -o thread1 thread1.c -lpthread
user4@ubuntu:~/Desktop$ gdb ./thread1
GNU gdb (GDB) 7.2-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/user4/Desktop/thread1...(no debugging symbols found).
..done.
(gdb)
```

First set breakpoints on all thread functions; thread1, thread2, and main.

```
..done.
(gdb) break thread1
Breakpoint 1 at 0x80484da
(gdb) break thread2
Breakpoint 2 at 0x804851a
(gdb) break main
Breakpoint 3 at 0x8048557
(gdb)
```

Then run the program.

```
(gdb) run
Starting program: /home/user4/Desktop/thread1
[Thread debugging using libthread_db enabled]

Breakpoint 3, 0x08048557 in main ()
(gdb)
```

```

(gdb) info thread
* 1 Thread 0xb7ff0b30 (LWP 3240) 0x08048557 in main ()
(gdb) next
Single stepping until exit from function main,
which has no line number information.
E
[ Breakpoint 1, 0x080484da in thread1 ()
(gdb) next
Single stepping until exit from function thread1,
which has no line number information.
E
2
4
6
8
10
0x00134cc9 in start_thread () from /lib/libpthread.so.0
(gdb) next
Single stepping until exit from function start_thread,
which has no line number information.
After thread1 is executed
Before thread2 is created
[Thread 0xb7feeb70 (LWP 3243) exited]
[New Thread 0xb7feeb70 (LWP 3246)]
[Switching to Thread 0xb7feeb70 (LWP 3246)]

Breakpoint 2, 0x0804851a in thread2 ()
(gdb) █

62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
0x00134cc9 in start_thread () from /lib/libpthread.so.0
(gdb)

```

Fig. 1 Thread1: To display even numbers from 1-100

### **C. To write a c program to implement Threading and Synchronization Applications.**

#### **ALGORITHM:**

- Step 1: Start the process
- Step 2: Declare process thread, thread-id.
- Step 3: Read the process thread and thread state.
- Step 4: Check the process thread equals to thread-id by using if condition.
- Step 5: Check the error state of the thread.
- Step 6: Display the completed thread process.

Step 7: Stop the process

**PROGRAM:**

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_t tid[2];
void* doSomething(void *arg)
{
    unsigned long i = 0;
    pthread_t id = pthread_self();
    if(pthread_equal(id,tid[0]))
    {
        printf("\n First thread processing\n");
    }
    else
    {
        printf("\n Second thread processing\n");
    }
    for(i=0; i<(0xFFFFFFFF);i++);
    return NULL;
}
int main(void)
{
    int i = 0;
    int err;
    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &doSomething, NULL);
        if (err != 0)
        {
            printf("\n can't create thread :[%s]", strerror(err));
        }
        else
        {
            printf("\n Thread created successfully\n");
        }
        i++;
    }
    sleep(5);
    return 0;
}
```

## **D. User Management:**

1. Checking all/specific user accounts  
\$ lslogins < --user-accs/--system-accs>  
\$ lslogins <username>  
Example: \$ lslogins bilal
  
2. Locating executables files  
\$ which <filename>  
Example: \$ which find  
Example: \$ which  
Example: \$ which find  
Which command show that where is the file in the directory.
  
3. Finding binary and source files for commands  
\$ whereis <name>  
Example: \$ whereis find  
Example: \$ whereis cat
  
4. To show currently logged in users in the system  
\$ who  
\$ w  
\$ users  
\$ last  
\$ top
  
5. To show current user  
\$ whoiam
  
6. To Substitute user with user name  
\$ su -<username> (if logged in as root then no password is required)  
Example \$ su -bilal
  
7. To Substitute user as root  
\$ su - root
  
8. To execute command with Substitute user  
\$ su - c <command> - s <shell> <user>  
Example \$ su - c "echo Hello World" -s /bin/bash bilal

## **D. touch Command:**

1. Create file using touch command

```
$ touch <-c> <filename1> <filename2> <filename3>
```

-c : to suppress creation if file does not exist

-t : to create file using specified date and time

Example: \$ touch OSLab5.1 OSLab5.2

2. Create file using touch command by specifying date & time

```
$ touch <-t> <YYMMDDHHMM> <filename1>
```

-t : to create file using specified date and time

Example: \$ touch -t 2401231513 OSLab5.3

3. Updating access and modification timings

```
$ touch <-c-d|-a|-m|-d|-r> <filename1>
```

-c-d : to update access and modification time

-a : to update access time only

-m : to update last modification time only

-d : to update modification date only

-r : uses timestamp of one file to the other one

Example: \$ touch -d "05 May 2023" test.txt

\$ touch -r source.txt destination.txt



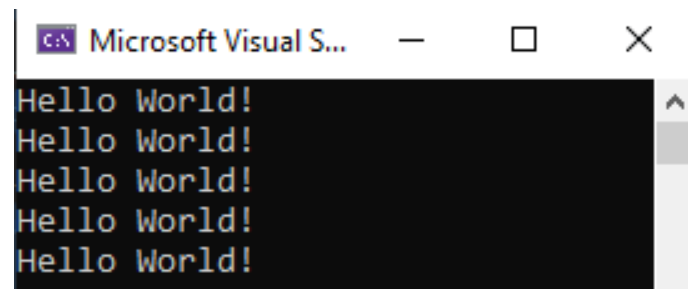
# Lab Tasks

1. Create a file OSLabFile5.1 with date & time 25<sup>th</sup> January, 2025 3:00 PM.
2. Create a file OSLab5.4 dated 01 Jan 2024 and copy its date to OSLab5.1
3. Find the binary and source file location of “ls” command
4. Create a user as testuser, and then execute command for creating file OSLabFile5.5 using testuser and shell /bin/bash
5. Creates 5 threads each thread prints a "Hello World!" message, and then terminates.

## PROGRAM

```
using System;
using System.Threading;
public class MyThread
{
    public void Thread1()
    {
        Console.WriteLine("Hello World!");
    }
}
public class ThreadExample
{
    public static void Main()
    {
        MyThread mt = new MyThread();
        Thread t1 = new Thread(new ThreadStart(mt.Thread1));
        Thread t2 = new Thread(new ThreadStart(mt.Thread1));
        Thread t3 = new Thread(new ThreadStart(mt.Thread1));
        Thread t4 = new Thread(new ThreadStart(mt.Thread1));
        Thread t5 = new Thread(new ThreadStart(mt.Thread1));
        t1.Start();
        t2.Start();
        t3.Start();
        t4.Start();
        t5.Start();
    }
}
```

## OUTPUT



A screenshot of a Microsoft Visual Studio console window. The window has a title bar with the text "Microsoft Visual S..." and standard minimize, maximize, and close buttons. The console area has a black background with yellow text. It displays five lines of "Hello World!". A vertical scrollbar is visible on the right side of the console area.

```
Hello World!  
Hello World!  
Hello World!  
Hello World!  
Hello World!
```