

# **OPERATING SYSTEMS LAB**

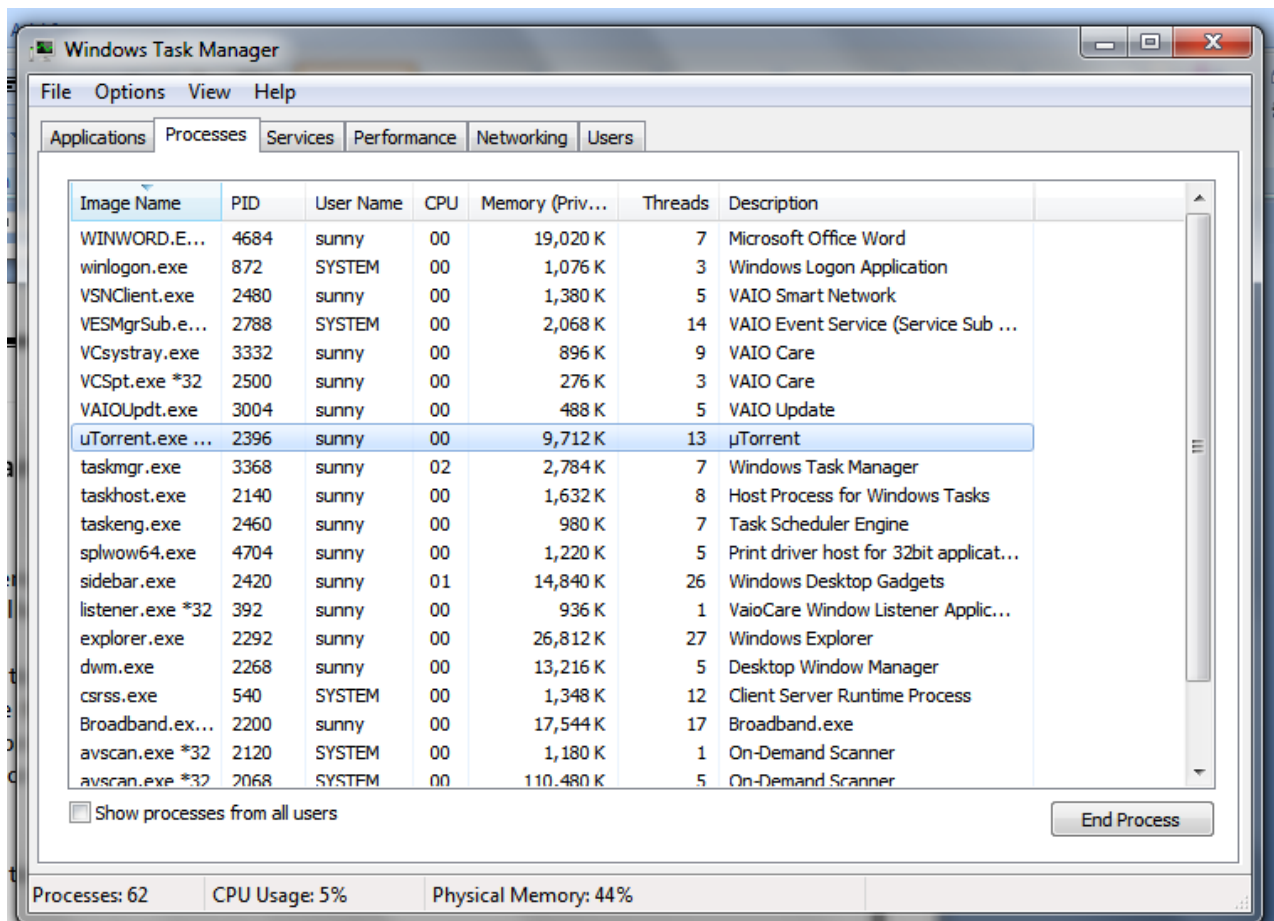
## **LAB # 10** **Threads**

## Simultaneous execution of two or more threads

### Threads:

A thread is a single sequence stream within a process. Threads are executed within a process. They are sometimes called *lightweight processes*. In a process, threads allow multiple executions of streams.

Threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. Like a traditional process i.e., process with one thread, a thread can be in any of several states (Running, Blocked, Ready or terminated).



**Thread Programming:**

**pthread\_create** : It is used to create a new thread.

**Syntax** : `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine, void*), void *arg);`

**Pthread\_t** : It is defining a thread pointer. When a thread is created identifier is written into the variable to which the pointer points. *This identifier helps to refer to thread.*

**Pthread\_attr\_t** : It is used to set the thread attributes. Usually there is no need to set the thread attributes. Pass this argument as NULL.

**Name of function** : The name of the function to be started by the thread for execution.

**Arguments to be passed to the function**: When a new thread is created it executes the function pointed by the function variable name.

**pthread\_join**: It is used to wait for the thread represented in the thread\_join call. It waits for the thread represented in the call to finish executing. It waits for the specified thread to complete, and gathers information about the thread's exit status.

**Syntax** : `int pthread_join(pthread_t thread, void **threadreturn);`

First argument is the thread for which to wait. Second argument is pointer that points to return value from the thread.

**Pthread\_exit**: This function is used to terminate the calling thread.

Syntax : `void pthread_exit(void *retval);`

**Example:**

```

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

void *print_message_function( void *ptr );

main()
{
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;

    /* Create independent threads each of which will execute function */

    iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
    iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

    /* Wait till threads are complete before main continues. Unless we */
    /* wait we run the risk of executing an exit which will terminate */
    /* the process and all threads before the threads have completed. */

    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

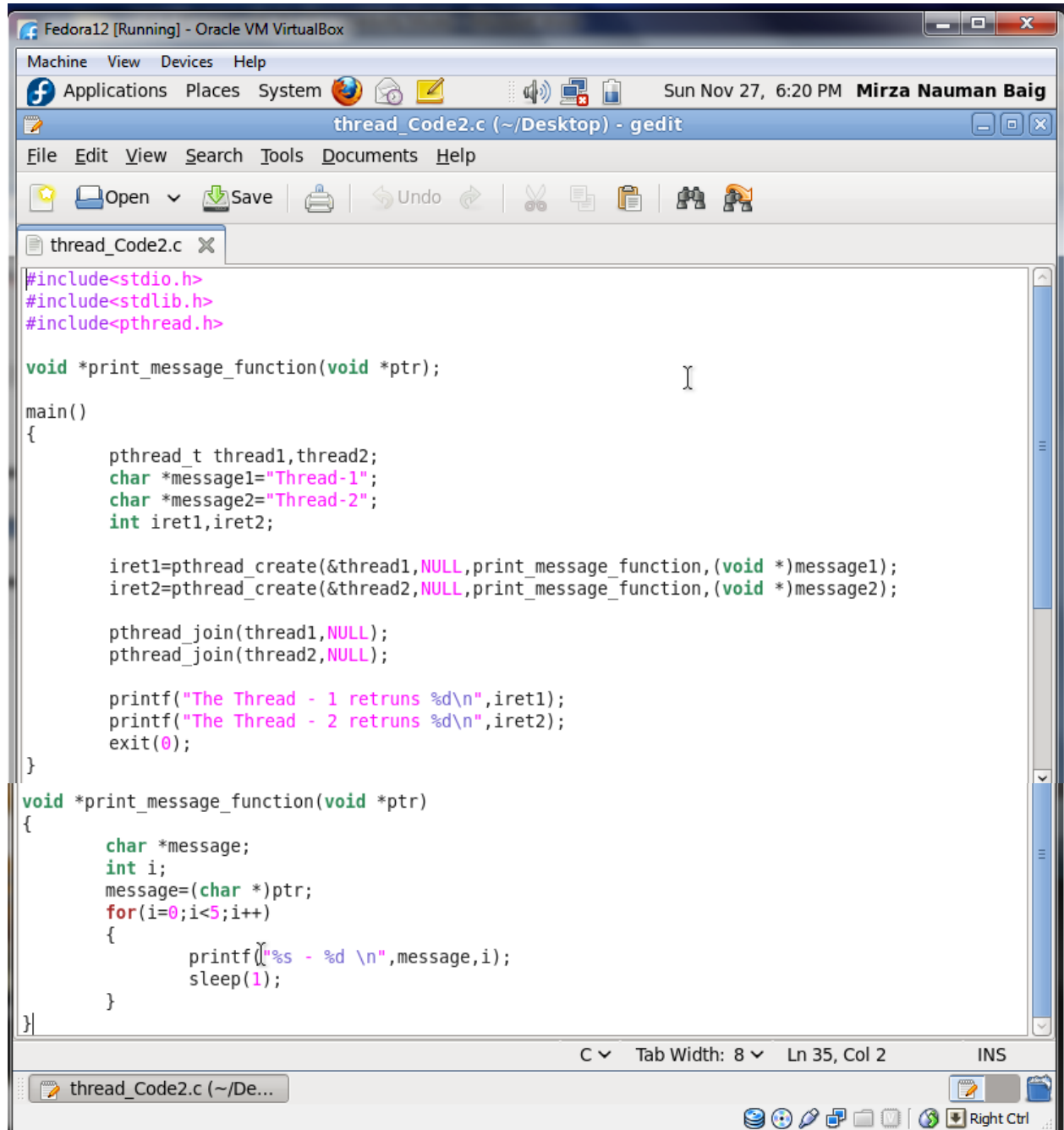
    printf("Thread 1 returns: %d\n",iret1);
    printf("Thread 2 returns: %d\n",iret2);
    exit(0);
}

void *print_message_function( void *ptr )
{
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}

```

Compile: gcc -pthread -o a mt.c  
 Run: ./a.out

## Code :



The screenshot shows a gedit editor window titled 'thread\_Code2.c (~/.Desktop) - gedit'. The code is as follows:

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

void *print_message_function(void *ptr);

main()
{
    pthread_t thread1,thread2;
    char *message1="Thread-1";
    char *message2="Thread-2";
    int iret1,iret2;

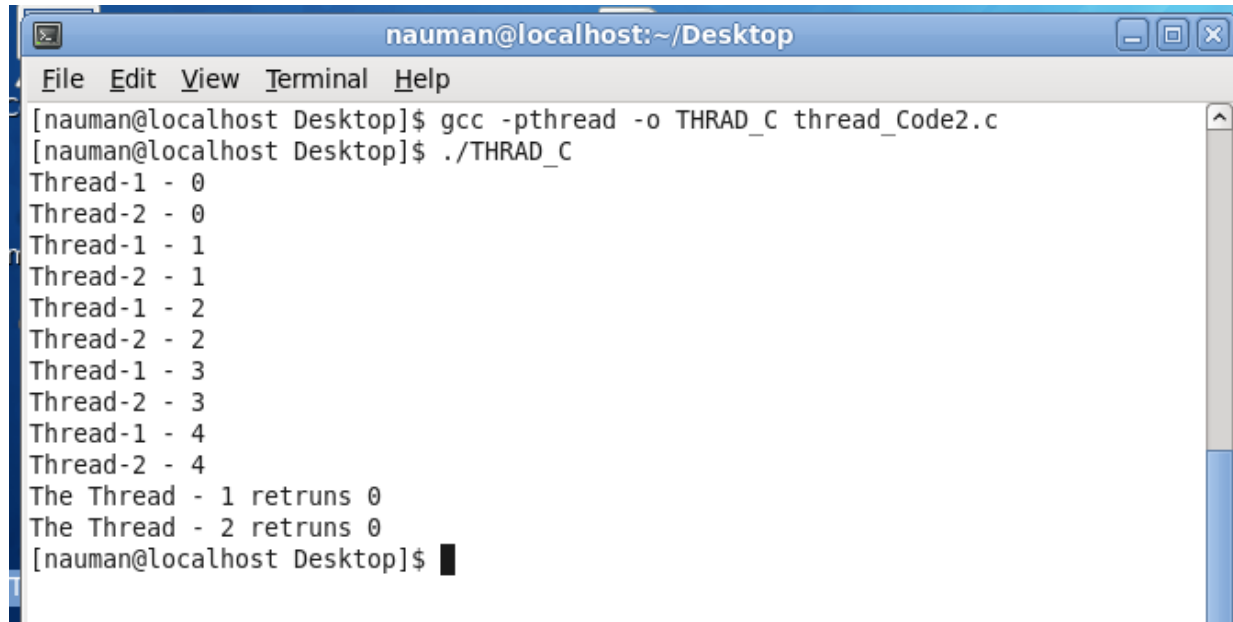
    iret1=pthread_create(&thread1,NULL,print_message_function,(void *)message1);
    iret2=pthread_create(&thread2,NULL,print_message_function,(void *)message2);

    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);

    printf("The Thread - 1 retruns %d\n",iret1);
    printf("The Thread - 2 retruns %d\n",iret2);
    exit(0);
}

void *print_message_function(void *ptr)
{
    char *message;
    int i;
    message=(char *)ptr;
    for(i=0;i<5;i++)
    {
        printf("%s - %d \n",message,i);
        sleep(1);
    }
}
```

The editor window includes a menu bar (File, Edit, View, Search, Tools, Documents, Help), a toolbar with icons for Open, Save, Undo, and other functions, and a status bar at the bottom showing 'C', 'Tab Width: 8', 'Ln 35, Col 2', and 'INS'.

**OUTPUT :**A terminal window titled 'nauman@localhost:~/Desktop' with a menu bar (File, Edit, View, Terminal, Help). The terminal shows the compilation and execution of a C program using pthreads. The output displays two threads, Thread-1 and Thread-2, each printing values from 0 to 4. At the end, it shows 'The Thread - 1 retruns 0' and 'The Thread - 2 retruns 0'.

```
nauman@localhost Desktop]$ gcc -pthread -o THRAD_C thread_Code2.c
nauman@localhost Desktop]$ ./THRAD_C
Thread-1 - 0
Thread-2 - 0
Thread-1 - 1
Thread-2 - 1
Thread-1 - 2
Thread-2 - 2
Thread-1 - 3
Thread-2 - 3
Thread-1 - 4
Thread-2 - 4
The Thread - 1 retruns 0
The Thread - 2 retruns 0
nauman@localhost Desktop]$
```

**Comment Pthread\_join these lines and see what happened**

### Another CODE for understanding threads

```

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

void *print_message_ftn1();
void *print_message_ftn2();

main()
{
    pthread_t thread1,thread2;
    int iret1,iret2;

    iret1=pthread_create(&thread1,NULL,print_message_ftn1,NULL);
    iret2=pthread_create(&thread2,NULL,print_message_ftn2,NULL);

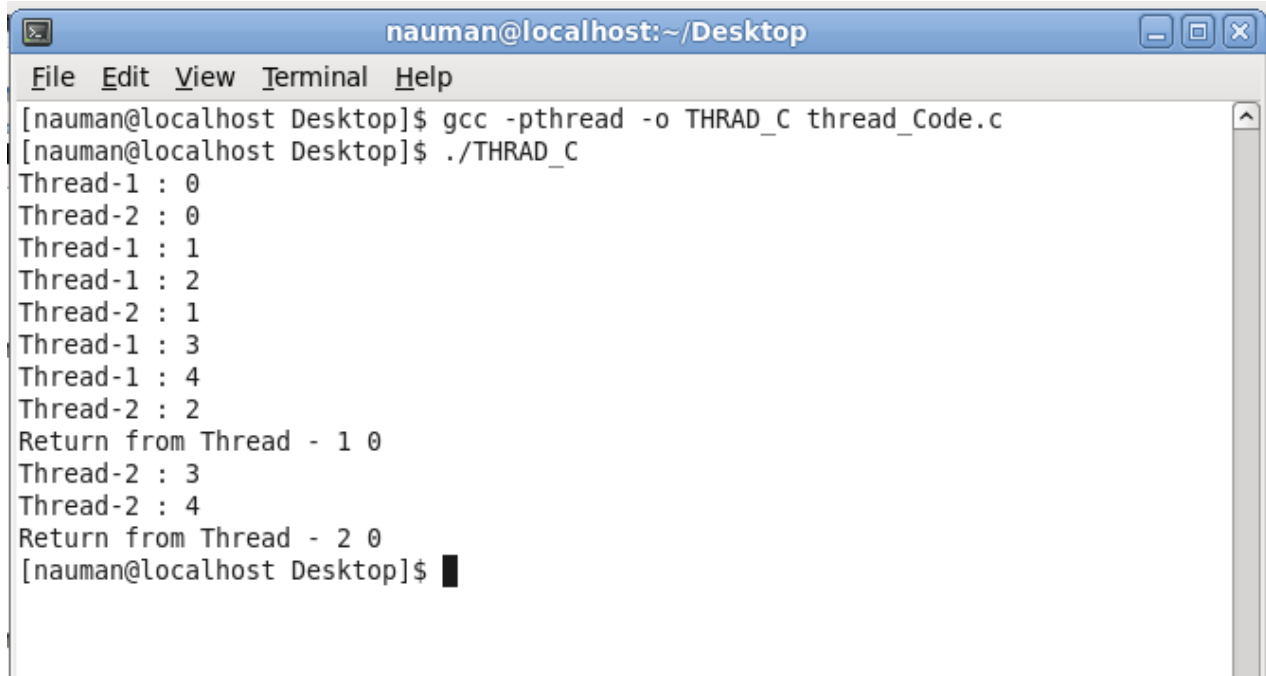
    pthread_join(thread1,NULL);
    printf("Return from Thread - 1 %d\n",iret1);
    pthread_join(thread2,NULL);
    printf("Return from Thread - 2 %d\n",iret2);

    exit(0);
}

void *print_message_ftn1()
{
    int i;
    for(i=0;i<5;i++)
    {
        printf("Thread-1 : %d \n",i);
        sleep(1);
    }
}

void *print_message_ftn2()
{
    int i;
    for(i=0;i<5;i++)
    {
        printf("Thread-2 : %d \n",i);
        sleep(2);
    }
}

```

**OUTPUT:**A terminal window titled 'nauman@localhost:~/Desktop' with a menu bar (File, Edit, View, Terminal, Help). The terminal shows the compilation and execution of a C program using pthreads. The output displays the progression of two threads, Thread-1 and Thread-2, with their respective values of 'i' and return codes.

```
nauman@localhost:~/Desktop
File Edit View Terminal Help
[nauman@localhost Desktop]$ gcc -pthread -o THRAD_C thread_Code.c
[nauman@localhost Desktop]$ ./THRAD_C
Thread-1 : 0
Thread-2 : 0
Thread-1 : 1
Thread-1 : 2
Thread-2 : 1
Thread-1 : 3
Thread-1 : 4
Thread-2 : 2
Return from Thread - 1 0
Thread-2 : 3
Thread-2 : 4
Return from Thread - 2 0
[nauman@localhost Desktop]$
```

**LAB TASK: Create 2 treads**

**Use while loop in thread 1 while( $i \leq 10$ ) and increment  $i$  in thread 2**

**Hint :** use pthread\_exit when work is done by thread