Threads

Lab 2

Marin Iuliana

Threads

- A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called *lightweight processes*.
- Threads (or Pthreads) is a POSIX standard for threads. Implementation of pthread is available with gcc compiler.

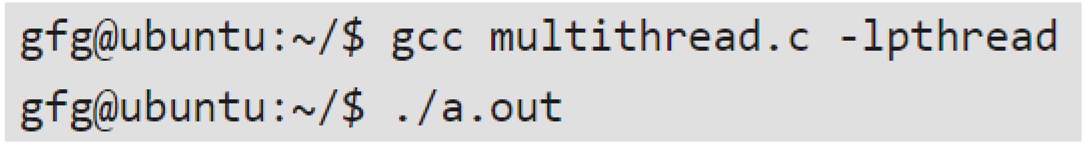
- **What are the differences between process and thread?**

  Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

USE OF PTHREAD BASIC FUNCTIONS

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //Header file for sleep(). man 3 sleep for details.
#include <pthread.h>

// A normal C function that is executed as a thread
// when its name is specified in pthread_create()
void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Printing UPB from Thread \n");
    return NULL;
}

int main()
{
    pthread_t thread_id;
    printf("Before Thread\n");
    pthread_create(&thread_id, NULL, myThreadFun, NULL);
    pthread_join(thread_id, NULL);
    printf("After Thread\n");
    exit(0);
}
```

In main() we declare a variable called thread_id, which is of type pthread_t, which is an integer used to identify the thread in the system. After declaring thread_id, we call pthread_create() function to create a thread.
pthread_create() takes 4 arguments.
The first argument is a pointer to thread_id which is set by this function. The second argument specifies attributes. If the value is NULL, then default attributes shall be used. The third argument is name of function to be executed for the thread to be created. The fourth argument is used to pass arguments to the function, myThreadFun. The pthread_join() function for threads is the equivalent of wait() for processes. A call to pthread_join blocks the calling thread until the thread with identifier equal to the first argument terminates.

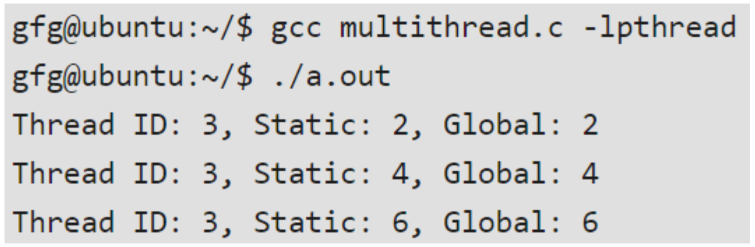


```
gfg@ubuntu:~/$ gcc multithread.c -lpthread
gfg@ubuntu:~/$ ./a.out
```

MULTIPLE THREADS WITH GLOBAL AND STATIC VARIABLES

- Global and static variables are stored in data segment. Therefore, they are shared by all threads.

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

// Let us create a global variable to change it in threads
int g = 0;

// The function to be executed by all threads
void *myThreadFun(void *vargp)
{
    // Store the value argument passed to this thread
    int *myid = (int *)vargp;

    // Let us create a static variable to observe its changes
    static int s = 0;

    // Change static and global variables
    ++s; ++g;

    // Print the argument, static and global variables
    printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}

int main()
{
    int i;
    pthread_t tid;

    // Let us create three threads
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, myThreadFun, (void *)&tid);

    pthread_exit(NULL);
    return 0;
}
```



```
gfg@ubuntu:~/$ gcc multithread.c -lpthread
gfg@ubuntu:~/$ ./a.out
Thread ID: 3, Static: 2, Global: 2
Thread ID: 3, Static: 4, Global: 4
Thread ID: 3, Static: 6, Global: 6
```

EXERCISES

1) Consider the case when a person introduces her/his personal details and based on a provided amount of money which the person has, multiple threads compute how much the person can earn if the money are deposited for maximum 10 years at a bank where the interest is of 0.7% per month. Solve the problem using global and static variables.
2) Write a program based on threads which will compute the number of occurrences of a specific word inside multiple files that exist at a provided path.