Mawlana Bhashani Science and Technology University

Lab-Report

Report No: 03

Course code: ICT-3110

Course title: Operating Systems Lab

Date of Performance :

Date of Submission : 30/09/2020

Submitted by

Name: Md Humayun Kabir

ID:IT-18026

3rd year 1stsemester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Experiment No : 03

Experiment Name : Threads on Operating System.

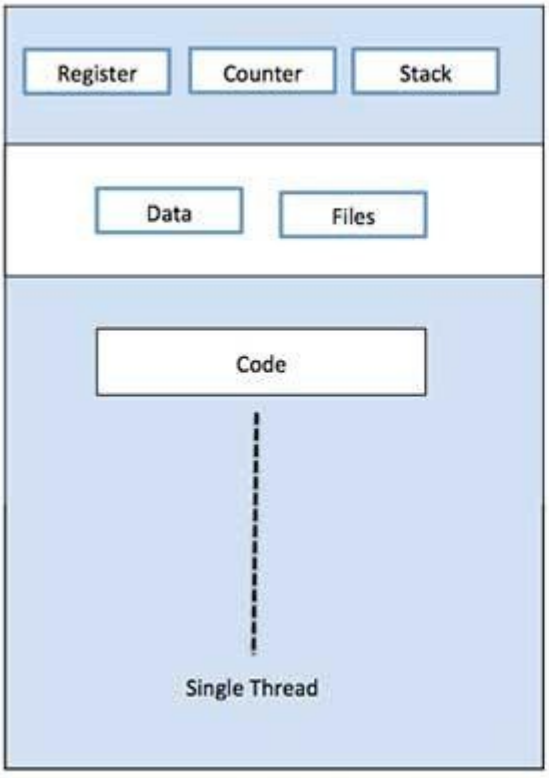
Theory :

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

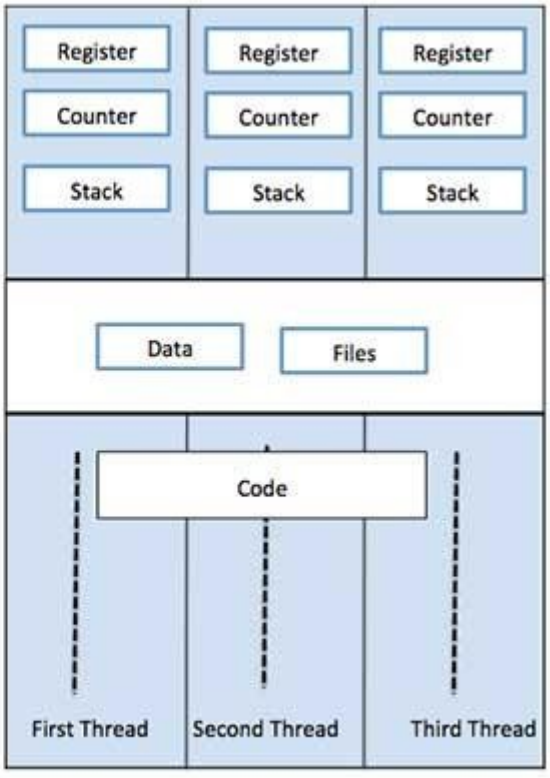
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Types of Thread :

Threads are implemented in following two ways –

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

User Level Threads : In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Kernel Level Threads : In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Working process :

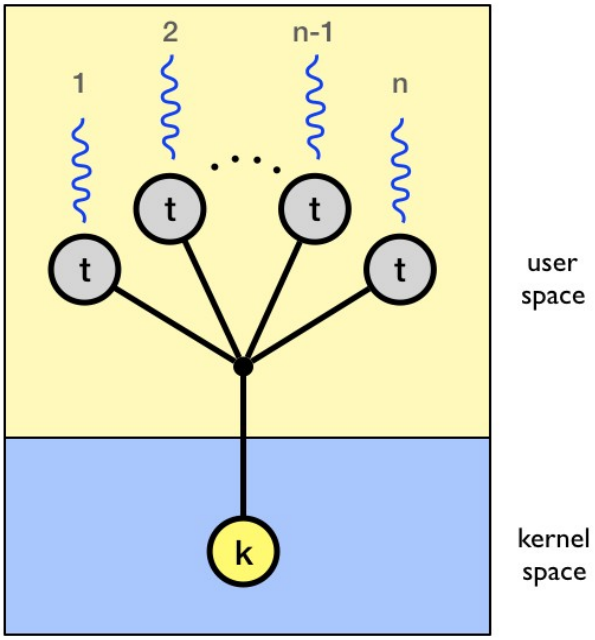
In general, user-level threads can be implemented using one of four models.

- Many-to-one
- One-to-one
- Many-to-many

- Two-level

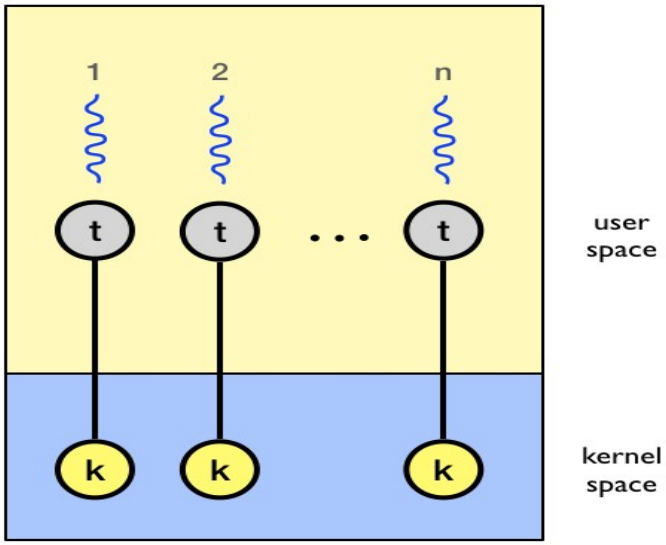
All models maps user-level threads to kernel-level threads. A **kernel thread** is similar to a process in a non-threaded (single-threaded) system. The kernel thread is the unit of execution that is scheduled by the kernel to execute on the CPU. The term **virtual processor** is often used instead of kernel thread.

Many to one : In the many-to-one model all user level threads execute on the same kernel thread. The process can only run one user-level thread at a time because there is only one kernel-level thread associated with the process.



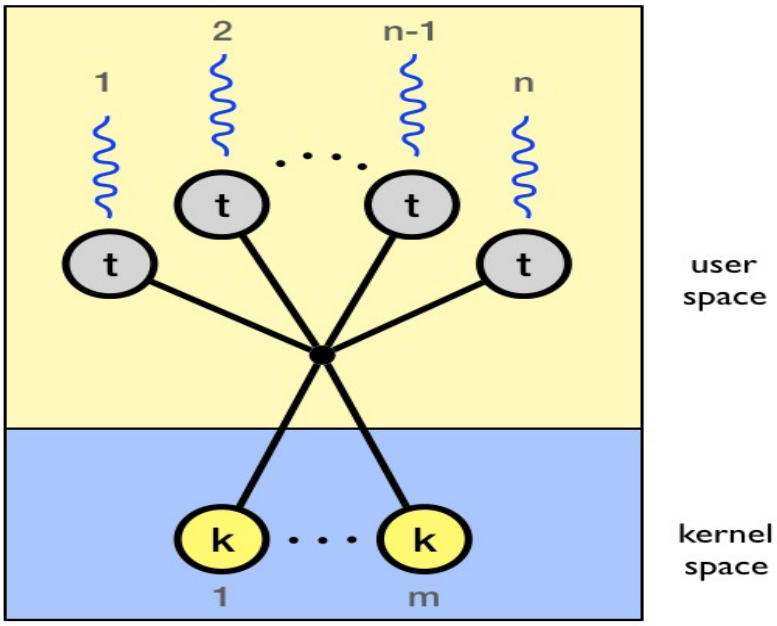
The kernel has no knowledge of user-level threads. From its perspective, a process is an opaque black box that occasionally makes system calls.

One to one : In the one-to-one model every user-level thread execute on a separate kernel-level thread.

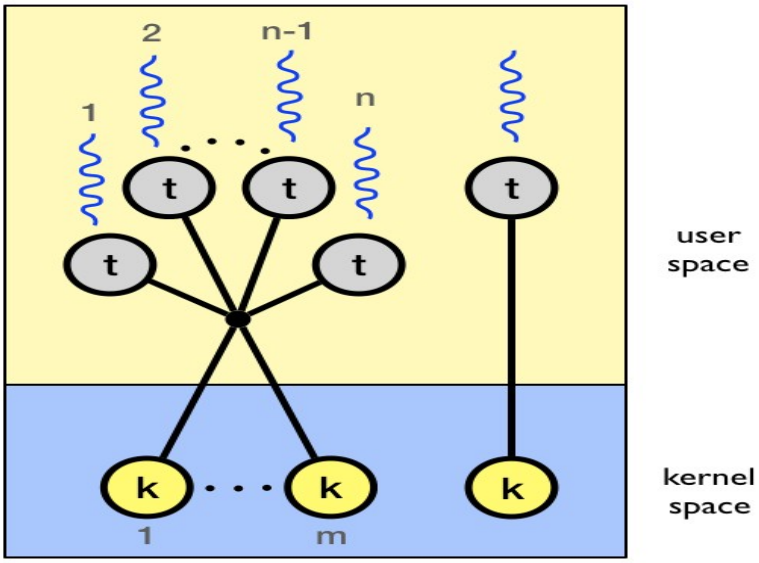


In this model the kernel must provide a system call for creating a new kernel thread.

Many to many: In the many-to-many model the process is allocated m number of kernel-level threads to execute n number of user-level thread.



Two level : The two-level model is similar to the many-to-many model but also allows for certain user-level threads to be bound to a single kernel-level thread.



Discussion : This lab helps us to learn about Threads on Operating System. We have seen how the implementation of threads happen in Operating System. In future we can know more about it.