**Lab Report No :** 03

**Lab Report Name :** Threads on Operating System.

**ID :** IT- 17025

**Q1 : What is Thread ?**

**Ans :** A thread is the smallest unit of processing that can be performed in an OS. In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads. In general , A thread is a path of execution within a process. A process can contain multiple threads.

**Q2 : Write Types of Thread ?**

**Ans :** Threads are implemented in following two ways

**1. User Level Threads.**

**2. Kernel Level Threads.**

* **User Level Threads :**

In this case, application manages 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 begins with a single thread and begins running in that thread.

**#Advantages of ULT :**

* 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.

**#Disadvantages of ULT :**

* In a typical operating system, most system calls are blocking.
* Multithreaded application cannot take advantage of multiprocessing.

**Kernel Level Threads :**

In this case, thread management 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.

**#Advantages of KLT :**

* 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 multithreaded.

**#Disadvantages of KLT :**

* Kernel threads are generally slower to create and manage than the user threads.
* Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

**Q3 : Write the implementation of threads.**

**Ans :**

**1.Threads implementation in User Space :**

In this model of implementing the thread package completely in user space, the kernel do not know anything about that.

The advantage of implementing threads package in user space is that a user level threads package that can be implemented on an OS that doesn’t support threads. All of these implementations have some general structure.

**2. Threads implementations in Kernel :**

The kernel's thread table holds each thread's registers, state, and other information. The information is the same as with user-level threads, but it is now in the kernel instead of in user space .This information is a subset of the information that traditional kernels maintain about each of their single-threaded processes, that is, the process state. In addition, the kernel also maintains the traditional process table to keep track of processes.

All calls that might block a thread are implemented as system calls, at considerably greater cost than a call to a run-time system procedure. When a thread blocks, the kernel, at its option, can run either another thread from the same process or a thread from a different process. With user-level threads, the run-time system keeps running threads from its own process until the kernel takes the CPU away from it .

Due to the relatively greater cost of creating and destroying threads in the kernel, some systems take an environmentally correct approach and recycle their threads. When a thread is destroyed, it is marked as not runnable, but its kernel data structures are not otherwise affected. Later, when a new thread must be created, an old thread is reactivated, saving some overhead. Thread recycling is also possible for user-level threads, but since the thread management overhead is much smaller, there is less incentive to do this.

Kernel threads do not require any new, nonblocking system calls. In addition, if one thread in a process causes a page fault, the kernel can easily check to see if the process has any other runnable threads, and if so, run one of them while waiting for the required page to be brought in from the disk. Their main disadvantage is that the cost of a system call is substantial, so if thread operations) are common, much more overhead will be incurred.

**Conclusion :**

From this lab , we learn the importance of Threads in Operating System. We also learn various types of threads and how they implemented to make User Interface more easy to use . the blessing of multitasking can possible only by using thread. We learn both of these from these lab .