### Threads

Despite of the fact that a thread must execute in process, the process and its associated threads are different concept. Processes are used to group resources together and threads are the entities scheduled for execution on the CPU.

*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*. In a process, threads allow multiple executions of streams. In many respect, 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). Each thread has its own stack. Since thread will generally call different procedures and thus a different execution history. This is why thread needs its own stack. An operating system that has thread facility, the basic unit of CPU utilization is a thread. A thread has or consists of a program counter (PC), a register set, and a stack space. Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section, OS resources  also known as task, such as open files and signals.

### Processes Vs Threads

As we mentioned earlier that in many respect threads operate in the same way as that of processes. Some of the similarities and differences are:

**Similarities**

* Like processes threads share CPU and only one thread active (running) at a time.
* Like processes, threads within a processes, threads within a processes execute sequentially.
* Like processes, thread can create children.
* And like process, if one thread is blocked, another thread can run.

**Differences**

* Unlike processes, threads are not independent of one another.
* Unlike processes, all threads can access every address in the task .
* Unlike processes, thread are design to assist one other. Note that processes might or might not assist one another because processes may originate from different users.

**Why Threads?**

Following are some reasons why we use threads in designing operating systems.

1. A process with multiple threads make a great server for example printer server.
2. Because threads can share common data, they do not need to use interprocess communication.
3. Because of the very nature, threads can take advantage of multiprocessors.

**Threads are cheap in the sense that**

1. They only need a stack and storage for registers therefore, threads are cheap to create.
2. Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources.
3. Context switching are fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

But this cheapness does not come free - the biggest drawback is that there is no protection between threads.

### User-Level Threads

User-level threads implement in user-level libraries, rather than via systems calls, so thread switching does not need to call operating system and to cause interrupt to the kernel. In fact, the kernel knows nothing about user-level threads and manages them as if they were single-threaded processes.

**Advantages:**

The most obvious advantage of this technique is that a user-level threads package can be implemented on an Operating System that does not support threads. Some other advantages are

* User-level threads does not require modification to operating systems.
* Simple Representation:   
      Each thread is represented simply by a PC, registers, stack and a small control block, all stored in the user process address space.
* Simple Management:   
      This simply means that creating a thread, switching between threads and synchronization between threads can all be done without intervention of the kernel.
* Fast and Efficient:  
      Thread switching is not much more expensive than a procedure call.

**Disadvantages:**

* There is a lack of coordination between threads and operating system kernel. Therefore, process as whole gets one time slice irrespective of whether process has one thread or 1000 threads within. It is up to each thread to relinquish control to other threads.
* User-level threads requires non-blocking systems call i.e., a multithreaded kernel. Otherwise, entire process will blocked in the kernel, even if there are run able threads left in the processes. For example, if one thread causes a page fault, the process blocks.

### Kernel-Level Threads

In this method, the kernel knows about and manages the threads. No runtime system is needed in this case. Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system. In addition, the kernel also maintains the traditional process table to keep track of processes. Operating Systems kernel provides system call to create and manage threads.  
  
The implementation of general structure of kernel-level thread is 

**Advantages:**

* Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads.
* Kernel-level threads are especially good for applications that frequently block.

**Disadvantages:**

* The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.
* Since kernel must manage and schedule threads as well as processes. It require a full thread control block (TCB) for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity.

### Advantages of Threads over Multiple Processes

* **Context Switching**    Threads are very inexpensive to create and destroy, and they are inexpensive to represent. For example, they require space to store, the PC, the SP, and the general-purpose registers, but they do not require space to share memory information, Information about open files of I/O devices in use, etc. With so little context, it is much faster to switch between threads. In other words, it is relatively easier for a context switch using threads.
* **Sharing**    Treads allow the sharing of a lot resources that cannot be shared in process, for example, sharing code section, data section, Operating System resources like open file etc.

### Disadvantages of Threads over Multi-processes

* **Blocking**    The major disadvantage if that if the kernel is single threaded, a system call of one thread will block the whole process and CPU may be idle during the blocking period.
* **Security**    Since there is, an extensive sharing among threads there is a potential problem of security. It is quite possible that one thread over writes the stack of another thread (or damaged shared data) although it is very unlikely since threads are meant to cooperate on a single task.

Video Links:

<https://www.youtube.com/watch?v=o2Sf0GHWpmk&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O&index=34>

<https://www.youtube.com/watch?v=wNns0kIDC68&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O&index=35>

REFERENCES:

1. Galvin, Peter B., Silberchatz, A., “*Operating System Concepts*”, Addison Wesley, 8th Edition.
2. Flynn, *“Operating Systems”*, Cengage Learning.
3. Dhamdhere, D.M., "*Operating System:A Concept Based Approach*",  
   Tata Mc-Graw- Hill.
4. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-828-operating-system-engineering-fall-2012/lecture-notes-and-readings/
5. **https://www.youtube.com/watch?v=OrM7nZcxXZU**
6. <https://computing.llnl.gov/tutorials/>
7. https://nptel.ac.in/courses/106/105/106105214
8. <https://www.guru99.com/operating-system-tutorial.html>
9. <https://www.geeksforgeeks.org/operating-systems/>