Introduction to threads

A thread is a path of execution within a process. A process can contain multiple threads. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

A thread is also known as lightweight process.

- Each thread runs strictly sequentially and has its own program counter and stack to keep track of where it is.
- Threads share the CPU just as processes do: first one thread runs, then another does.
- Threads can create child threads and can block waiting for system calls to complete.
- A traditional (heavyweight) process has single thread of control. If a process has multiple thread of control, it can perform more tasks at a time.
- All threads have exactly the same address space. They share code section, data section, and OS resources (open files & signals). They share the same global variables.
- One thread can read, write, or even completely wipe out another thread’s stack.
- Threads can be in any one of several states: running, blocked, ready, or terminated.

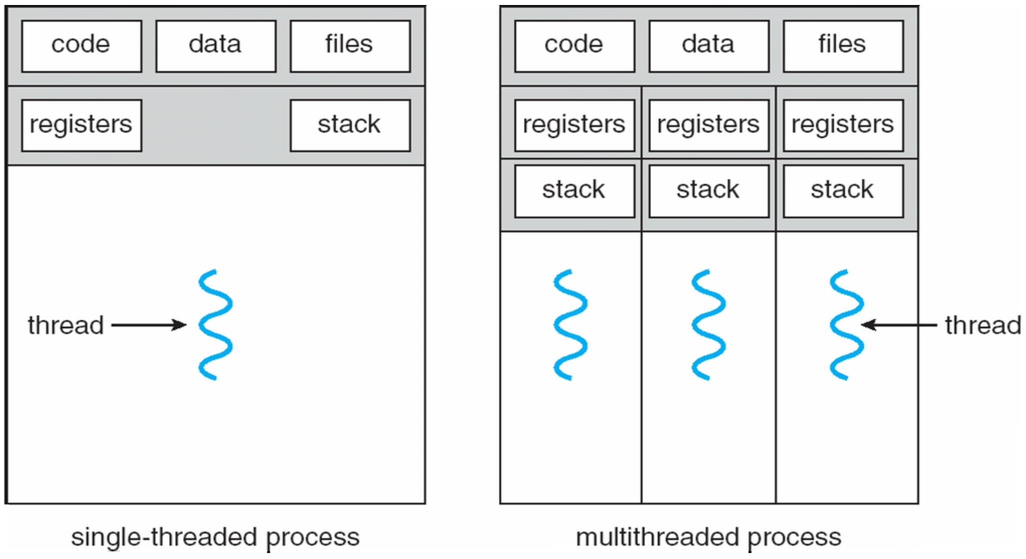
Example

- **A process as a house**
  - A house is really a container, with certain attributes (such as the amount of floor space, the number of bedrooms, and so on).
  - The house really doesn't actively do anything on its own — it's a passive object. This is effectively what a process is.
- **The occupants as threads**
  - The people living in the house are the active objects — they're the ones using the various rooms, watching TV, cooking, taking showers, and so on.
  - The occupants of a house can be considered as thread.

Single and Multithreaded Processes

- **Single threaded**
  - If you've ever lived on your own, then you know what this is like — you know that you can do anything you want in the house at any time, because there's nobody else in the house. If you want to turn on the stereo, use the washroom, have dinner — whatever — you just go ahead and do it.

- **<u>Multi threaded</u>**
  - Things change dramatically when you add another person into the house. Let's say you get married, so now you have a spouse living there too.



Benefits of using Thread

- Responsiveness: Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. For example, a multithreaded web browser could still allow user interaction in one thread while an image was being loaded in another thread.

- Resource Sharing: By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

- Economy: Allocating memory and resources for process creation is costly. As thread shares resource of the process to which they belong, it is more economical to create and context-switch threads than processes. In general, it is more time consuming to create and manage threads than processes.

- Utilization of multiprocessor architectures: The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

- Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Similarity between Threads and Processes –

- Only one thread or process is active at a time
- Within process both execute sequentially
- Both can create children

Differences between Threads and Processes –

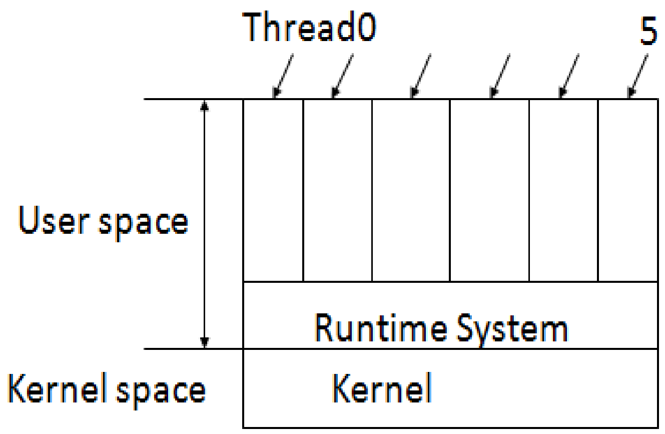
- Threads are not independent, processes are.
- Threads are designed to assist each other, processes may or may not do it

Types of Threads

User Level thread (ULT)

ULT is implemented in the user level library, they are not created using the system calls. Thread switching does not need to call OS and to cause interrupt to Kernel. Kernel doesn't know about the user level thread and manages them as if they were single threaded processes.

Implementing threads in user space



Advantages of ULT –

- Can be implemented on an OS that doesn't support multithreading. For example: The Unix system.
- Simple representation since thread has only program counter, register set, stack space.
- Simple to create since no intervention of kernel.
- Thread switching is fast since no OS calls need to be made.

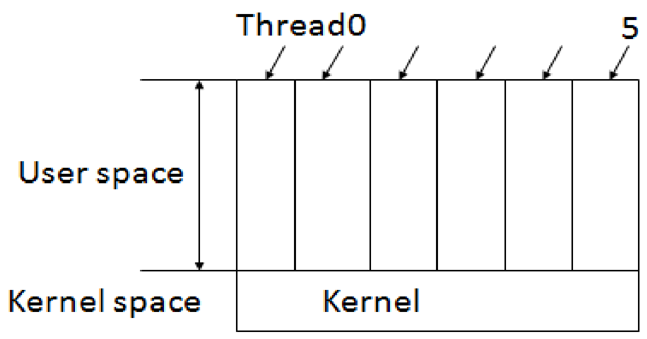
Disadvantages of ULT –

- No or less co-ordination among the threads and Kernel.
- If a thread causes a page faults, the kernel does not know about the threads. It will block the entire process until the page has been fetched, even though other threads might be runnable.
- If a thread starts running, no other thread in that process will ever run unless the first thread voluntarily gives up the CPU.

Kernel Level Thread (KLT)

Kernel knows and manages the threads. Instead of thread table in each process, the kernel itself has thread table (a master one) that keeps track of all the threads in the system. In addition kernel also maintains the traditional process table to keep track of the processes. OS kernel provides system call to create and manage threads.

Implementing threads in the kernel



Advantage of threads in the kernel space

- The kernel knows about and manages the threads. No runtime system is needed. When a thread wants to create a new thread or destroy an existing thread, it makes a kernel call, which then does the creation and destruction.
- To manage all the threads, the kernel has one table per process with one entry per thread.
- When a thread blocks, the kernel can run either another thread from the same process or a thread from a different process.

Disadvantage

- Here, every thread operation (creation, deletion, synchronization etc.) will have to be carried out by the kernel which requires a system call.
- Again switching thread context here is as expensive as switching process context.