Threads

Thread

- Thread is a kind of light-weight process within a process.
- We create thread for doing multiprogramming and use of Main Memory efficiently.
- Single task of a process is divided into multiple parts and perform each part with a separate thread
- Thread uses Process's user space
- We do not provide separate user space in main memory to run Thread

Thread

- In traditional operating systems, each process has an address space and a single thread of control.
- In many situations, it is desirable to have multiple threads of control in the same address space running in quasi-parallel, as though they were (almost) separate processes (except for the shared address space).

Thread

- A web browser
  - display images or text
  - retrieves data from the network
  - Multiple users can share the contents of a web page concurrently
- Word Processor
  - displaying graphics
  - responding to keystrokes from the user
  - performing spelling and grammar checking in the background
  - automatically save the contents to disk

Why Thread?

- Threads share an address space and all of its data among themselves.
- This ability is essential for certain applications, where multiple processes with separate address spaces will not work.

Why Thread?

- Threads are lighter weight than processes, they are easier (i.e., faster) to create and destroy than processes.
- In many systems, creating a thread goes 10–100 times faster than creating a process.
- When the number of threads needed changes dynamically and rapidly, this property is useful to have.
- Threads are created in user space
- System call from User Level to Kernel Level is not required to create a Thread

Thread Performance

- Threads yield no performance gain when all of them are CPU bound.
- Process speeds up when some threads are CPU bound and some are I/O bound; as these activities can overlap then.
- If multiple CPUs are available, different threads can be run on different processors, to gain real-parallelism.