Multithreading

Light-weight "processes" that do the same task parallely. All threads share address space, global process data etc. but have their own registers, stack etc..

Motivation for threads

Threads have become prominent due to trends in:

- *Software Design*: More naturally expresses inherently parallel tasks
- *Performance*: Scales better to multiprocessor systems
- *Cooperation*: Shared address space incurs less overead than IPC

Why do we need threads?

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU
- Prioritize work within the processes

Processes vs Threads

Processes have:

- virtual address space that holds the PCB.
- protected access to processors, other processes, files, and I/O resources.

While threads have:

- execution state
- thread context (program counter etc.)
- private storage for local variables and stack
- shared access to address space and resources of their process
    - When one thread alters data, all other threads can see
    - they can then communicate using these shared variables
    - file opened by one thread is acessible to all

Threads over Processes

- Threads are faster to create, terminate and switch b/w.
- Easier to communicate b/w threads; You don't need to set up IPC.
