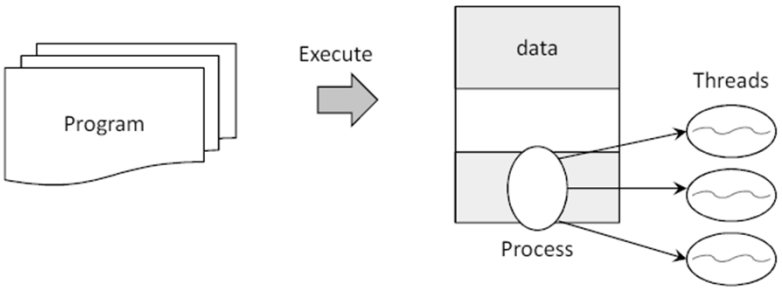
ET0023 Operating Systems

4. Threads

Threads



http://en.wikipedia.org/wiki/Thread_%28computer_science%29

Processes vs Threads

- Processes
  - Expensive (in terms of time) to switch
  - Requires file, resource, accounting structures
  - Require a fully Process Control Block
  - Heavy weight
- Threads
  - Scheduled in a user-dependent way
  - Independent sub-processes that share the data/address space of the process
  - Requires on stack and registers
  - Light weight

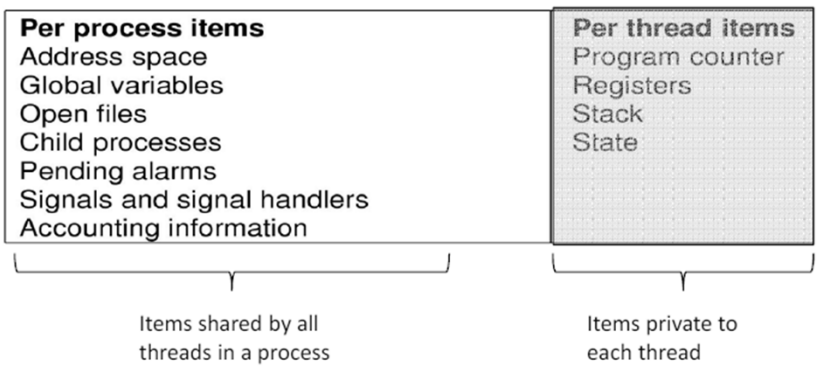
Threads

- Mini-versions of a process.
- Enables the user to split up a process into logically separate parts (threads)
- Each thread is run separately and independently of one another until they need to communicate.
- Allow certain functions to be executed in parallel with others.

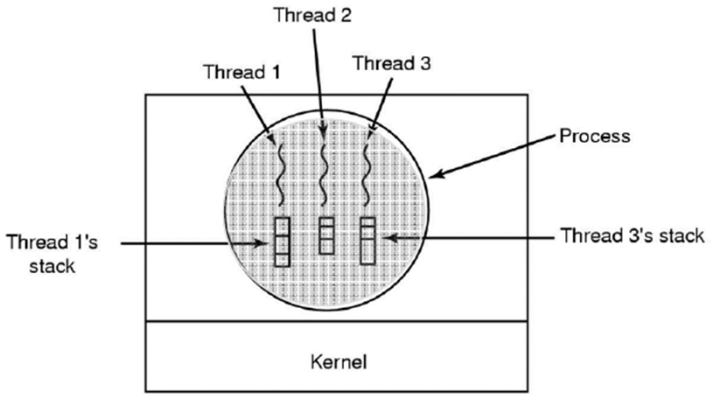
Threads

- Multi-threaded tasks are comprised of several independent processes which work on common data, and can run in parallel.
- Threads only require stack and some registers. All other resources are shared.
- Lightweight
- Can be assigned priorities
- User controlled in the program.

Thread Model



The Thread Model



Each thread has its own
- Thread ID
- Program Counter
- Register Set
- Stack Space

Why use threads?

- If the scheduler knows what resources a process requires, then the sharing of resources could be made more efficient.
  (Scheduler is a program – programs cannot think!!)
- If there are multiple CPUs, you could run multiple processes in parallel – but what about resources ?
- When you switch processes, you have to save the whole PCB and restore the next PCB (slow, CPU intensive)
- What if we could "think" for the scheduler?

Advantages of threads

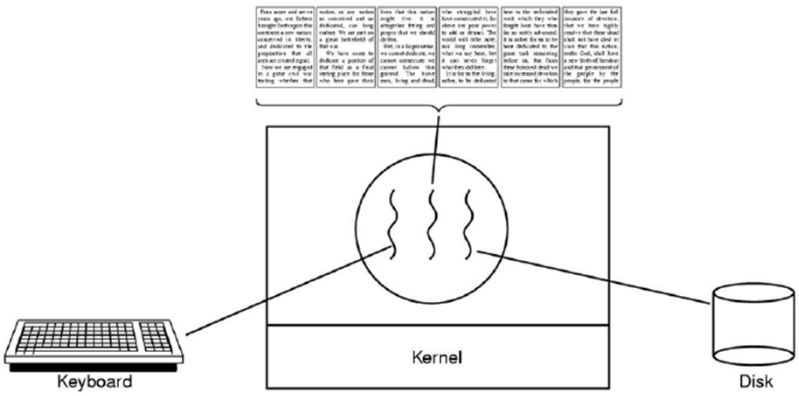
- Allows a programmer to switch between lightweight processes when it is best for the program
- Each process gets a slice of the CPU time, but what if we can make parts of the process work in parallel
- Responsiveness: threads increase responsiveness to a process
- Resource sharing: threads share the resources
- Economy: thread CPU cost is low
- Multiprocessor architectures allow multiple threads to run.

http://en.wikipedia.org/wiki/Multithreading_%28computer_architecture%29

Use of Threads

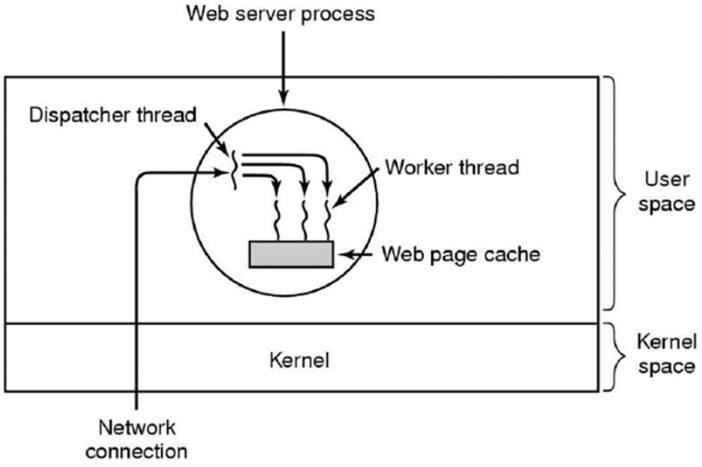
- Interactive applications.
  - Helps make apps more interactive.
  - Eg. Word processor, Excel spreadsheet.
- Server applications.
  - Serve multiple requests at the same time.
  - Eg. Web server.
- Application that process large amounts of data.
  - Input thread, Processing thread, Output thread.

Threaded Word Processor

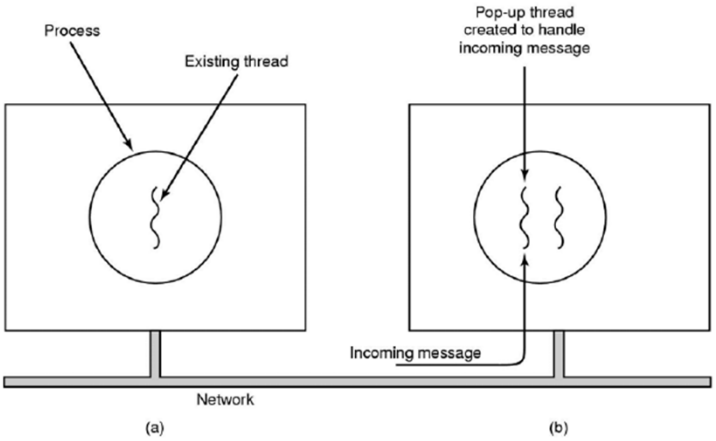


A word processor with three threads

Multi-threaded Web Server



Use of Threads

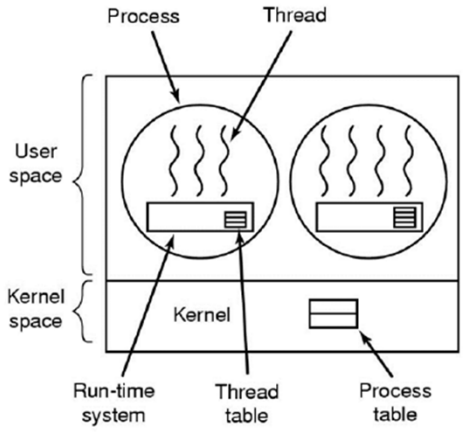


Creation of a new thread when message arrives:
(a) Before message arrives
(b) (b) After message arrives

Thread Types - User

- User Thread
  - Supported above the kernel
  - Implemented by a thread library at user level.
  - Library provides support for thread creation, scheduling and management.
  - No support from the kernel, no kernel intervention
  - Fast to create and manage
  - Problem if one thread performs a blocking system call, all threads will be blocked.

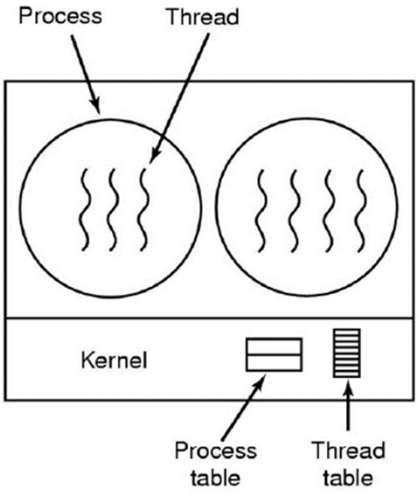
User-level threads



Thread Types - Kernel

- Kernel Thread
  - Supported directly by kernel (operating system)
  - Kernel performs thread creation, scheduling and management in kernel space.
  - Slower to create and manage
  - In multiprocessor environments, kernel can schedule threads on different processors.
  - Blocking calls are handled by kernel, the kernel can schedule another thread in the application.

Kernel-level Threads



QUESTIONS?