COP 4225 Advanced Unix Programming

Processes and Threads

Chi Zhang
czhang@cs.fiu.edu

Process Concept

- Process – a program in execution; process execution must progress in sequential fashion.
- A process includes
  - program counter
  - stack
  - data section
  - (p.168 Figure 7.3):

Process State

- As a process executes, it changes *state*
  - **new**: The process is being created.
  - **running**: Instructions are being executed.
  - **waiting**: The process is waiting for some event to occur.
  - **ready**: The process is waiting to be assigned to a process.
  - **terminated**: The process has finished execution.

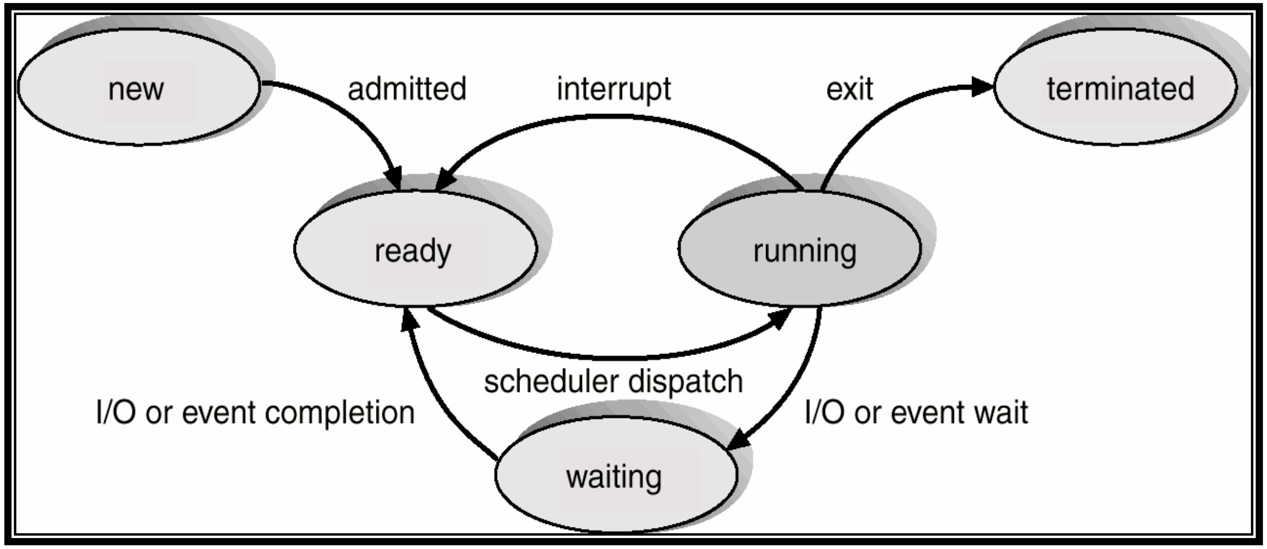
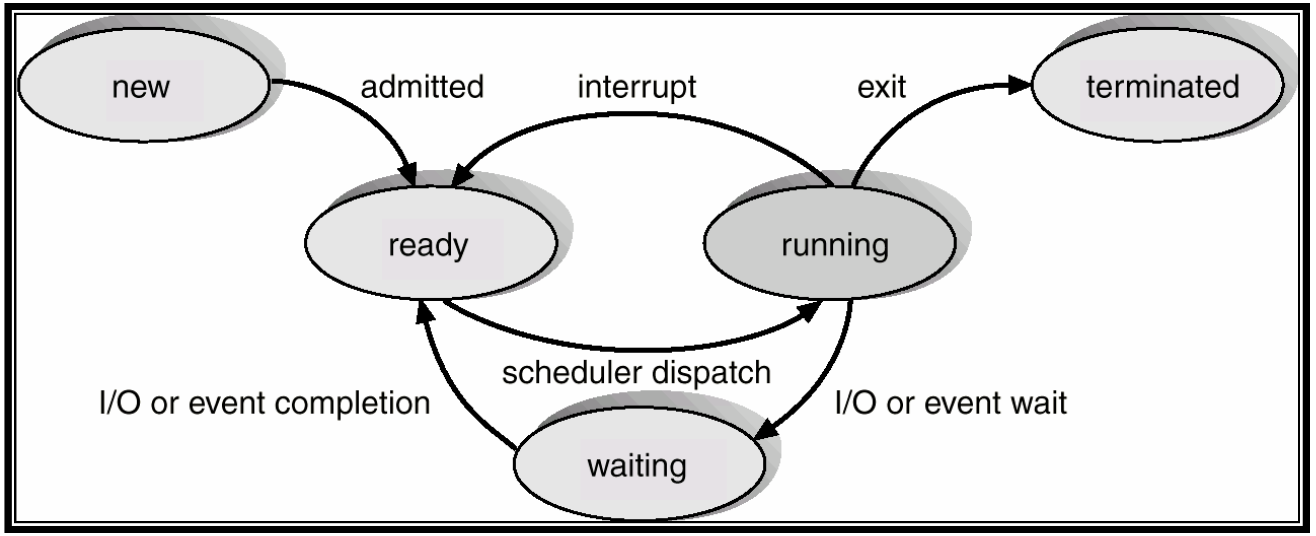
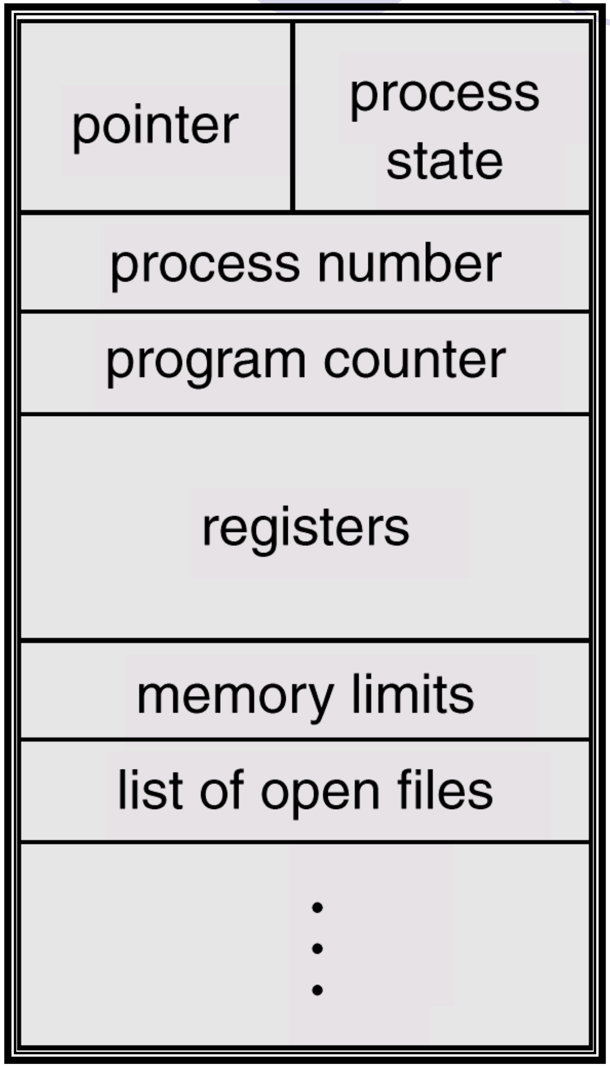


Diagram of Process State

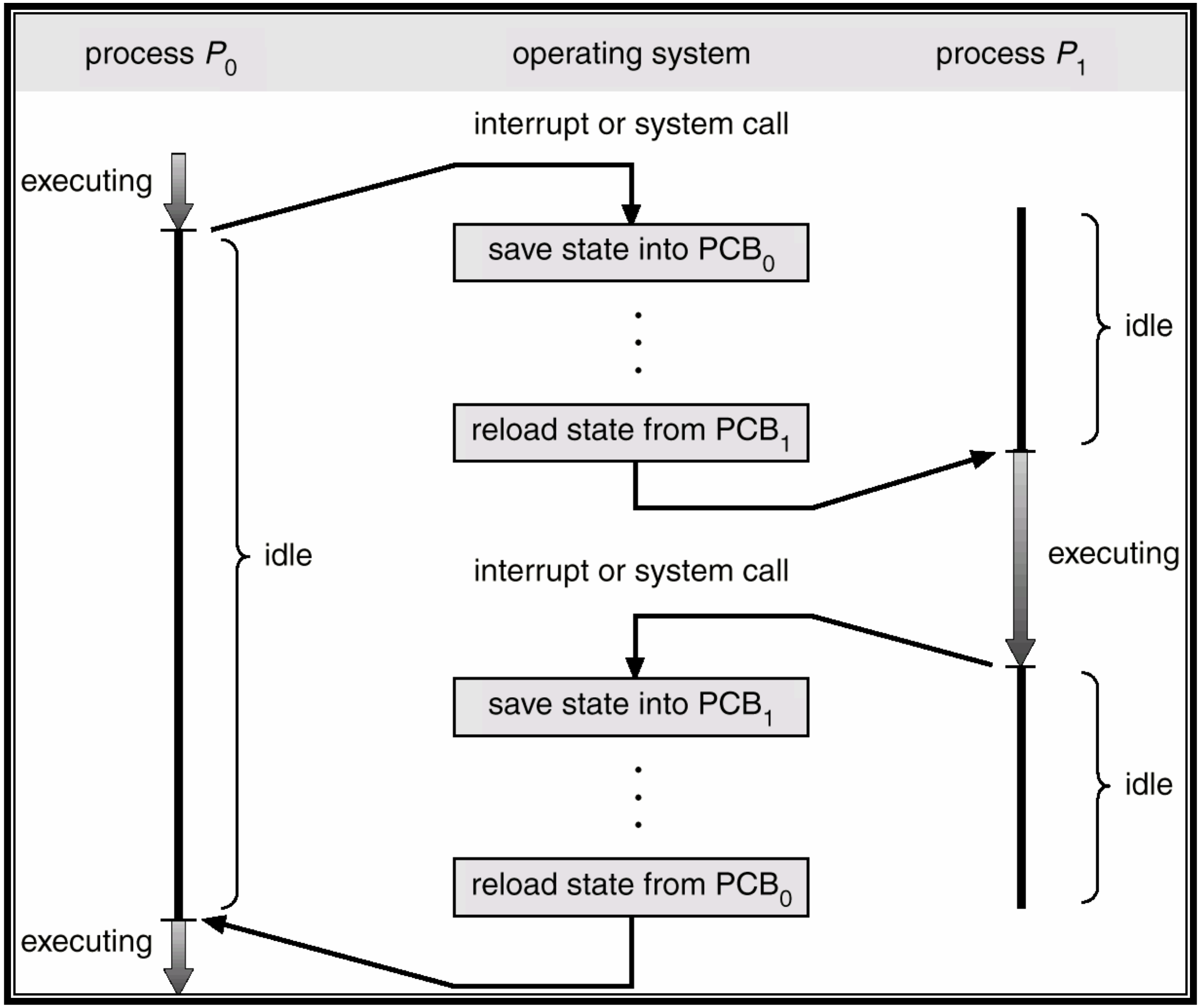


Process Control Block (PCB)

Pointer to the next process



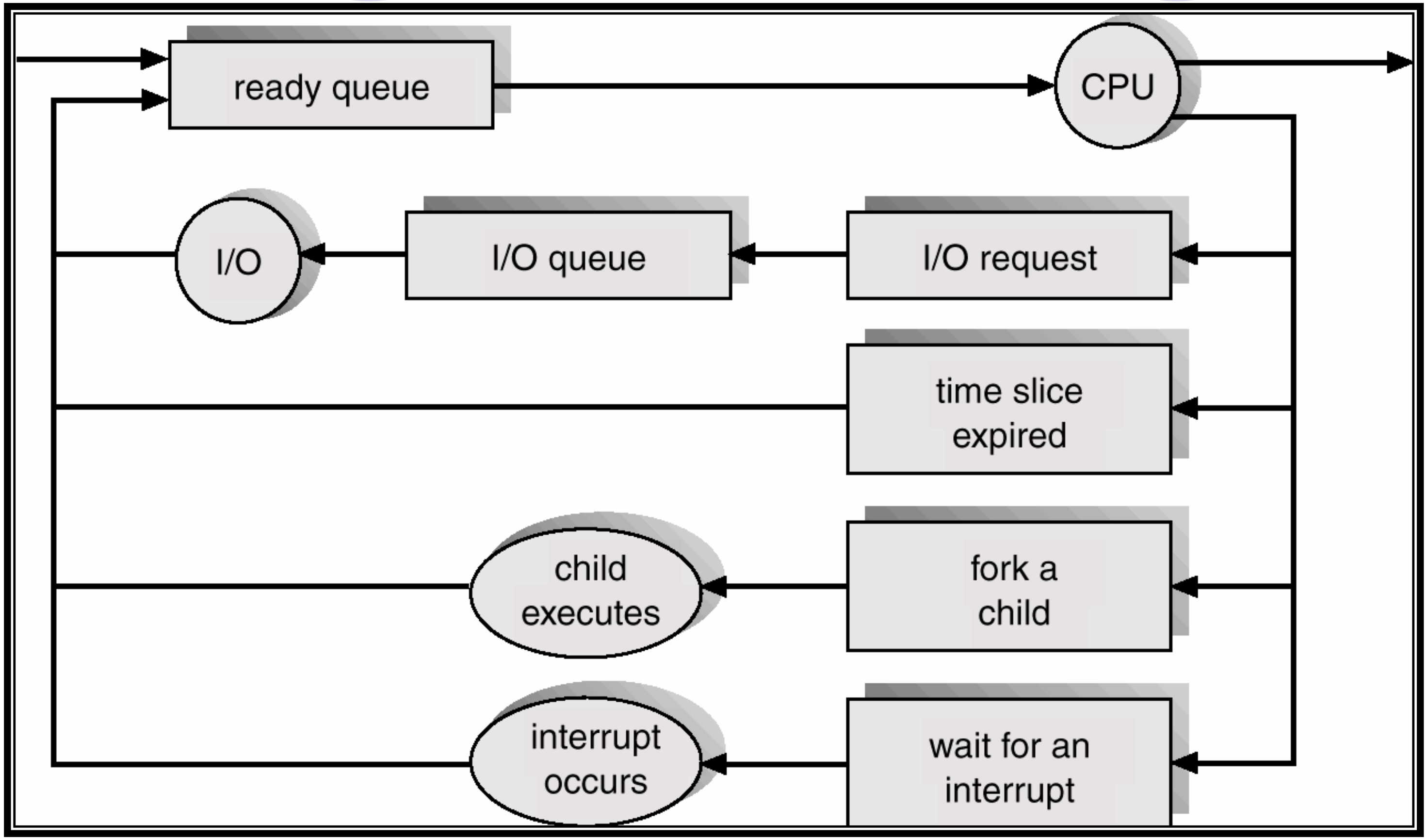
CPU Switch From Process to Process



Process Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for a particular I/O device.
- Process migration between the various queues.

Representation of Process Scheduling



Schedulers

- Long-term scheduler
  - which processes should be brought into the ready queue (in memory rather than on disk).
  - invoked very infrequently (when a process leave the system)
- Short-term scheduler
  - selects which process should be executed next and allocates CPU.
  - Invoked frequently
- Midterm scheduler
  - Swapping improves the process mix.

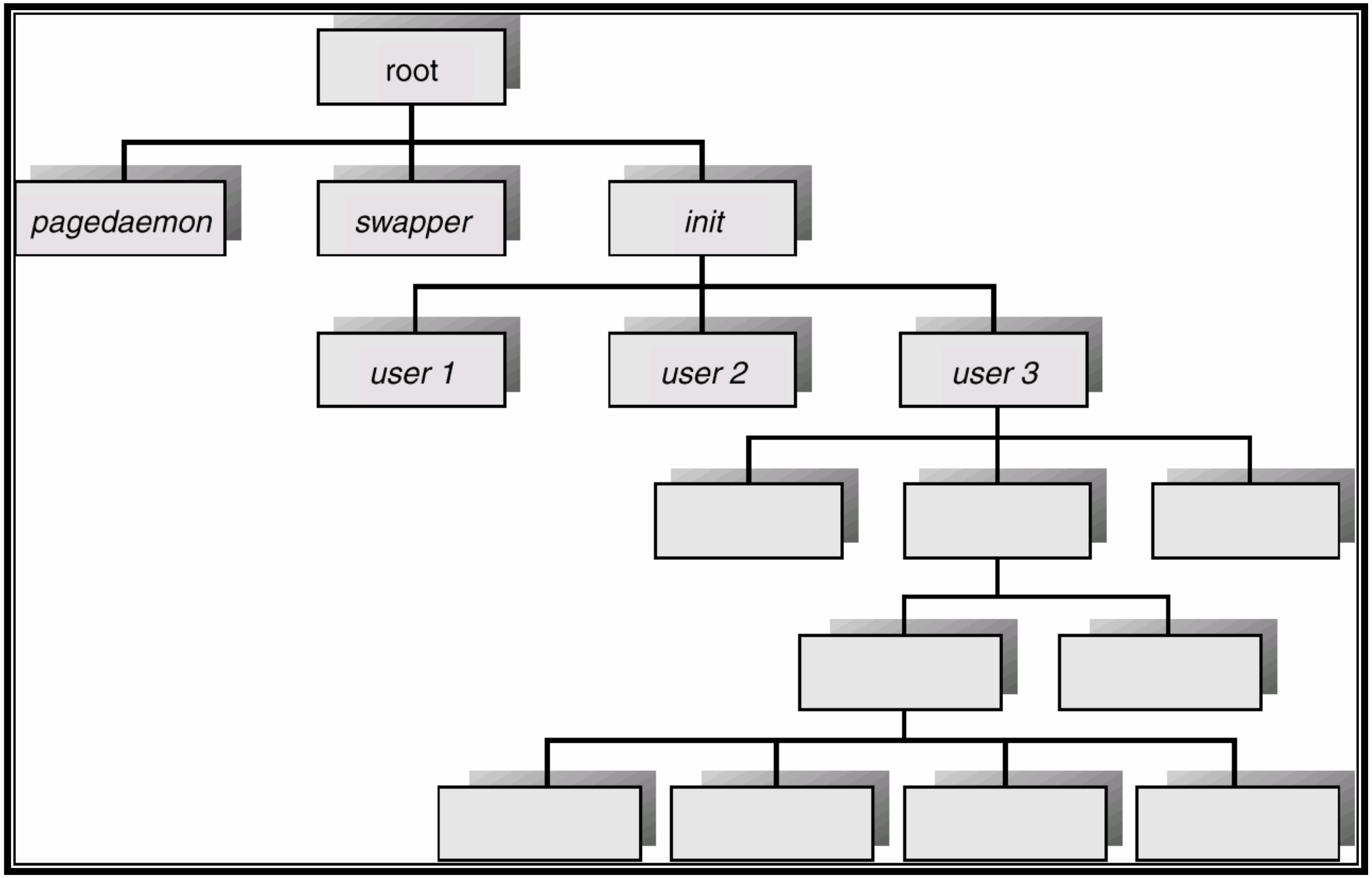
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent’s resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

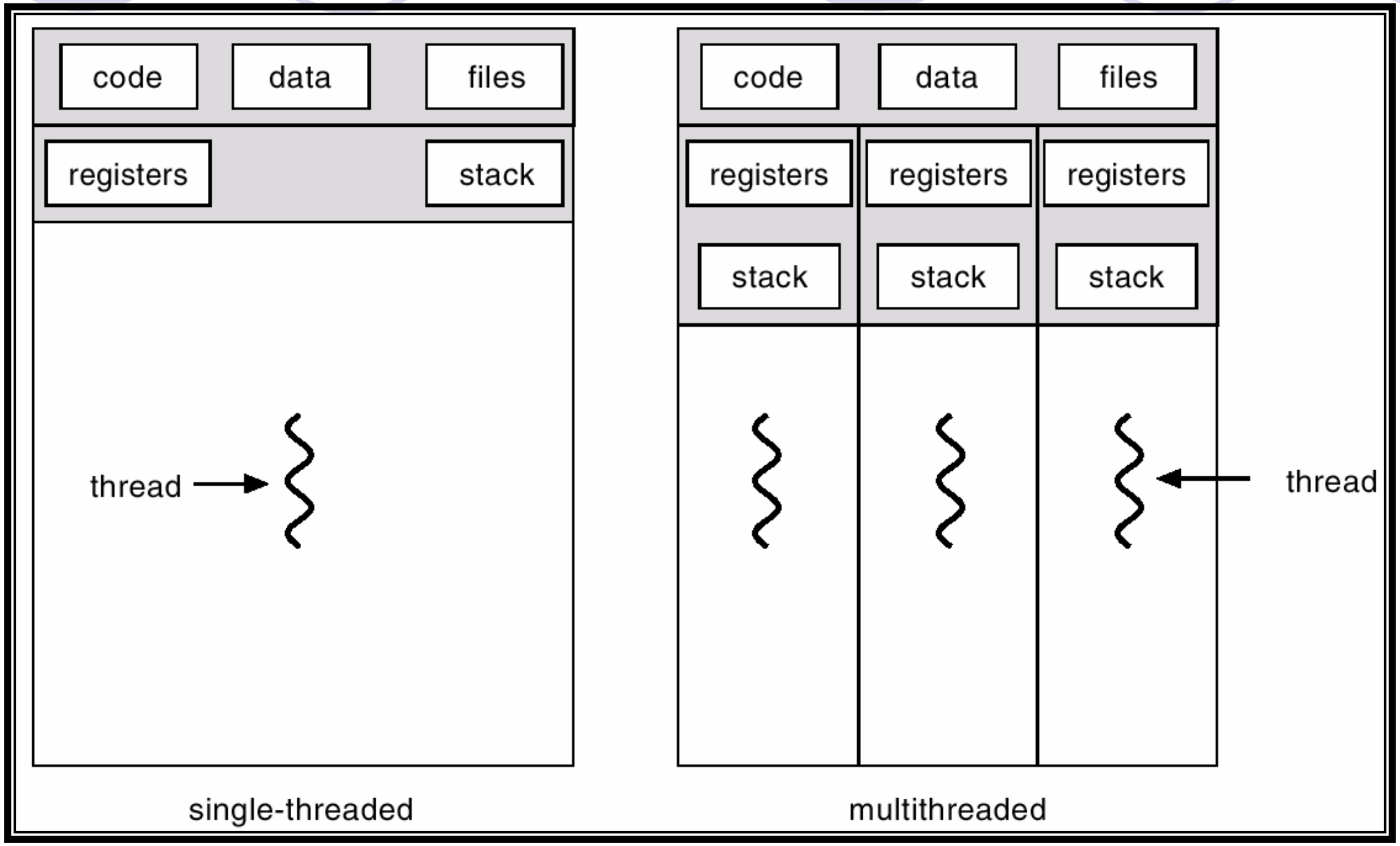
Processes Tree on a UNIX System



Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
  - Output data from child to parent (via **wait**).
  - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes (**abort**).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

Single and Multithreaded Processes



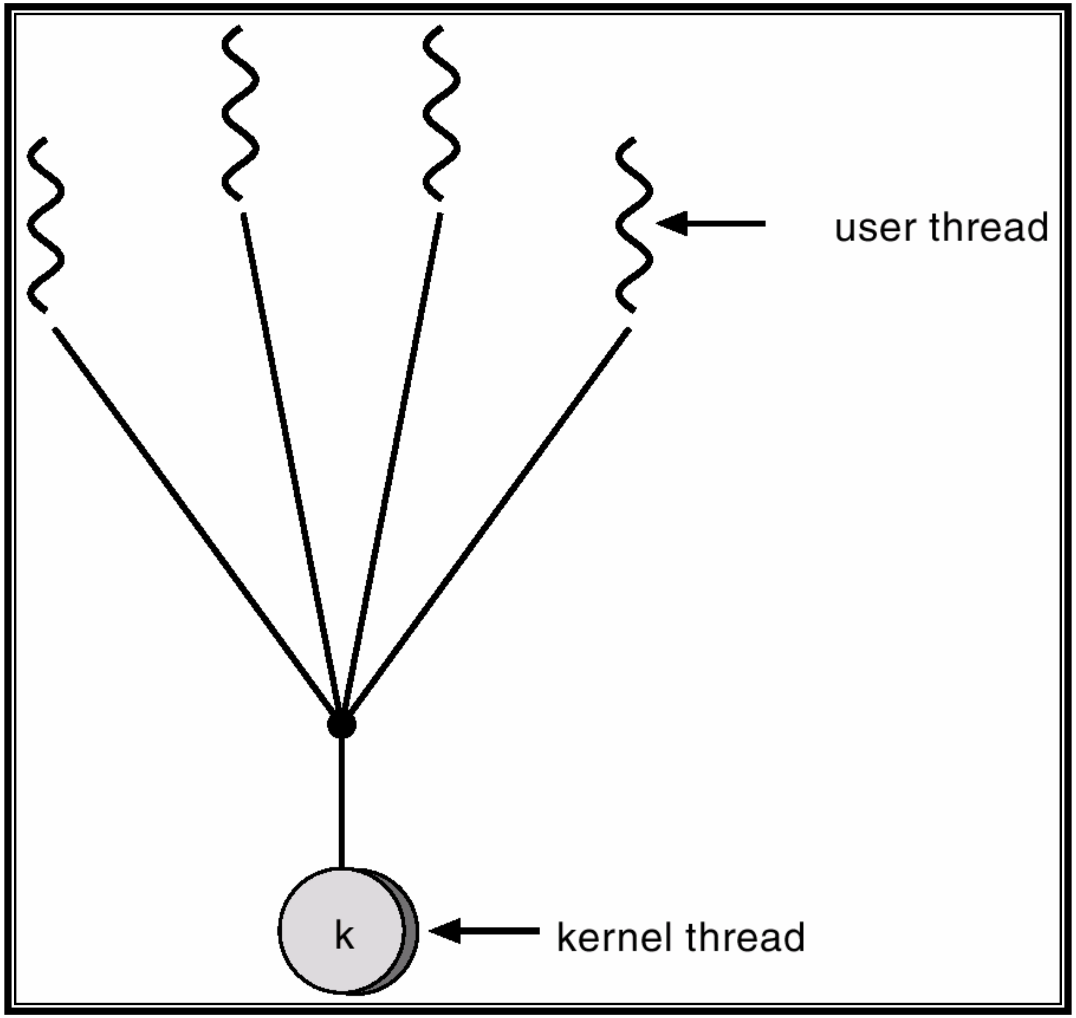
Benefits

- Responsiveness
  - User interaction in parallel with data retrieval
- Resource Sharing
- Economy
  - In Solaris 2, creating a process is about 30 times slower than threads
  - Context switch is about 5 times slower.
- Utilization of MP Architectures

User Threads

- Thread management done by user-level threads library
- A blocking system call will cause the entire process to block
  - OS is unaware of threads
- The kernel cannot schedule threads on different CPUs.

Many-to-One Model (User Threads)

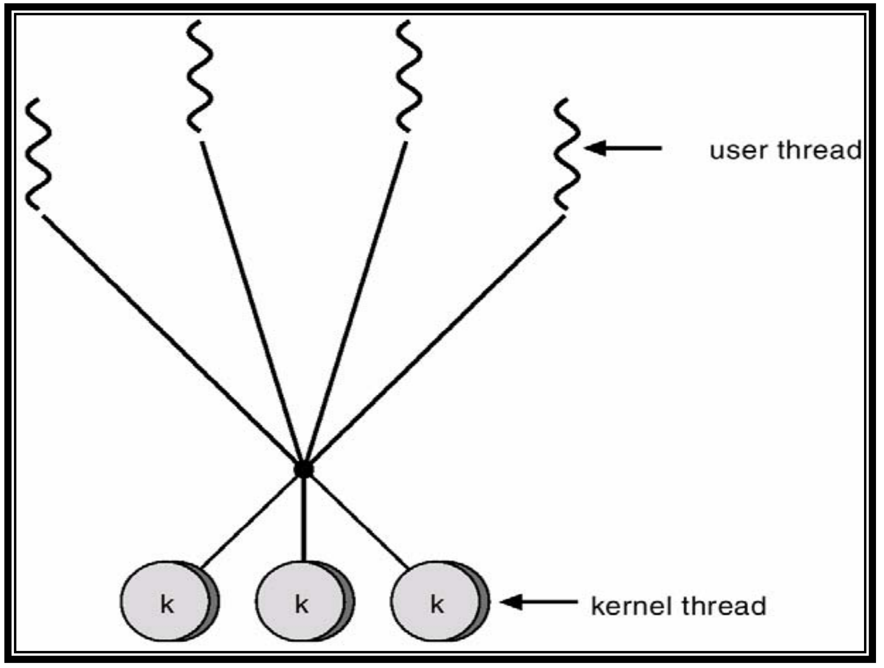


Kernel Threads

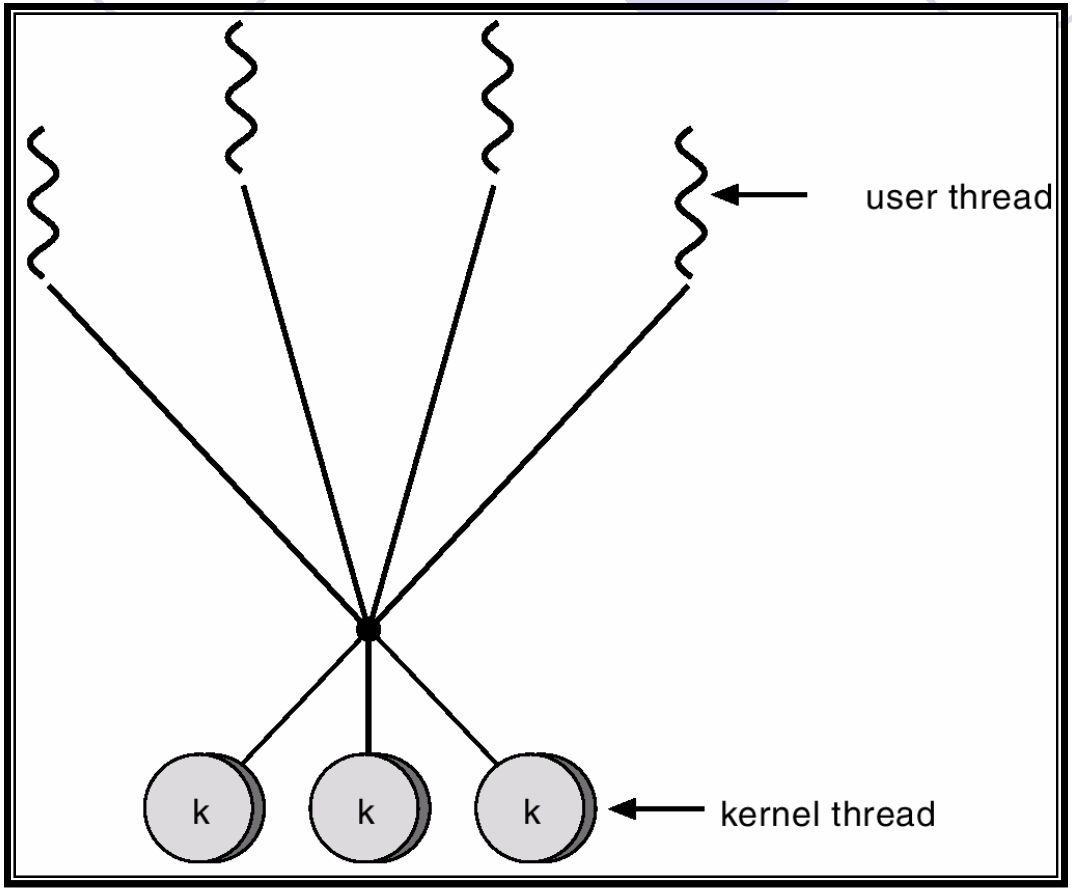
- Supported by the Kernel
- OS manages threads
  - Slower to create and manage because of system calls
  - A blocking system call will not cause the entire process to block.
  - The kernel can schedule threads on different CPUs.

Many-to-Many Model (Solaris 2)

- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.



Many-to-Many Model



Threading Issues

- Semantics of fork() and exec() system calls.
  - Duplicate all threads in the child process?
- Thread cancellation.
  - Asynchronous Cancellation
    - One thread immediately terminates the target thread
    - OS reclaims resources (but not all) allocated to the threads
  - Deferred Cancellation
    - The target thread checks periodically if it should terminate (if so, terminate gracefully)

Threading Issues

- Signal handling
  - Which thread should a signal be delivered
- Thread pools
  - Creating threads upon incoming request is expensive
  - Unlimited Threads can exhaust system resources
  - Request queue + thread pool
- Thread specific data

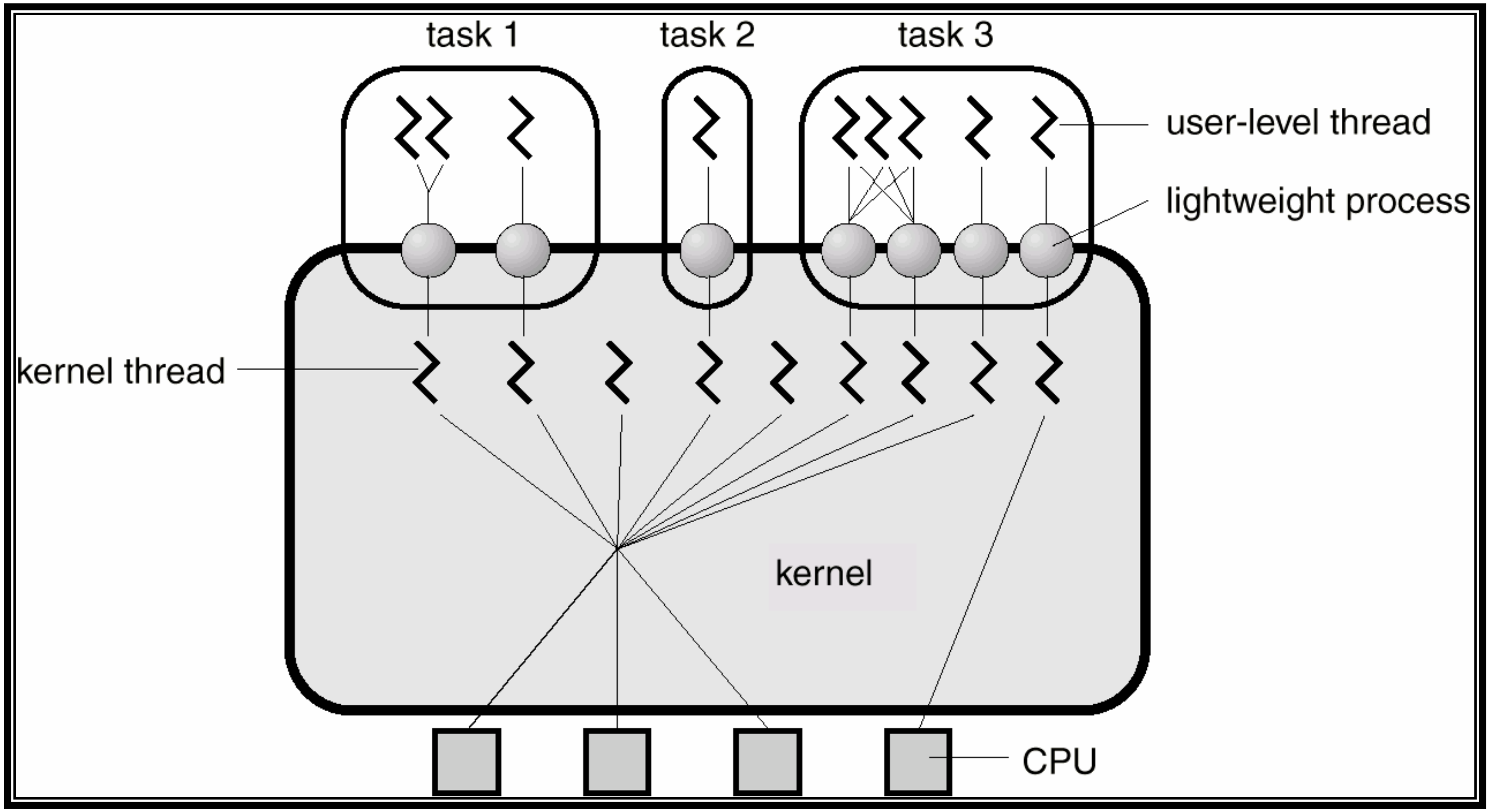
Pthreads

- a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library.
- Common in UNIX operating systems.

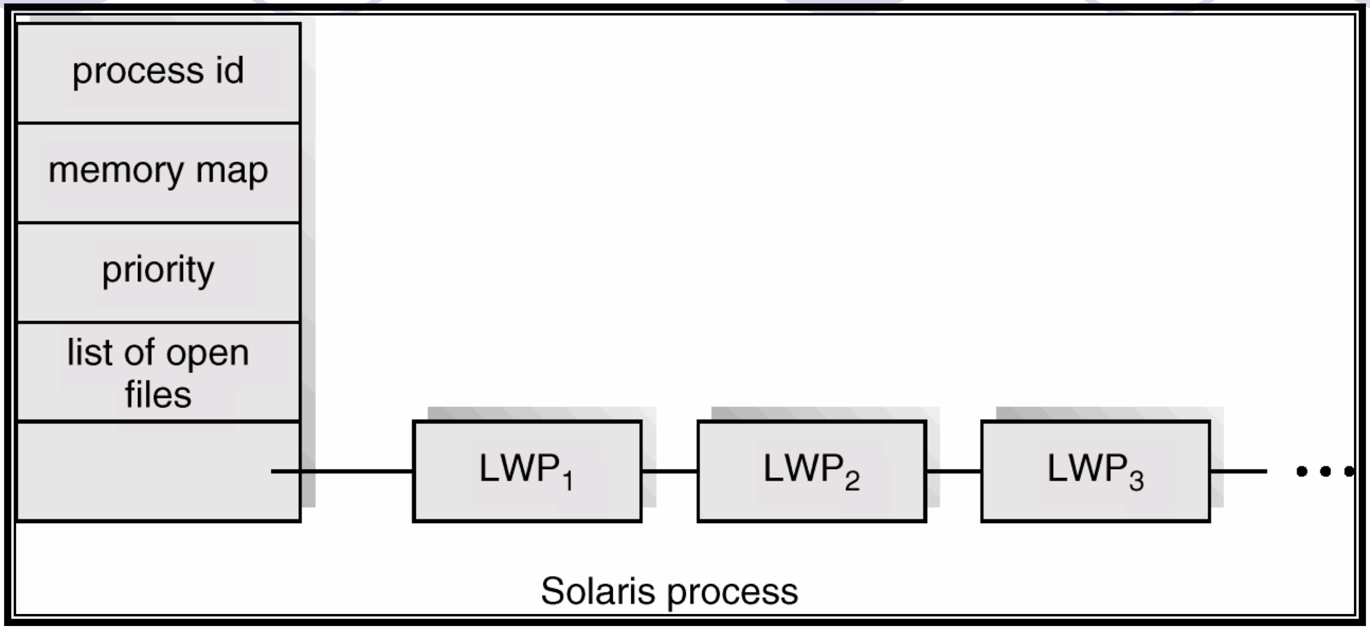
Solaris 2 Threads

- Light Weight Threads (LWP) between user- and kernel- level threads.
- Each LWP is mapped to one kernel-level thread
- The thread library (user level) multiplexes (schedules) user-level threads on the pool of LWPs for the process.
  - Only user-level threads currently connected to an LWP accomplish work
  - For one process, one LWP is needed for every thread that may block concurrently in system calls.

Solaris 2 Threads



Solaris Process



The kernel maintains Process control block, kernel threads, and LWPs.

The user-level threads is maintained in the user space.

Linux Threads

- Linux refers to them as *tasks* rather than *threads*.
  - Linux actually does not distinguish between processes and threads
- Thread creation is done through clone() system call.
- Clone() allows a child task to share the address space of the parent task (process)
  - A set of parameters decides how much of the parent process is to be shared with the child.
- User-level Pthread implementation is also available