PRINCIPLES OF OPERATING SYSTEMS

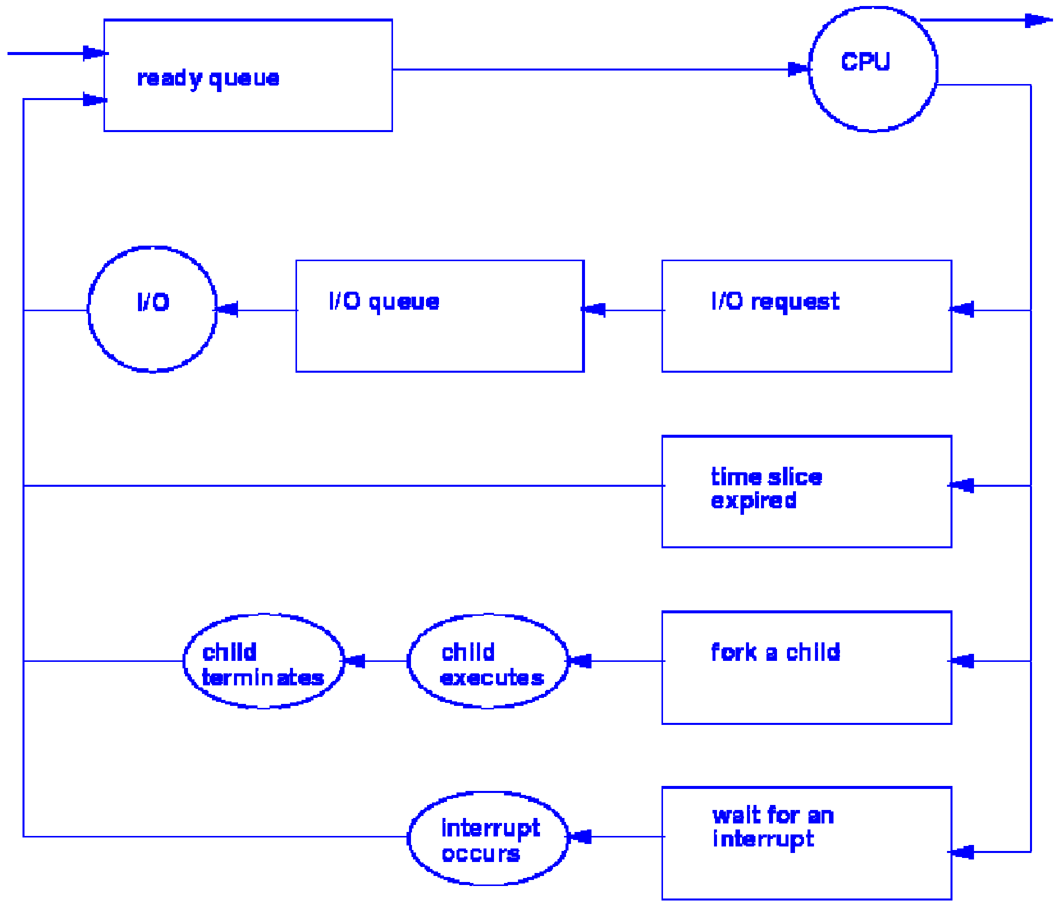
LECTURE- 6
Principles of Operating Systems

PROCESS SCHEDULING, SCHEDULERS

Process Scheduling

Process (PCB) moves from queue to queue

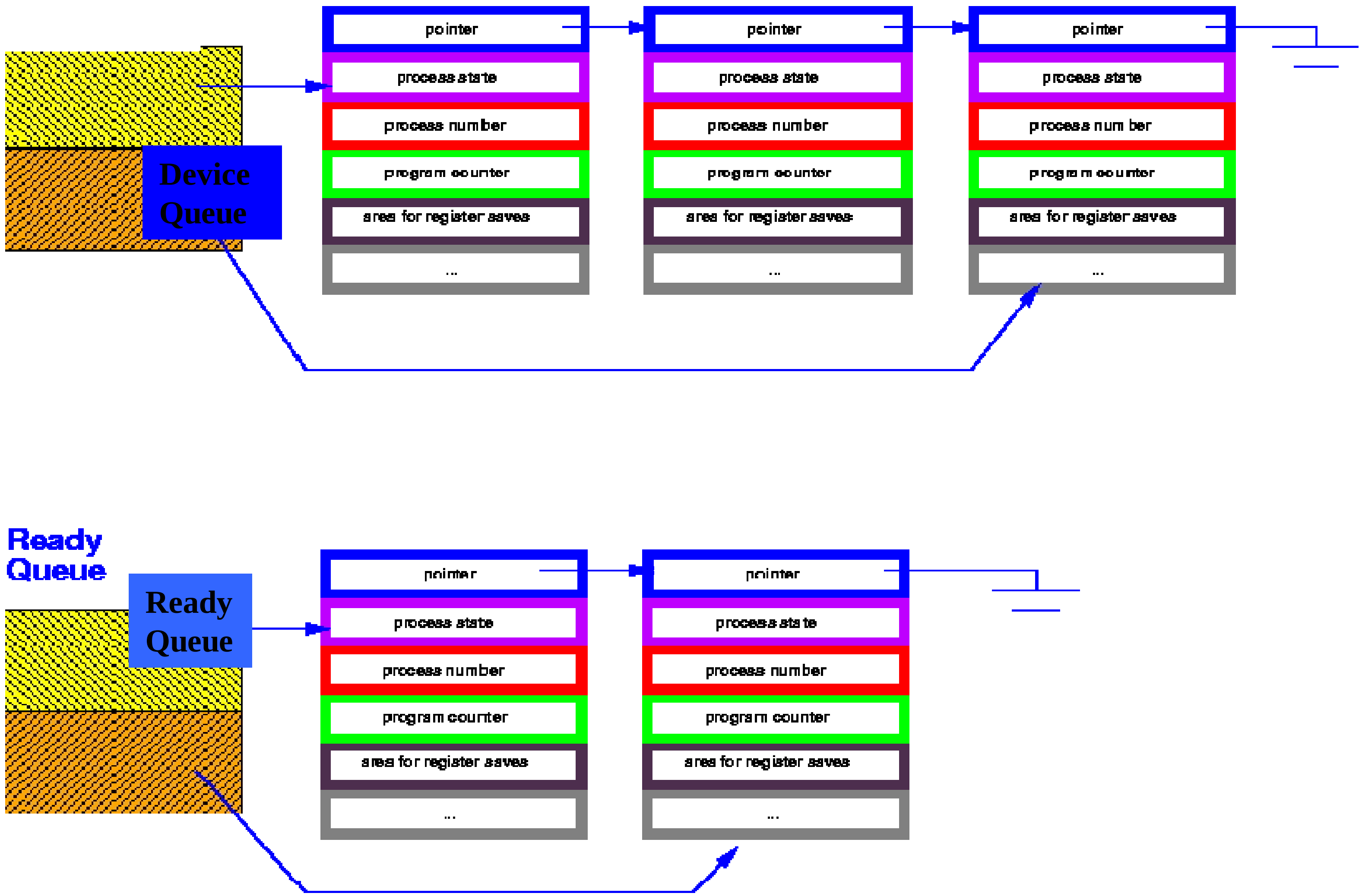
When does it move? Where? A scheduling decision



Process Scheduling Queues

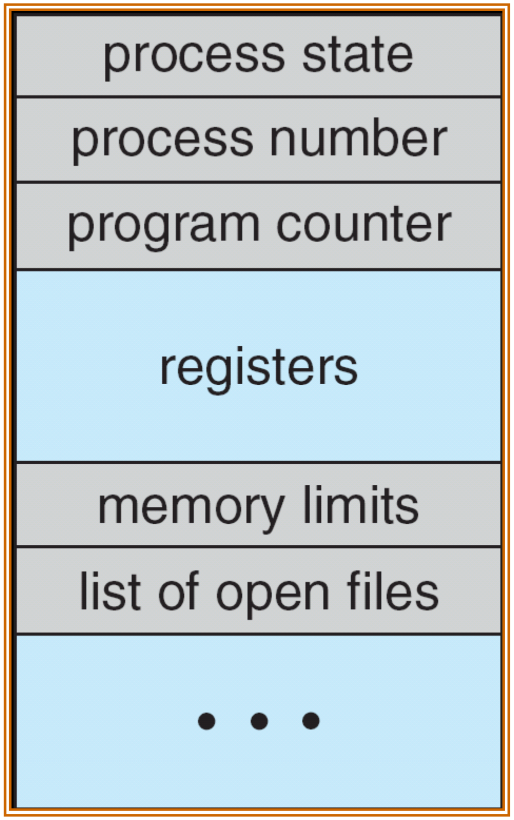
- Job Queue - set of all processes in the system
- Ready Queue - set of all processes residing in main memory, ready and waiting to execute.
- Device Queues - set of processes waiting for an I/O device.
- Process migration between the various queues.
- Queue Structures - typically linked list, circular list etc.

Process Queues



Enabling Concurrency and Protection: Multiplex processes

- Only one process (PCB) active at a time
  - Current state of process held in PCB:
    - "snapshot" of the execution and protection environment
  - Process needs CPU, resources
- Give out CPU time to different processes (Scheduling):
  - Only one process "running" at a time
  - Give more time to important processes
- Give pieces of resources to different processes (Protection):
  - Controlled access to non-CPU resources
    - E.g. Memory Mapping: Give each process their own address space

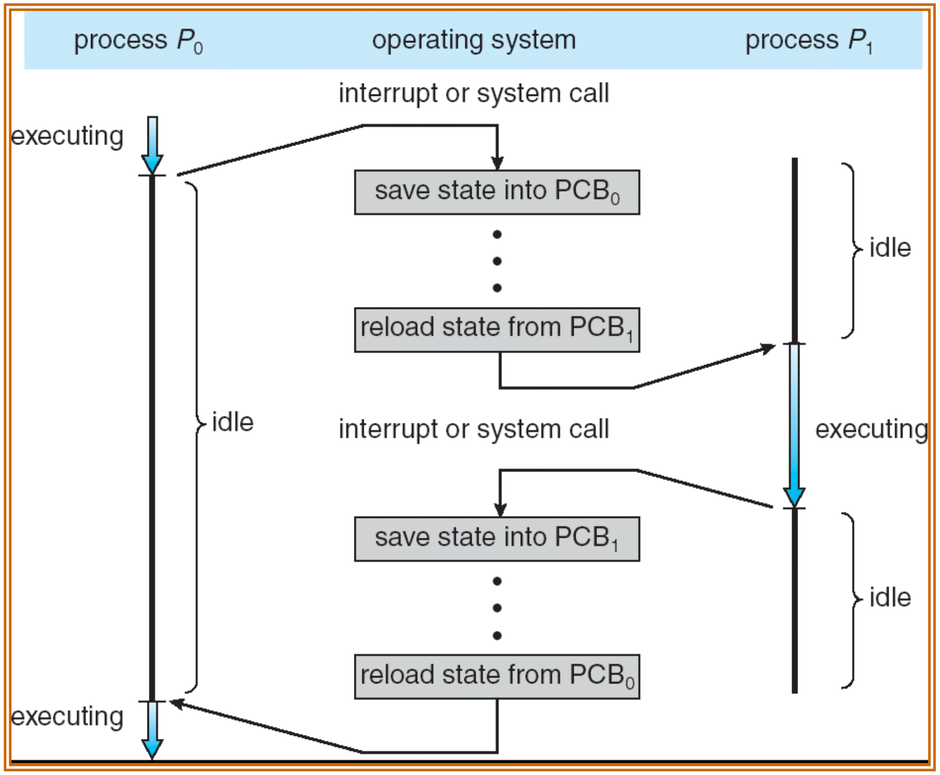


Process Control Block

Enabling Concurrency: Context Switch

- Task that switches CPU from one process to another process
  - the CPU must save the PCB state of the old process and load the saved PCB state of the new process.
- Context-switch time is overhead
  - System does no useful work while switching
  - Overhead sets minimum practical switching time; can become a bottleneck
- Time for context switch is dependent on hardware support (1- 1000 microseconds).

CPU Switch From Process to Process

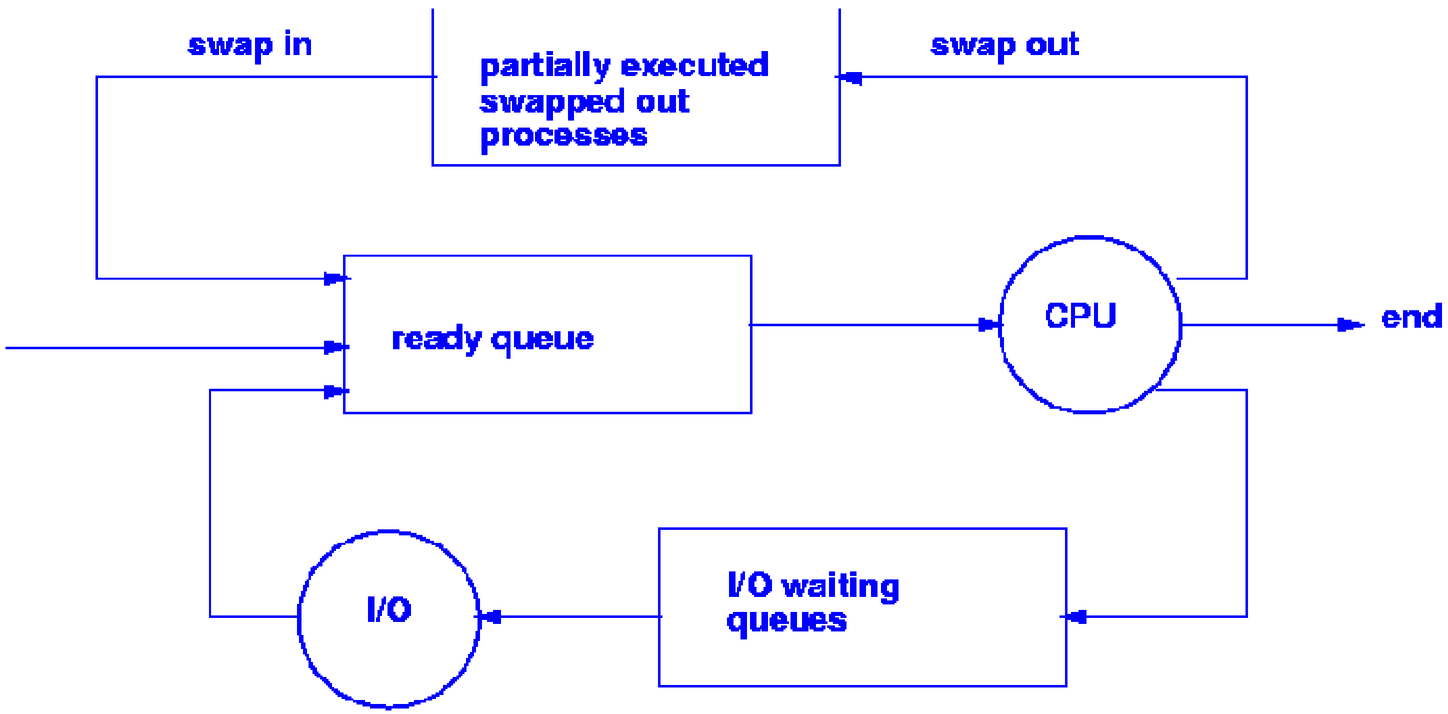


- Code executed in kernel above is overhead
  - Overhead sets minimum practical switching time

Schedulers

- Long-term scheduler (or job scheduler) -
  - selects which processes should be brought into the ready queue.
  - invoked very infrequently (seconds, minutes); may be slow.
  - controls the degree of multiprogramming
- Short term scheduler (or CPU scheduler) -
  - selects which process should execute next and allocates CPU.
  - invoked very frequently (milliseconds) - must be very fast
- Medium Term Scheduler
  - swaps out process temporarily
  - balances load for better throughput

Medium Term (Time-sharing) Scheduler



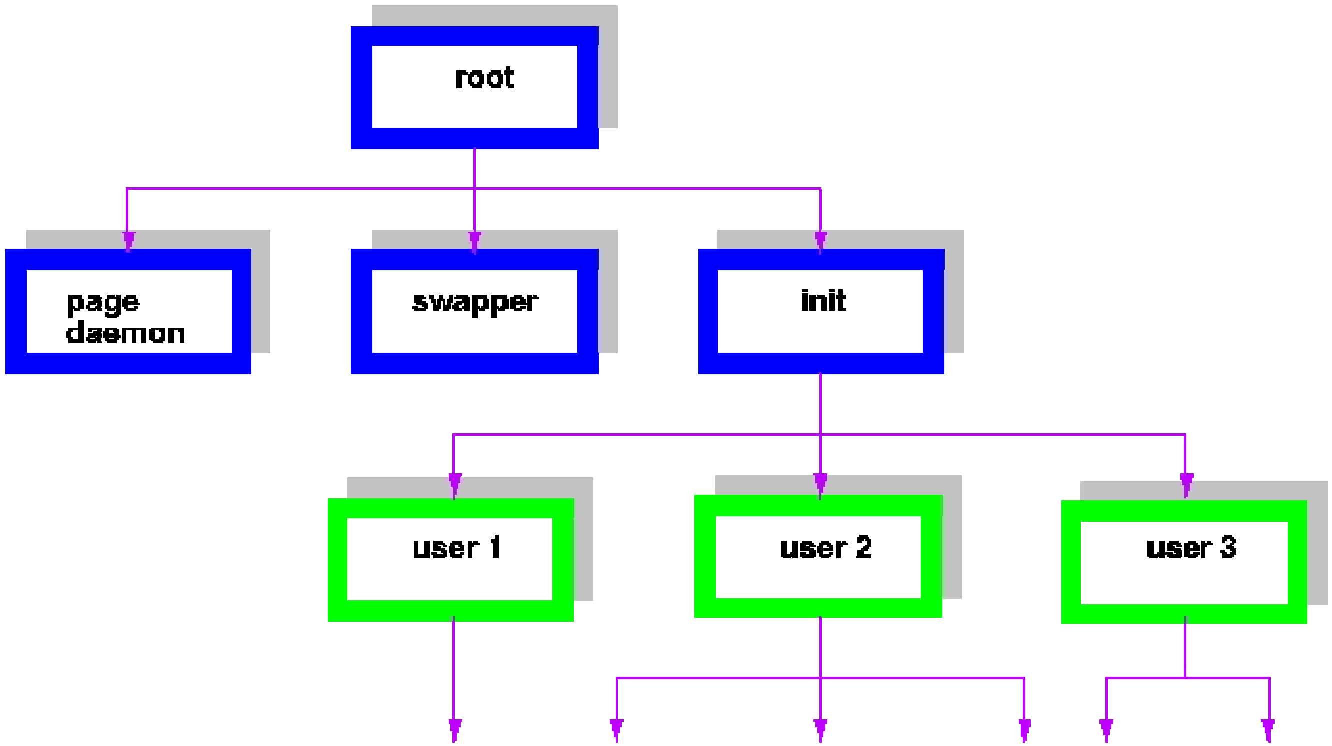
Process Profiles

- I/O bound process -
  - spends more time in I/O, short CPU bursts, CPU underutilized.
- CPU bound process -
  - spends more time doing computations; few very long CPU bursts, I/O underutilized.
- The right job mix:
  - Long term scheduler - admits jobs to keep load balanced between I/O and CPU bound processes
  - Medium term scheduler – ensures the right mix (by sometimes swapping out jobs and resuming them later)

Process Creation

- Processes are created and deleted dynamically
- Process which creates another process is called a *parent* process; the created process is called a *child* process.
- Result is a tree of processes
  - e.g. UNIX - processes have dependencies and form a hierarchy.
- Resources required when creating process
  - CPU time, files, memory, I/O devices etc.

UNIX Process Hierarchy



•

What does it take to create a process?

- Must construct new PCB
  - Inexpensive
- Must set up new page tables for address space
  - More expensive
- Copy data from parent process? (Unix `fork()`)
  - Semantics of Unix `fork()` are that the child process gets a complete copy of the parent memory and I/O state
  - Originally *very* expensive
  - Much less expensive with “copy on write”
- Copy I/O state (file handles, etc)
  - Medium expense

Process Creation

- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources - prevents many processes from overloading the system.
  - Parent and children share no resources.
- Execution
  - Parent and child execute concurrently.
  - Parent waits until child has terminated.
- Address Space
  - Child process is duplicate of parent process.
  - Child process has a program loaded into it.

UNIX Process Creation

- Fork system call creates new processes
- execve system call is used after a fork to replace the processes memory space with a new program.

Process Termination

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