Process Scheduling

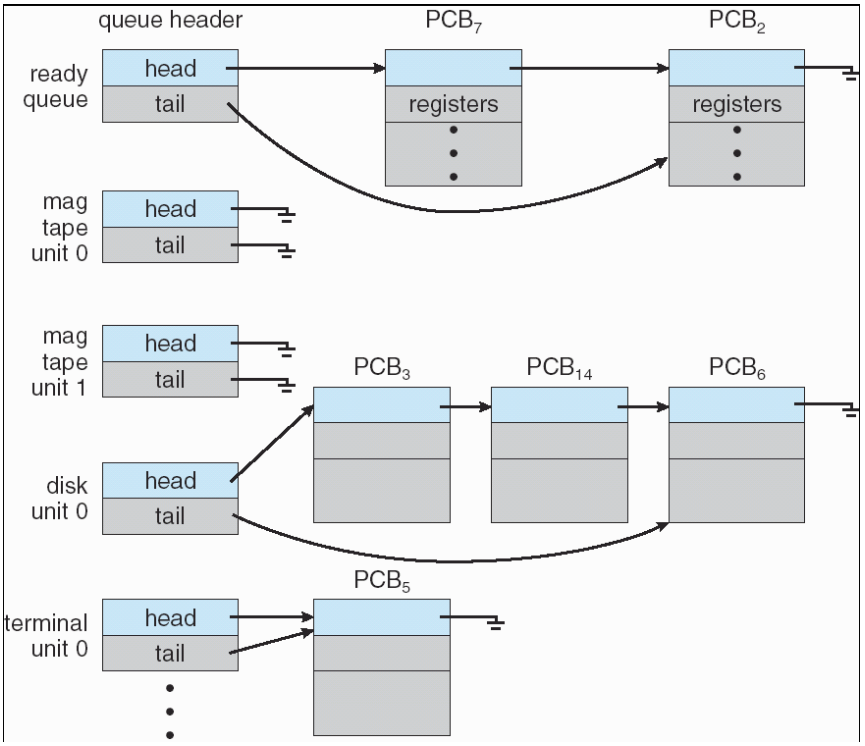
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization
    - The objective of time sharing is to switch the CPU among processes so frequently that the user can interact with each program while it is running
    - To meet this objectives, the process scheduler selects an available process for execution on the CPU
    - For single-processor system, there will never be more than one running process
    - If more than one process, it will have to wait until CPU is free and can be rescheduled

Scheduling Queues

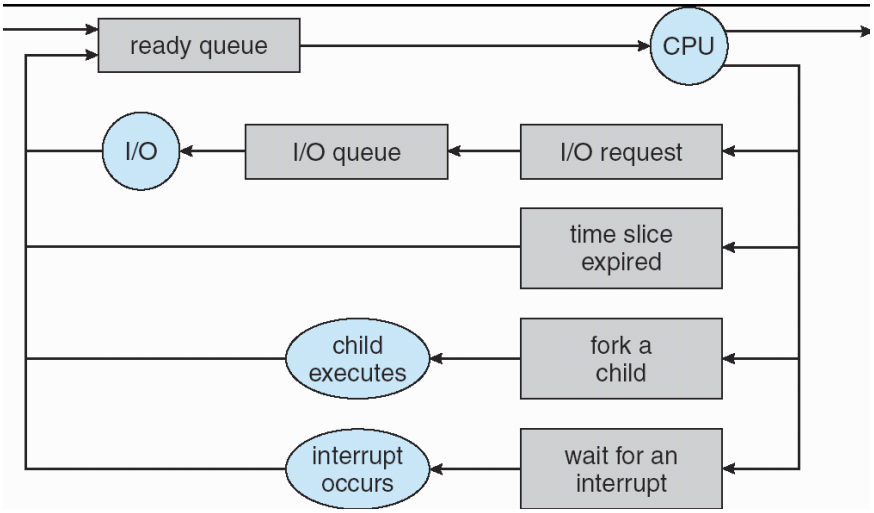
- As processes enter the system, they are put into a **job queue**
- Processes in memory, waiting to execute, are in the **ready queue**
- A ready queue header contains pointers to the fist & last PCBs in the list, each of which has a pointer to the next PCB
- **Device queue** = the list of processes waiting for an I/O device
- After a process in the ready queue is selected for execution…
    - it could issue an I/O request and be put in the I/O queue

- it could create a sub-process and wait for its termination
- it could be interrupted and go to the ready queue

- Processes migrate among the various queues



- Queuing-diagram representation of process scheduling



Schedulers

- A process migrates between the various scheduling queues throughout its lifetime
- The appropriate scheduler selects processes from these queues
- In a batch system, more processes are submitted than can be executed immediately
  - These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution
- The **long-term scheduler** / **job scheduler** selects processes from this pool and loads them into memory for execution

- The **short-term scheduler** / **CPU scheduler** selects from among the processes that are ready to execute, and allocates the CPU to it
- The main **difference** between these two schedulers is the **frequency of execution** (short-term = more frequent)
- The degree of multiprogramming (= the number of processes in memory) is controlled by the long-term scheduler
- I/O-bound process = spends more time doing I/O than computations, many short CPU bursts
- CPU-bound process = spends more time doing computations; few very long CPU bursts
- The long-term scheduler should select a good process mix of I/O-bound and CPU-bound processes for good performance
- Some time-sharing systems have a **medium-term scheduler**:
    - It **removes processes from memory** and thus **reduces the degree of multiprogramming**
    - Later, the process can be reintroduced into memory and its execution can be continued where it left off (= Swapping)
    - Swapping may be necessary to improve the process mix, or because memory needs to be freed up

Context Switch

- Context switch = saving the state of the old process and switching the CPU to another process
- The context of a process is represented in the PCB of a process
- (It includes the value of the CPU registers, process state, and memory-management information)
- Context-switch time is pure overhead, because the system does no useful work while switching
- Context-switch time is highly dependent on hardware support (e.g. some processors provide multiple sets of registers)

Operations on

Processes

Process

Creation

- Parent process = the creating process
- Children = new processes created by parent ones
- Sub-processes may…
    - get resources directly from the OS
    - be constrained to a subset of the parent's resources (This prevents sub-processes from overloading the system)
- When child processes are created, they may obtain initialization data from the parent process (in addition to resources)
- Execution possibilities when a process creates a new one:
    - The parent continues to execute concurrently with children
    - The parent waits until some / all children have terminated
- Address space possibilities when a process creates a new one:

    - The child process is a duplicate of the parent
    - The child process has a program loaded into it
- **UNIX example**
    - **fork** system call creates new process