**Chapter 3 Process**

**Process Concept**

* An OS executed a variety of programs:
* Batch system --- jobs
* Time shared system ---- user programs or tasks
* Textbook uses the terms **job** and **process** almost interchangeably
* **Process**--- a program in execution; process execution must progress in sequential fashion
* A process includes:

Program counter

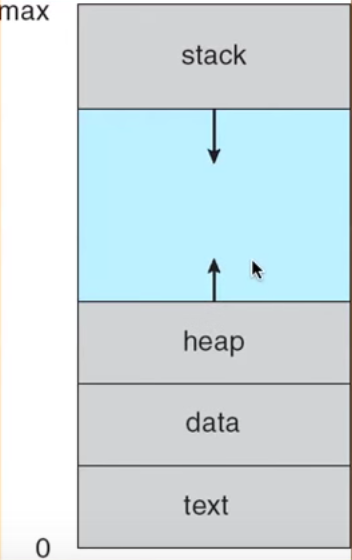
Stack

Data section

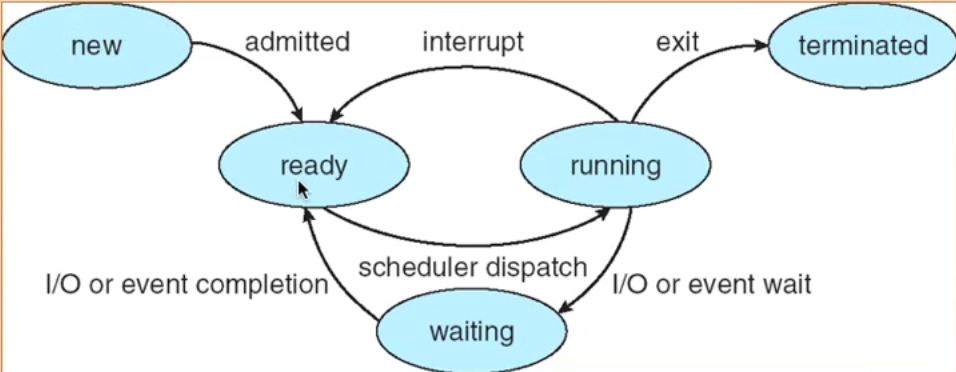
Heap: dynamic allocate memory, but never reallocate

Text: reform to the executable file

Data: variable



Process State



New: Process is being created

Running: Instructions are being executed

Waiting: process is waiting for some event to occur

Ready: The process is waiting to be assigned to a process

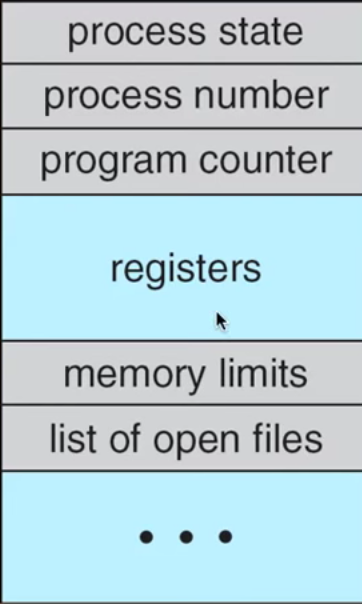
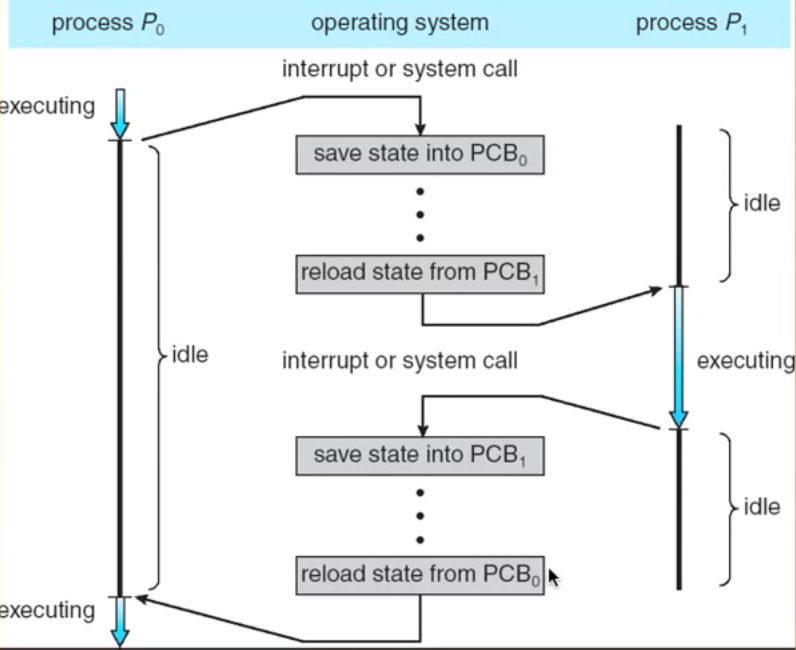
Terminated: process has finished execution

Process Control Block

* The kernel owns this data structure called PCB

CPU Switch from Process to Process

* Also refers to context Switch
* CPU scheduling is nothing more than take the process out of the queue, and looking for some information in the PCB, and loading it to do some work. And then stop it, put it back, update in the PCB, start a new process

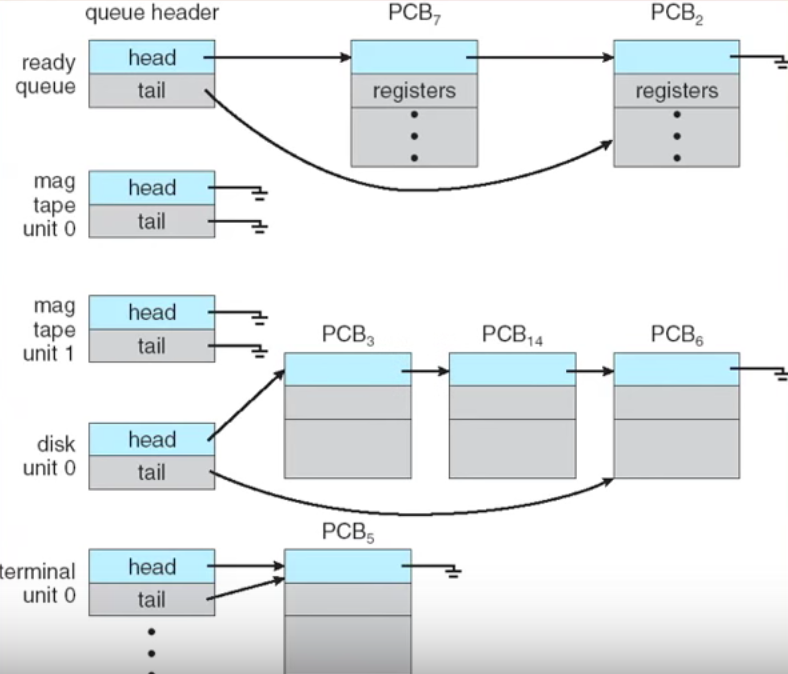
 

And Switching goes on between the first process stop and second process gets to start. Then second Stop, go back to the first one. And keep going.

Process Scheduling Queues

* **Job queue** --- set of processes in the system
* **Ready queue** --- set of processes residing in main memory
* **Device queues** --- set of processes waiting for an I/O device
* Process migrate among the various queues

Ready queue and various I/O device queues



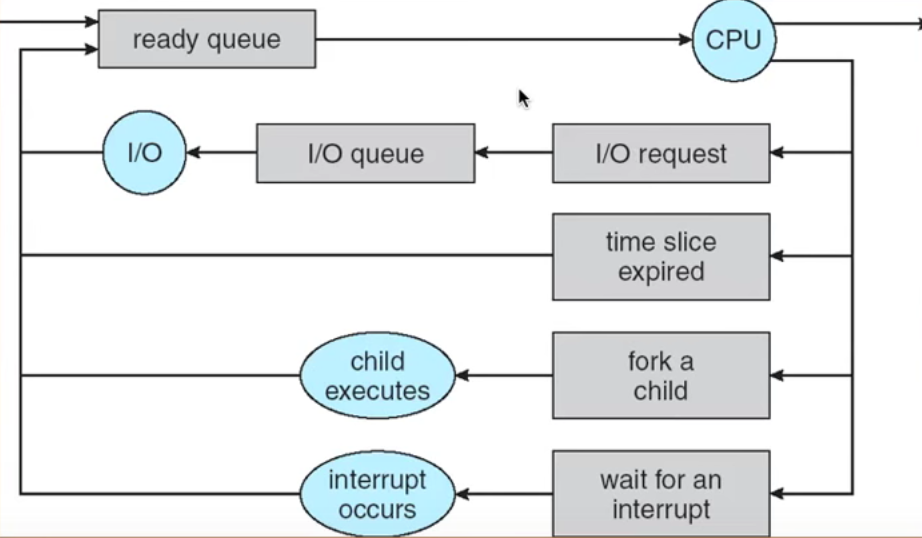
It is a linked list.

Process can control the walk is keeping track of the number which is running.

It is easier to schedule the process

Keeping going to check the next process

Representation of Process Scheduling



Everything goes to the ready queue at the very beginning.

Fork a child and child executes and go back to ready queue

Scheduler to creating this map

Schedulers

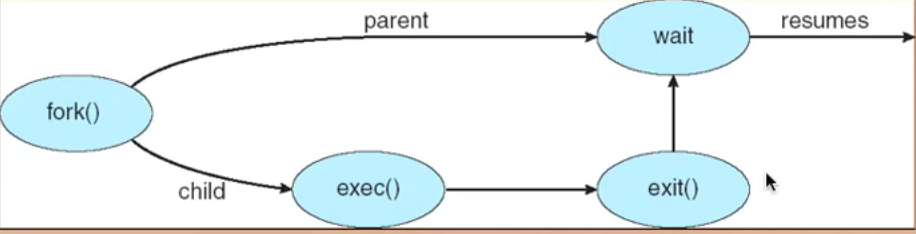
* **Long-term scheduler** (or job scheduler) --- selects which process should be brought into the ready queue
* **Short-term scheduler** (or CPU scheduler) --- selects which process should be executed next and allocates CPU
* Short-term scheduler is invoked very frequently (milliseconds) 🡪 (must be fast)
* Long-term scheduler is invoked very in frequently (seconds, minutes)🡪(may be slow)
* The long-term scheduler controls the degree of multiprogramming
* Process can be described as either:
* **I/o –bound process** --- spend more time doing I/O than computation, many short CPU bursts
* **CPU-bound process ---** spends more time doing computations; few very long bursts

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 not useful work while switching
* Time depend on hardware support

Process Creation

* Parent process create children process, which, in turn create other process, forming a tree of process
* 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
* Address space
* Child duplicate of parent
* Child has a program loaded into it
* UNIX examples
* **Fork ()** system call creates new process
* **Exec ()** system call used after a **fork** to replace the process’s memory space with 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’s resources are deal located by operating system
* Parent may terminate execution of children process (**abort**)
* Child has exceeded allocated resources
* Task assigned to child is no longer required
* If parent is exiting
* Some operating system do not allow child to continue if its parent terminates

All children terminated --- cascading termination

Cooperating Processes

* Independent process cannot affect or be affected by the execution of another process
* Cooperating process can affect or be affected by the execution of another process
* Advantage of process cooperation
* Information sharing
* Computation speed up
* Modularity
* Convenience

Interposes Communication (IPC)

* Mechanism for processes to communicate and to synchronize their actions
* Message system --- process communicate with each other without resorting to shared variables
* IPC facility provides two operations:
* **Send** (message) --- message size fixed or variable
* **Receive** (message)
* If P and Q wish to communicate, they need to:
* Establish a communication link between them
* Exchange messages via send/receive
* Implementation of communication link
* Physical (e.g. shared memory, hardware bus)
* Logical (e.g., logical properties)

Synchronization

* Message passing may be either blocking or non-blocking
* **Blocking** is considered synchronous
* **Blocking send** has the sender block until the message is received
* **Blocking receive** has the receiver block until a message is available
* Non-blocking is considered asynchronous
* **Non-blocking** send has the sender send the message and continue
* **Non-blocking** receive has the receiver receive a valid message or null