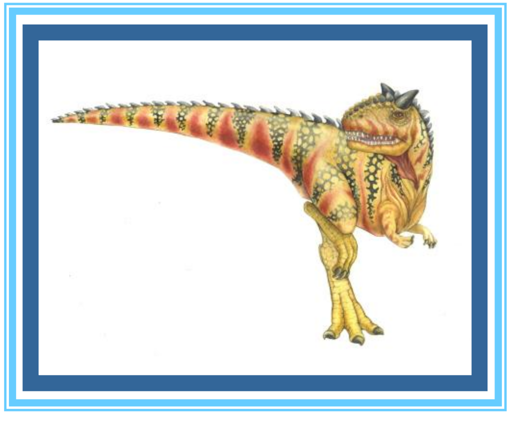
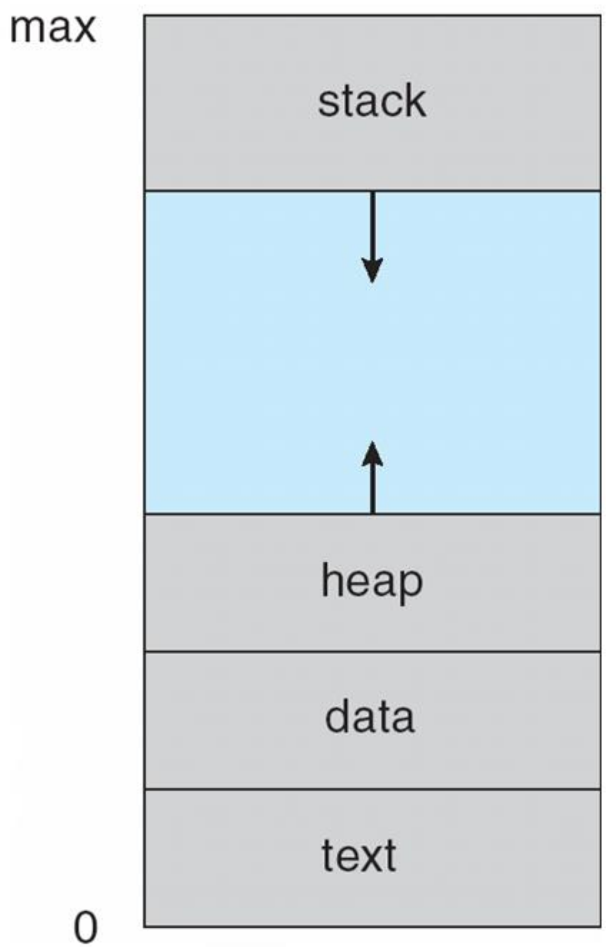
Process Concepts



Process Concept

- **Process** – a program in execution;
  - A process will need certain resources—such as CPU time, memory, files, and I/O devices to accomplish its task.
  - These resources are allocated to the process either when it is created or while it is executing
- Process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Process in Memory



A process is the unit of work in a modern time-sharing system

Process Memory Map

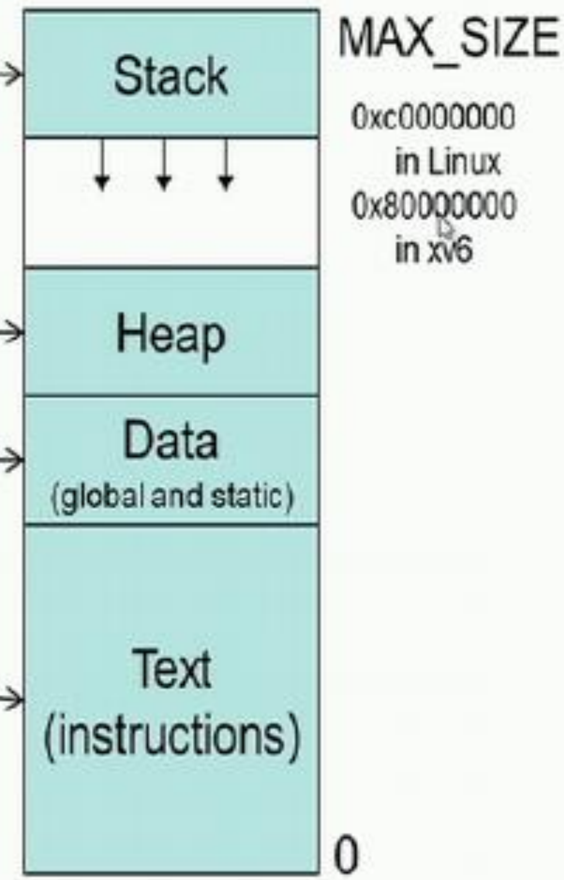
```c
#include <stdio.h>
#include <stdlib.h>

int calls;

void fact(int a, int *b)
{
    calls++;
    if (a == 1) return;
    *b = *b * a;
    fact(a - 1, b);
}

int main(){
   int n, *m;

   scanf("%d", &n);
   m = malloc(sizeof(int));
   *m = 1;
   fact(n, m);
   printf("Factorial(%d) is %d\n", n, *m);
   free(m);
}
```



Courtesy: Introduction to OS by Prof. Chester Rebeiro, IITM

Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

- Operating system processes executing system code and user processes executing user code.
- Potentially, all these processes can execute concurrently, with the CPU (or CPUs) multiplexed among them

Process State

- As a process executes, it changes **state**
- The state of a process is defined in part by the current activity of that process
- A process may be in one of the following states:
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur (such as an I/O completion or reception of a signal)
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution
- Only one process can be *running* on any processor at any instant. Many processes may be *ready* and *waiting*

Diagram of Process State

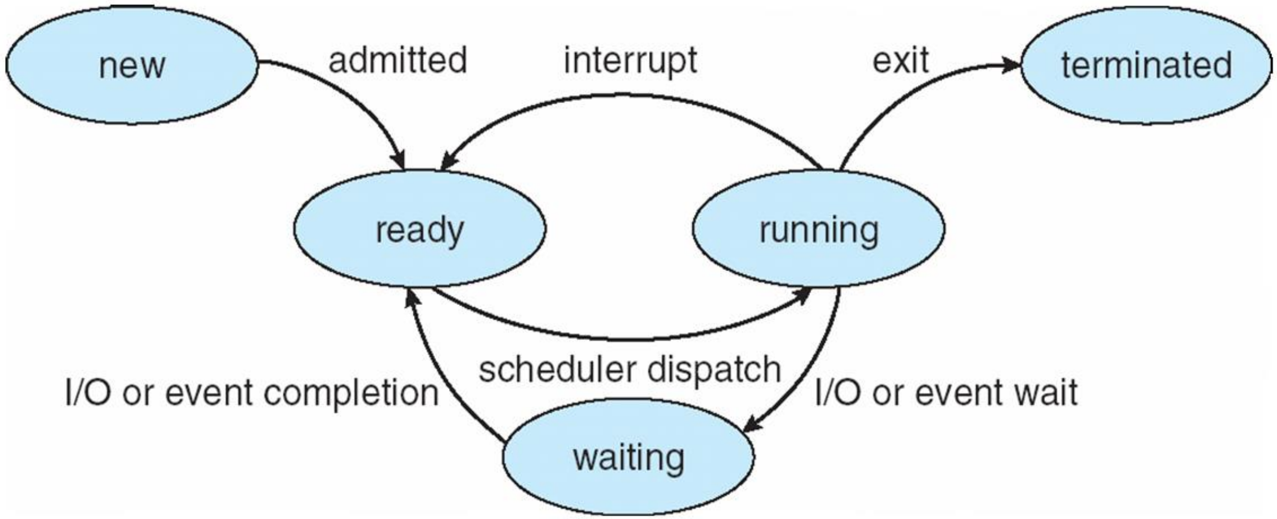
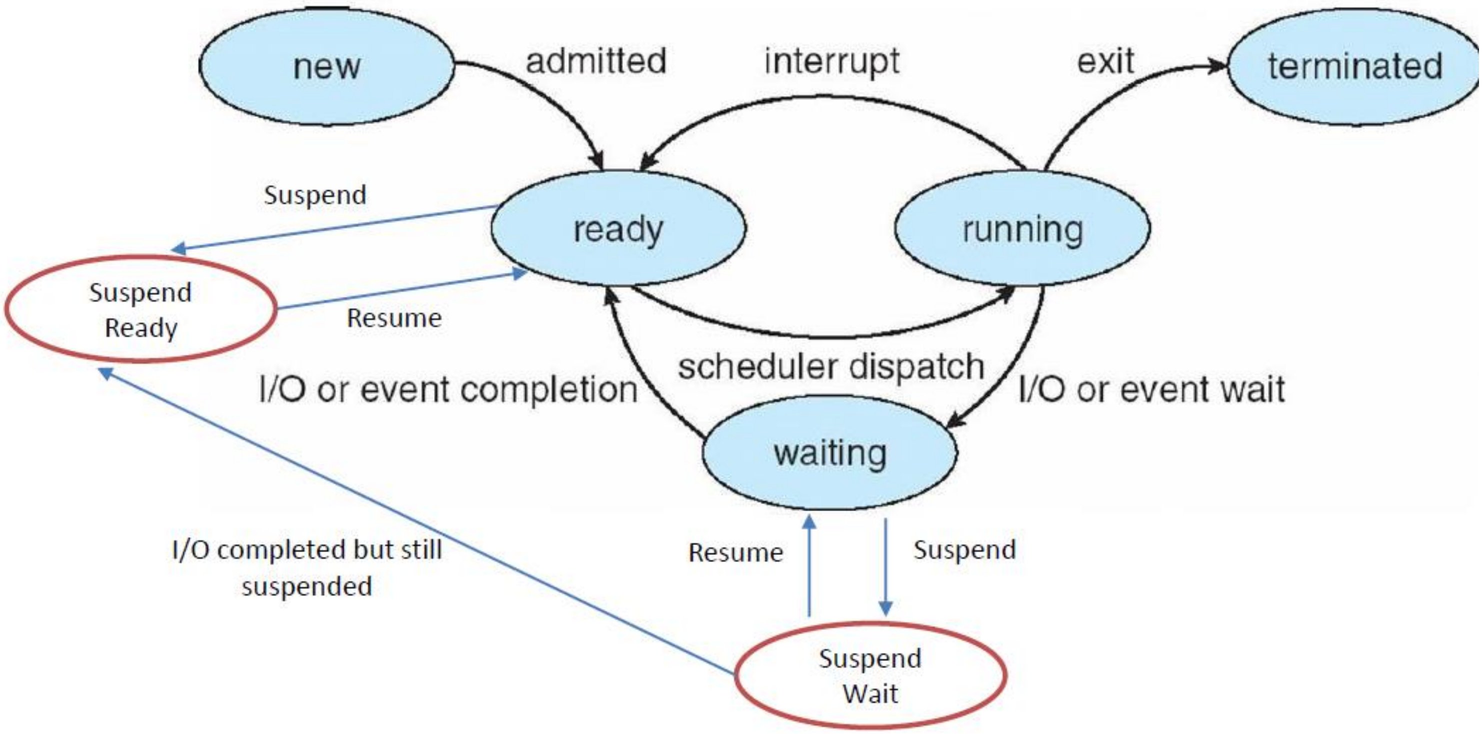


Diagram of Process State (Extended)



Process Control Block (PCB)

Information associated with each process (also called **task control block**)

- **Process state** – running, waiting, etc
- **Program counter** – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- **CPU scheduling information-** priorities, scheduling queue pointers
- **Memory-management information** – memory allocated to the process
- **Accounting information** – CPU used, clock time elapsed since start, time limits
- **I/O status information** – I/O devices allocated to process, list of open files

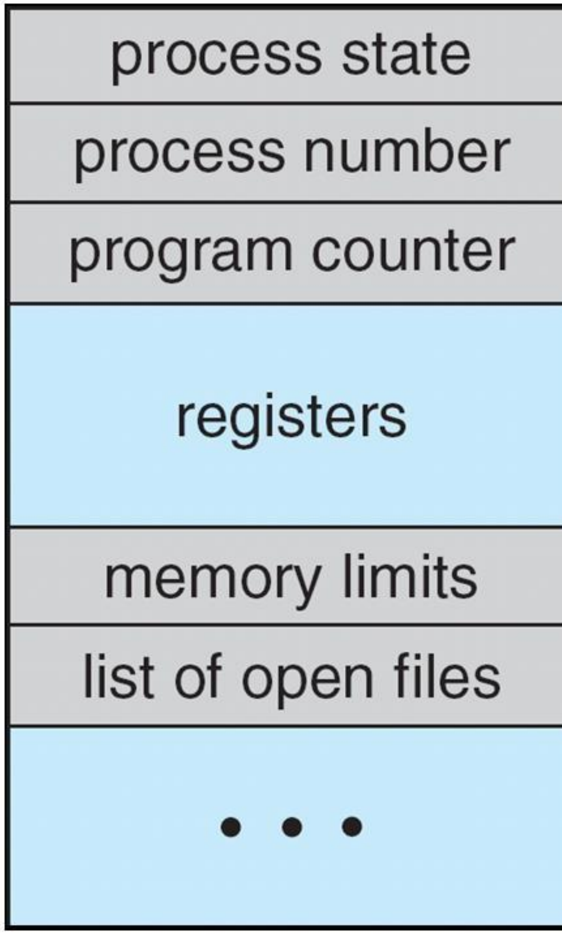
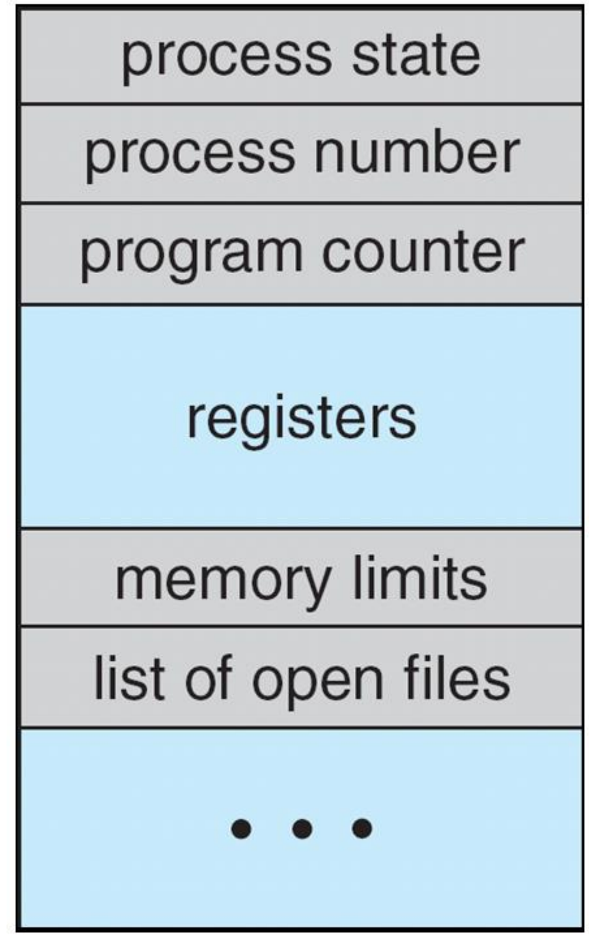


Fig:- Process Control Block (PCB)

Process Control Block (PCB)

- Each process is represented in the operating system by a **process control block (PCB)**—also called a **task control block**
- It contains many pieces of information associated with a specific process, including these:



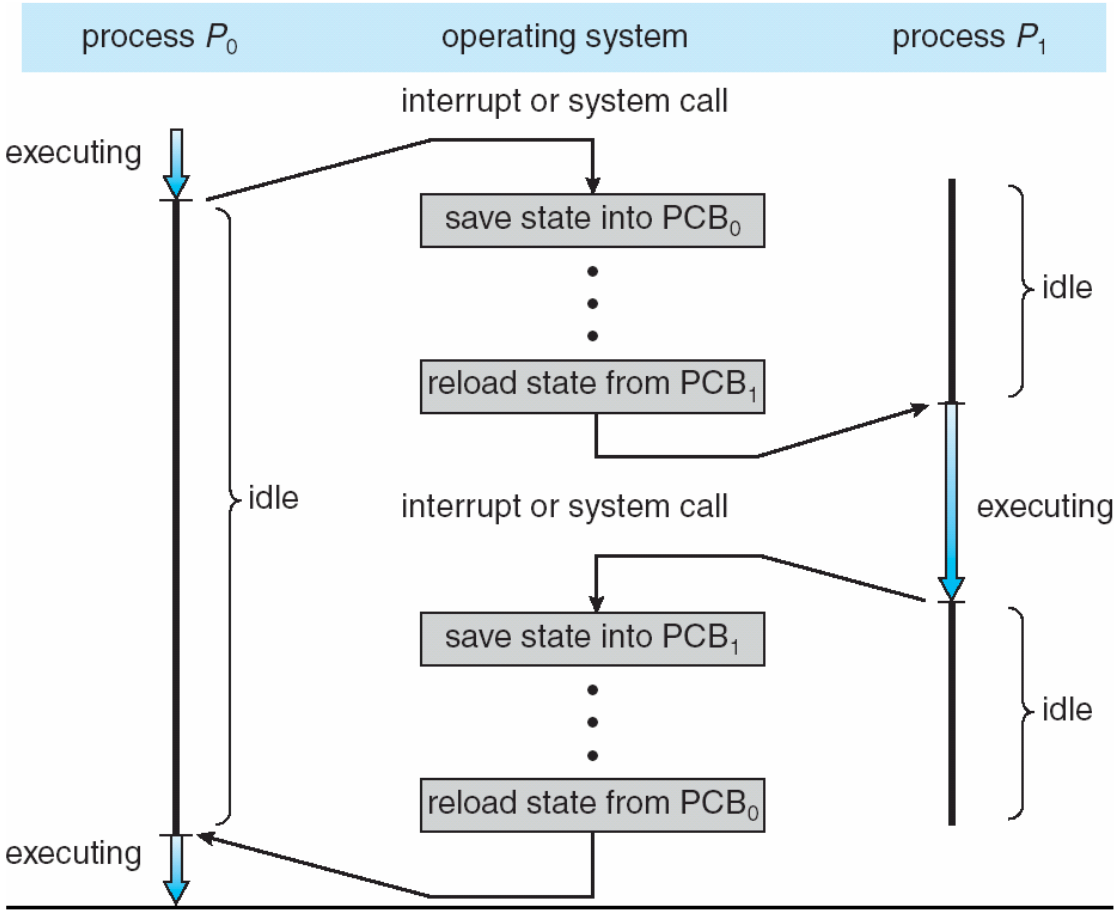
Process Control Block (PCB)

- **Process state**. The state may be new, ready, running, waiting, halted, and so on.
- **Program counter**. The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers**. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. **Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.**
- **CPU-scheduling information**. This information includes a **process priority**, **pointers to scheduling queues**, and any **other scheduling parameters**.

Process Control Block (PCB)

- **Memory-management information**. This information may include such items as the **value of the base and limit registers** and the page tables, or the segment tables, depending on the memory system used by the operating system (Chapter 8).
- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

CPU Switch From Process to Process



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 via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Process Scheduling

- **WHY:**
  - Several Processes competing at a time to get the CPU for their execution
- **Scheduling**
  - strategy and methods used by OS to decide which process is going to be allocated to CPU next among the several process in the queue for CPU time
- The objective **of multiprogramming** is
  - to have some process running at all times, to maximize CPU utilization
- Time sharing multiprogramming system

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing

- Process gives up the CPU under two conditions:
  - I/O request
  - After N units of time have elapsed

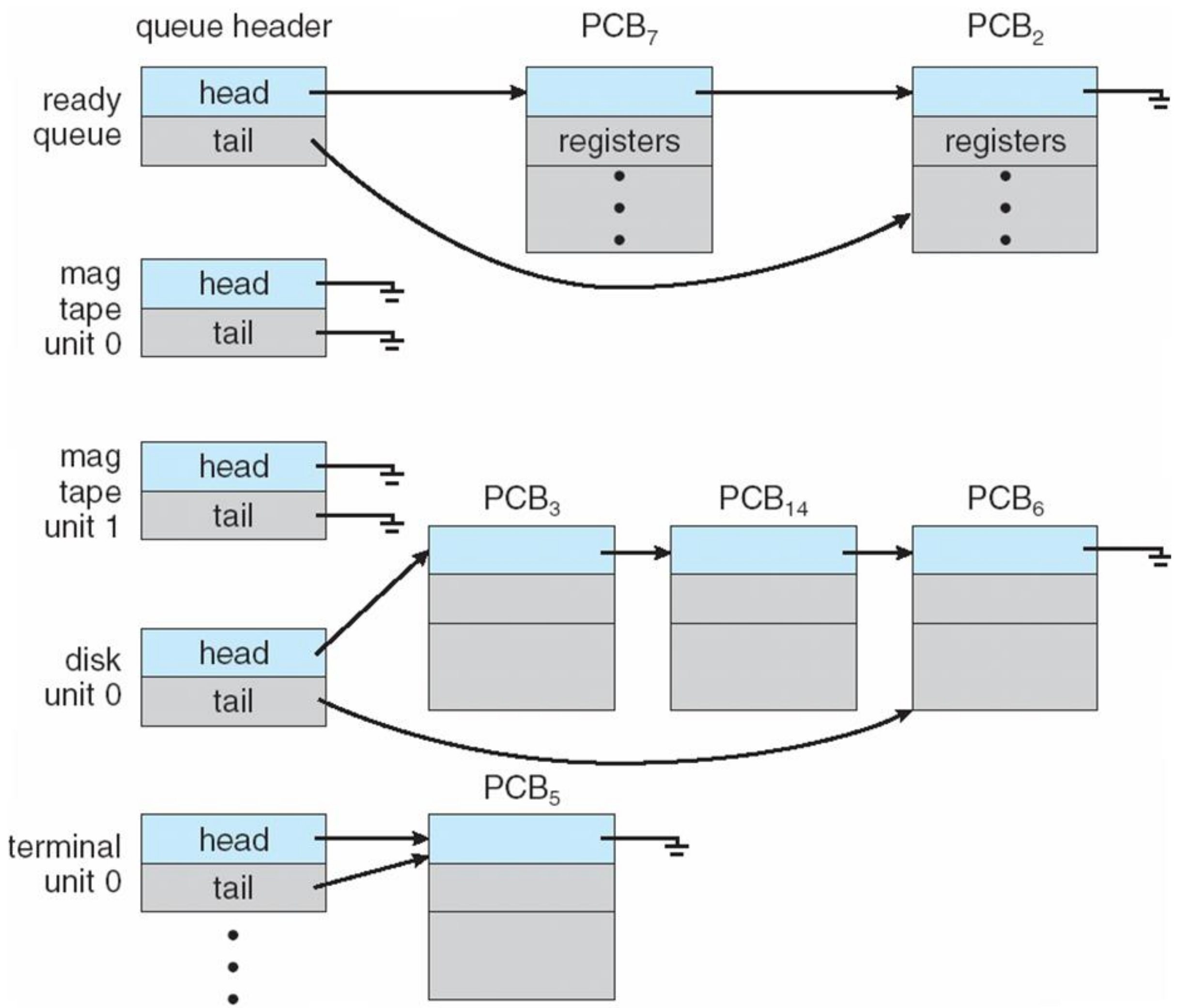
- Once a process gives up the CPU it is added to the ready queue

- **Process scheduler** selects among available processes for next execution on CPU

Process Scheduling

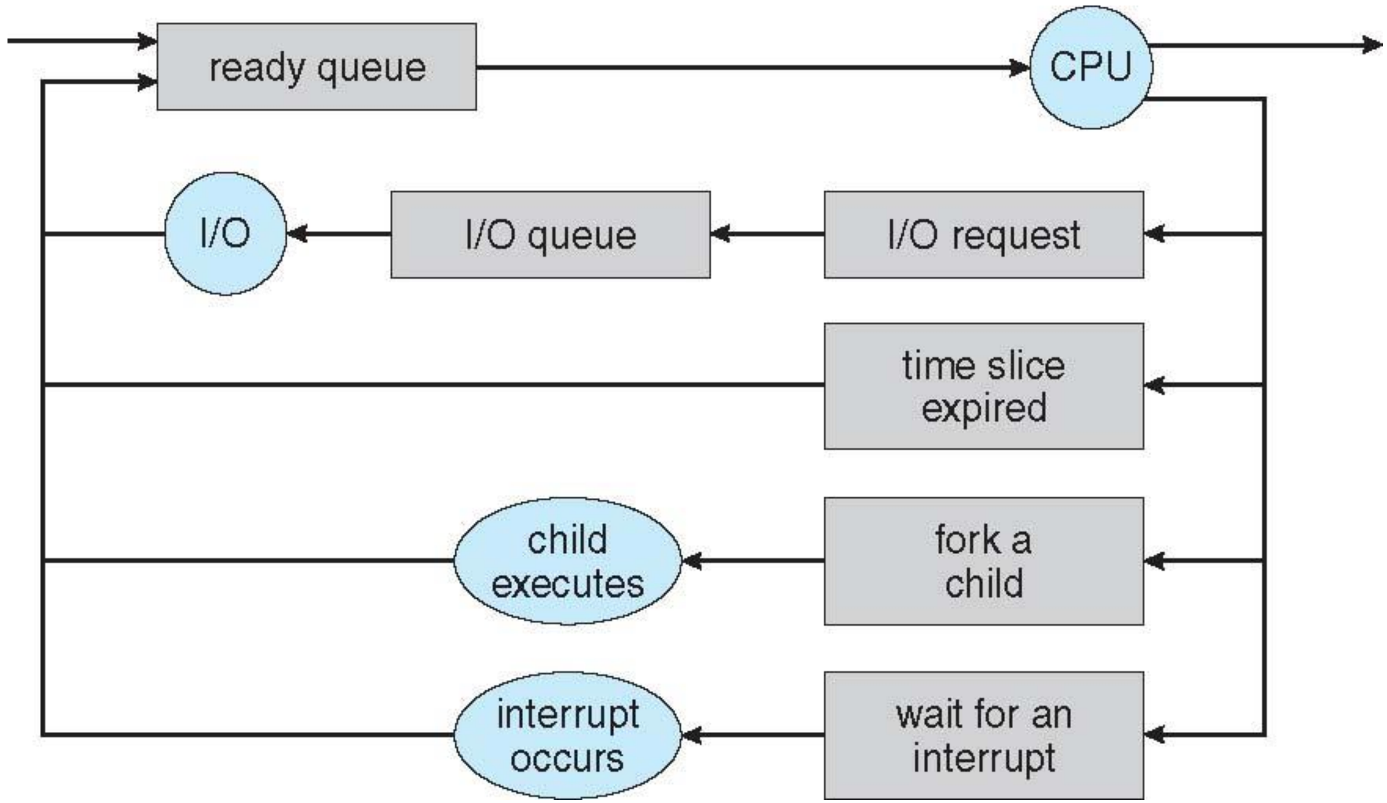
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows



Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **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
- Long-term scheduler strives for good ***process mix***

Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**