# Process Management

Alok Jhaldiyal
Assistant Professor
SoCSE, UPES
Dehradun

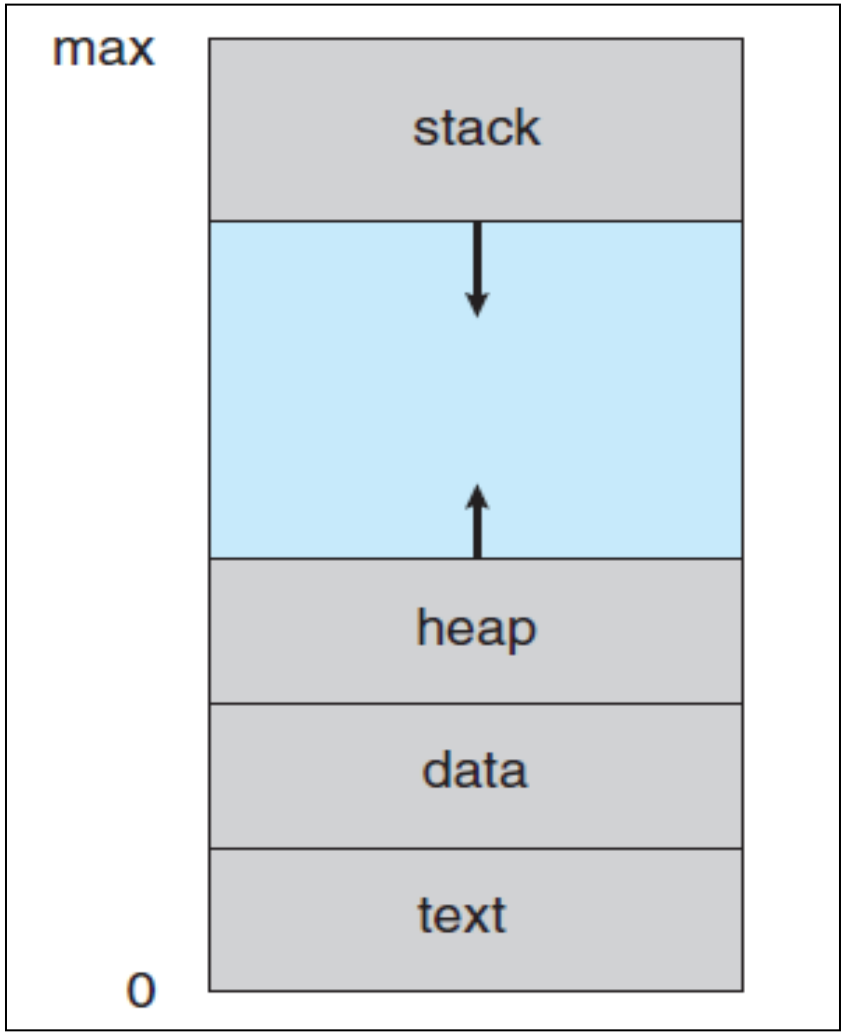
# What a process is ?

- Process is
  - A program in execution
  - Unit of work of most operating systems
  - Requires resources – CPU time, memory, I/O devices
- A system may consist of multiple running processes: System process running system code and user process running user code.
- Traditionally process used to have a single thread but modern operating system contains multiple thread.
- Important aspects of process management: Thread management, creation and deletion of process, scheduling process and maintain a synchronized and a communicated state.

# The Process

- A program is set of instruction stored in computers memory.
- A program may be active or in a passive state.
- Program in active state is a process.
- Process includes a program counter and data stored in the processor registers.
- A process generally includes process stack which contains temporary data, a data section,  and also a heap.

- Process Stack contains temporary data such as function parameters, return addresses, and local variables.
- Data section contains global variables
- Heap is the dynamically allocated memory during runtime.
- And text is program instructions



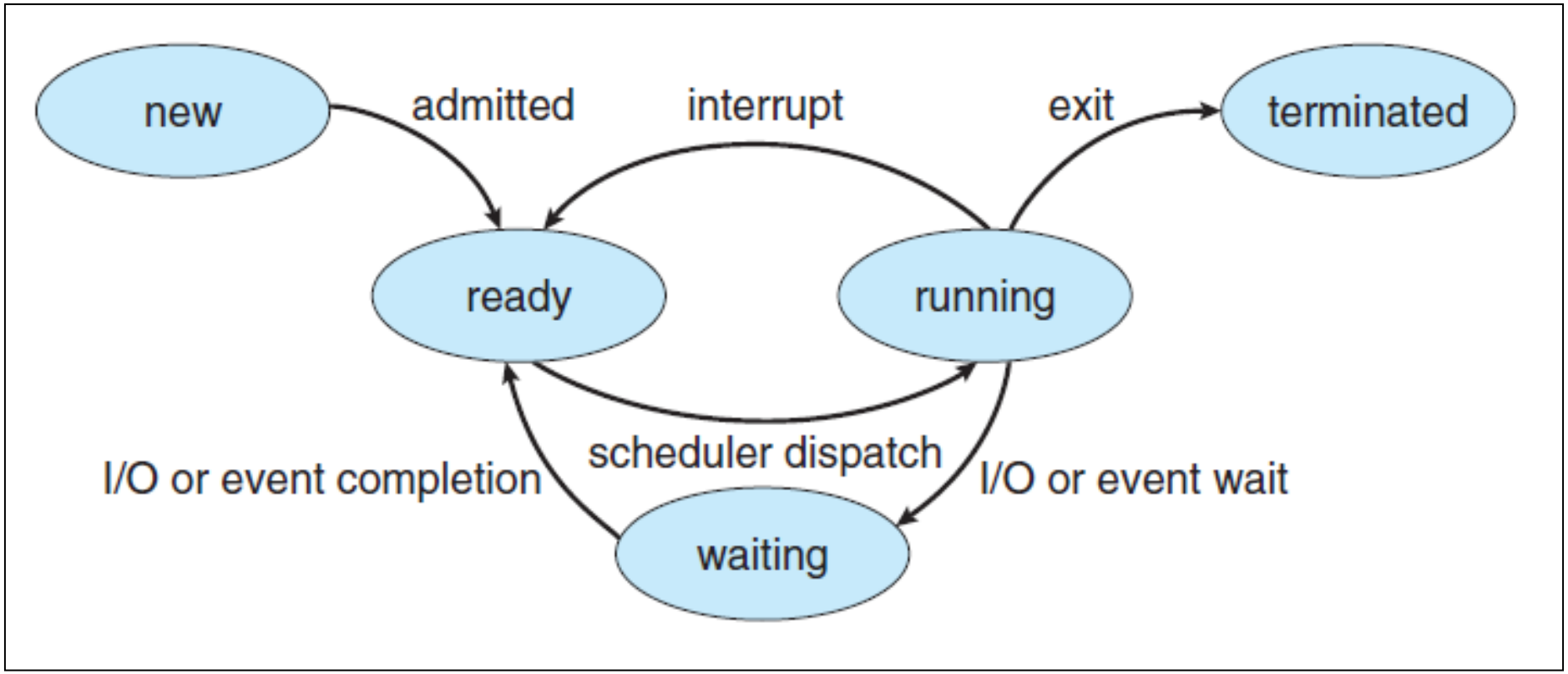

**Process in Memory**

# More about process

- A program becomes a process when an executable file is loaded into memory.
- Two processes may be associated with the same program, they are nevertheless considered two separate processes.
- Example can be a browser in which user has opened multiple tabs, but active tab is a separate process.

# Process State

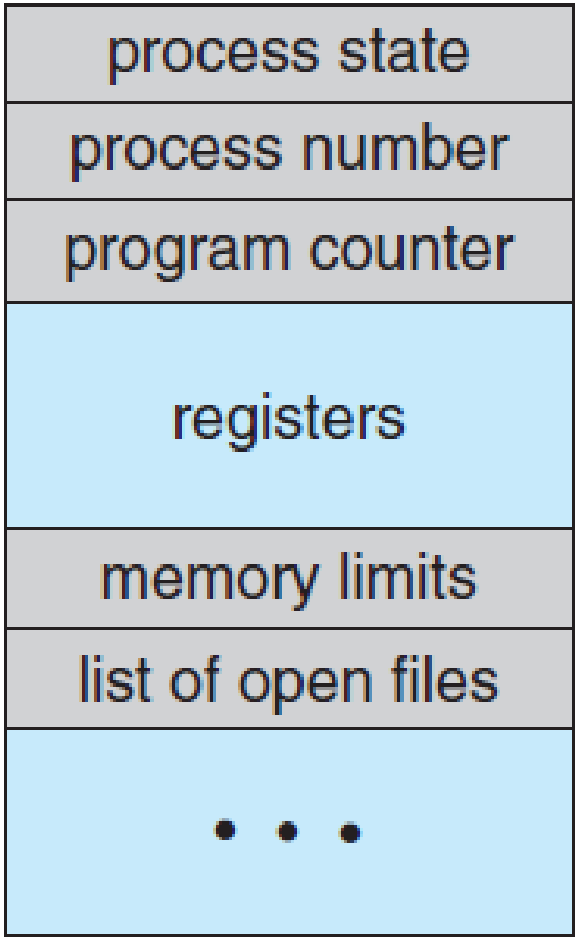
- When a process executes, it is in some state that we call process state.
- A process may be in one of the following states:
  - New. The process is being created.
  - Running. Instructions are being executed.
  - Waiting. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - Ready. The process is waiting to be assigned to a processor.
  - Terminated. The process has finished execution




**Process State Diagram**

# Process Control Block

- Representation of a process in operating system is done using a Process Control Block (PCB) or Task Control Block.
- A PCB is tired block and each block holding information associated with the particular process.

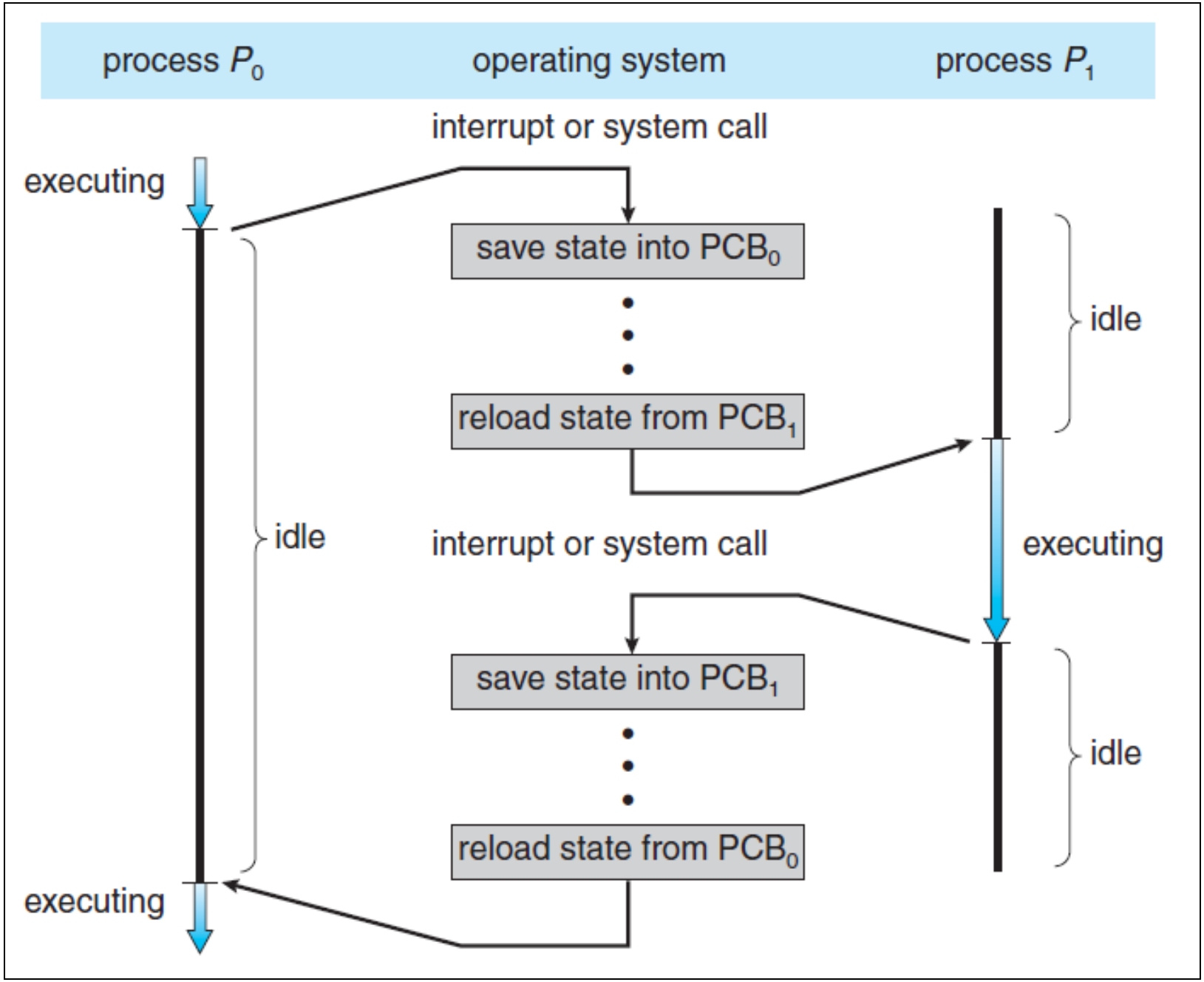



**Process Control Block**

# Process Control Block

- **Pieces of information in PCB:**
  - **Process state:** New, Ready, Running, Waiting, Halted and Terminated.
  - **Program counter**: Indicates the address of the next instruction.
  - **CPU registers:**
    - Vary in type and number
    - Includes accumulators, index registers, stack pointers, and general-purpose registers
    - During interrupt process state and program counter information is stored in registers.
  - **CPU-scheduling information:** Includes a process priority, pointers to scheduling queues, and other scheduling parameters.

- **Memory-management information:** Might include value of the base and limit registers and the page tables, based on the memory system used by the underlying system.
- **Accounting information:** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers.
- **I/O status information:** Includes the list of I/O devices allocated to the process and list of open files.




**CPU Switching between two concurrent processes**

**Class Activity: Assume that a process is running in which user is recording his voice and at the same time it is getting converted into text form. Based on the above scenario:**

1. **List all the required resources in context to operating system to run this process successfully.**
2. **Assume that due to system calls this process suffers two interrupts, using a diagram show the change in state on one side and highlight the changes in the PCB on the other side of your diagram.**

# Context Switching

- Context Switch: A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block (PCB) so that a process execution can be resumed from the same point at a later time.
- Process to switch the CPU between multiple programs which want to run on the system is context switching.
- Act of context switching can only be performed in kernel mode as only in kernel mode page map can be changed.

# Context Switching

- Get into kernel mode.
- Save the old program's registers somewhere, and the address of the next instruction it was about to execute, so they can be restored later, save in its PCB.
- Save the mappings in the current page map, also into the PCB.
- Unmap all of the old program's pages from the page map.
- Choose a new program to re-start. Find its PCB.
- Re-map the pages of the new program from its PCB.
- Re-load the registers of the new program from its PCB.
- Return to the next instruction of the new program, and return to user-mode at the same time.

# Process Scheduling

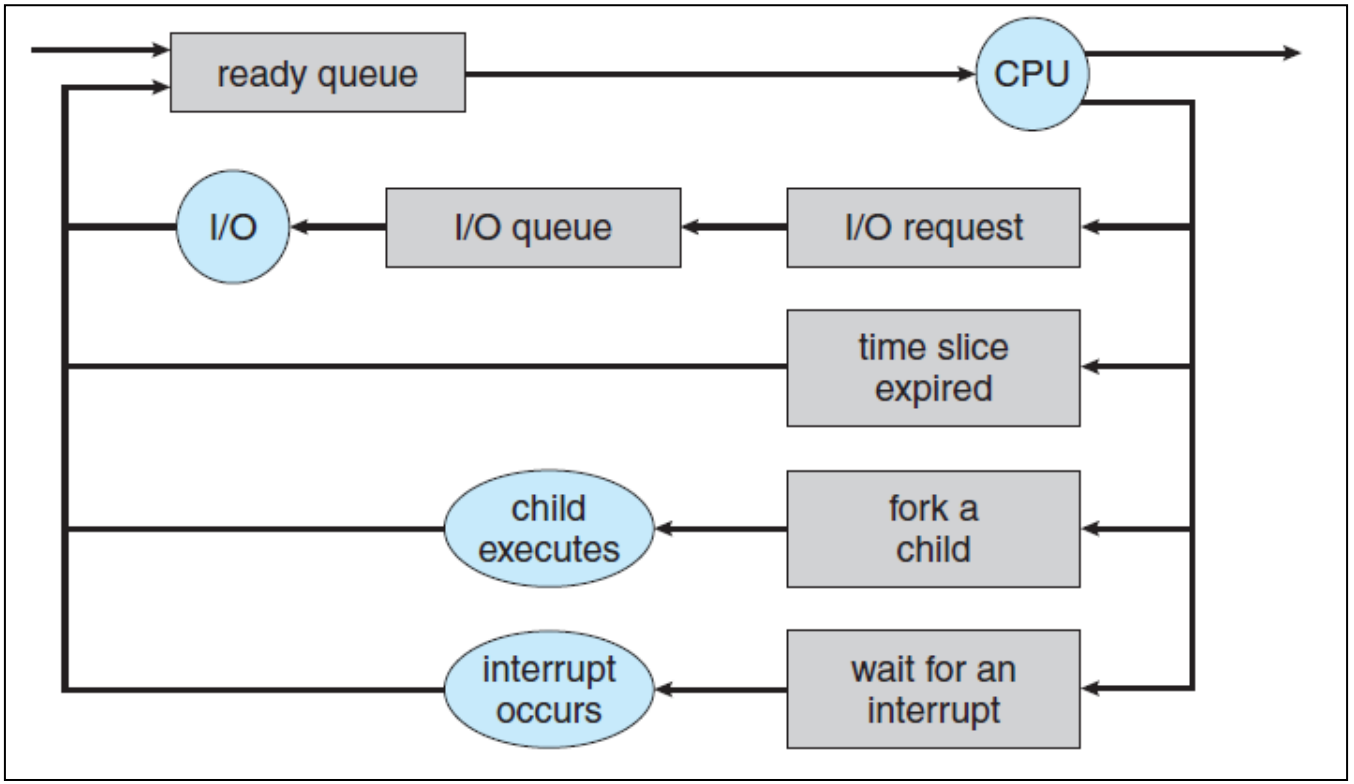
- Multiprogramming aims to maximize CPU utilization and it requires CPU to have some process running at all times, to maximize CPU utilization.
- Idea is to switch between CPU so frequently that the CPU can interact with each program.
- A process scheduler helps CPU find a available process.

# Scheduling Queues

- All incoming jobs are kept in  a job queue.
- Processes residing in main memory and are ready to be executed are kept in ready queue.
- Ready queue is a linked list and headers contain pointers to the first and last process PCB.
- Each PCB has a header field that holds the pointer to the next process PCB.
- A process waiting for a shared device is kept in a device queue.
- Each device has its own device queue and a oval denotes a resource.

# Queuing Diagram

- A common representation of process scheduling is a queueing diagram.
- A queuing diagram is made of ovals and rectangles and connected using arrows.
- Two types of queues are present: the ready queue and a set of device queues.
- Rectangular box represents a queue




- A new process waits in a ready queue.
- Once process is assigned a CPU, it is executing.
- Once a process is executing several events may occur:
  - The process could issue an I/O request and then be placed in an I/O queue.
  - The process could create a new child process and wait for the child's
  - termination.
  - The process could be removed forcibly from the CPU, as a result of an
  - interrupt, and be put back in the ready queue.

# Schedulers

- A process since its creation passes through several queues.
- A schedulers responsibility is to select an appropriate and available process.
- Before getting a CPU time a process is spooled in memory disk.
- Schedulers types:
  - Long term scheduler
  - Short term scheduler
- The two schedulers are distinguished by the frequency of there operation.

# Schedulers

- Short term scheduler:
  - It is CPU scheduler
  - Selects from among the processes that are ready to execute and allocates the CPU to one of them.
  - A CPU might run for few milliseconds before waiting for a input.
  - A short term scheduler needs to be very fast to select a process and assign CPU.
  - It often switches process at every 100 milliseconds.

# Schedulers

- Long term schedulers:
  - These schedulers function less frequently, often minutes may separate two process.
  - Long term schedulers control the degree of multiprogramming in the system i.e. process loaded in the memory.
  - The rate to process creation must match the process departure rate.
  - Long term schedulers have flexibility to give more time to select a process.

# Schedulers

- For efficiently utilizing CPU time a long term scheduler should select a proper mix of processes.
- A process is usually Input bound or CPU bound:
  - Input Bound: It is a process that spends more of its execution time waiting for inputs.
  - CPU Bound: A CPU bound generates I/O requests infrequently, using more of its time doing computations
- If all processes are I/O bound, the ready queue will almost always be empty, short term scheduler will have very less to do.
- If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused

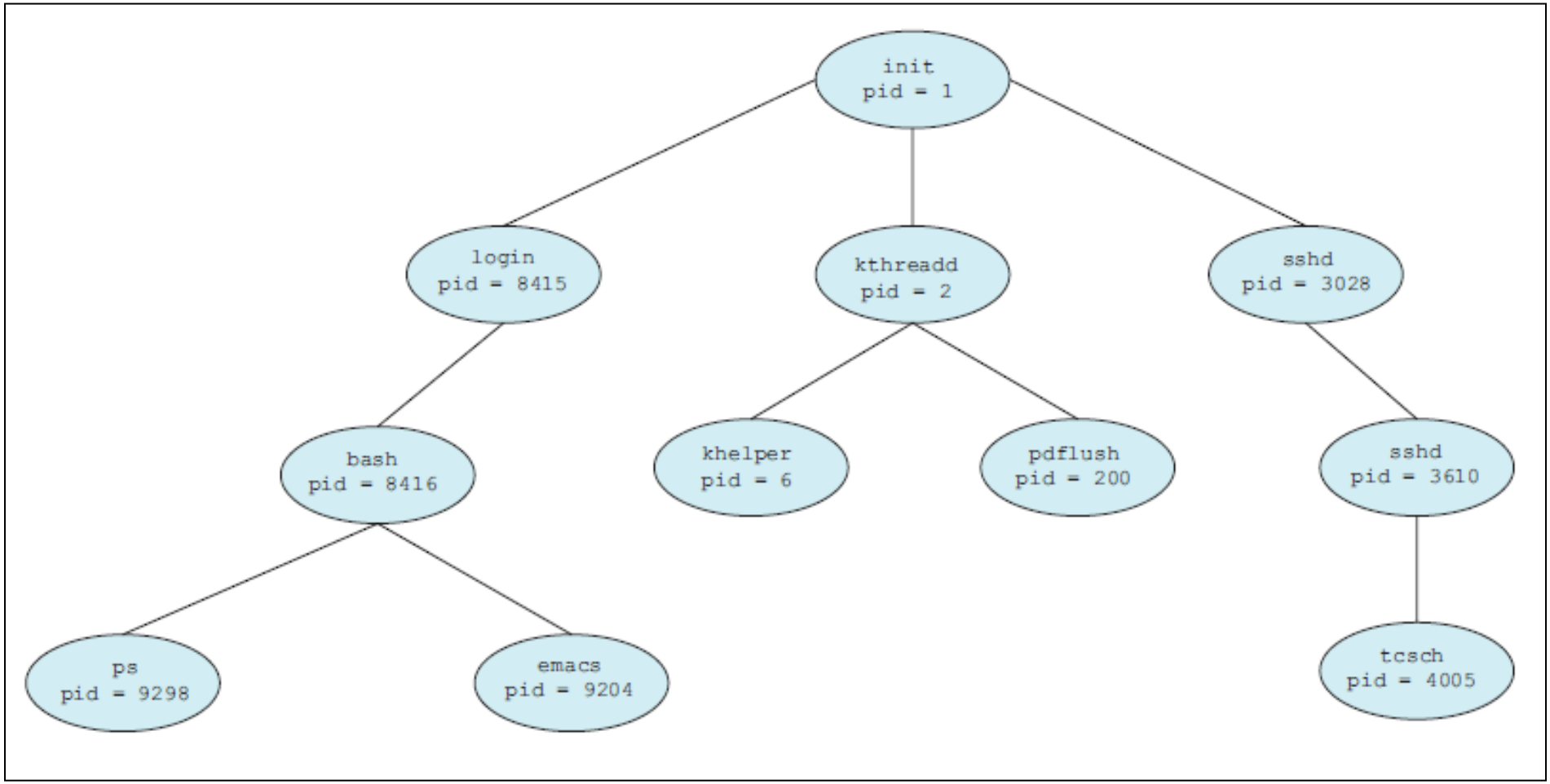
# Operations on Process

- Most systems run processes dynamically and for this to be implemented, multiple processes needs to be created and destroyed.
- Broadly operations on process include:
  - Process Creation
  - Process Destruction

# Process Creation

- During the course of execution a process may create multiple new processes.
- Creating process is parent process and new process created is child process.
- Operating Systems (UNIX, Linux and Windows) identify process using a unique identifier called PID.
- PID is a integer and is used as index to various attributes of a process.
-

# A tree of Processes in a Linux System



# Process Creation

- When a new process is created there may exist some scenarios:
  - The child program may request the required resources directly from the OS.
  - Parent may partition its resources among its children.
- Partitioning the resources avoids over burdening the OS
- Also two possibilities of execution exist:
  - The parent continues to execute concurrently with its children.
  - The parent waits until some or all of its children have terminated.
- Two address-space possibilities for the new process:
  - The child process is a duplicate of the parent process (it has the same program and data as the parent).
  - The child process has a new program loaded into it.

# Process Creation

- A new process is created by a fork() system call (UNIX).
- New process consists of a copy of the address space of the original process, which allows communication between parent and child.
- Both parent and child execute after fork()
  - Child process has a return identifier 0.
  - Parent process has a non-negative return identifier
- Child process is an identical copy of parent process.
- The new processes can start execution using exec() system call.
- In PCB every process has a reference to its parent process, hence sharing of resources is possible.
- If a parent process has noting to do and is waiting for child to finish then it can enter to waiting using wait() sys call.

# fork() Demonstration

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
        pid t pid;
        /* fork a child process */
        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                return 1;
}
        else if (pid == 0) { /* child process */
        execlp("/bin/ls","ls");
    }
    else { /* parent process */
    /* parent will wait for the child to complete */
    wait();
    printf("Child Complete");
    }
    return 0;
    }
```

# Process Termination

- Process terminates when it finishes executing.
- Terminates using exit() system call.
- A terminating process informs its parent by returning a integer.
- All the resources i.e. physical and virtual memory, open files, and I/O buffers are deallocated by OS.
- A parent can also abruptly terminate a process based on following reasons:
  - Child has exceeded its usage of some of the resources.
  - The task assigned to the child is no longer required.
  - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates

**Class Activity: For the following scenario draw a queuing diagram and demonstrate creation of 5 different processes after bootstrap (Assume it process P0).**

| Process | Time | Event/Resources | Running Time |
|---|---|---|---|
| P1 | 6 | I/O Request | 3 |
| P2 | 3 | System Call | 2 |
| P3 | 7 | CPU Bound | 7 |
| P4 | 1 | I/O Request | 1 |

**Home Work: Demonstrate process creation in a windows environment.**

| Process | Req. Running Time | Initial Queue | Second Queue | Actual Running Time | Enters to | Remaining running time |
|---|---|---|---|---|---|---|
| P1 | 6 | Job Queue | Ready Queue | 3 | I/O Queue | 3 |
| P2 | 3 | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |