Process Concept

Moumita Patra
July-Nov 2019
Ref: Galvin- Gagne

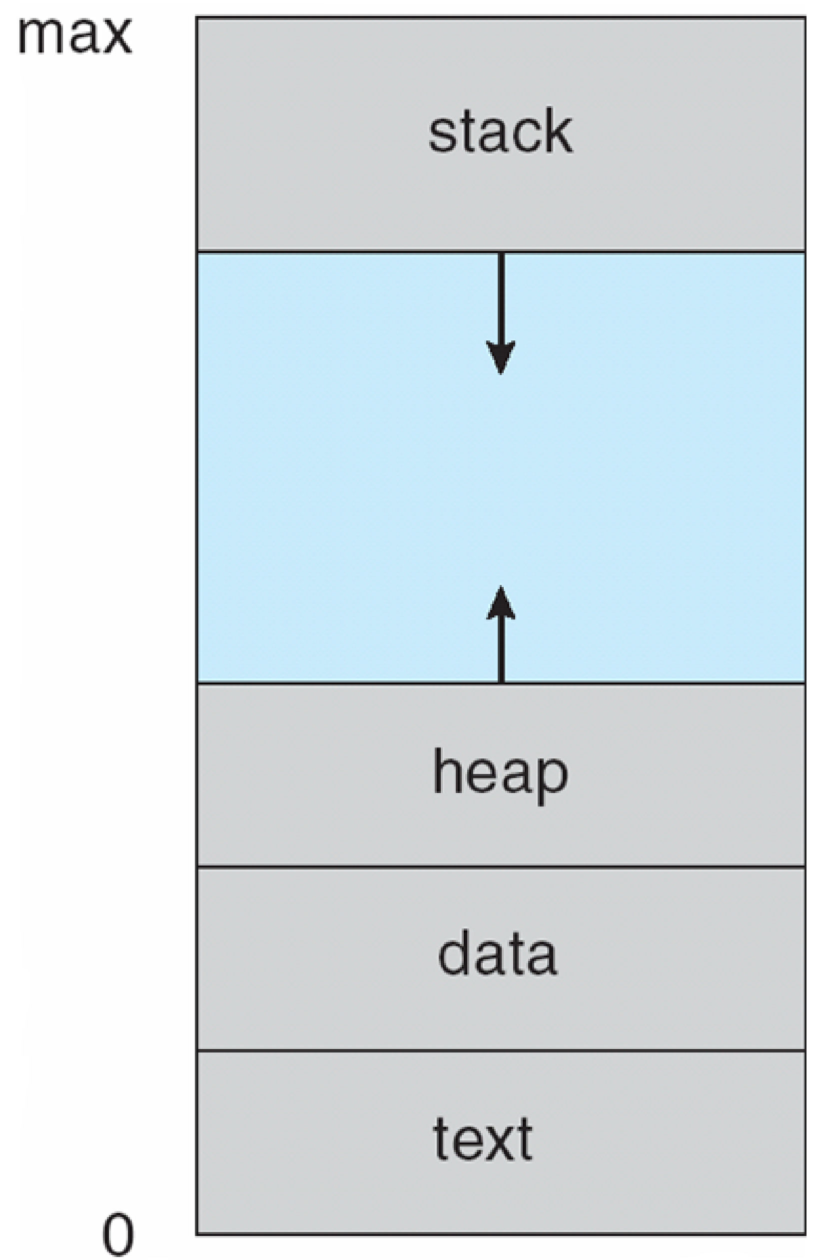
What is a process?

- **Process**- it is an instance of a program in execution.
- Multiple instances of the same program are different processes
- Process execution must progress in sequential fashion
- Process has resources allocated to it by the OS during execution
  - CPU time
  - Memory space for code, data, stack
  - Open files
  - Signals, etc.

- Each process is identified by a unique, positive integer id called process id.
- Program- is a passive entity stored on disk (executable file)
- Process is an active entity
- Program becomes process when executable file is loaded into memory
- One program can be several processes

Process- more than program code !

- Has multiple parts
  - Text section- the program code
  - Stack- contains temporary data (function parameters, return addresses, local variables, etc)
  - Data section- contains global variables
  - Heap- dynamically allocated memory during run time

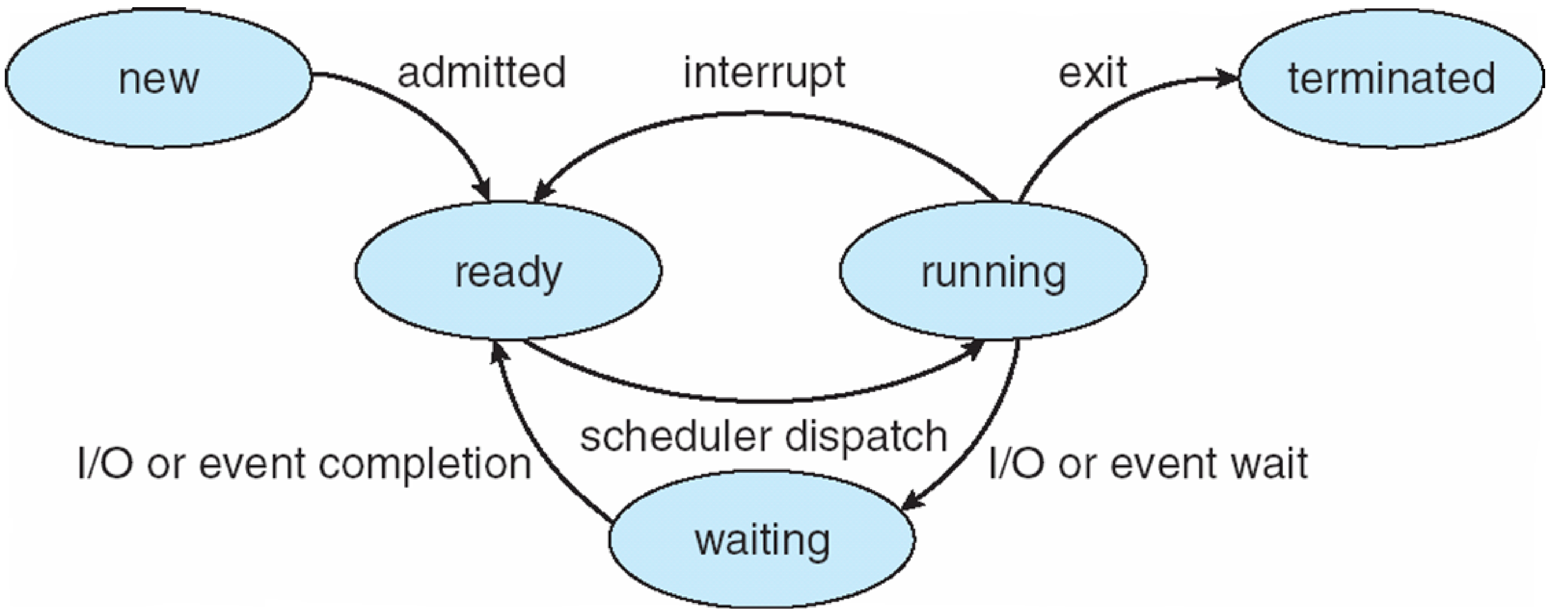


Process States

As a process executes, it changes **state.**

- **New-** The process is being created
- **Running-** Instructions are being executed
- **Waiting-** The process is waiting for some event to occur
- **Ready-** The process is waiting to be assigned to a processor
- **Terminated-** The process has finished execution

Process States

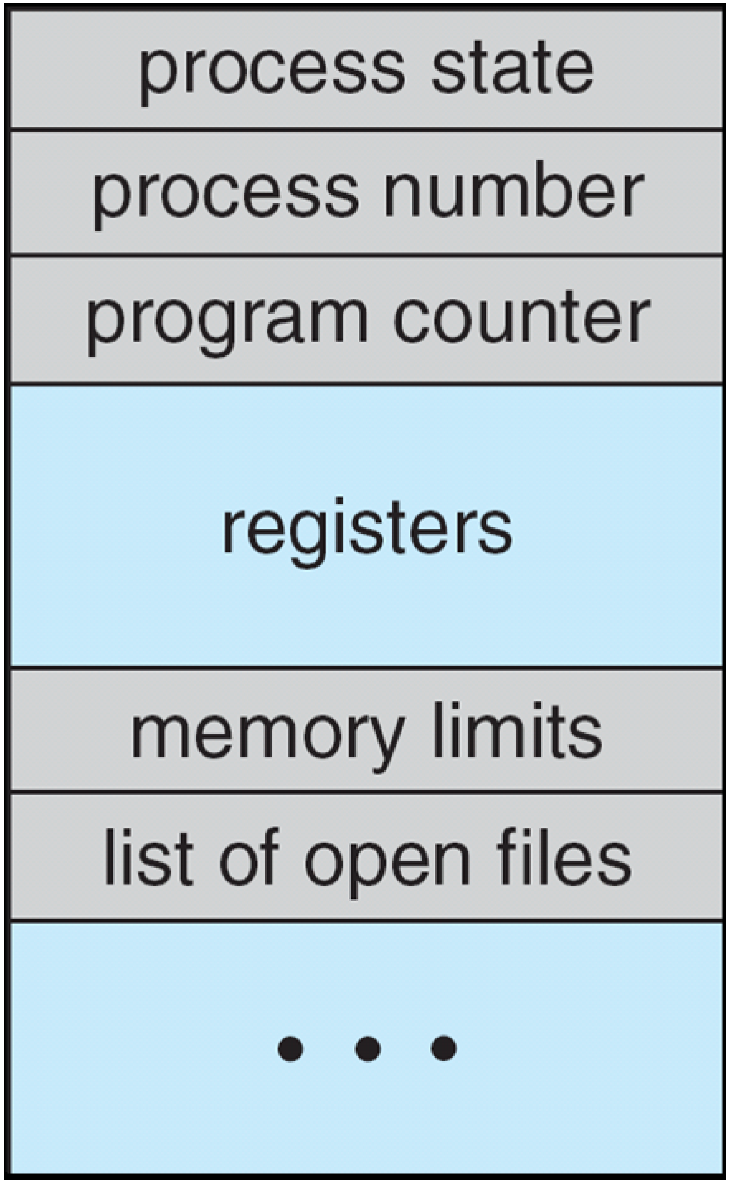


Process Control Block (PCB)

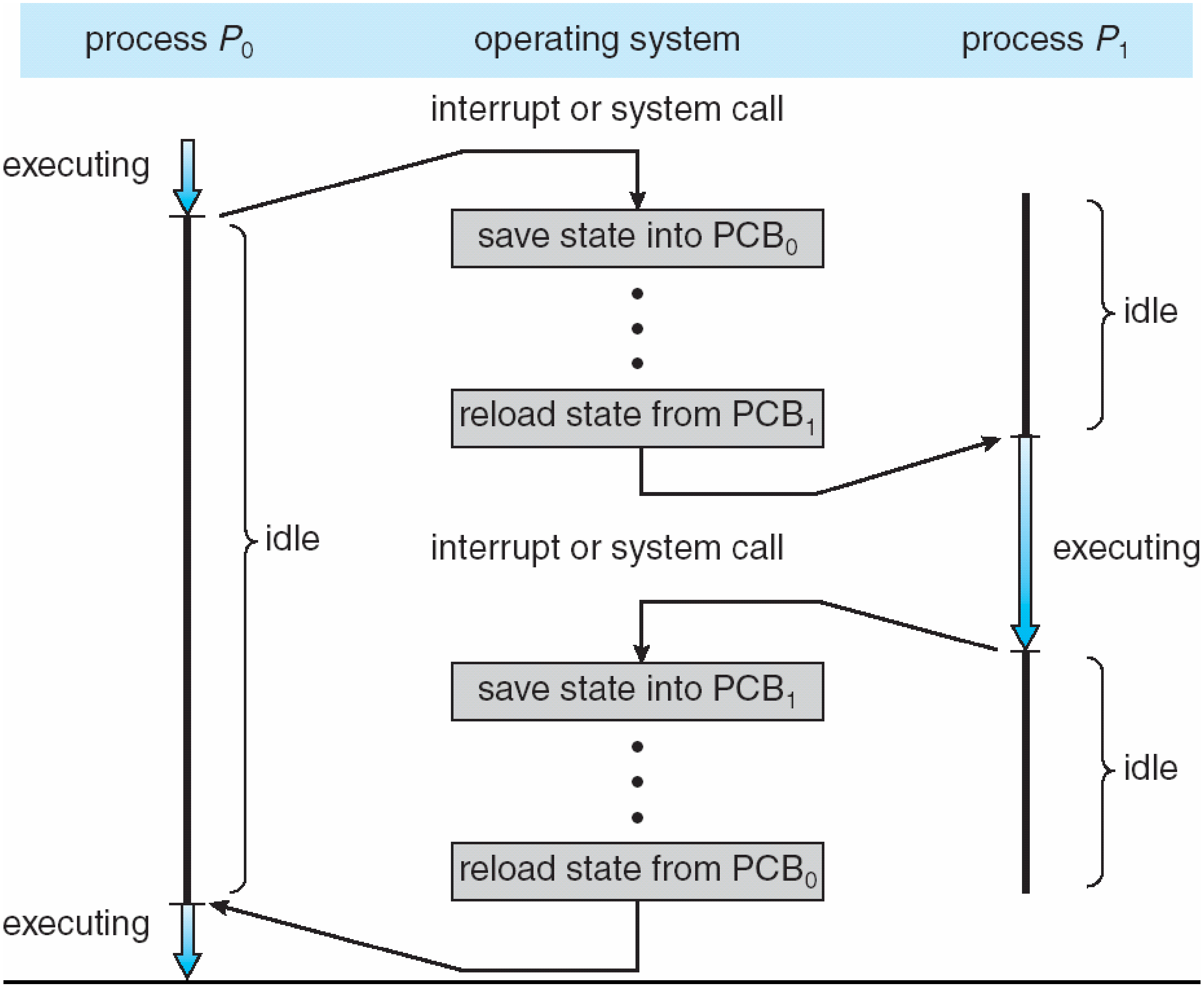
- Primary data structure maintained by the OS that contains information about a process
- One PCB per process
- OS maintains a list of PCBs for all processes
- **Interrupt**- A signal to the processor emitted by the hardware or software indicating an event that needs immediate attention.
- Processor responds to an interrupt by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an Interrupt Service Routine, ISR).

Typical contents of PCB

- Process state- running, waiting, etc
- Program counter- location of instruction to execute next
- CPU registers- contents of all process-centric registers
- CPU scheduling information
- Memory management information
- Accounting information- CPU used, time elapsed, etc
- I/O status information- I/O devices allocated to process, list of open files



CPU Switch from Process to Process



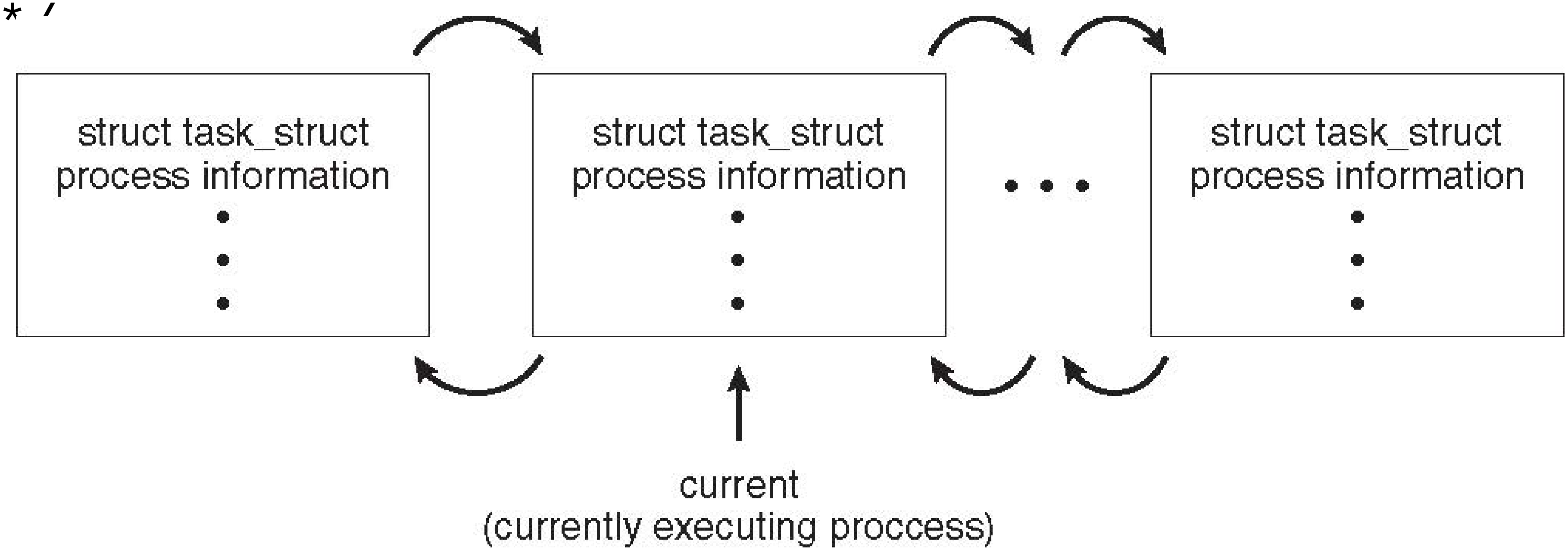
Threads

- A basic unit of CPU utilization.
- It is the smallest sequence of programmed instructions that can be executed independently.
- Traditionally, processes have a single thread of control.
- Multiple threads to perform more than one task at a time-- multicore systems.
- Therefore, need to have storage for thread details

Process Representation in Linux

Represented by the C structure `task_struct`

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children*/
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process
*/
```



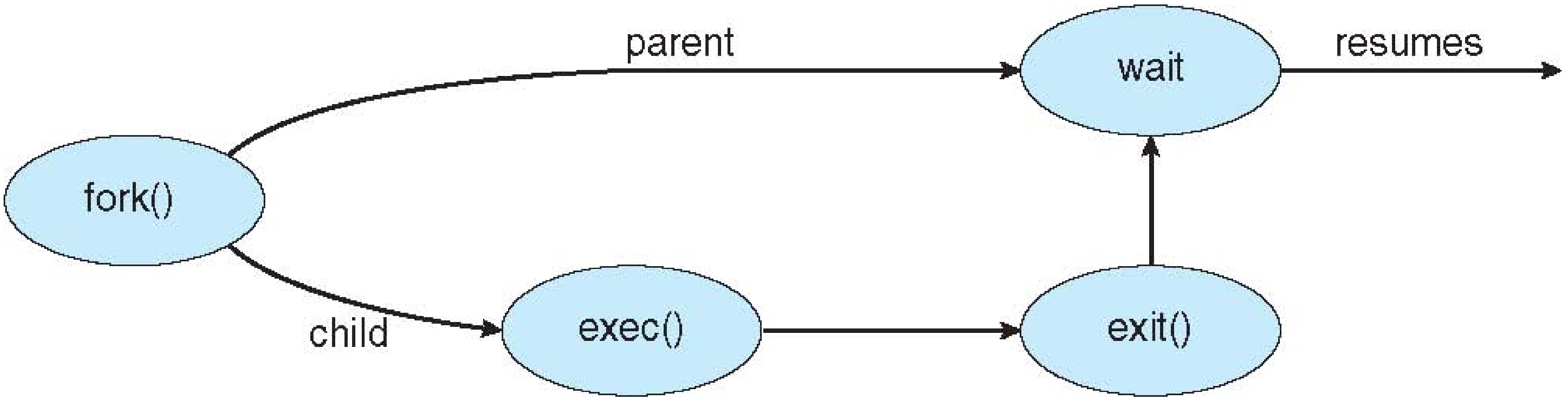
Main Operations on a Process

- **Process creation**
  - Data structures like PCB is set up and initialized
  - Initial resources allocated and initialized if needed
  - Process added to ready queue
- **Process scheduling**
  - CPU is allotted to the process, process runs
- **Process termination**
  - Process is removed
  - Resources are reclaimed

Process Creation

- A process (***parent***) can create another process (***child***), which in turn can create other processes, forming a ***tree*** of processes
  - By making a system call, ***fork()***
  - **Parent process**- process that invokes the call
  - **Child process**- the new process created
- Processes are identified and managed via ***process identifier (pid)***
- ***exec()*** system call used after a ***fork()*** to replace the process' memory space with a new program

Process Creation

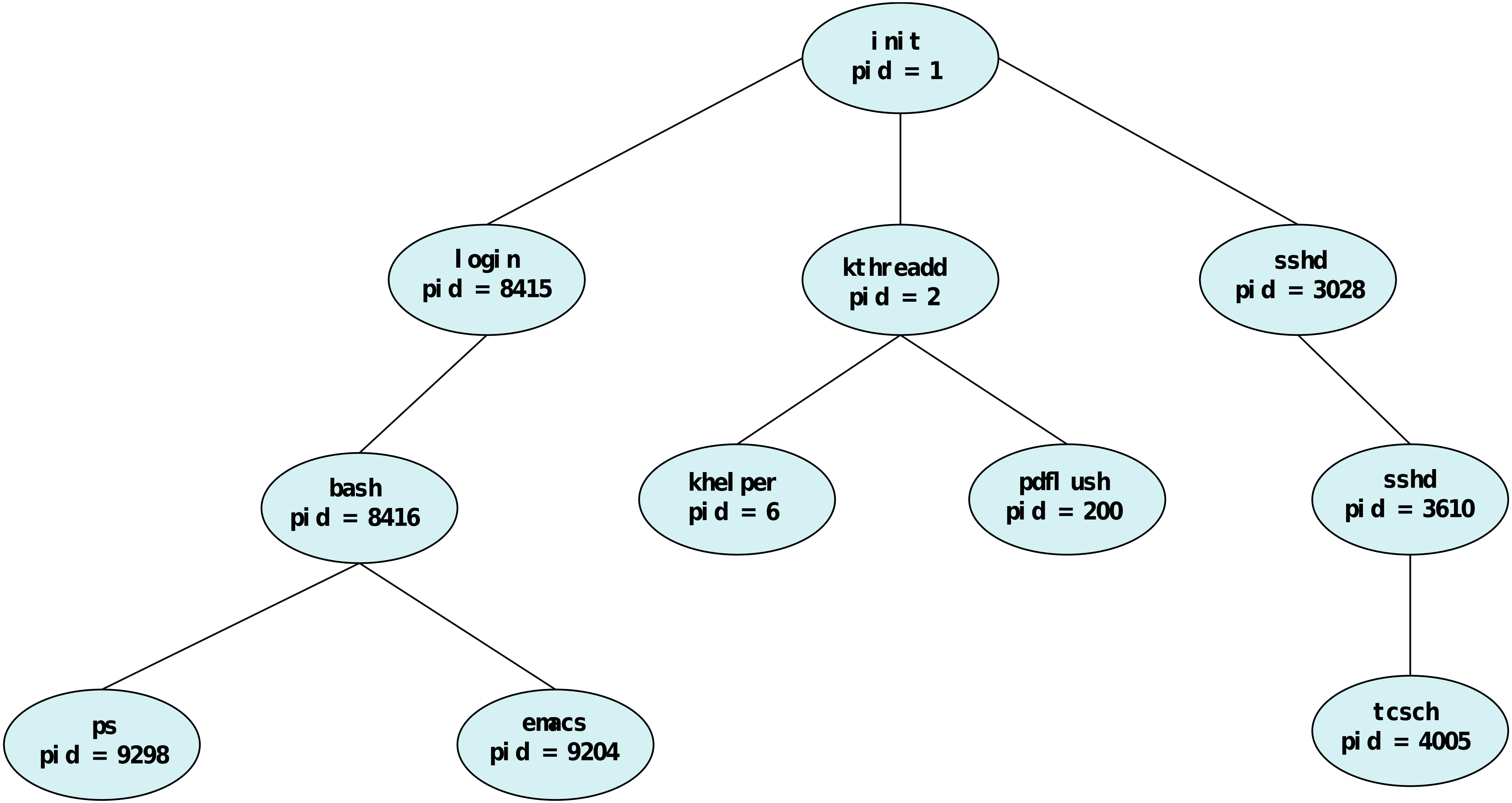


C Program Forking Separate Process

```
#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",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

- **Resource sharing possibilities**
  - ➢ Parent and children share all resources
  - ➢ Children share subset of parent’s resources
  - ➢ Parent and child share no resources
- Restricting a child process to a subset of the parent’s resources prevents any process from overloading the system by creating too many child processes.
- **Execution possibilities**
  - ➢ Parent and children execute concurrently
  - ➢ Parent waits until children terminate
- **Memory address space possibilities**
  - ➢ Address space of child duplicate of parent
  - ➢ Child has a new program loaded into it

A Tree of Processes in Linux



Process Termination

- Process executes last statement and asks the OS to terminate it (e.g- ***exit()*** system call)
- Returns status data from child to parent (via ***wait()***)
- Process's resources are deallocated by OS
- Parent may terminate execution of a child using ***abort()*** due to-
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and OS does not allow a child to continue if its parent terminates.

Process Termination

- Some operating systems do not allow child to exist if its parent has terminated.
- If parent terminates then children must also terminate
- Cascading termination- all children, grandchildren, etc are terminated. Termination is initiated by OS.
- Parent process may wait for termination of a child process by using wait(). It returns status information and the pid of the terminated process.
- Zombie process- A process that has terminated but whose parent has not yet called wait()
- Orphan process- Parent terminated without wait()
- Linux and UNIX address this scenario by assigning init() process as the new parent to the orphan process

Process Scheduling

- Objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler/ dispatcher** selects among available processes (from ready queue according to some algorithm) for next execution on CPU
- Selected process runs till

➢ It needs to wait for some event to occur (e.g- disk read)

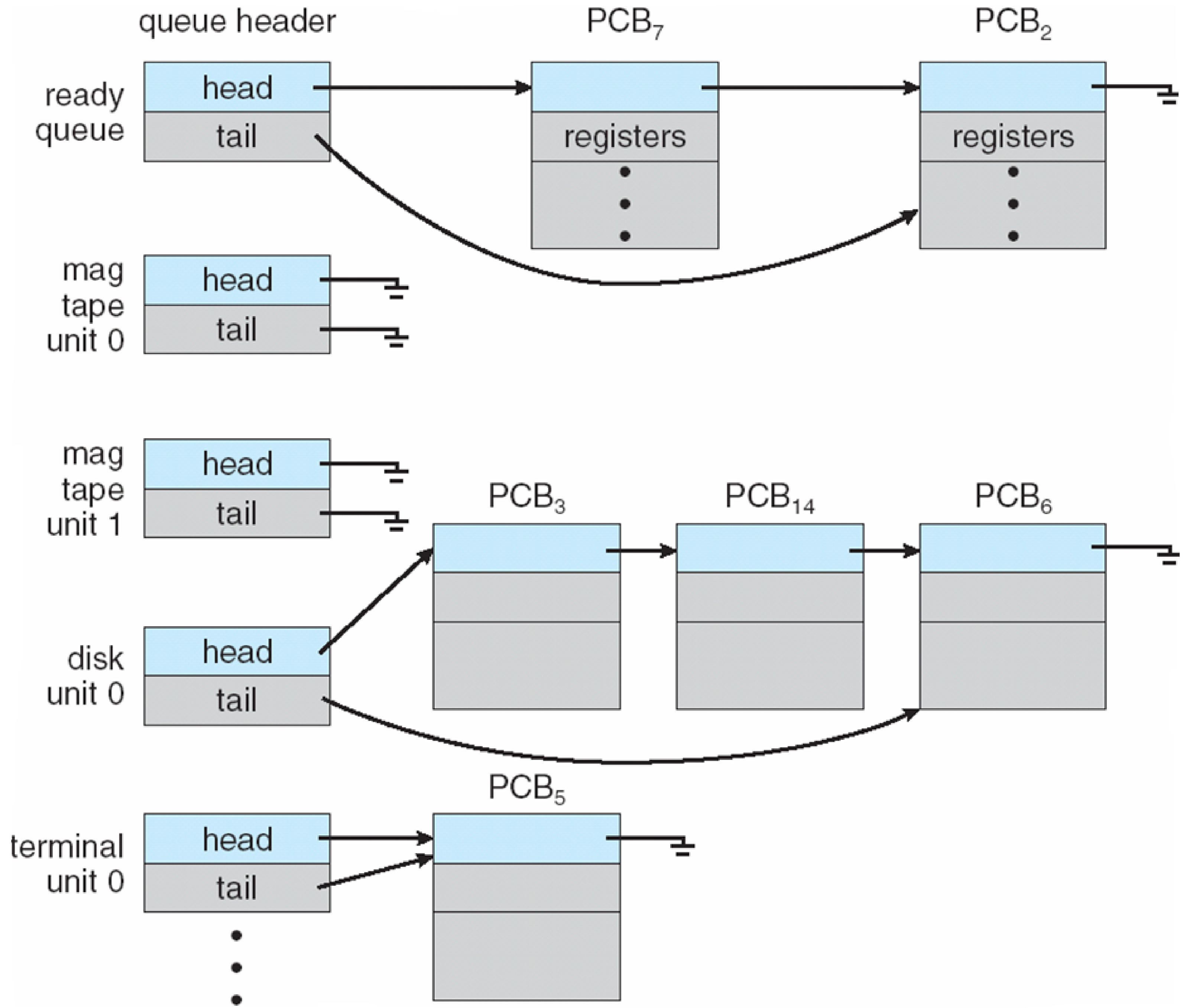
➢ CPU time allotted expires

➢ Arrival of a higher priority process

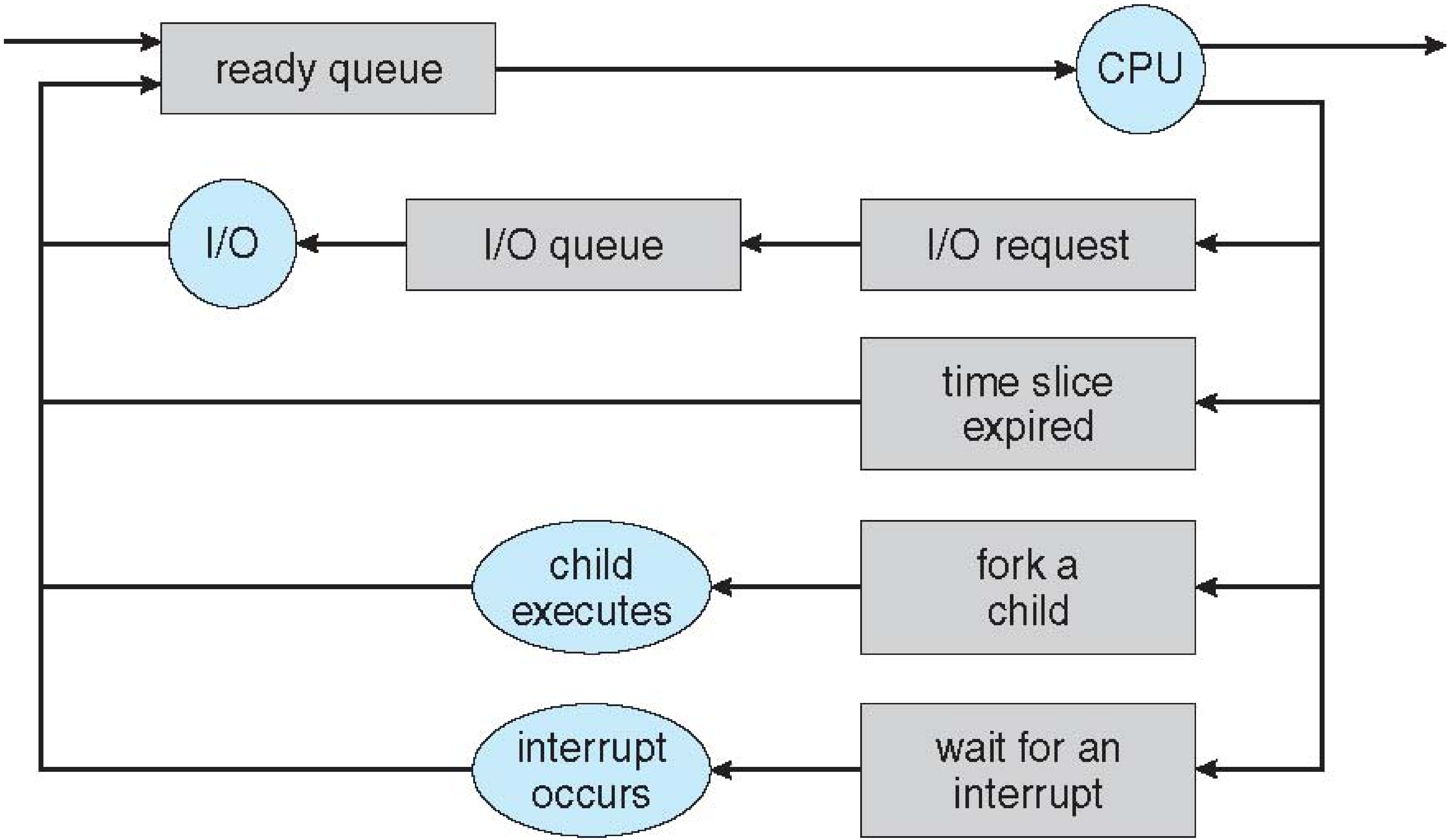
- When process is ready to run again, it goes back to the ready queue
- Scheduler is invoked again to select the next process from the ready queue

- 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



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 must select a new process for the CPU frequently.
- Long term scheduler (or job scheduler)- selects which processes should be brought into the ready queue
  - Invoked less frequently.
  - Controls the degree of multiprogramming (the number of processes in memory)

Schedulers

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
- Job of long term scheduler to make a careful selection.
- It is important that the long term scheduler selects a good mix of CPU and I/O bound processes.
- What if all processes are I/O bound?
- What if all processes are CPU bound?

- Note: In some systems, long time scheduler may be absent or minimal, e.g- UNIX and Windows. Every process is put into memory.
- Stability of such systems depends on physical limitations or on user's demand of work.
- What if all processes do not fit in memory?

➢ Partially executed jobs in secondary memory (swapped out)

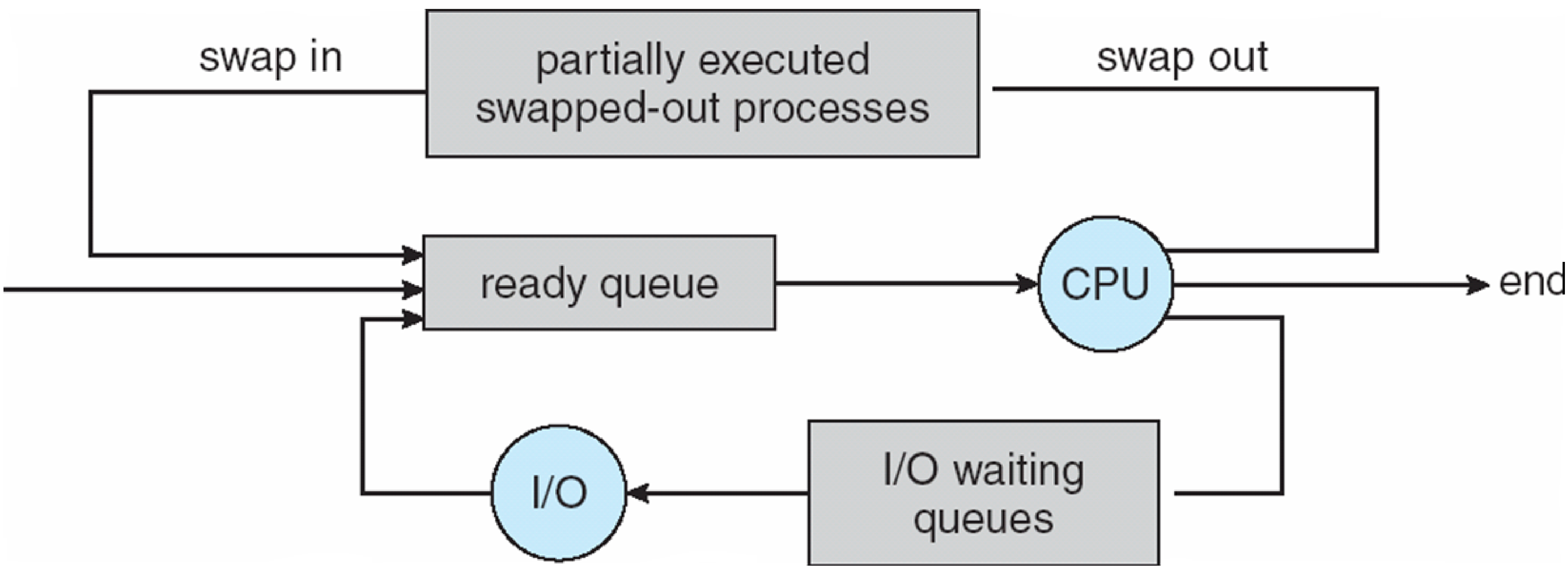
➢ Copy the process image to some pre-designated area in the disk (swap out)

➢ Bring in again later and add to ready queue later

Medium Term Scheduler

Some time-sharing systems may introduce additional, inetrmediate level of scheduling

- Helps in removing process from memory, store on disk, bring back in from disk to continue execution : swapping



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- Information that is required to be saved to be able to restart the process later from the same point
- Context of a process is represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- More complex the OS and the PCB-> longer the context switch
- Switching time dependent on hardware support- memory speed, number of registers that must be copied, special instructions, etc.

Interprocess Communication(IPC)

- Processes within a system may be Independent or cooperating
- **Independent process**- that cannot affect or be affected by other processes executing in the system
- **Cooperating process**- It can affect or be affected by other processes executing in the system
- Reasons for process cooperation-
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication

Models of IPC

- **Shared memory**- A region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region
- **Message passing**- Communication takes place by means of messages exchanged between the cooperating processes.
- Which ones better? - Shared memory
- But shared memory suffers from cache coherency issues.

Communication Models

(a) Message passing (b) Shared memory