Chapter 3: Processes

NARZU TARANNUM(NTR)
DEPT. OF CSE, BRAC UNIVERSITY

Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operation on Processes
- Interprocess Communication

Process Concept

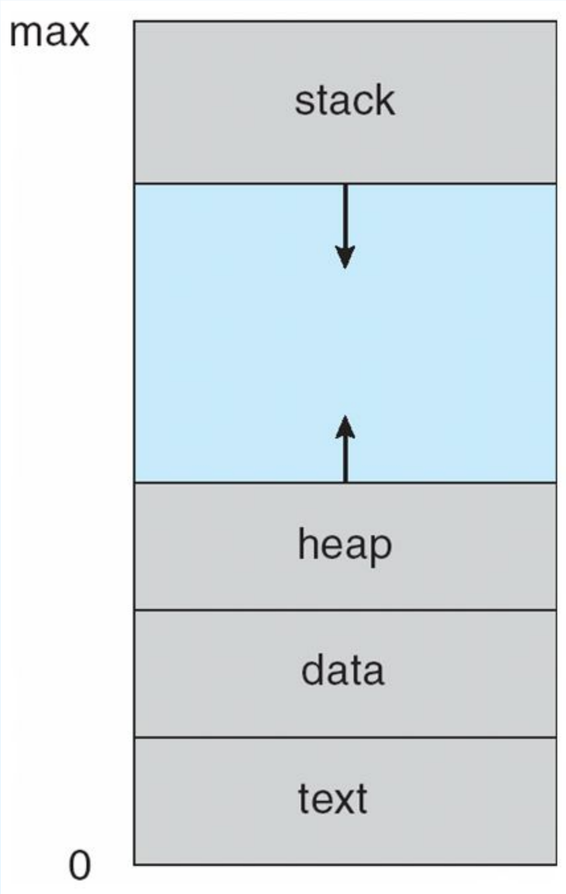
- An operating system executes a variety of programs
- Textbook uses the terms *job* and *process* almost interchangeably

- **Process – a program in execution**; process execution must progress in sequential fashion.
- A process includes:
  - program counter
  - stack
  - data section

Multiple parts of a process

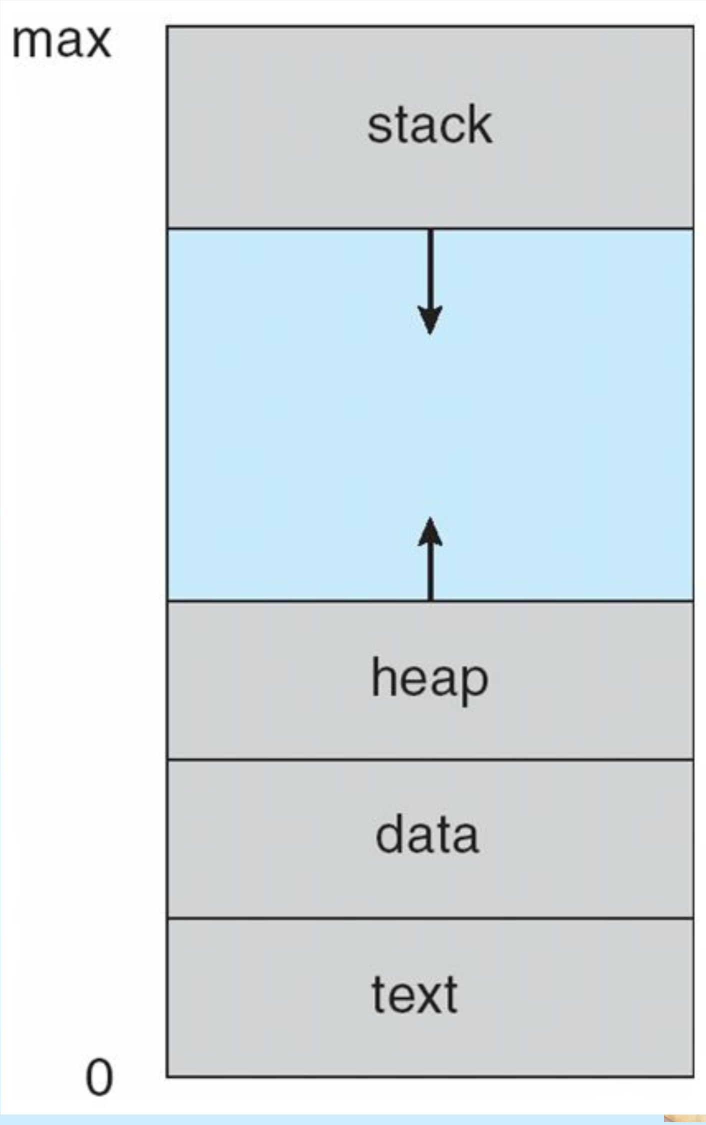
- The program code, also called **text section**
- Current activity including **program counter**, processor registers
- **Stack** containing temporary data, function parameters, return addresses, local variables, etc.
- **Data section** containing global variables
- **Heap** containing memory dynamically allocated during run time

- Process in Memory

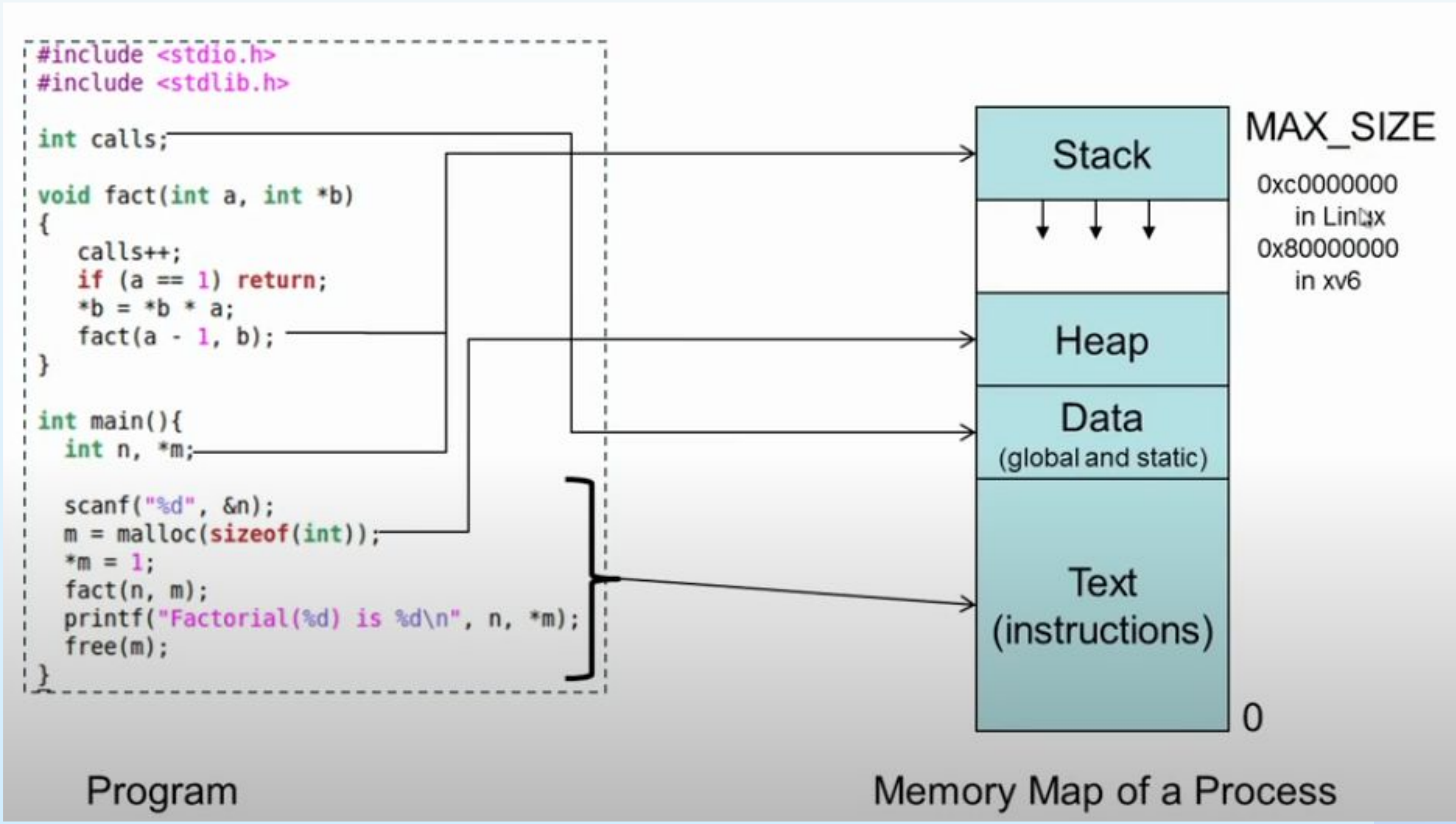


Process Memory Map

```
#include <stdio.h>
#include <stdio.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);
        print(“Factorial (%d) is %d\n”,n,*m);
        free (m);
}
```



Process Memory Map

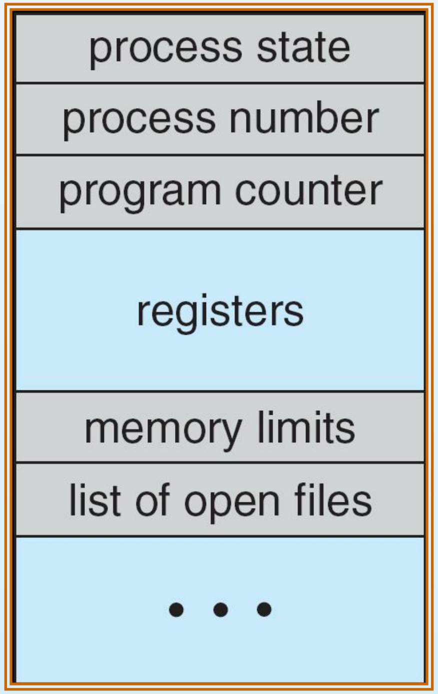


Process Management

- **A process is a program in execution**. It is a unit of work within the system. Program is a *passive entity* stored on disk (**executable file)**, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Program becomes process when executable file loaded into memory.
- One program can be several processes
  - Consider multiple users executing the same program
- Process termination requires reclaim of any reusable resources
- Processes maybe single-threaded or multi-threaded
  - Process executes instructions sequentially, one at a time, until completion
- Typically a system has many processes running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Control Block (PCB)

- Information associated with each process is represented in the Operating system is Process Control Block(PCB).
  - Process id/number
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information/ Priority
  - Memory-management information/ protection
  - List of open files
  - I/O status information



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.*
  - **new**: The process is being created
  - **ready**: The process is waiting to be assigned to a processor.
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **terminated**: The process has finished execution

Diagram of Process State

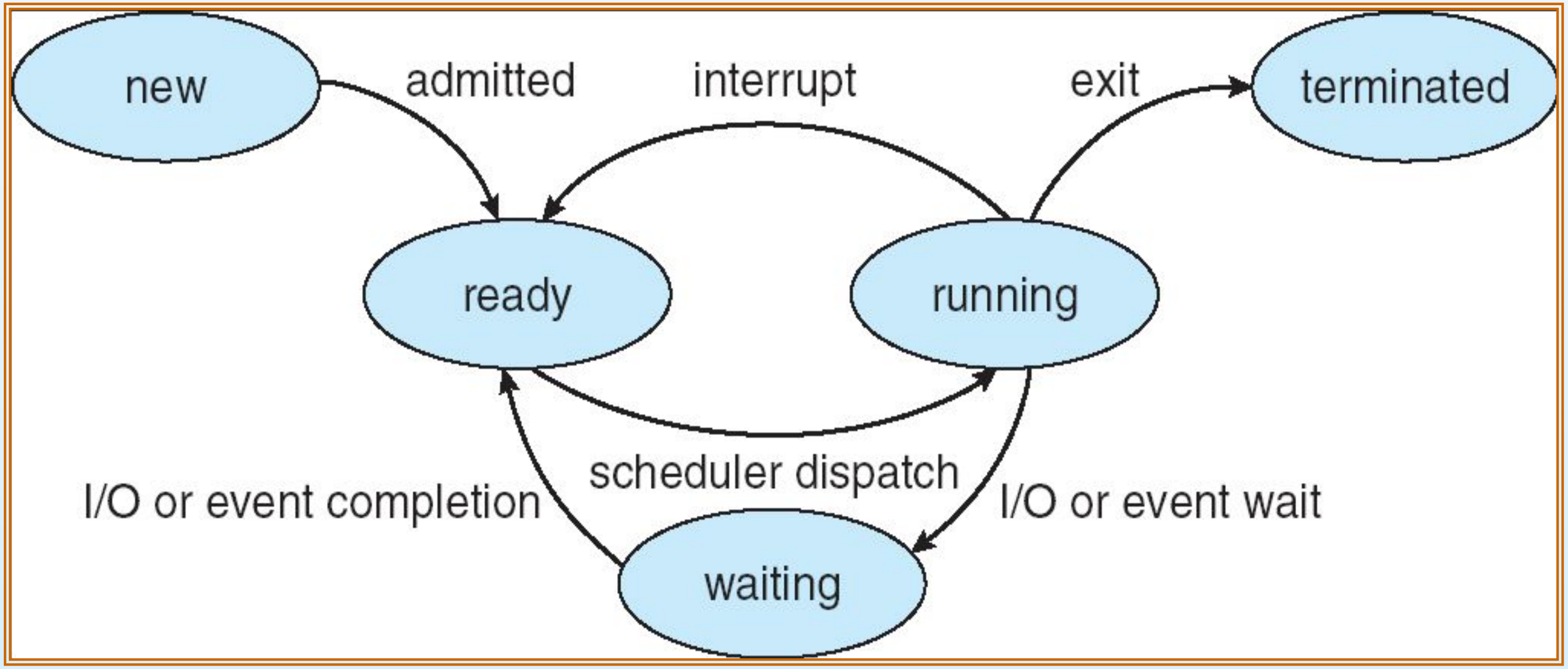
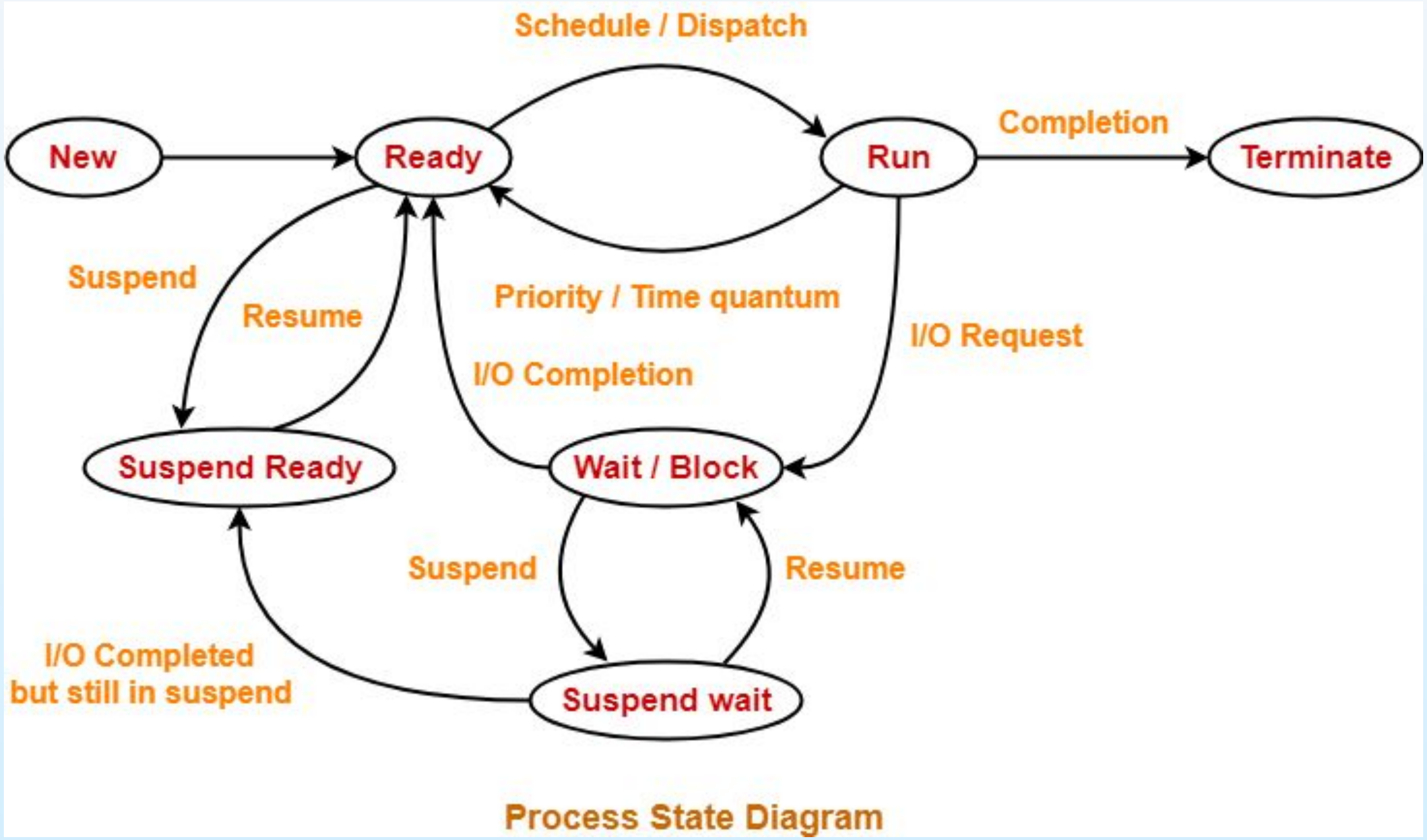


Diagram of Process State



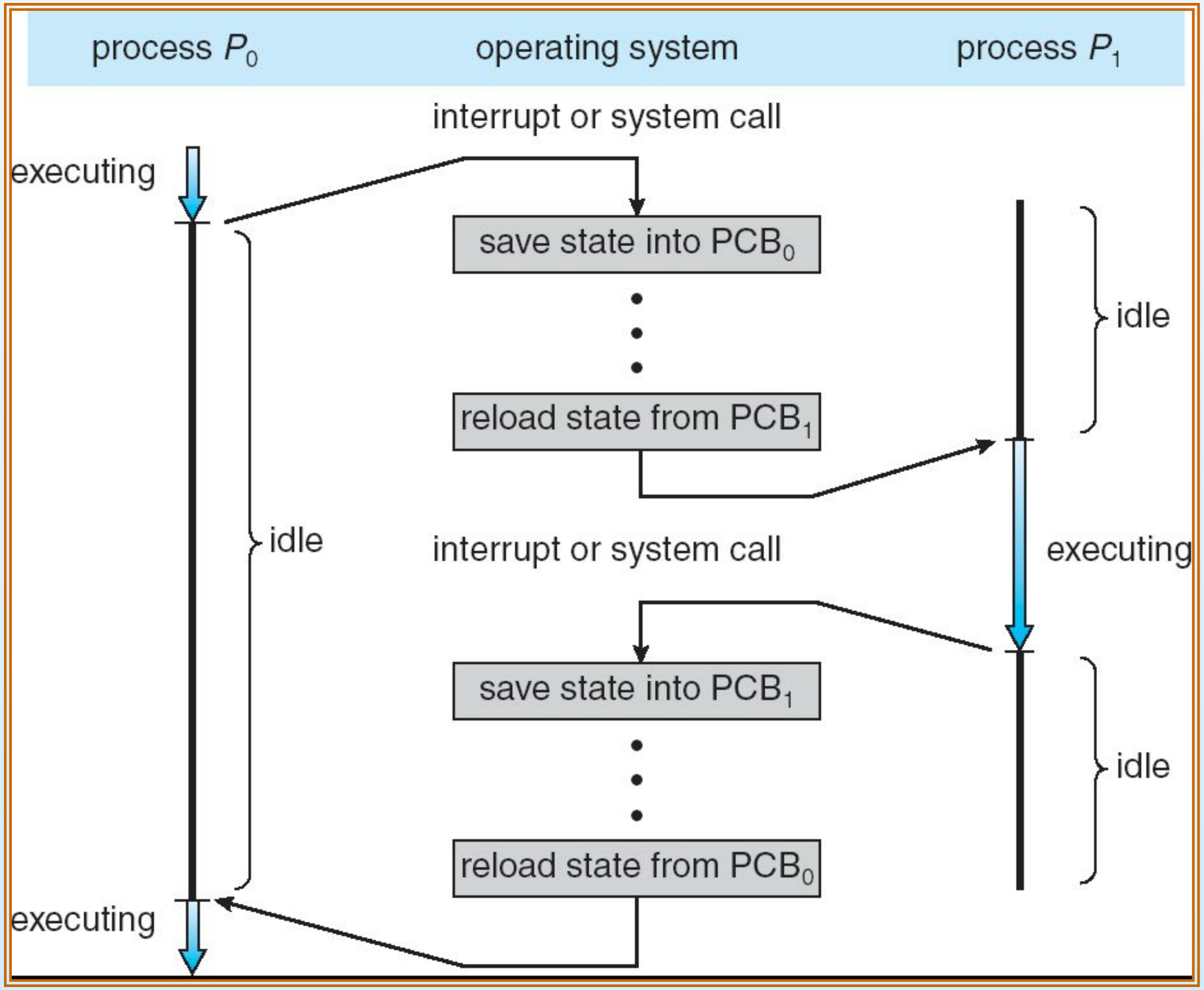
Process Scheduling Info

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations.
  - **CPU-bound process** – spends more time doing computations.

Context Switch

- Context switching operations happen frequently on general purpose system.
- 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 no useful work while switching
- **Context** of a process represented in the PCB
- The more complex the OS and the PCB □ the longer the context switch
- Time dependent on hardware support

CPU Switch From Process to Process

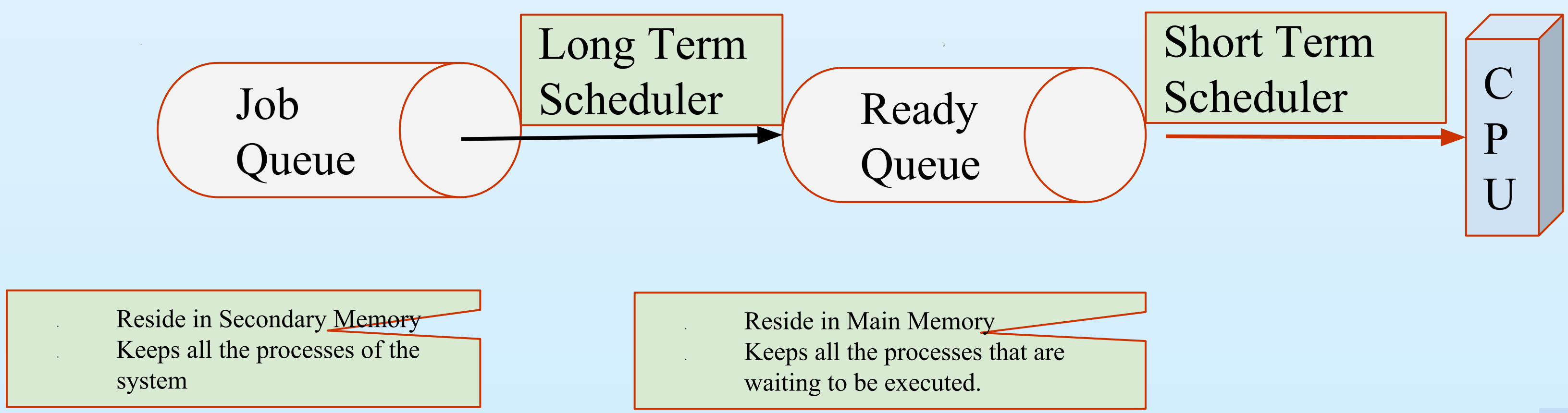


Process Scheduling

- The objective of Multi-programming 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 users can interact with each program while it is running.
- To meet these objectives, the process schedulers select an available process for program execution on CPU.

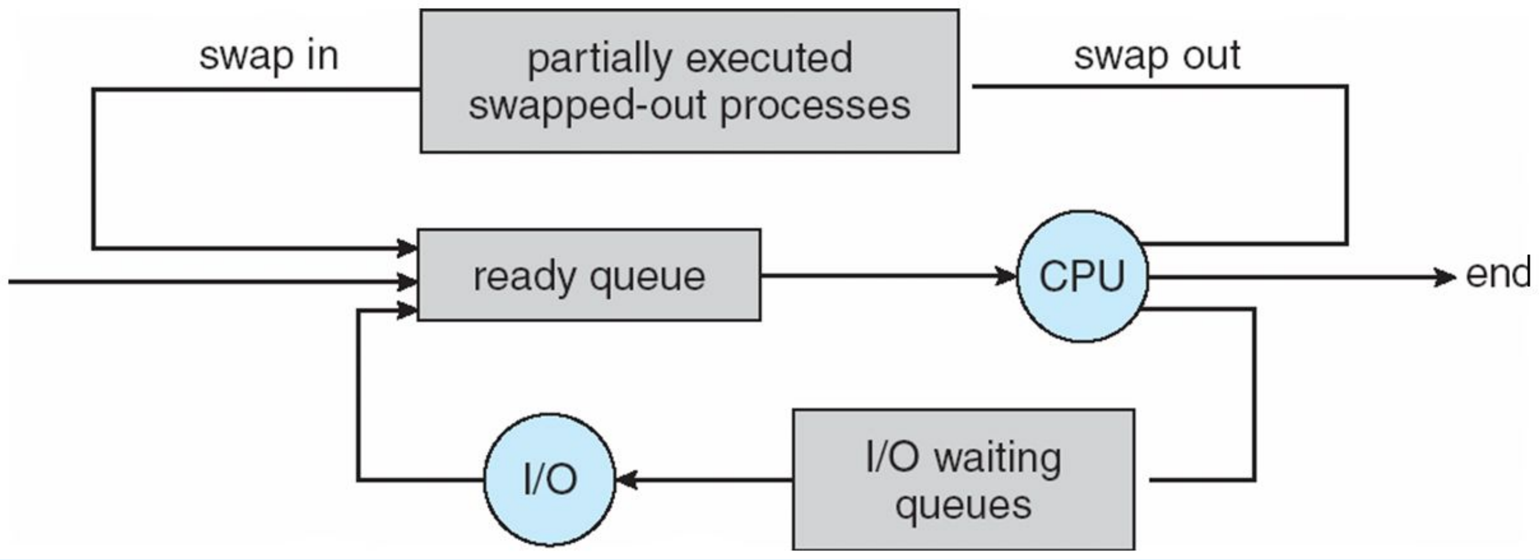
Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - The long-term scheduler controls the *degree of multiprogramming*
  - Long-term scheduler strives for good ***process mix***
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Short-term scheduler is invoked very frequently



Mid-term scheduler

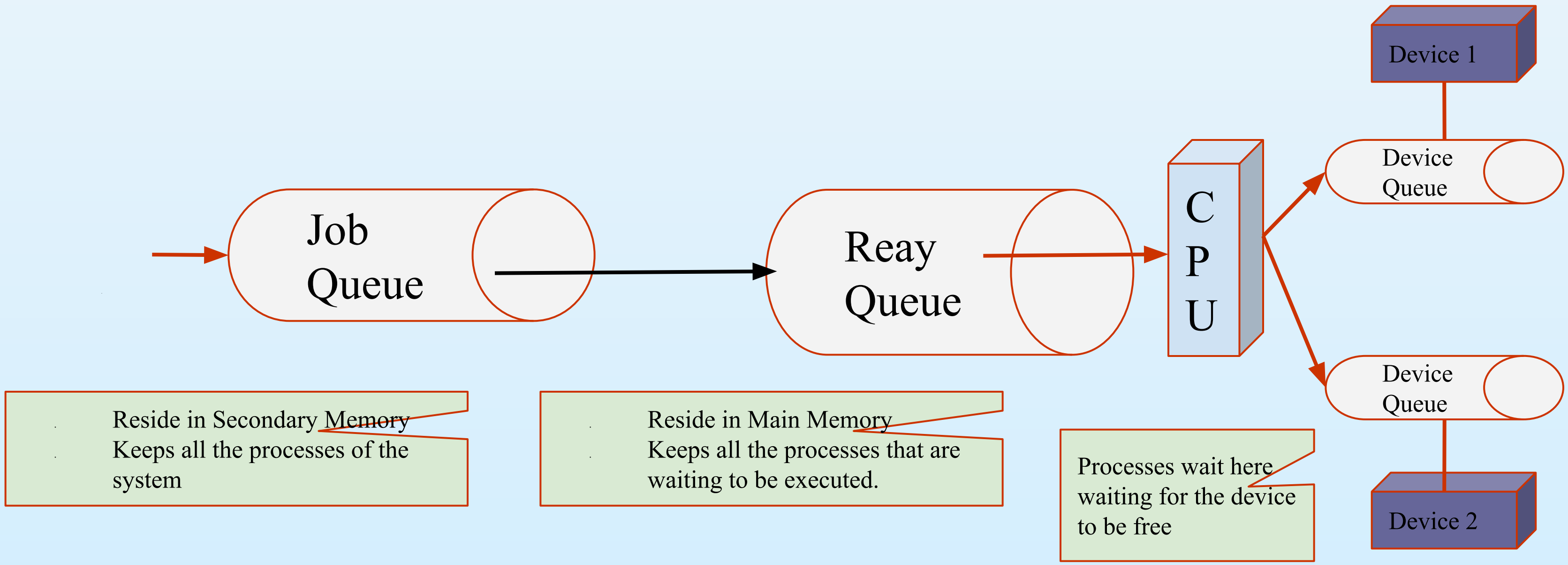
- sometime it can be advantageous to remove process from the memory and thus to reduce the degree of multi-programming.
- Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



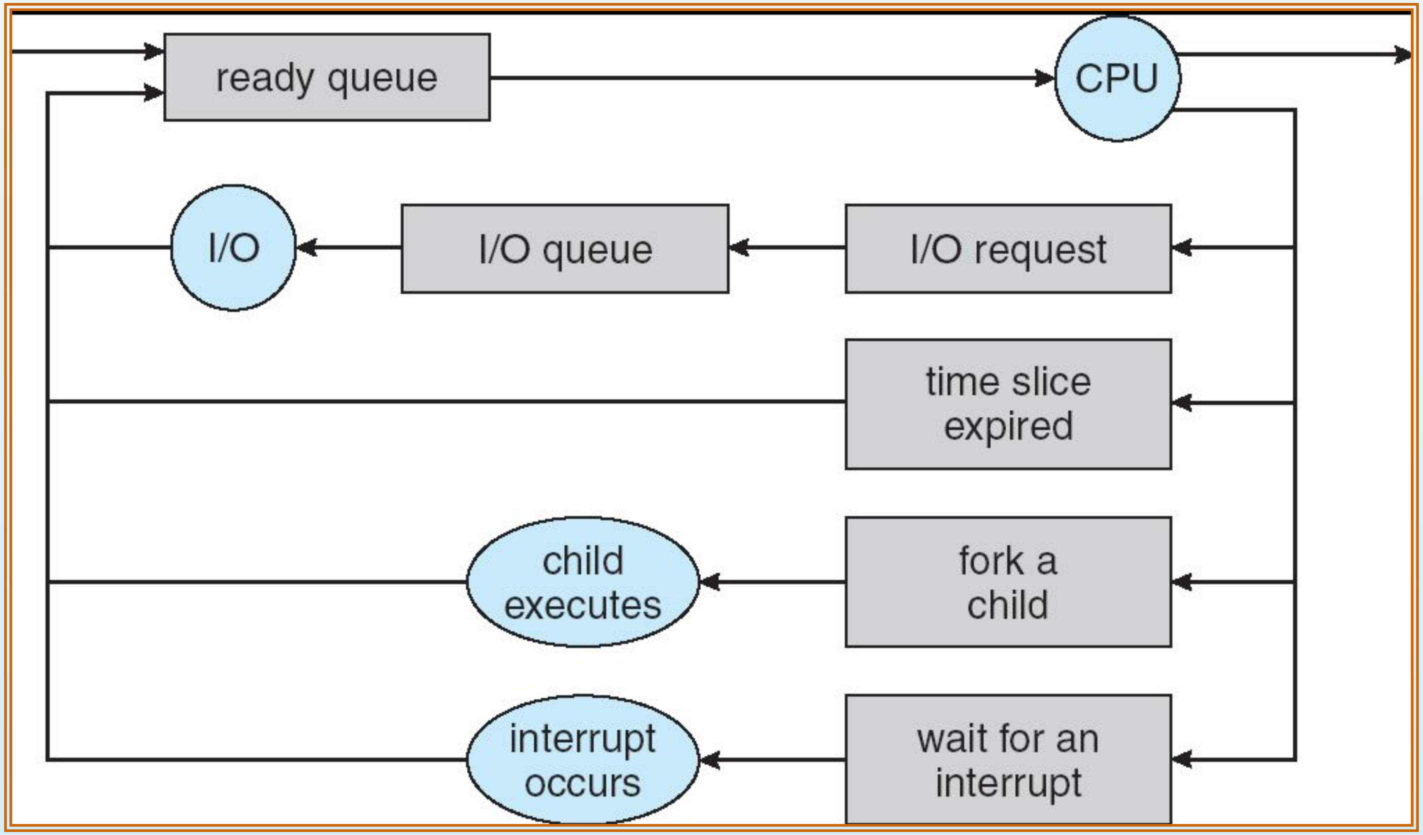
Process Scheduling Queues

- Process scheduler selects among available processes for next execution CPU.
- Different queues are maintained for different state of the process.
- Processes migrate among the various queues
- Maintains scheduling queues of process:
  - **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

Process Scheduling Queues



Process Scheduling Queue/Queueing diagram



Process scheduling as a queueing diagram

- A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution and is given the CPU. Once the process is allocated the CPU and is executing, one or several events could occur.
  - The process could issue an I/O request, and then be placed in an I/O queue.
  - The process could create a new sub process and waits for its termination.
  - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back to ready queue.
- A process continue this cycle until it terminates.

Operation on process

The processes in the system can execute concurrently, and must be created and deleted dynamically. OS must provide a mechanism for process creation and termination.

Process Creation

- A process may create several new processes via a **create()** process system call during it's execution.
- The creating process is called a parent process, and the new processes are called the children of that process
- Each of the new processes may create other processes, forming a tree of processes.
- Every process identified by a unique PID.

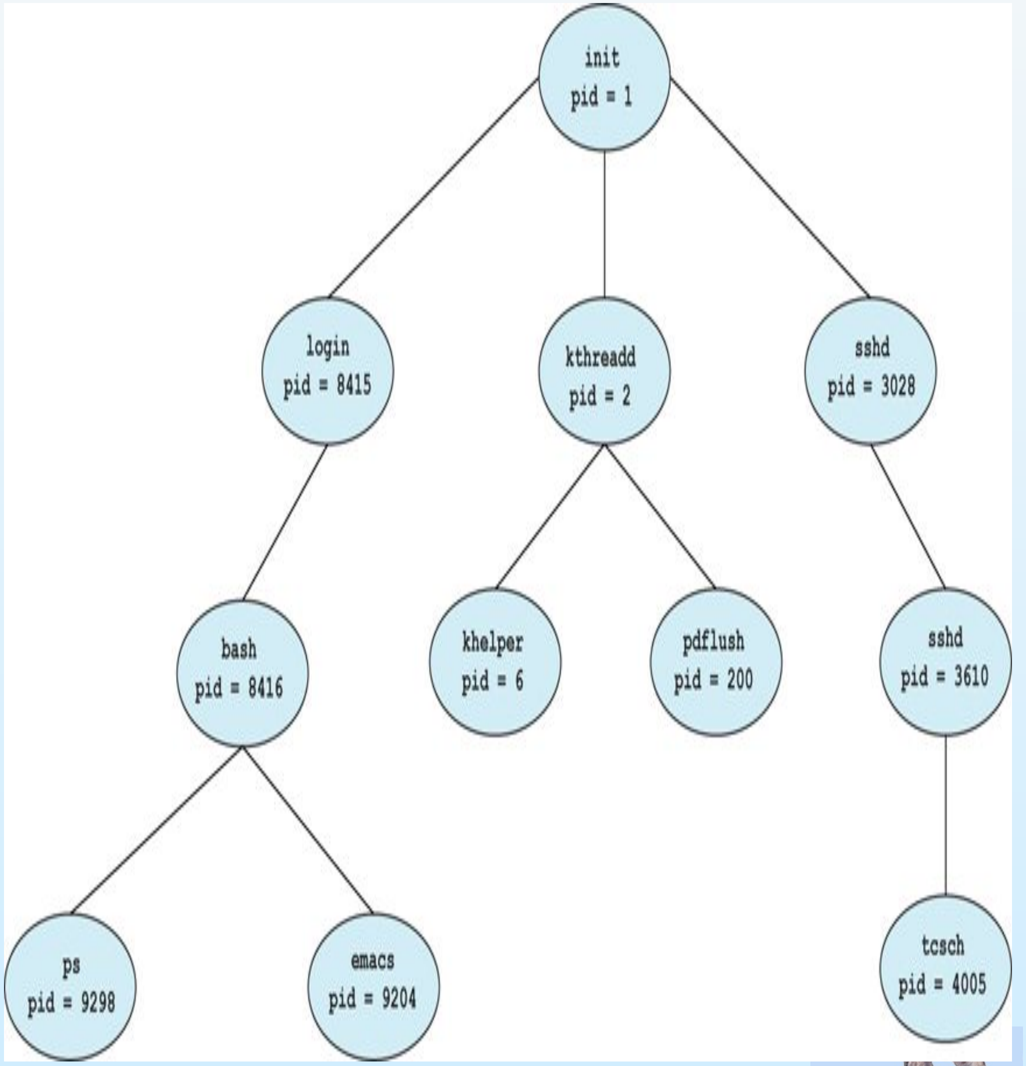
- **When a process creates a new process, two possibilities exist in term of execution:**
  - Parent and children execute concurrently
  - Parent waits until some or all of its children have terminated.

Process Creation

- **<u>Possibilities of the address space of the new process:</u>**
  - The child process is duplicate of the parent process.
  - The child process has a new program loaded into it.
- **<u>Possibilities of resource uses of new process:</u>**

  Child process obtain resources from OS or are restricted to Parent's resources.
  Parent process may pass initializing data to child process

Process creation for UNIX OS



Process Creation in UNIX

- System Call: offers the services of the operating system to the user programs.
- ***fork()***: create a new process, which becomes the child process of the caller
- ***exec()***: runs an executable file , replacing the previous executable
- ***wait()***: suspends execution of the current process until one of its children terminates.
- **exit():** is used to terminate program execution.

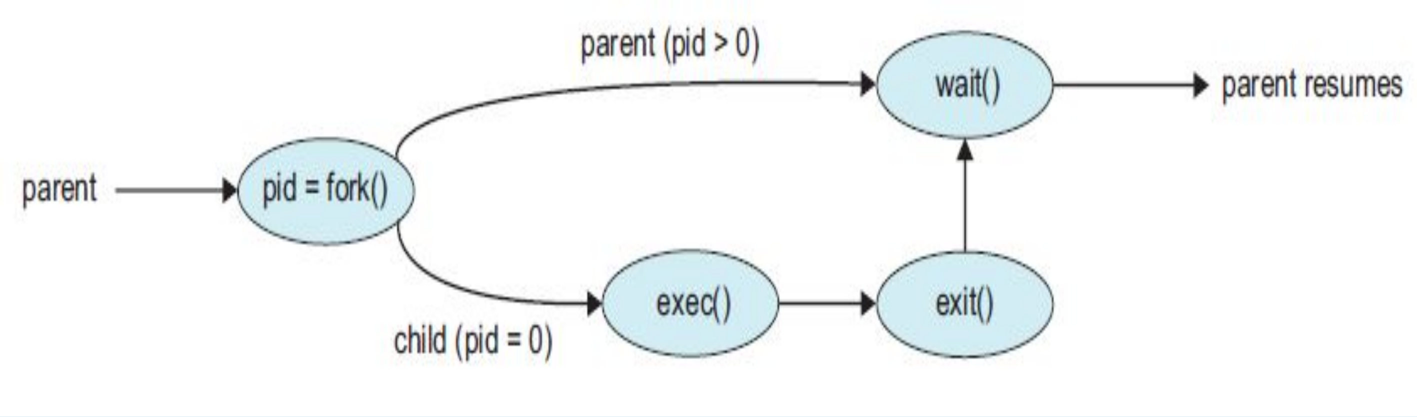


Fig: Process creation using fork() system call

Process Creation in UNIX

- In Unix like operating system, fork system call is used for creating a new process where a process is a copy of itself, which is called the child process.
- The fork system call is usually a system call, implemented in the kernel.
- After a new child process is created, both processes will execute the next instruction following the fork() system call.
- If n times fork then total number of process created is 2^n in which number of child process created is 2^n -1

```
int main(){
   fork();
   fork();
   printf("A");
}
```

```
int main(){
   fork();
   fork();
   fork();
   printf("A");
}
```

Process Creation in UNIX (Example)

```
int main(){
    a = fork();
    if(a==0) fork();
    fork();
    printf(“A”);
}
```

```
int main(){
    int x = 1;
    a = fork();
    if(a==0){
        x = x -1;
        printf(“value of x is: %d”, x);
    }
    else if (a>0){
        wait(NULL);
        x = x +1;
        printf(“value of x is: %d”, x);
    }
}
```

Process Creation in UNIX (Example)

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       pid1 = getpid();
       printf("child: pid = %d",pid); /* A */
       printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
       pid1 = getpid();
       printf("parent: pid = %d",pid); /* C */
       printf("parent: pid1 = %d",pid1); /* D */
       wait(NULL);
    }

    return 0;
}
```

Figure 3.34 What are the pid values?

```c
int main(){
        int id;
        static int x = 10;
        int y = 5;
        id = fork();
        if (id < 0){
                printf("fork failed\n");
        }
        else if(id == 0){
                printf("child started\n");
                printf("child finished\n");
        }
        else{
                wait(NULL);
                printf("parent started\n");
                x=x-2;
                y=y+5;
                printf("values of x: %d & y: %d\n",x,y);
                printf("parent finished\n");
        }
        x=x+5;
        y=y-5;
        printf("values of x: %d & y: %d\n",x,y);
        printf("terminating\n");

        return 0;
}
```

Process Termination

- A process terminates when it finishes executing its last statement and asks the operating system to delete it using the **`exit()`** system call.
- At that point the process may return a status value(typically an integer) to its parent process(via the wait() system call).
- All the resources of the process-including physical and virtual memory, open files, and I/O buffers are deallocated by the OS.

Process Termination (Termination cause by another process)

- Parent may terminate the execution of children processes using the **`abort()`** system call for a verity of reasons:
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates.

Inter-process Communication

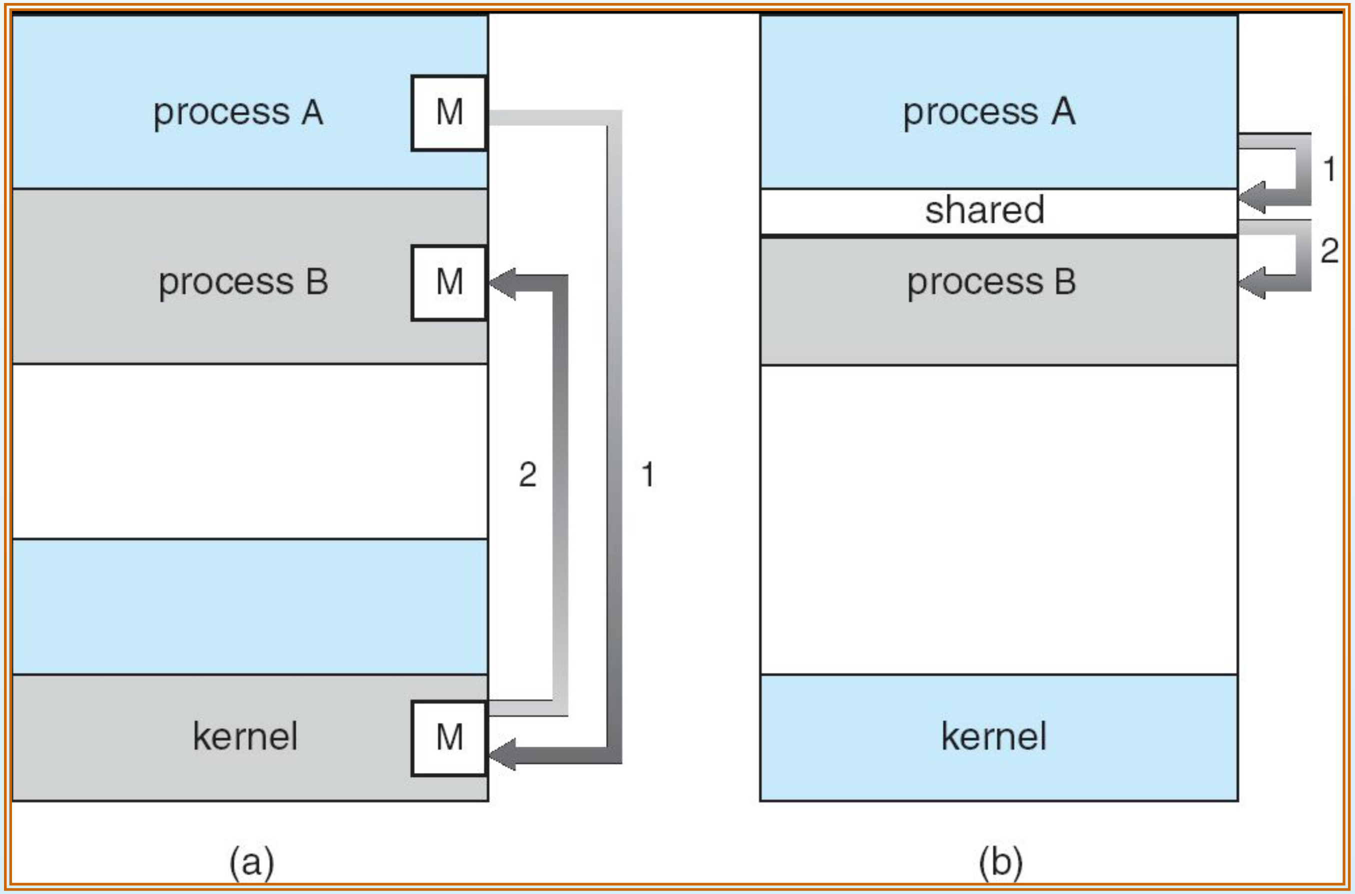
- Processes within a system may be ***independent*** or ***cooperating*** processes can execute concurrently.
- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process (any process that shared data with other process is a cooperating process)

  - **Advantages or reason of cooperating process**
    - Information sharing
    - Computation speed-up
    - Modularity
    - Convenience

Interprocess Communication

- Cooperating processes required **interprocess communication** (**IPC**) mechanism that will allow them to exchange data and information.
  - IPC provides a mechanism to allow processes to communicate and to synchronize there actions.
    - 4 There are two fundamental models of IPC
      - **Shared memory**
      - **Message passing**

Interprocess Communication (IPC) Models



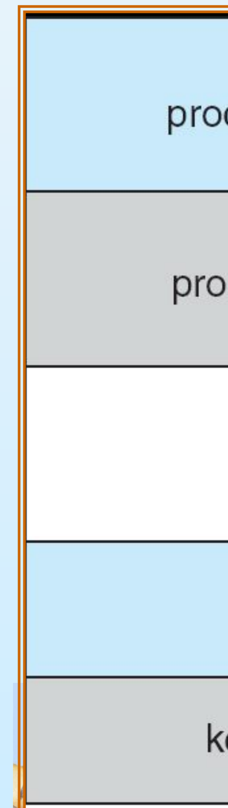
a) Message Passing b) Shared Memory

Shared-Memory

- Establish a region of shared memory for IIPC between processes.
- Other process that wish to communicate using this shared memory segment must attach it to their address space.
- Read and write data in the shared area
- Processes responsible for synchronization
- Must ensure that same memory location is not being modified by multiple processes at the same time.
- Shared memory requires that two or more processes agree to remove this restriction.

Message-Passing systems

- In the message passing model, communication takes place by means of messages exchanged between the cooperating processes
- Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.
- IPC facility provides at least two operations in message passing system:
  - **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



End of Chapter 3