

Operating Systems

Prepared by Dr. Alaa Eldeen Ahmed

IT263 / CYB510
Processes

Lecture 2



- Objectives:
 - Process Concept
 - Process Scheduling
 - Operations on Processes
 - Interprocess Communication

- Topics:
 - Identify the separate components of a process and illustrate how they are represented in an operating system.
 - Describe how processes are created and terminated in an operating system.
 - Describe and contrast inter-process communication using shared memory and message passing.

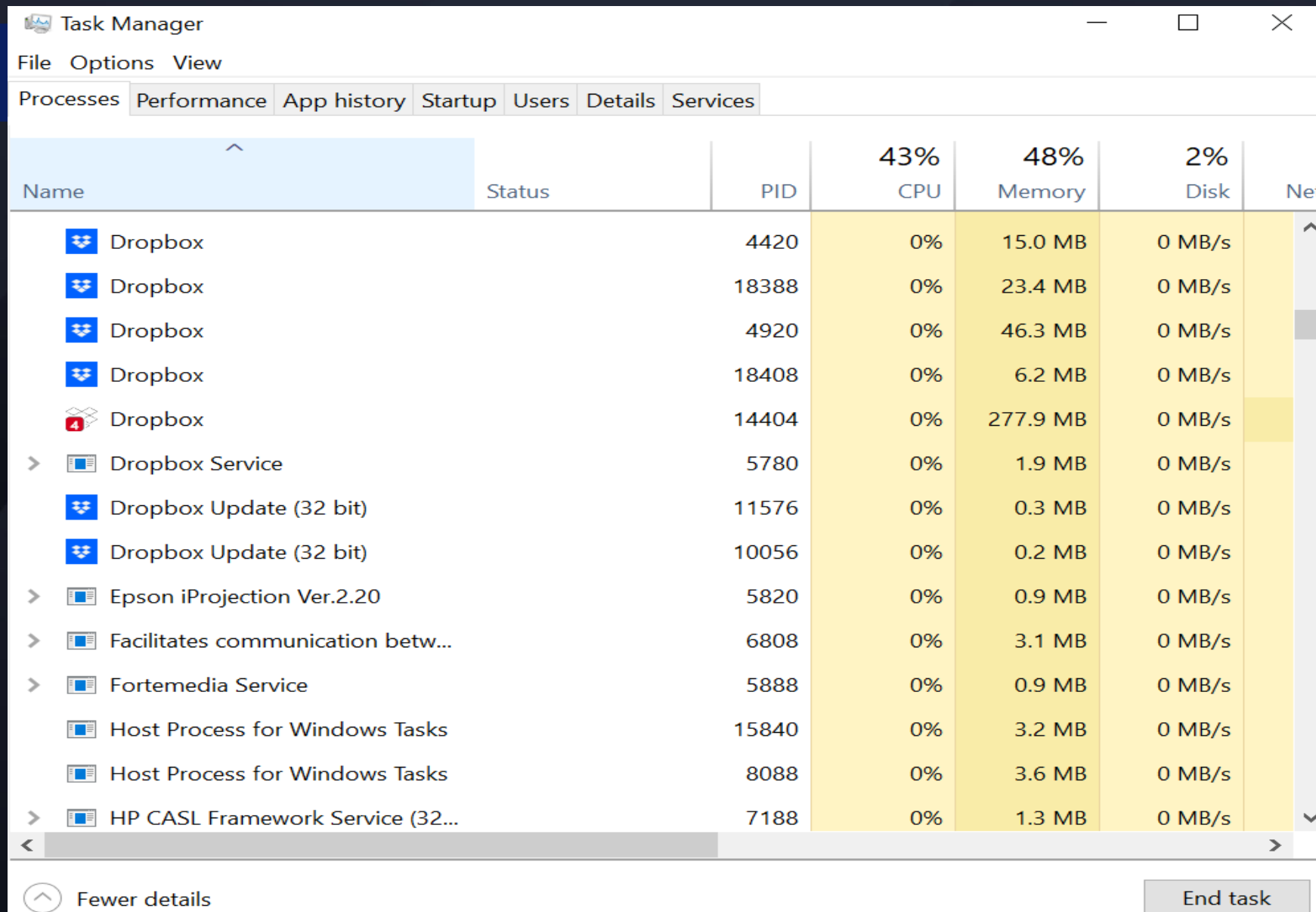
- Any program to be developed and established, the following steps are followed”
 1. The program needed is written using some high-level language such as C++, Java etc.
 2. The basic computer system doesn't understand High level languages, it understands only binary codes which are 0's and 1's
 3. Programs are converted to binary code using compilers.
 4. Program are converted into executable binary format and then it is ready for execution.

5. How should the program inform the computer what should be done by the program instructions --→ by starting execution
6. The program to execute, it has to be loaded into the memory and use the different resources of the computer system.
7. **The operating system** is responsible for loading the executable program into the memory and allocate its resources to allow the program to start execution.

- Program is a ***passive*** entity stored on disk (**executable file**). Passive means that the program is not doing any activities until that moment.
- When the **program** start execution then the program becomes a **processes** and it will be an ***active*** entity.
 - 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

- **Process** – a program in execution; process execution must progress in a specific order.
- A process refers to an entity that helps in representing the fundamental unit of work that must be implemented in any Operating system.
- In other words, we write the computer programs in the form of a text file, thus when we run them, these turn into processes that complete all of the duties specified in the program.

Lecture 02



The screenshot shows the Windows Task Manager window with the 'Performance' tab selected. The window title is 'Task Manager'. The menu bar includes 'File', 'Options', and 'View'. The tabs at the top are 'Processes', 'Performance', 'App history', 'Startup', 'Users', 'Details', and 'Services'. The 'Performance' tab displays a table of system resource usage. The table has columns for 'Name', 'Status', 'PID', 'CPU', 'Memory', 'Disk', and 'Net'. The 'CPU' column shows 43%, 'Memory' shows 48%, and 'Disk' shows 2%. The 'Name' column lists various processes, including multiple instances of 'Dropbox', 'Dropbox Service', 'Dropbox Update (32 bit)', 'Epson iProjection Ver.2.20', 'Facilitates communication betw...', 'Fortemedia Service', 'Host Process for Windows Tasks', and 'HP CASL Framework Service (32...'. The 'PID' column shows the process ID for each entry. The 'Status' column shows the status of each process. The 'CPU', 'Memory', and 'Disk' columns show the percentage of resource usage. The 'Net' column shows the network usage. At the bottom of the window, there is a 'Fewer details' button and an 'End task' button.

Name	Status	PID	43% CPU	48% Memory	2% Disk	Net
 Dropbox		4420	0%	15.0 MB	0 MB/s	
 Dropbox		18388	0%	23.4 MB	0 MB/s	
 Dropbox		4920	0%	46.3 MB	0 MB/s	
 Dropbox		18408	0%	6.2 MB	0 MB/s	
 Dropbox		14404	0%	277.9 MB	0 MB/s	
>  Dropbox Service		5780	0%	1.9 MB	0 MB/s	
 Dropbox Update (32 bit)		11576	0%	0.3 MB	0 MB/s	
 Dropbox Update (32 bit)		10056	0%	0.2 MB	0 MB/s	
>  Epson iProjection Ver.2.20		5820	0%	0.9 MB	0 MB/s	
>  Facilitates communication betw...		6808	0%	3.1 MB	0 MB/s	
>  Fortemedia Service		5888	0%	0.9 MB	0 MB/s	
 Host Process for Windows Tasks		15840	0%	3.2 MB	0 MB/s	
 Host Process for Windows Tasks		8088	0%	3.6 MB	0 MB/s	
>  HP CASL Framework Service (32...		7188	0%	1.3 MB	0 MB/s	

 Fewer details End task

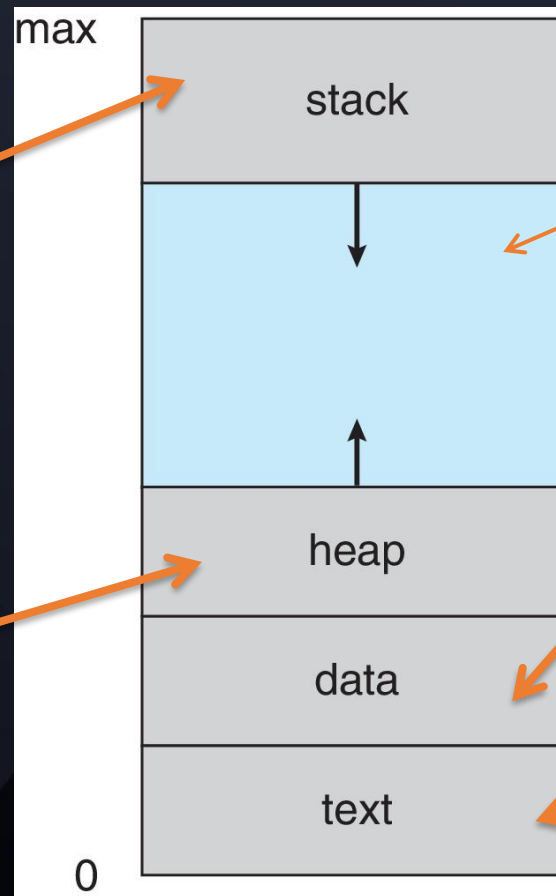
- A program can be divided into group of pieces when put into memory to become a process:
 - The program code, also called **text section**
 - Current activity including **program counter**, processor registers
 - **Stack** containing temporary data
 - Function parameters, return addresses, local variables
 - **Data section** containing global variables
 - **Heap** containing memory dynamically allocated during run time

Lecture 02

Process in Memory

Stack containing temporary data
Function parameters, return addresses, local variables

Heap containing memory dynamically allocated during run time



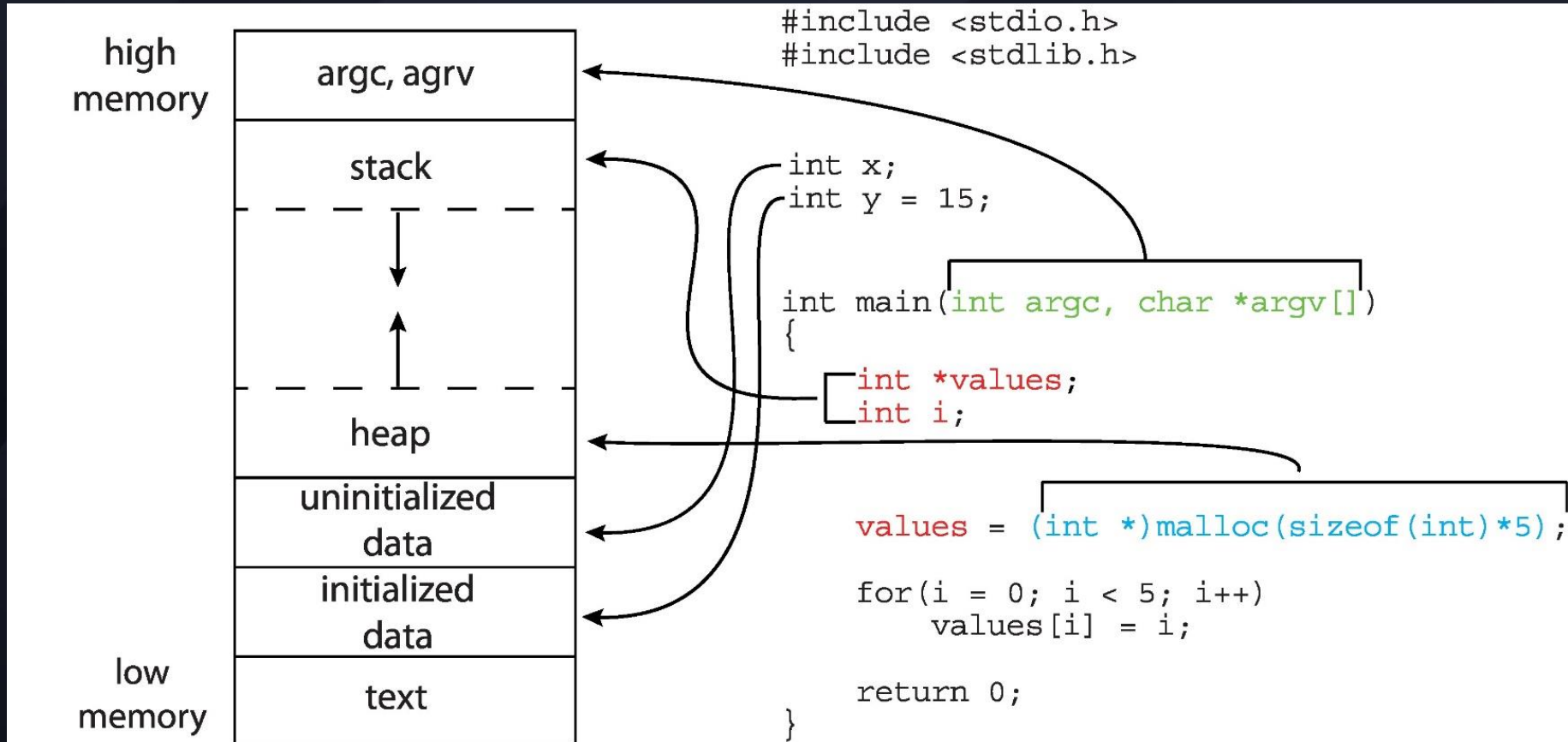
Current activity including **program counter**, processor registers

Data section containing global variables

The program code, also called **text section**

Lecture 02

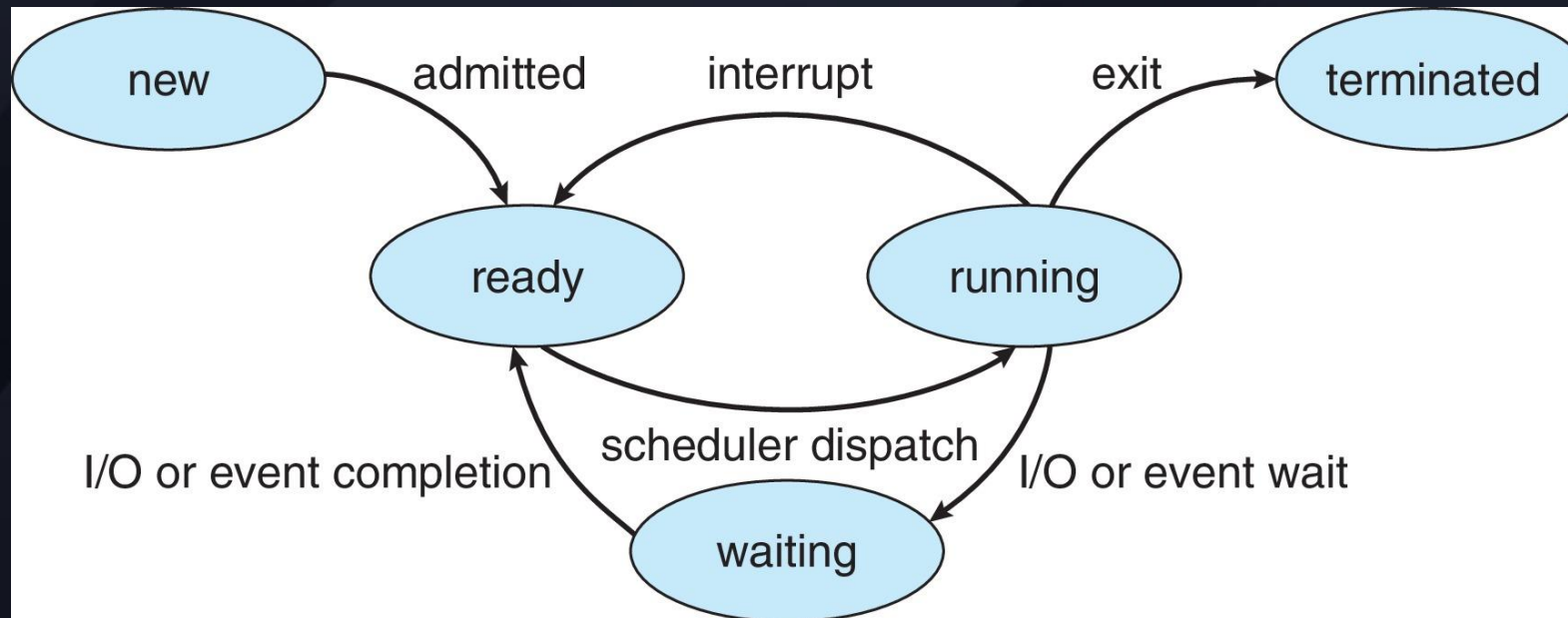
Memory Layout of a C Program



- 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

Lecture 02

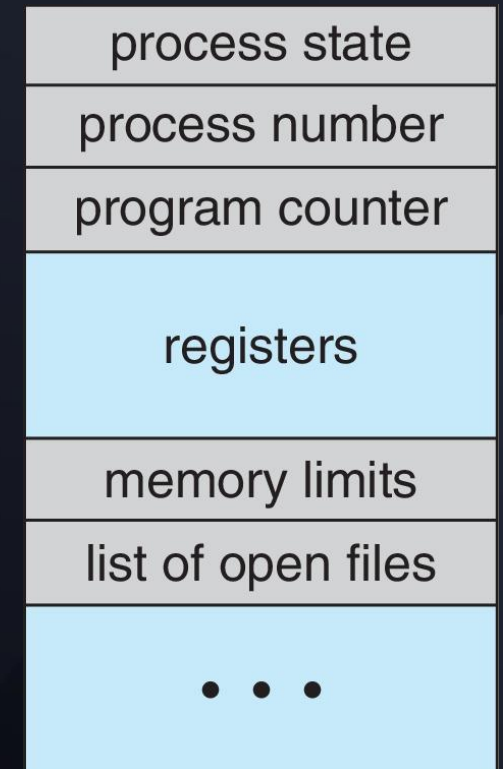
Diagram of Process State



Lecture 02

Process Control Block (PCB)

- Every process has a process control block, which is a data structure managed by the operating system.
- PCB stores all of the information required to maintain track of a process.
 - **Process state** – running, waiting, etc
 - **Process Number (ID)**. Each process in the OS has its own unique identifier.



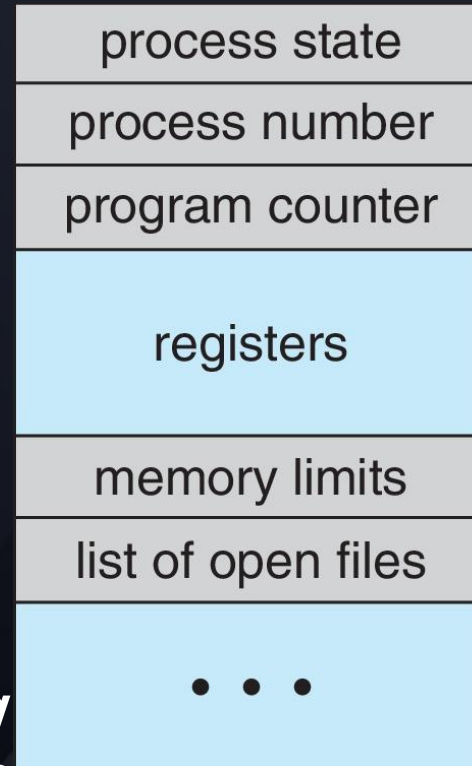
- **Program counter** – location (address) of instruction to next execute. (the next instruction to be executed)
- **CPU registers** – contents of all process-centric registers.
- **CPU scheduling information**- priorities.

process state
process number
program counter
registers
memory limits
list of open files
...

Lecture 02

Process Control Block (PCB)

- Memory-management information – memory allocated to the process
- Accounting information: Account of all resources used by the process such as (CPU time , memory,)
- I/O status information – I/O devices allocated to process, list of open files.
- *The PCB architecture is fully dependent on the operating system, and different operating systems may include different information.*



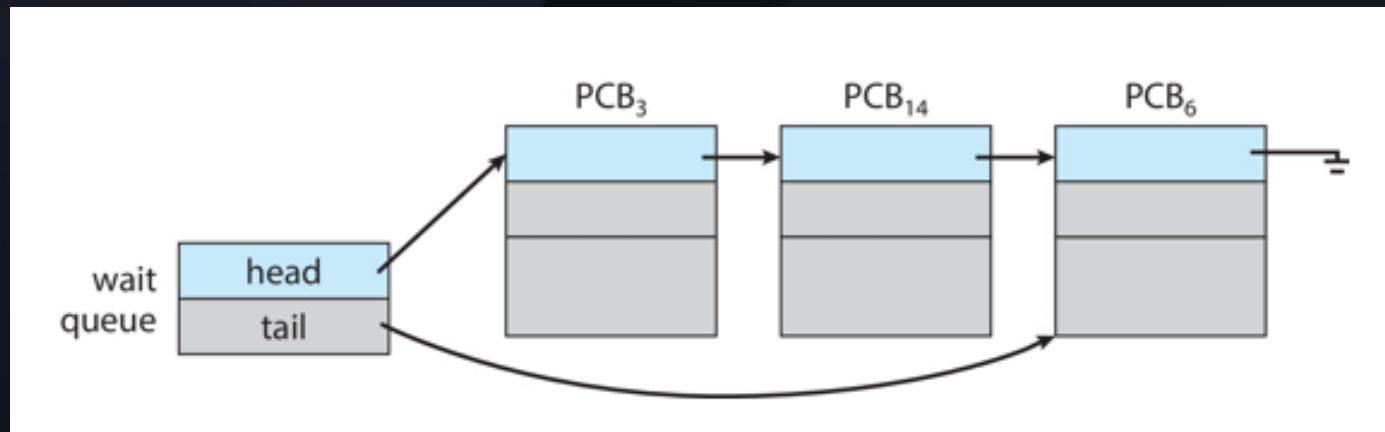
- **When there are several or more runnable processes, the operating system chooses which one to run first; this is known as process scheduling.**
- **A scheduler is a program that uses a scheduling algorithm to make choices.**

- OS, Maintains **scheduling queues** of processes
- All processes, upon entering the system, are stored in the **Job Queue** (set of all processes in the system)
- Processes in the Ready state are placed in the **Ready Queue** (set of all processes residing in main memory, ready and waiting to execute)

Lecture 02

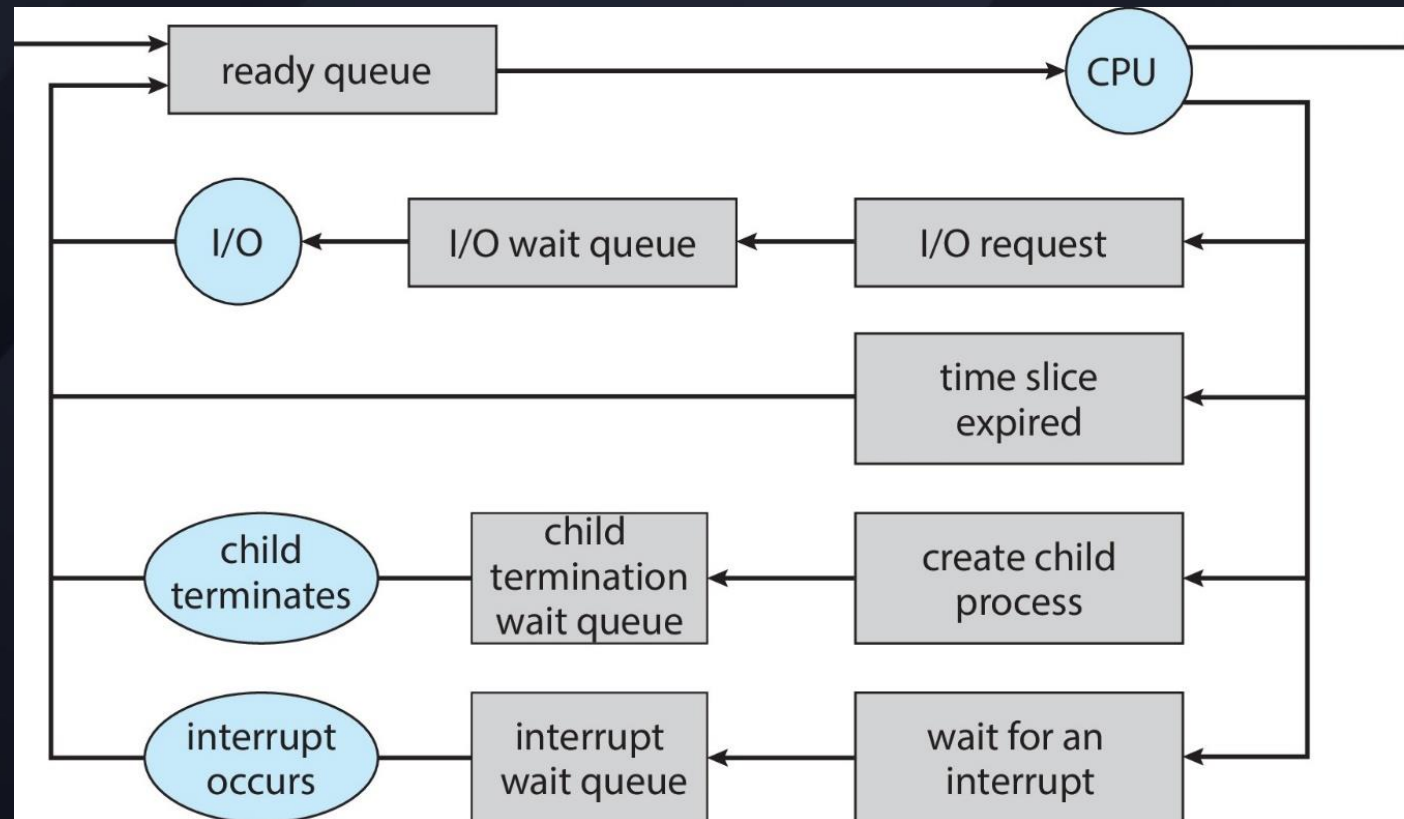
Process Scheduling

- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.
- Processes migrate among the various queues



Lecture 02

Representation of Process Scheduling



Representation of Process Scheduling

- A new process is initially put in the **Ready queue**.
- It waits in the ready queue until it is selected for execution(or dispatched).
- Once the process is assigned to the CPU and is executing, one of the following several events can occur:
 1. The process could issue an I/O request, and then be placed in the **I/O queue**.
 2. The process could create a new subprocess and wait for its termination.
 3. The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

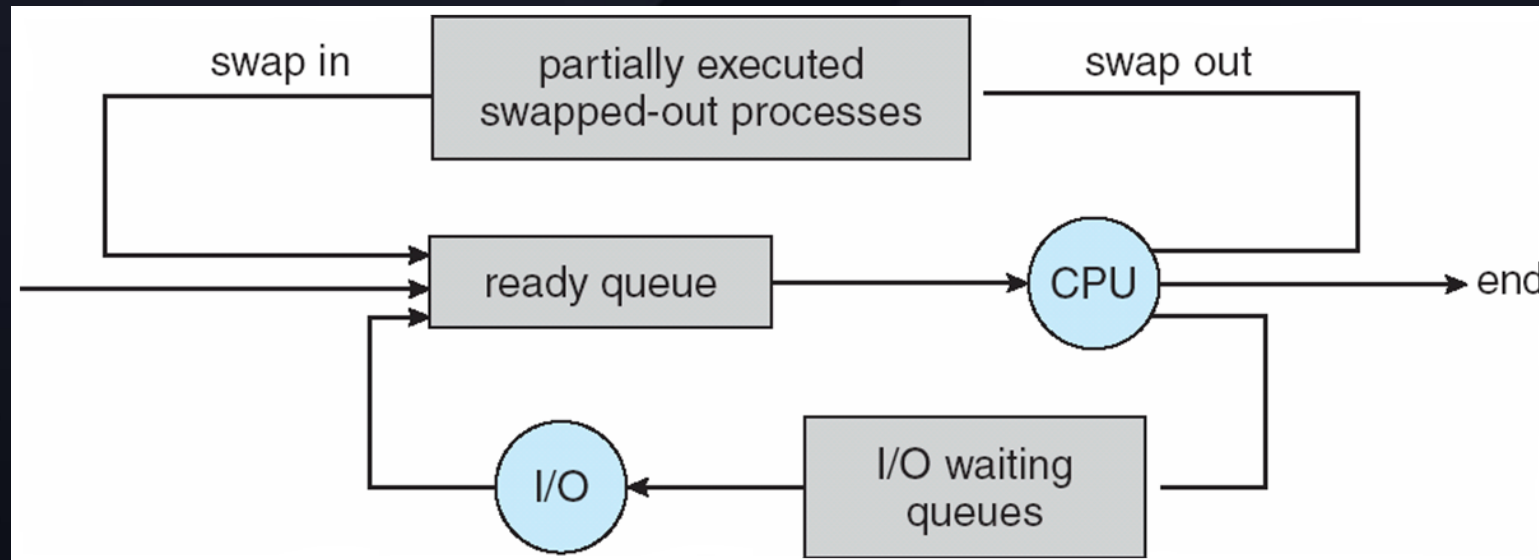
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
 - Sometimes the only scheduler in a system
 - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)

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*

- ❑ **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**

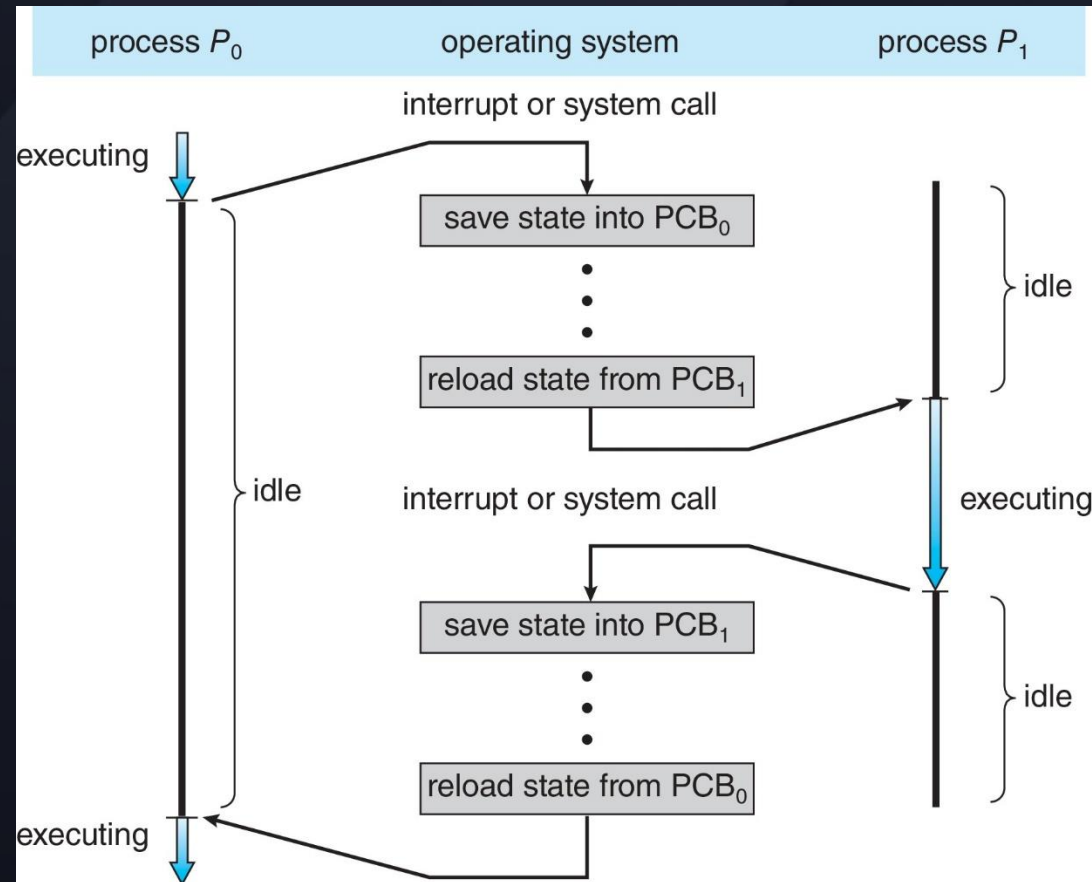


- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB. When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

- 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

Lecture 02

CPU Switch From Process to Process



- System must provide mechanisms for:
 - process creation
 - process termination

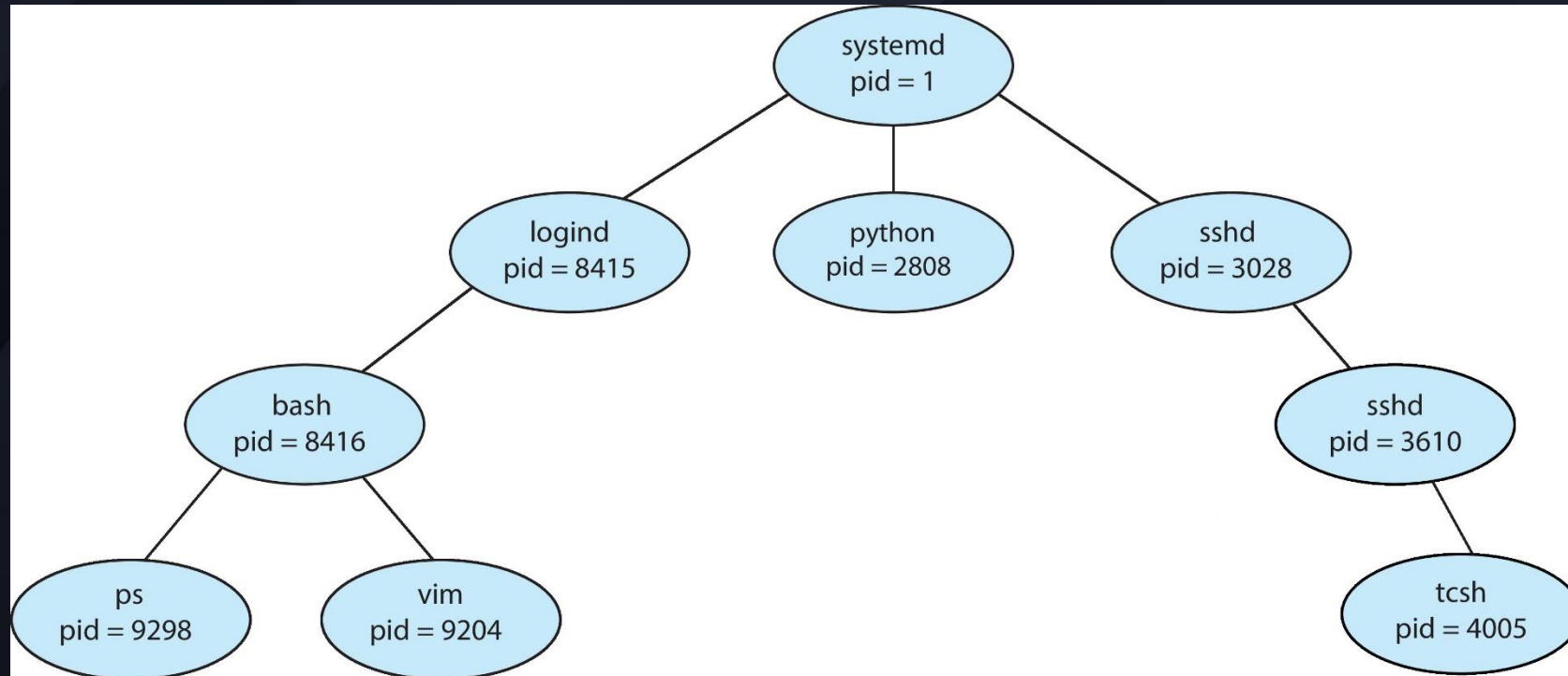
- A process may create several new processes, via create process system call during the course of execution.
- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier (pid)**
- **Resource sharing options**
 - Parent and children share all resources
 - Children share subset of parent's resources
 - Parent and child share no resources

- **Execution options**

- Parent and children execute concurrently.
- Parents Wait for the child process to terminate before proceeding. Parent process makes a `wait()` system call, for either a specific child process or for any particular child process, which causes the parent process to block until the `wait()` returns.

Lecture 02

Tree Of processes in Linux



Process Creation (Cont.)

- There are also two possibilities in terms of the address space of the new process:
 - 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' memory space with a new program
 - Parent process calls **wait()** for the child to terminate

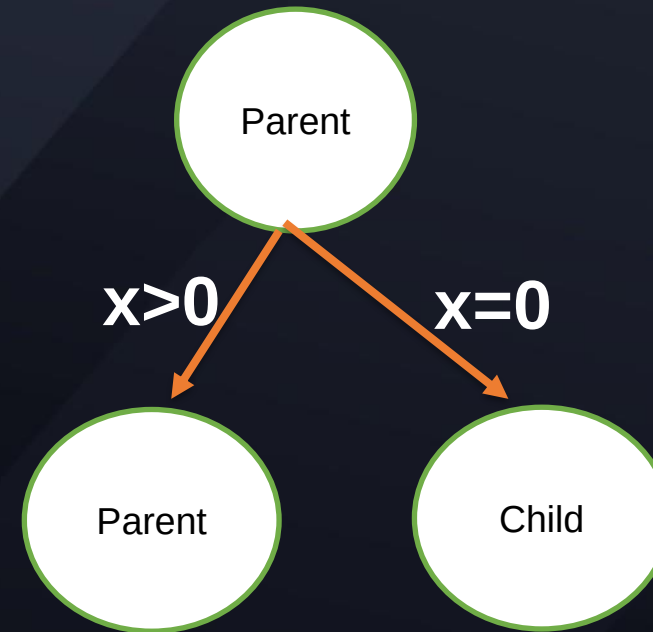
- The fork system call creates a new process.
- If the operation is successful, there are then both
- **parent** and **child** processes and both see fork return, but with different values:
 - it returns a value of 0 in the child process
 - It returns the child's process ID (+ve value) in the parent process.
 - It returns a value of -1 if the process creation failed, the parent process and no child is created.

Lecture 02

Fork() System Call

```
x = Fork()
```

```
If (x == 0) // child is running  
    print ("I'm child");  
else  
    print ("I'm parent");
```



Output :

I'm parent
I'm child

or

Output :

I'm child
I'm parent

Lecture 02

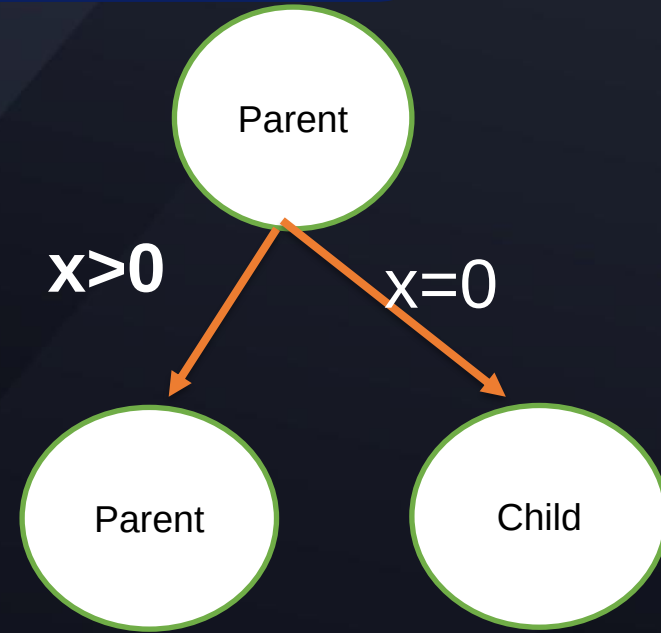
wait() System Call

```
x = Fork()
```

```
If (x == 0) // child is running  
    print ("I'm child");  
Else  
    wait();  
    print ("I'm parent");
```

Output :

I'm child
I'm parent



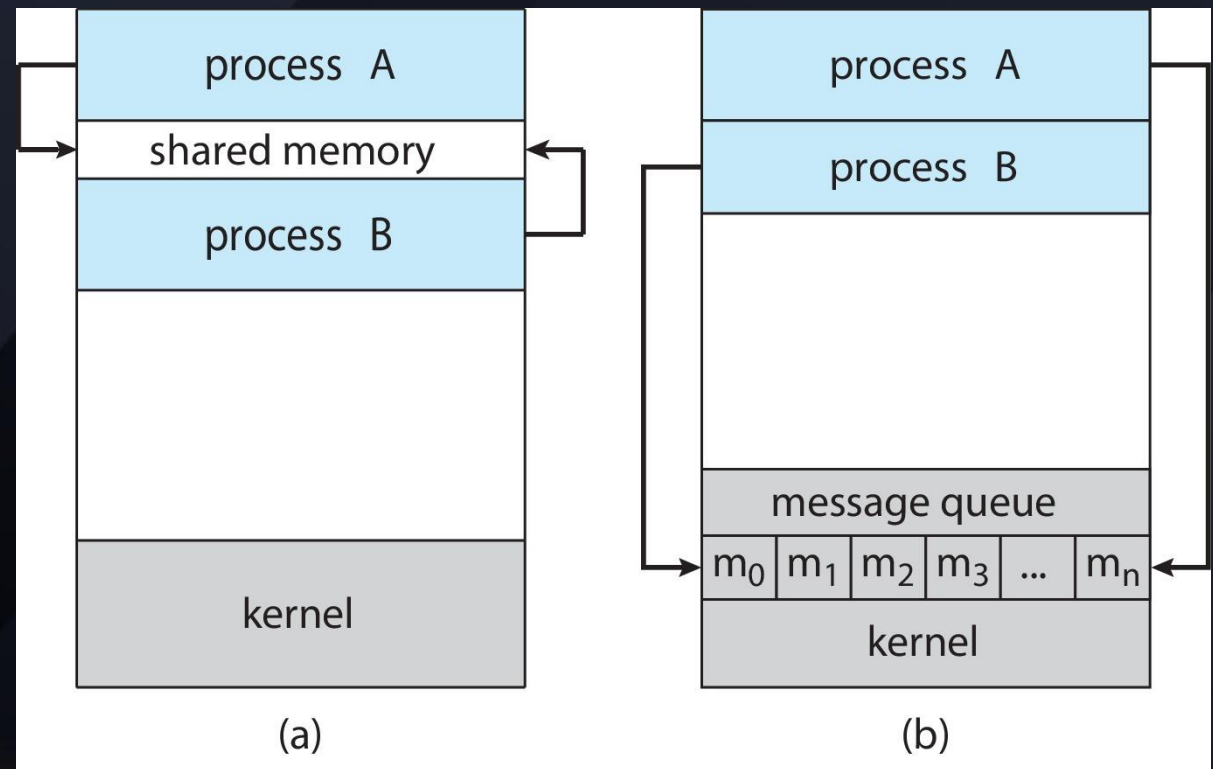
- A process terminate when it finish executing its last statement and then asks the operating system to delete it using the **exit()** system call.
- Process' resources are deallocated by operating system

- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
 - 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
- If no parent waiting (did not invoke wait()) process is a zombie
- If parent terminated without invoking wait , process is an orphan

- Some operating systems do not allow child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated.
 - **cascading termination.** All children, grandchildren, etc. are terminated.
 - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the **wait()** system call. The call returns status information and the pid of the terminated process

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
 - Information sharing
 - Computation speedup
 - Modularity
 - Convenience

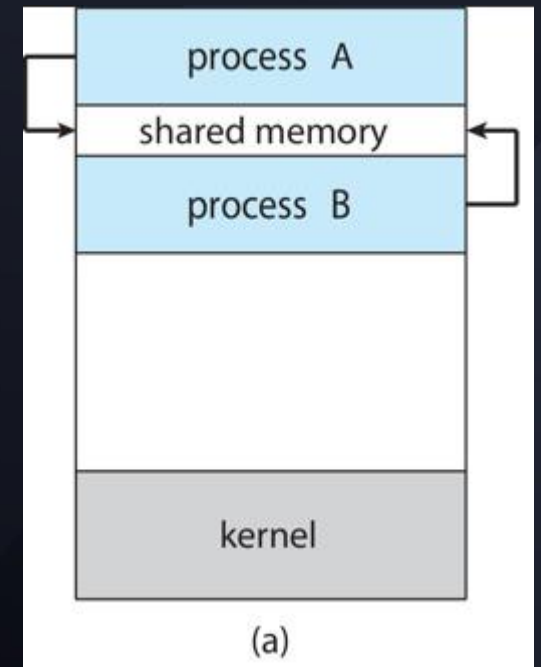
- Cooperating processes need **inter-process communication (IPC)**
- Two models of IPC
 - **Shared memory**
 - **Message passing**



Lecture 02

Shared Memory

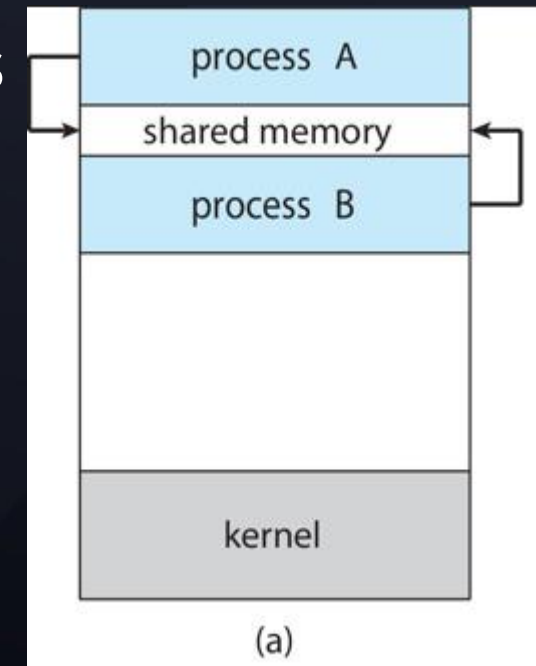
- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- **Synchronization is discussed in great details in Lecture 4**



Lecture 02

Shared Memory

- Typically, a shared memory region resides in the address space of the process creating the shared memory segment.
- Other processes that wish to communicate using this shared memory segment must attach it to its address space.
- Normally, the operating system tries to prevent one process from accessing another process memory.
- Shared memory requires that two or more processes agree to remove this restriction.



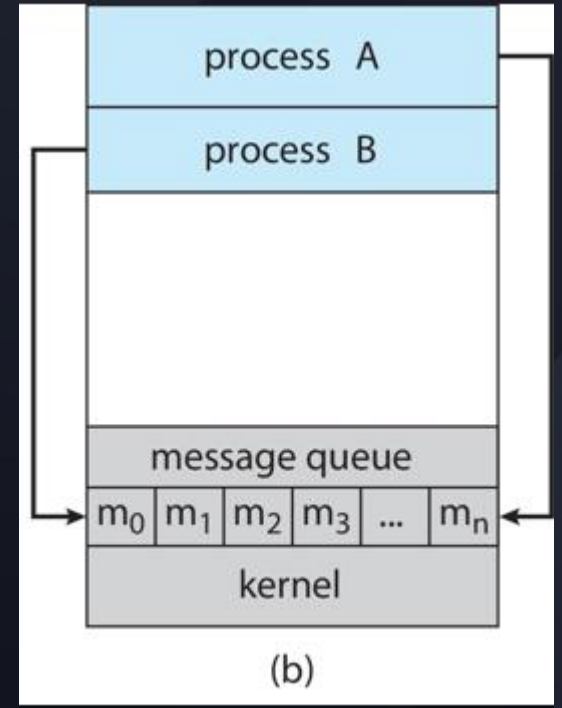
- Is a typical problem where the **producer process** produces information that's consumed by a **consumer process**
- For example, compiler may **produce** assembly code which is **consumed** the assembler, the assembler In turn **produce** object modules which are **consumed** by the loader program and so on...
- One solution to the producer consumer problem is the use of shared memory.
 - To allow the producer and consumer processes to run concurrently we must have a buffer of items that can be filled by the producer and emptied by the consumer

- The buffer reside in region of memory that is shared by the producer and the consumer processes
- The producer and consumer must be synchronized so that the consumer doesn't try to consume item that has not yet been produced

Lecture 02

Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- Message passing is useful in distributed environment where the communication processes may reside on computers connected by a network.
- IPC facility provides two operations:
 - **send**(*message*)
 - **receive**(*message*)
- [The *message* size is either fixed or variable]



- If processes 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:
 - Shared memory
 - Hardware bus
 - Network
 - Logical:
 - Direct or indirect
 - Synchronous or asynchronous

- Processes must name each other explicitly:
 - **send** (P , *message*) – send a message to process P
 - **receive**(Q , *message*) – receive a message from process Q
- **Properties of communication link**
 - Links are established automatically
 - A link is associated with exactly one pair of communicating processes
 - Between each pair there exists exactly one link
 - The link may be unidirectional, but is usually bi-directional

- Messages are directed and received from mailboxes (also referred to as ports)
 - Each mailbox has a unique id
 - Processes can communicate only if they share a mailbox
- **Properties of communication link**
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes
 - Each pair of processes may share several communication links
 - Link may be unidirectional or bi-directional

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
 - **Blocking send** -- the sender is blocked until the message is received
 - **Blocking receive** -- the receiver is blocked until a message is available

- **Non-blocking** is considered **asynchronous**
 - **Non-blocking send** -- the sender sends the message and continue
 - **Non-blocking receive** -- the receiver receives:
 - A valid message, or
 - Null message
- Different combinations possible
 - If both send and receive are blocking, we have a **rendezvous**



Thank You