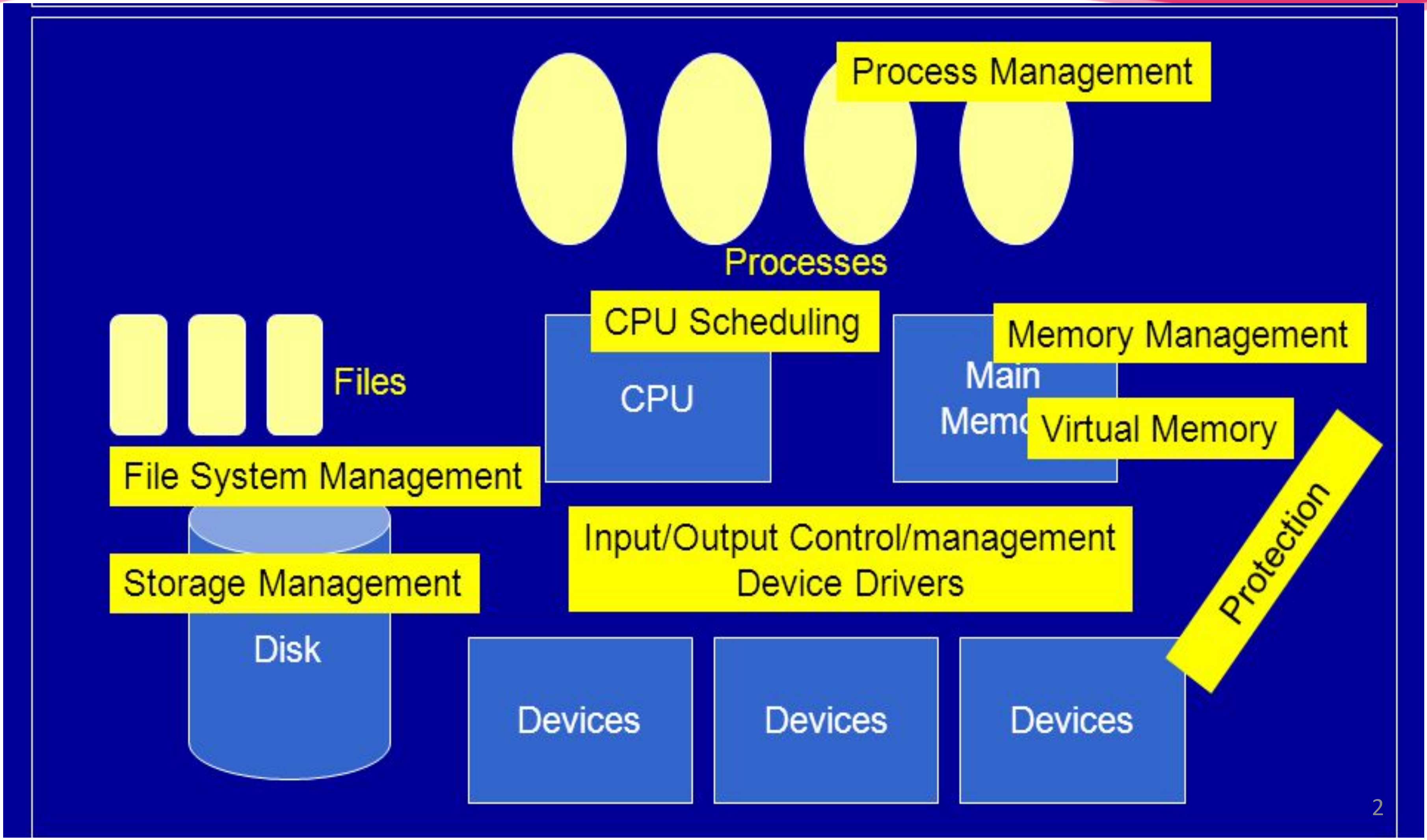
Lecture 02: Process Management

CIS 21012 – Platform Technologies

Shafana M. Shareef

South Eastern University of Sri Lanka

Major OS
Concepts / Components / Functions



Major Functions of OS

- Process Management
  - Providing process abstraction and managing processes
  - Ex:- starting execution by loading into main memory, interrupting, terminating.
- CPU Scheduling
  - Ex:- deciding to allocate CPU, time limit for a program
- Main Memory Management
  - Sharing memory among many process
  - EX:- deciding which part of the memory is used by which program
- Virtual Memory management
  - Ex:- we can also run programs whose memory need is larger than the physical memory what we have, this is possible by the virtual memory

Major Functions of OS…

- Secondary-Storage Management
  - Proving file abstraction
  - Mapping files to disk blocks, dick scheduling
  - Ex:- allowing to store files in internal drives or external drives
- I/O System Management
  - Device derivers, buffering, proving uniform access interface
  - Ex:- when a C program uses scanf and printf system calls, OS directly deals with hardware like keyboard and monitor.
- File Management
  - Ex:- allow programs to create a file to write something or read something from the existing file.
- Protection and Security
  - Controlled access to resources, preventing processes interfering with each other and OS

Lesson 02: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Inter-process Communication
- Examples of IPC Systems
- Communication in Client-Server Systems

Objectives

- To introduce the notion of a process – **“a program in execution, which forms the basis of all computation”**

- To describe the various features of processes, including **scheduling**, **creation** and **termination**, and **communication**

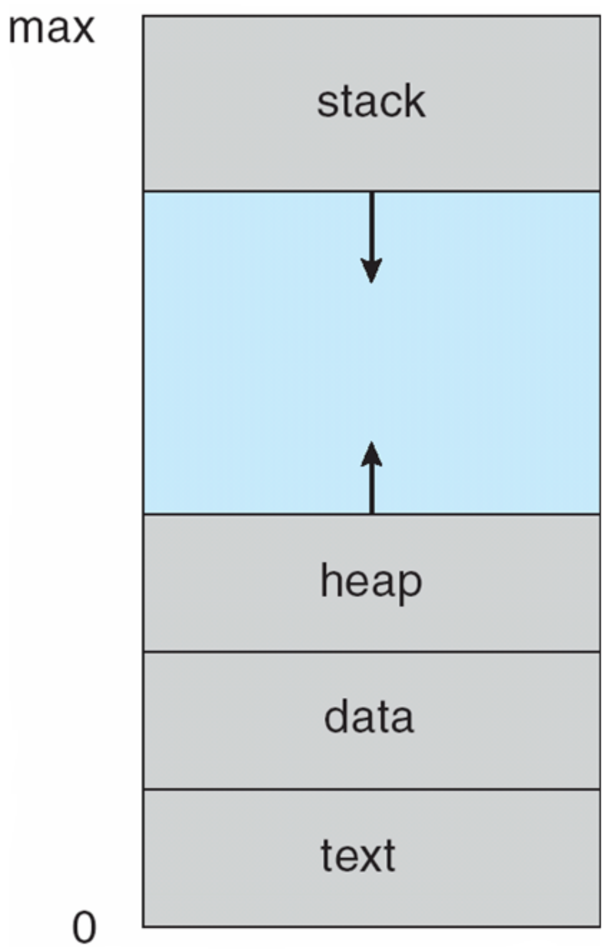
- To explore inter-process communication using **shared memory** and **message passing**

- To describe communication in client-server systems

Process Concept

- What is source code of a program?
- What is programs?
- **Process** – a program in execution; or when a program is loaded into main memory OS will create a kind of data structure which is called process.
- Process data structure contains 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



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
- The Process is the unit of protection which gives the authority to run a code.

Attributes of a process

1. Process ID :- *Unique number*
2. Program Counter
3. Process State
4. Priority:- *system process, user process*
5. General Purpose Registers
6. List of open files
7. List of open devices:- *list of scanners, printers, and hardware devices*
8. Protection :- *OS maintains one workspace of a process as can not be used by other processes*

Program Counter

- If a process is stopped at a point and restarted again, it should continue from the point where it stopped.
- OS provide number for each instruction
- For Example:-

```
I1    int    main()[
I2           char str[]="Hello world\n"
I3           printf("%s", str);
I4           printf("%s", str);
I5           printf("%s", str);
I6           }
```

- Program counter will contain what is the next instruction which needs to be executed.

General Purpose Registers

- CPU uses registers to store some values during the execution of process
- For Example:-
  - For P1➔ R1=1, R2=5, R3=3
    I3: ……. Stopped
    I4: R1=R2+R3
  - P1 executed and after few minutes P1 has preempted and P2 is started executing. And P2 changes the registers as follows
  - For P2➔ R2=2, R3=3
  - And P2 stopped after few minutes, and P1 restarted to execute. Now the value of R2 and R3 is wrong.
  - So, to avoid this problem the GPR also should be stored somewhere else in the memory, and it should be reloaded when the process started to execute again.
  - That is why OS keeps track of stages of GPR for each stages of each process.

List of open files

- When executing some process, some files will be opened for reading or writing. OS needs to keep track of open files.

- Example:- because, when you delete any file via program, it should show the message as the file is being used by other programs.

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

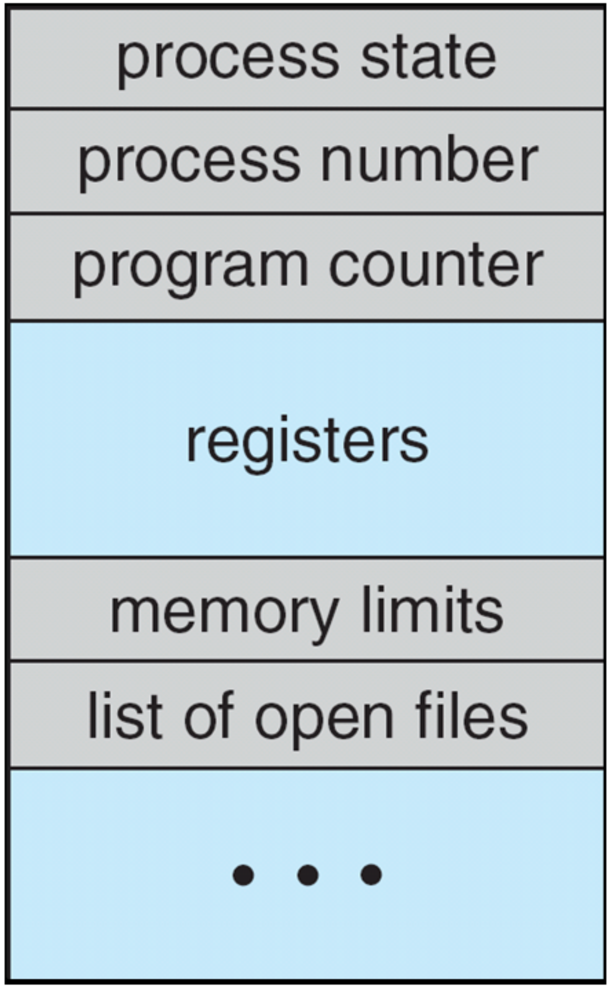
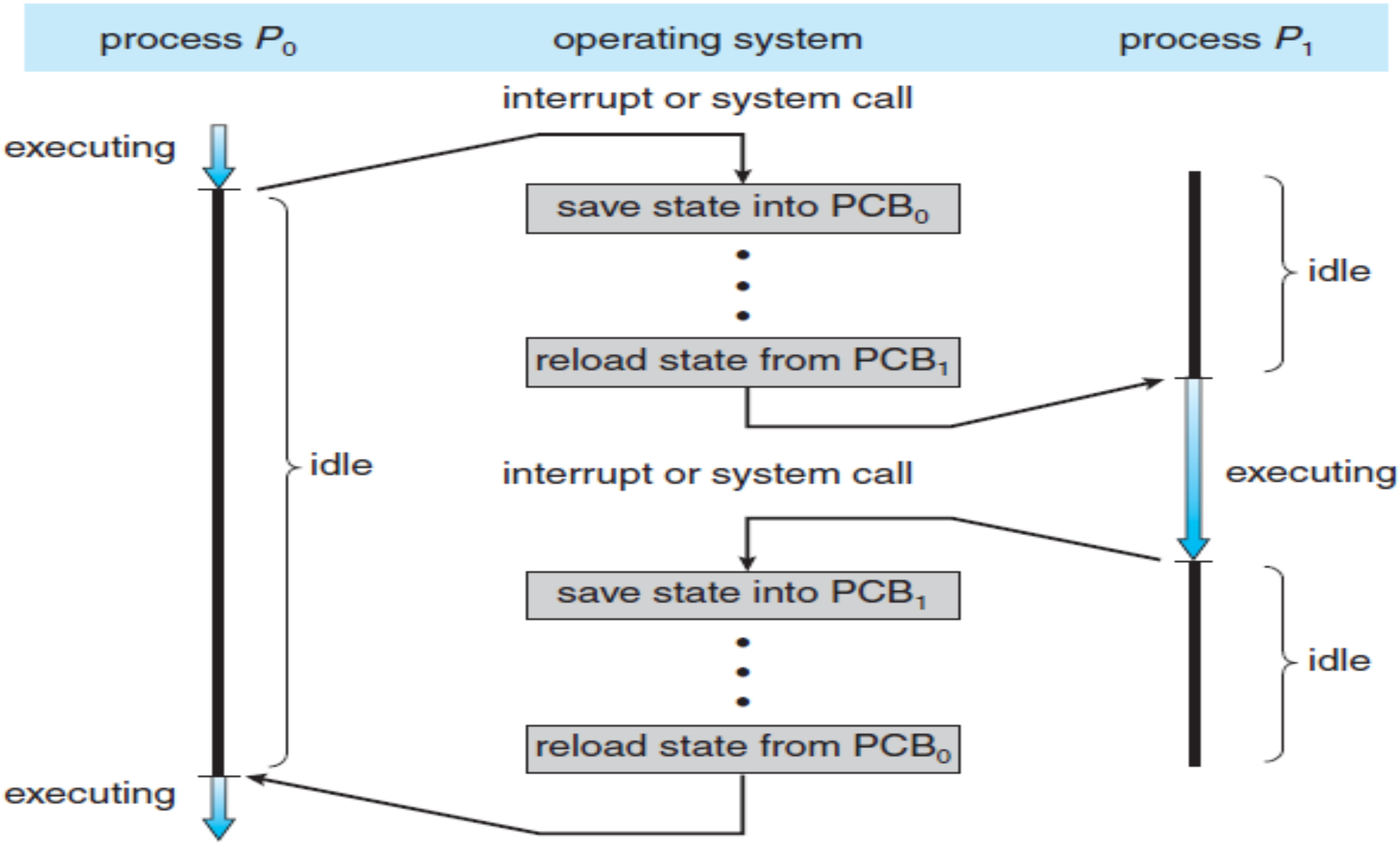


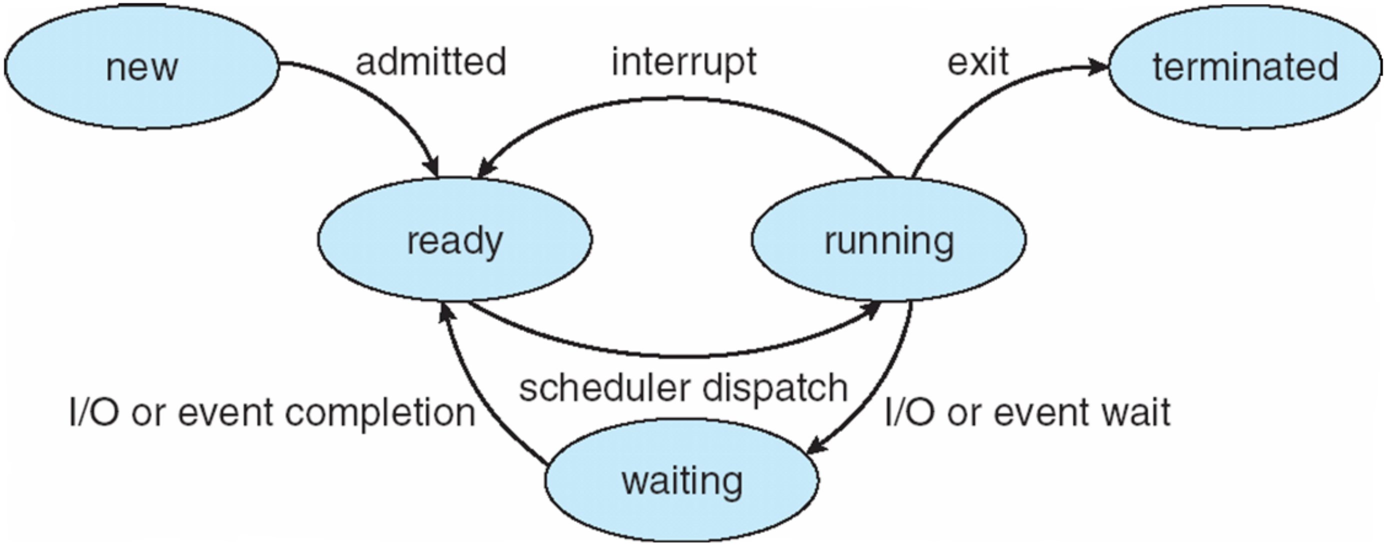
Diagram showing CPU switch from process to process



Process State

- From creation to termination, every process going to go through in various states.
  - **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



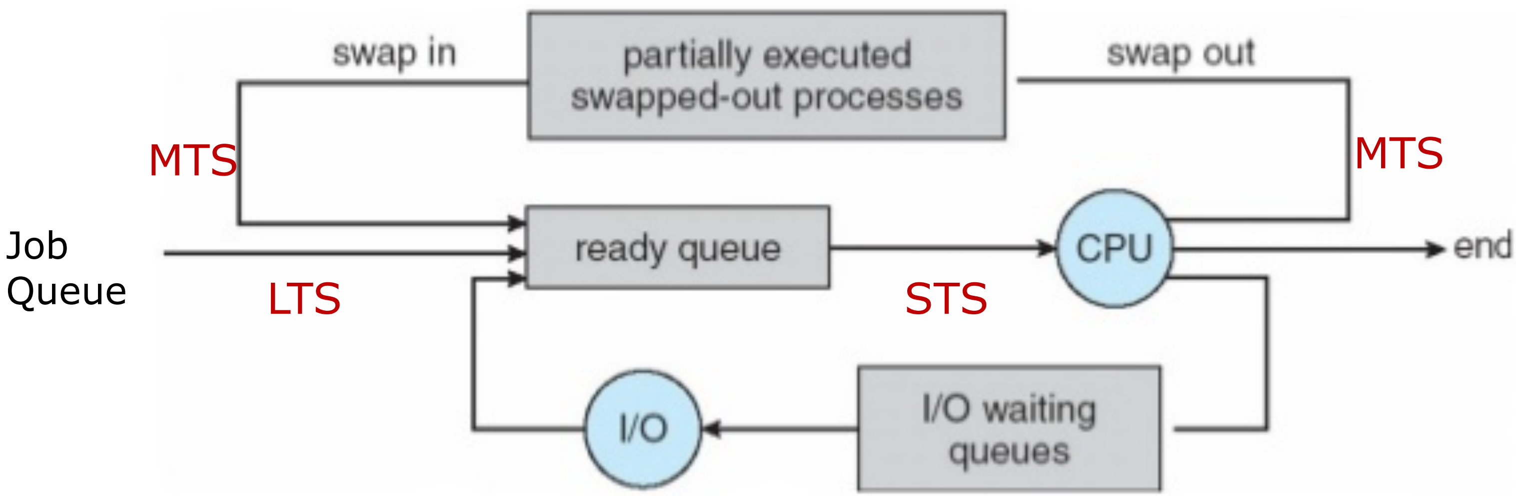
Process Scheduling queues

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- The OS maintains all PCBs in Process Scheduling Queues.
- 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

Schedulers

Schedulers are special system software which handle process scheduling in various ways

- Long-term scheduler (or job scheduler) –
  Selects which processes should be brought into the ready queue
- Short-term scheduler (or CPU scheduler) –
  Selects which process should be executed next and allocates CPU
- Medium-term scheduler (or process swapping scheduler) –
  Remove process from memory, store on disk, bring back in from disk to continue execution



- Long-term scheduler :- select the processes to bring it into Main memory (if it is need to be done, the following processes should be done)
    - Calculates the properties of processes
    - Examine the main memory space availability
    - Analysis the processes which not in use recently
    - Swap out the “not in use processes”
    - Assign a memory place for new process
    - Save the details in PCB

Short-term scheduler :- allocating CPU (if it is need to be done, the following processes should be done)

- Swap out / terminate the running process from CPU
- Based on any scheduler algorithm select the process and assign CPU

Medium-term scheduler :- managing interrupts of process (if it is need to be done, the following processes should be done)

- Swap out the process from CPU
- Find a place on disk to store based on already calculated properties of process
- And store it on disk
- Swap in the process into ready queue

Schedulers (Cont)

- STS is invoked very frequently (milliseconds)
  - Must be very fast
- LTS is invoked very infrequently (seconds, minuts)
  - May be slow
- Processes can be described as either:
  - I/O-bound processes – spends more time doing I/O than computations.
  - CPU-bound processes – spends more time doing computations

Context Switching

- 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-switching time is overhead: the system does no useful work while switching
- Time dependent on hardware support

Operations on Processes

Operations on Processes

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

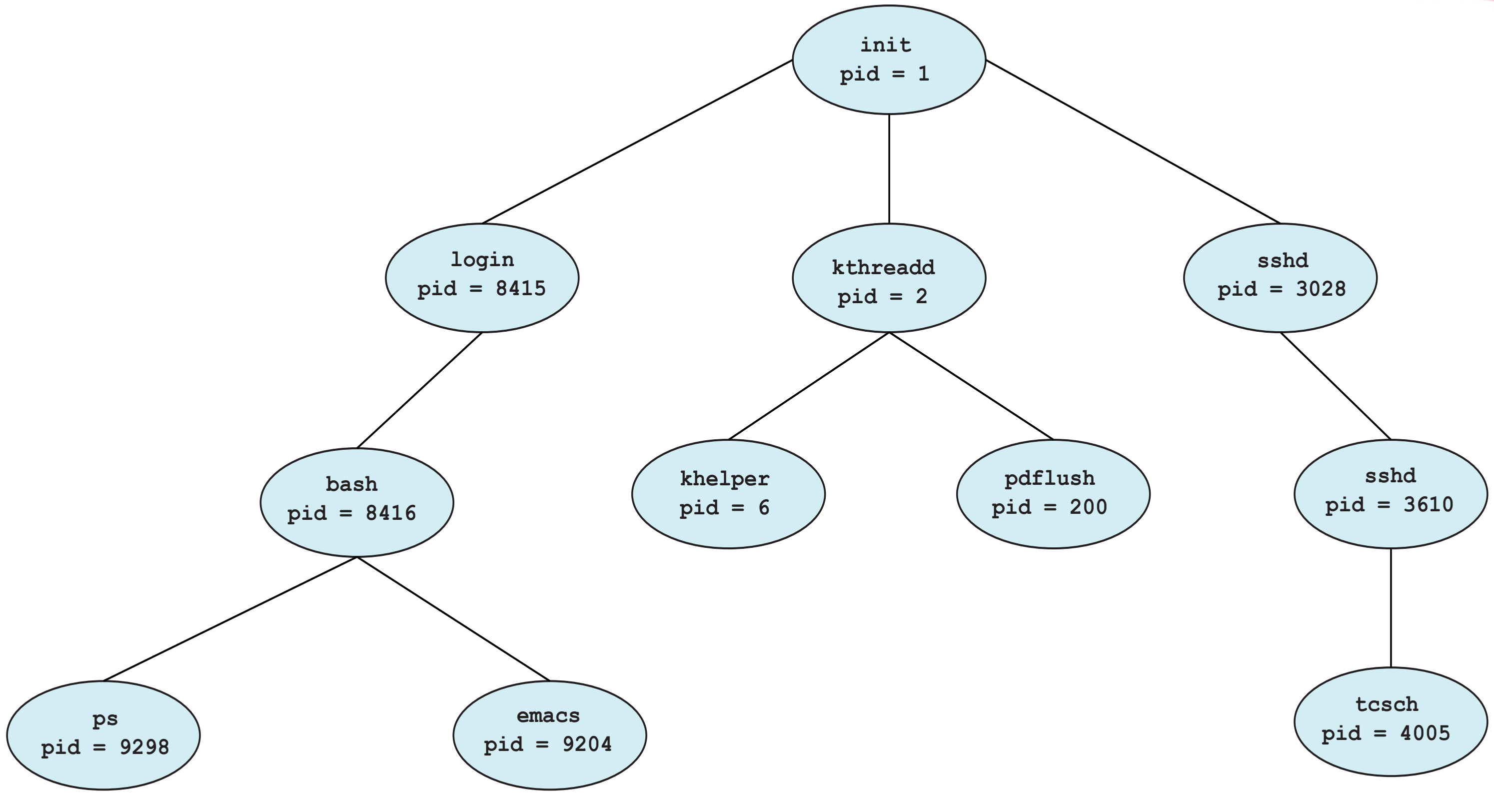
Process Creation

- **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**)
- When a process creates a new process - Resource sharing options
  1. Parent and children share all resources
  2. Children share subset of parent's resources
  3. Parent and child share no resources
- When a process creates a new process - execution possibilities
  1. Parent and children execute concurrently
  2. Parent waits until children terminate

Process Creation (Cont.)

- When a process creates a new process - Address space possibilities
  - Child duplicate of parent (it has the same program and data as the parent)
  - Child has a new program loaded into it

A Tree of Processes in Linux



Process Termination

- Process executes last statement and then asks the operating system to delete it using the **`exit()`** system call.
  - At that point, the process may returns a status value from child to parent (via **`wait()`**)
  - All the resources of Process – including physical and virtual memory, open files, and I/O buffers- are deallocated by operating system

Process Termination

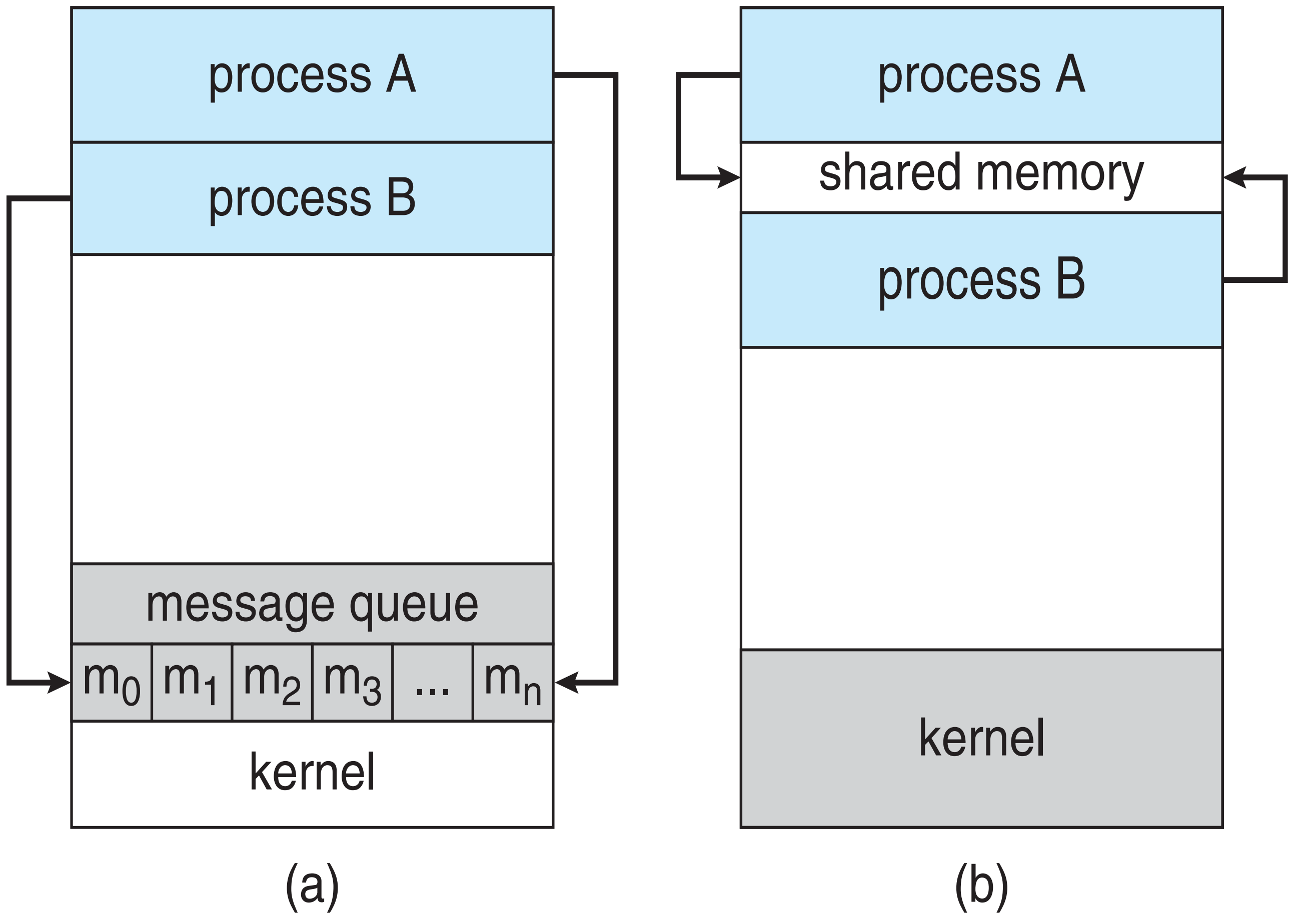
- Parent may terminate the execution of children processes using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting(it self terminating) and the some operating systems does not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

Inter-process Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes,
  - Any process that shares data with other processes is a cooperative process.
- Reasons for process cooperation:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **inter-process communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Communications Models

(a) **Message passing.** (b) **shared memory.**



Inter-process Communication: Shared Memory

- An portion of memory shared among the processes that wish to communicate.
  - Processes can then exchange information by reading and writing data to the shared region.

- 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.

Inter-process Communication
Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **`send`**(*message*)
  - **`receive`**(*message*)
- The *message* size is either fixed or variable

Message Passing (Cont.)

- If processes *P* and *Q* wish to communicate, they need to:
  - Establish a ***communication link*** between them
  - Exchange messages via send/receive calls

Message Passing (Cont.)

- The communication link can be implemented in a variety of ways
  - Physical implementation:
    - Shared memory
    - Hardware bus
    - Network
  - Logical implementation:
    - Direct or indirect communication
    - Synchronous or asynchronous communication
    - Automatic or explicit buffering

Synchronization

- 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

Buffering

- The message exchanged by communicating processes reside in a temporary queue.
- Queue can be implemented in one of three ways
  1. Zero capacity – no messages are queued on a link.
     Sender must wait for receiver
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits

Assignment 1

- The communication link can be implemented in a variety of ways
  - Physical implementation:
    - Shared memory
    - Hardware bus
    - Network
  - Logical implementation:
    - Direct or indirect communication
    - Synchronous or asynchronous communication
    - Automatic or explicit buffering
- Briefly explain each of the above ways.