SLIIT
Discover Your Future

IT2060/IE2061

Operating Systems and System Administration

Lecture 03

Introduction to Processes

U. U. Samantha Rajapaksha

M.Sc.in IT, B.Sc.(Engineering) University of Moratuwa

Senior Lecturer SLIIT

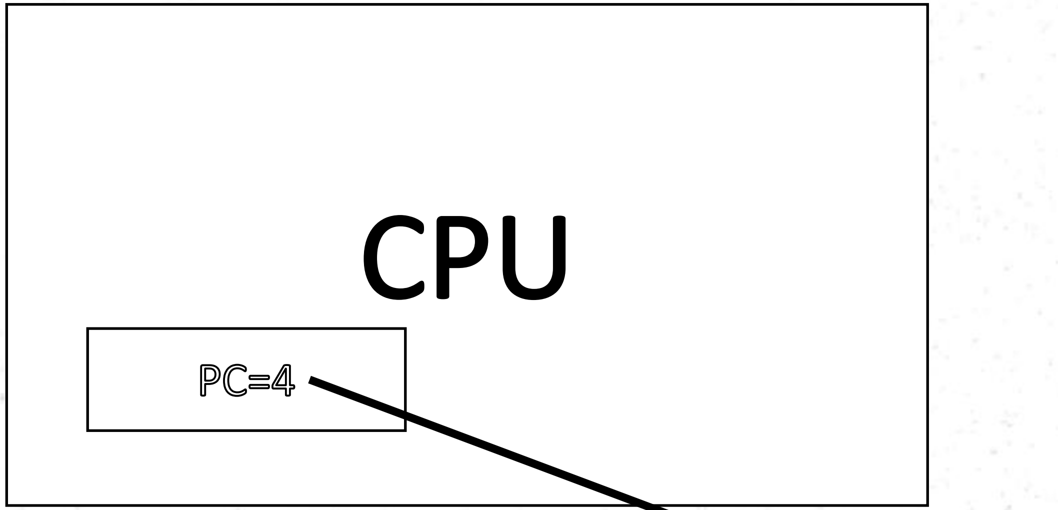
Samantha.r@slit.lk

Chapter 3: Processes

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

Process Concept

- **Process** – a program in execution.
- Program is a ***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 when multiple users executing the same program.



Memory Address	Content
0	INS 1(Executed)
1	INS 2(Executed)
2	INS 3(Executed)
3	INS 4(Executed)
4	INS 5
5	

RAM

Process Concept (Cont.)

- Process execution is sequential.

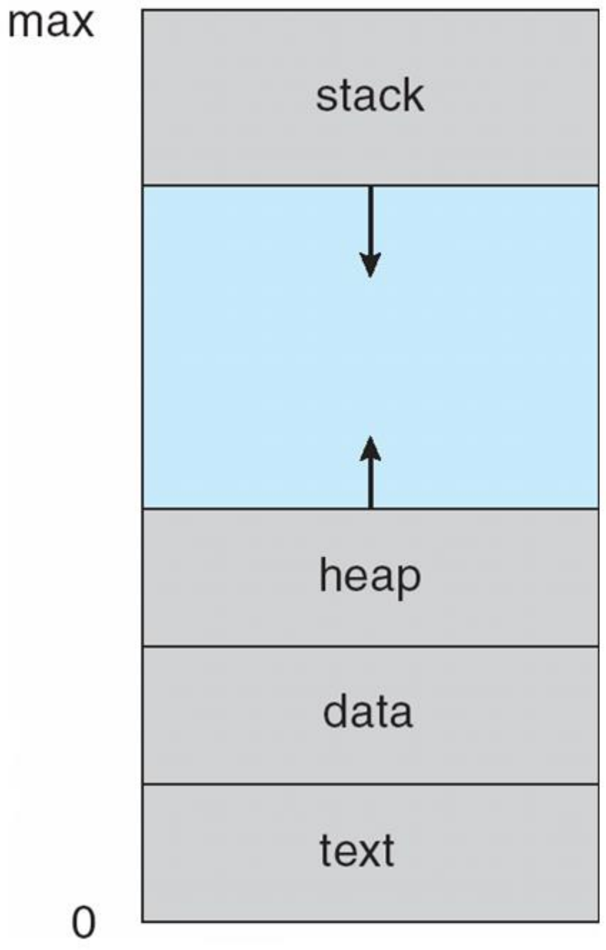
- A process has a **Program Counter**.
  - It’s a register entry which specifies the next instruction to execute.

- A process needs resources (CPU time, memory, files and I/O devices) to complete the execution successfully.

Process Concept (Cont.)

- A process consists of 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

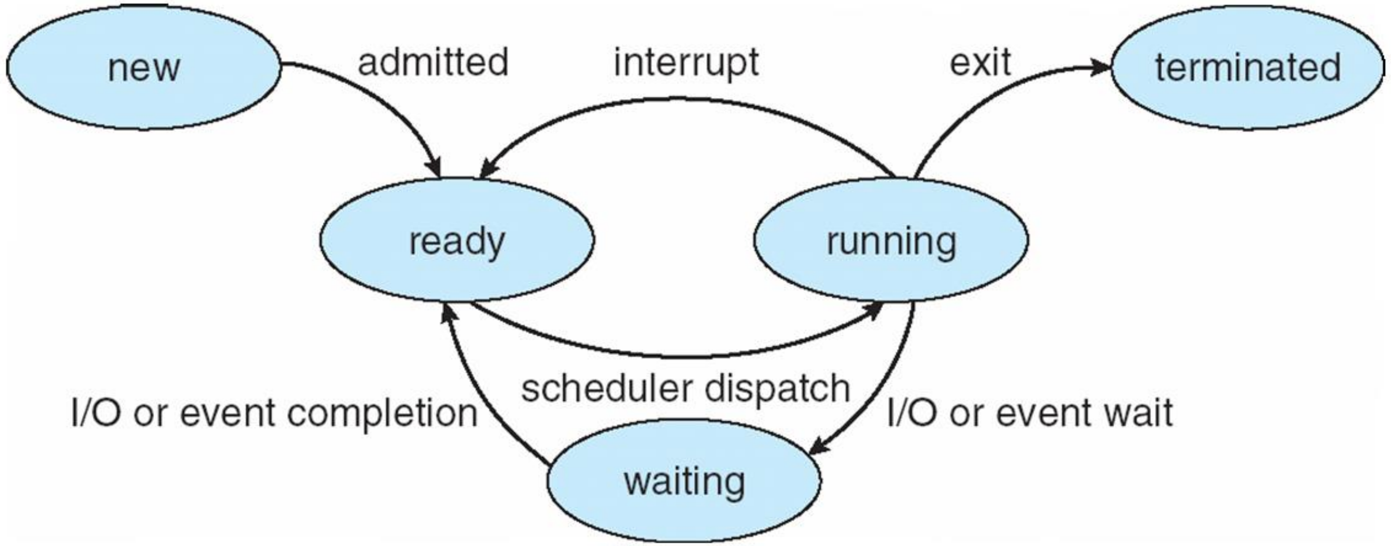


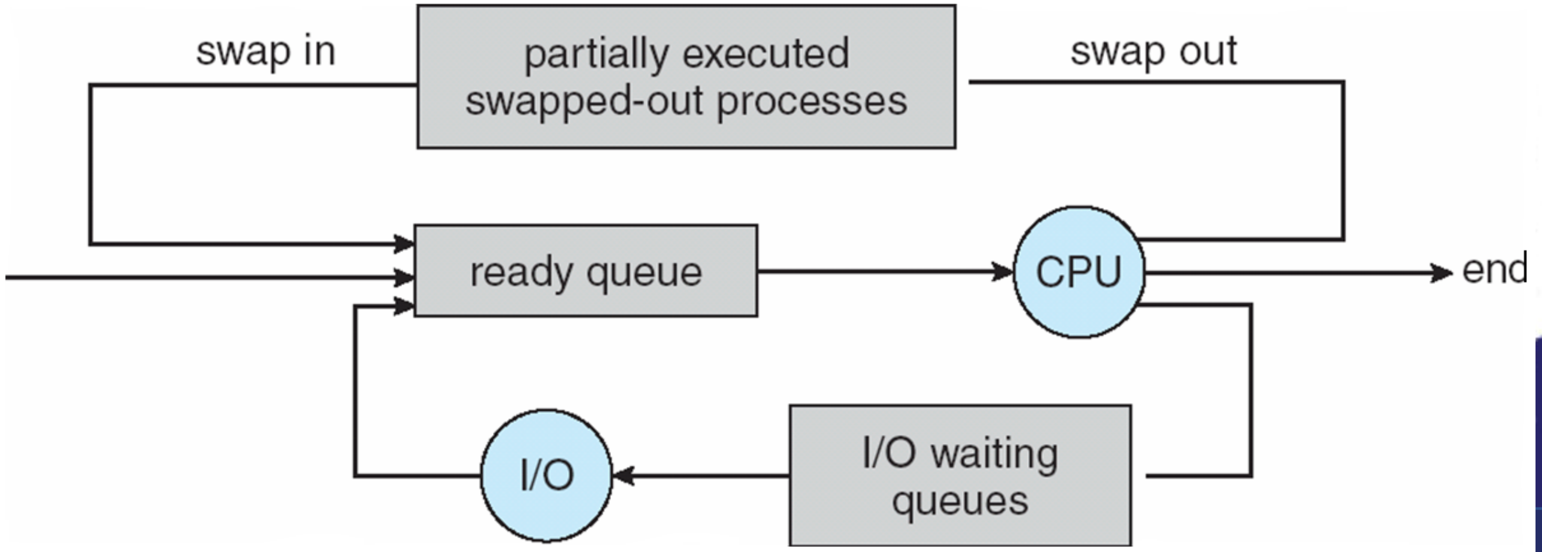
Process State

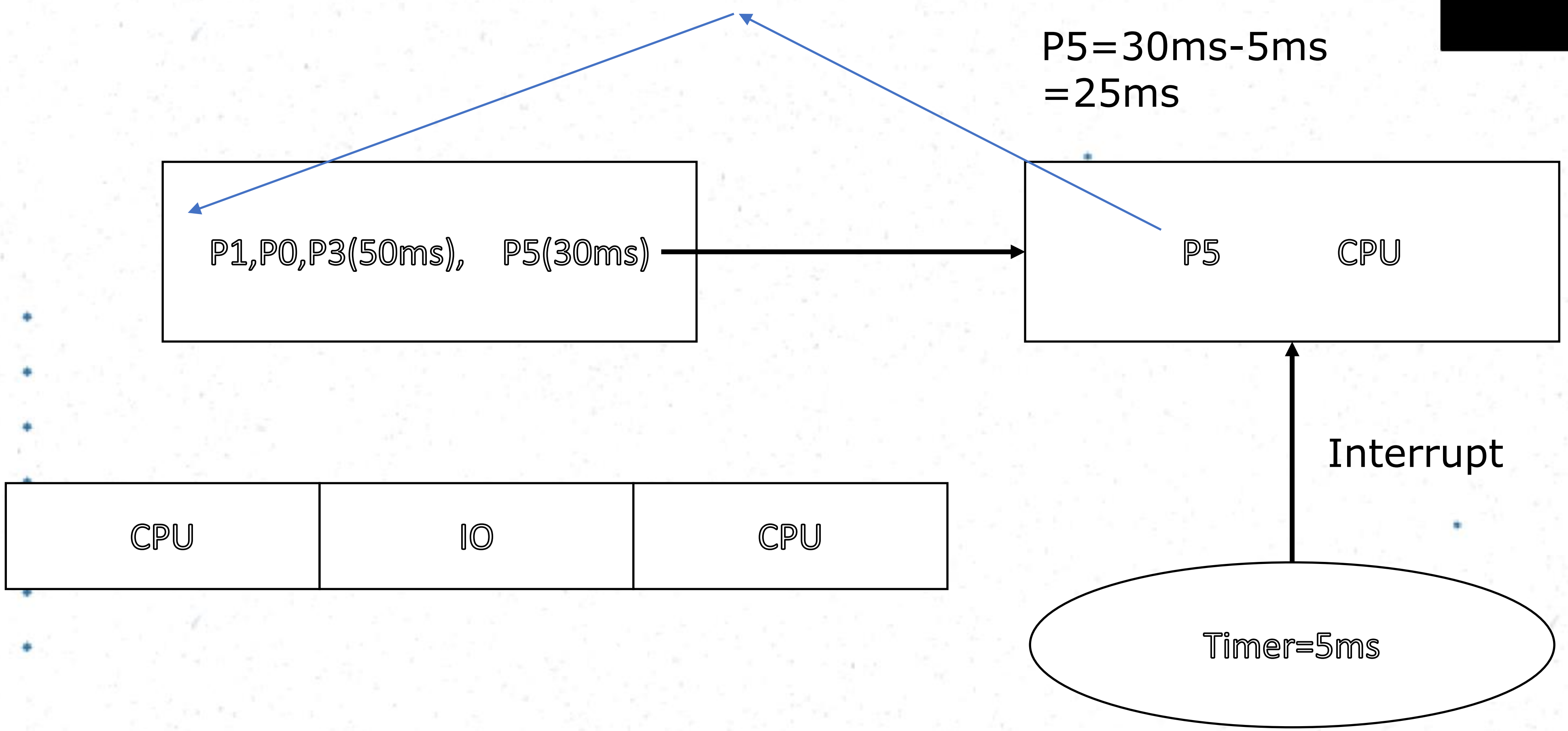
- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process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

Diagram of Process State

[redacted]

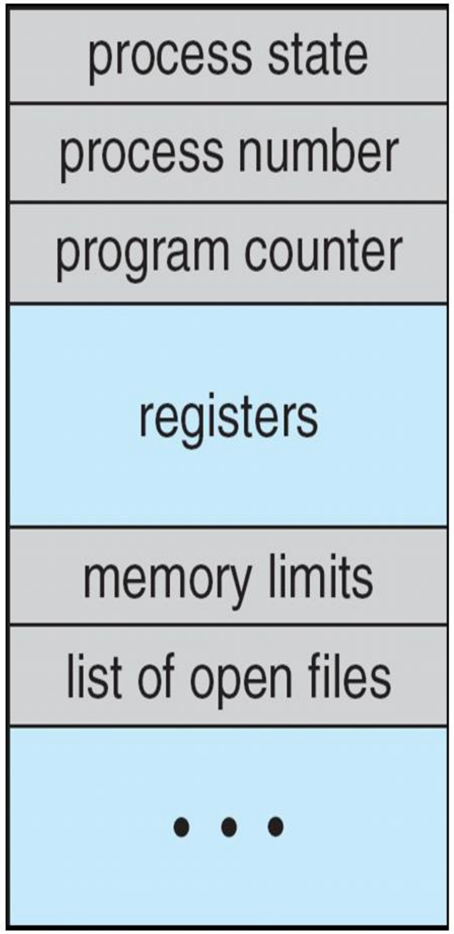


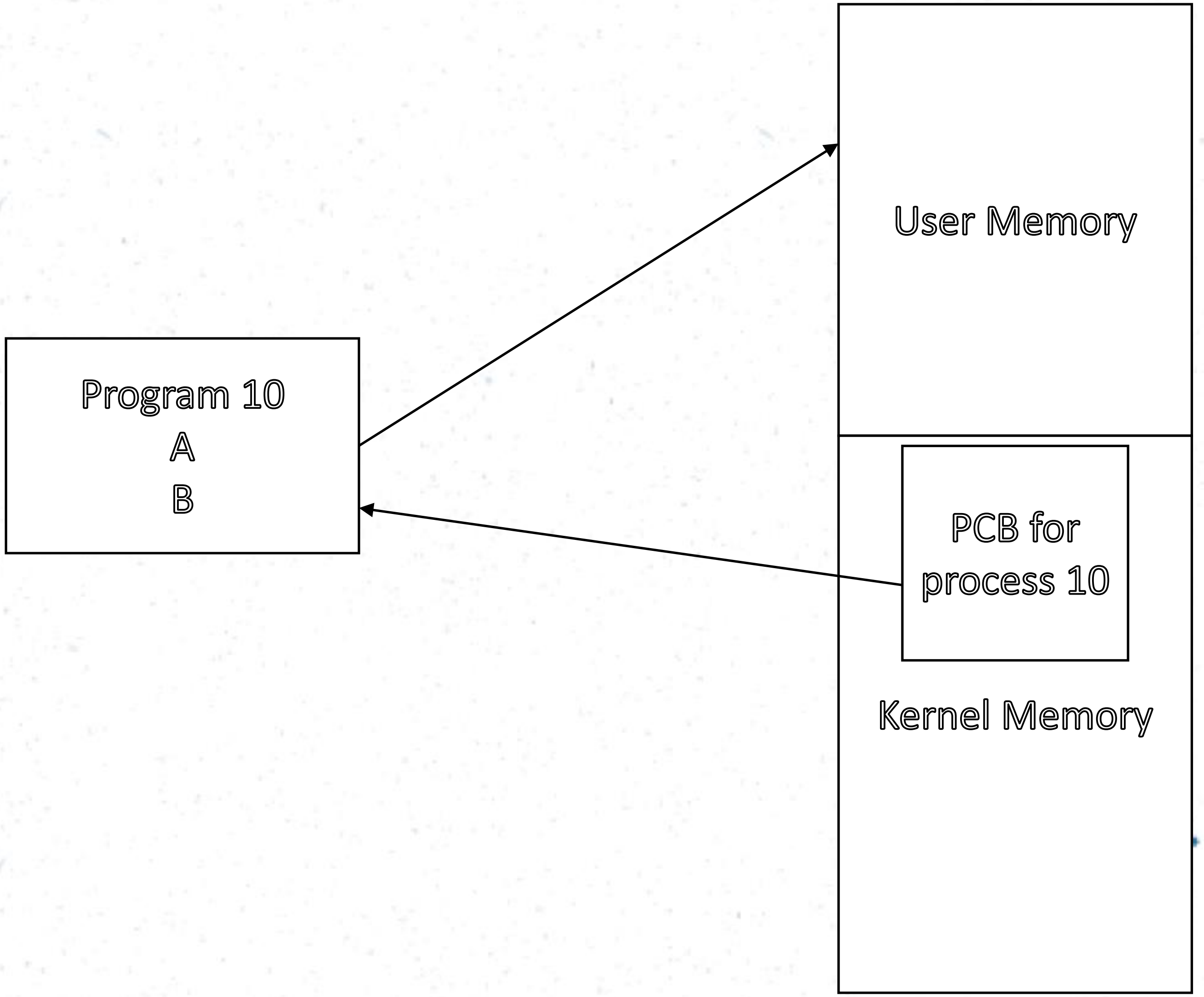



P5=30ms-5ms
=25ms
P1,P0,P3(50ms), P5(30ms)
P5
CPU
Interrupt
Timer=5ms
CPU
IO
CPU

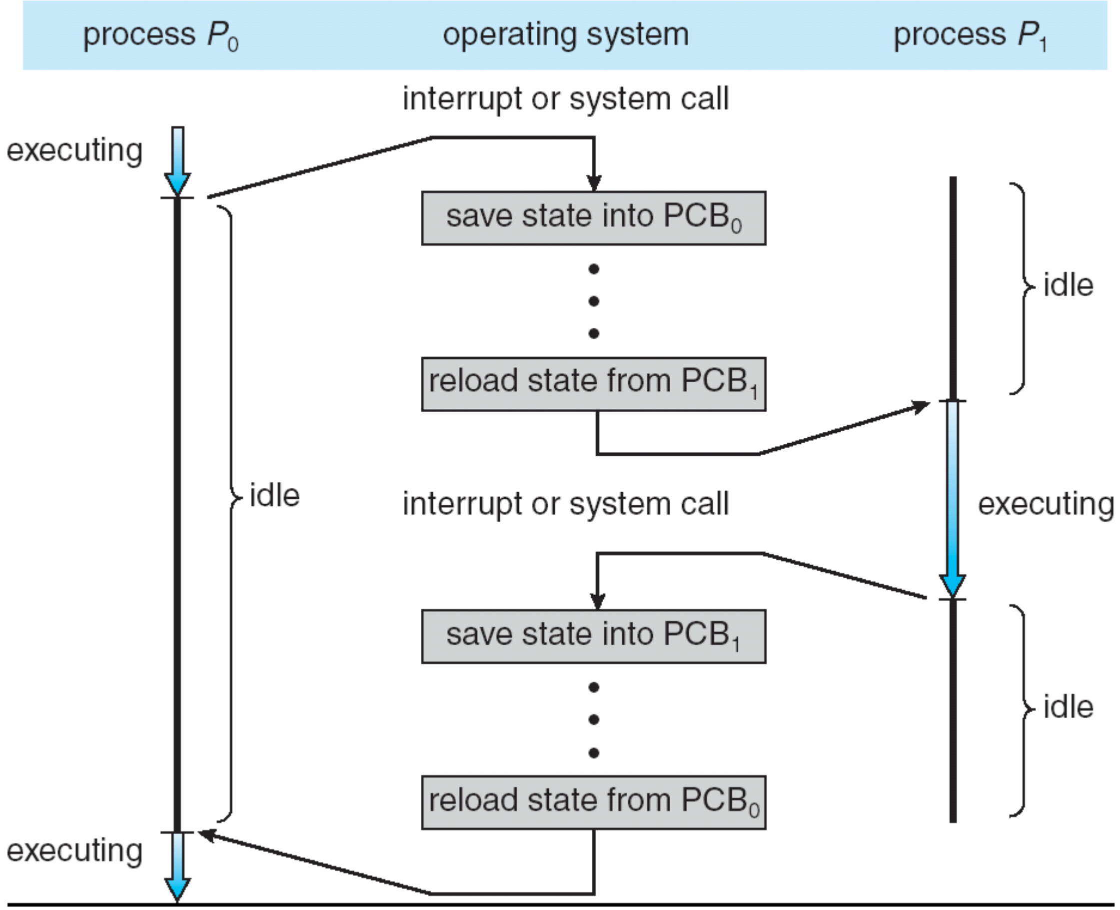
Process Control Block (PCB)

- Information of each process is in PCB
  (also called **task control block**)
- Each PCB contains:
  - Process state – running, waiting, etc
  - Process number (process ID)
  - 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




Program 10
A
B
User Memory
PCB for process 10
Kernel Memory

CPU Switch From Process to Process

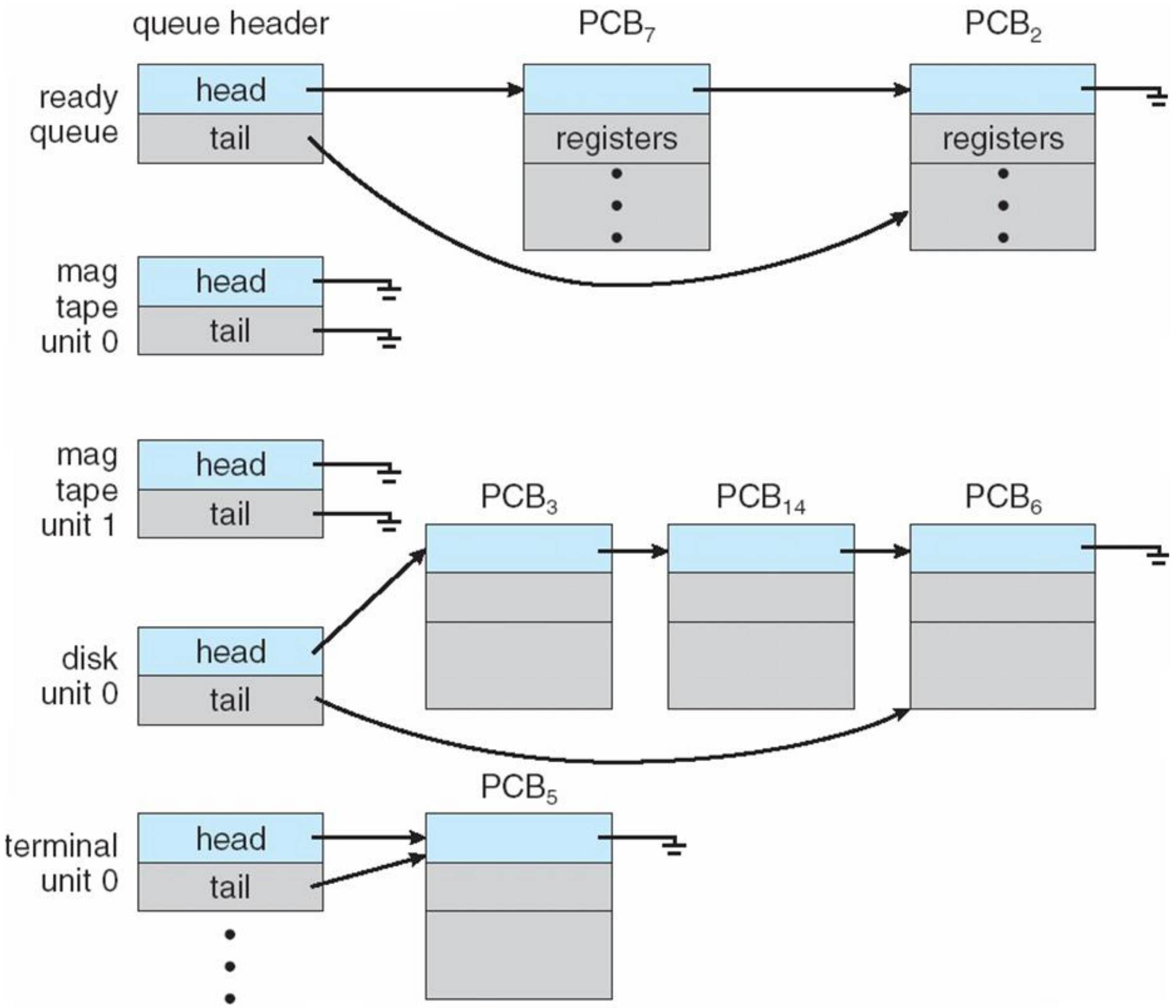


Process Scheduling

- **Process scheduler** selects among available processes for next execution on CPU
  - Scheduler in multiprogramming environment maximizes CPU use.
  - In time sharing, it quickly switches processes onto CPU

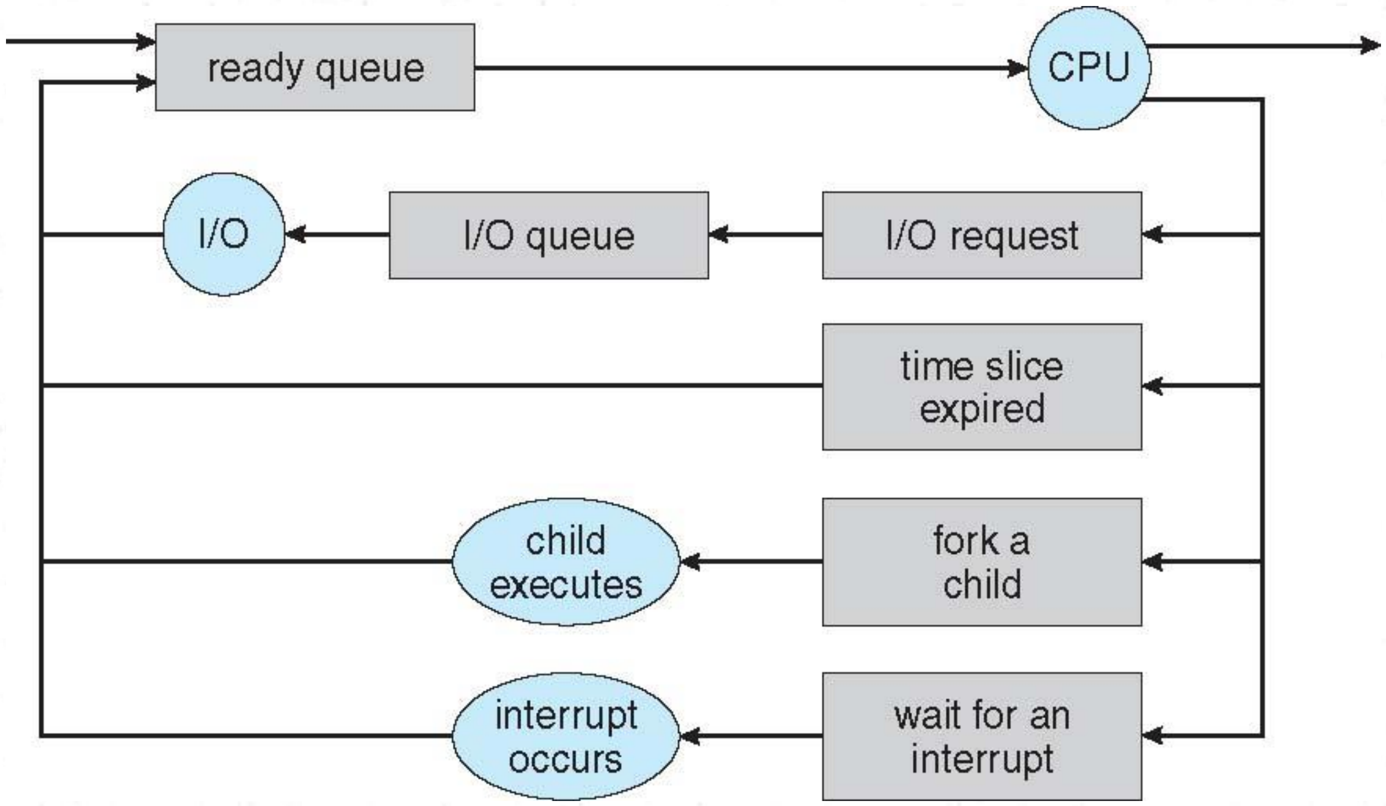
- There are several **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows

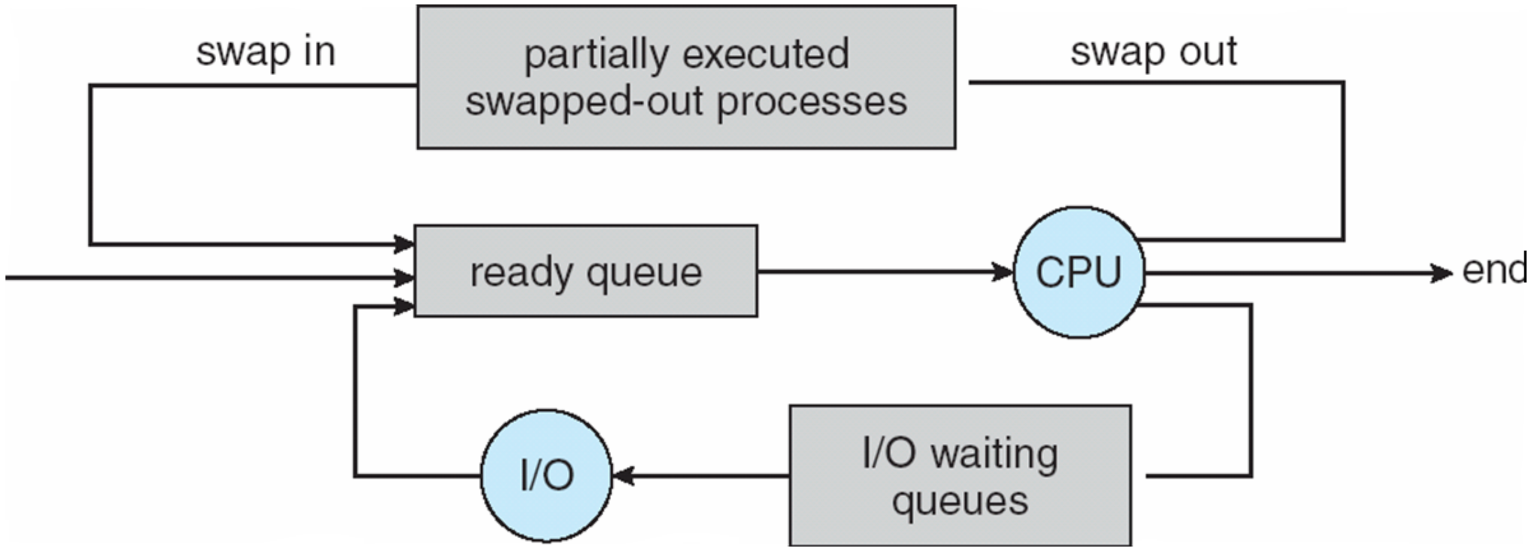


Schedulers

- **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 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***

Addition of Medium Term Scheduling

- **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**



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** of a process represented in the PCB
- 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

Operations on Processes

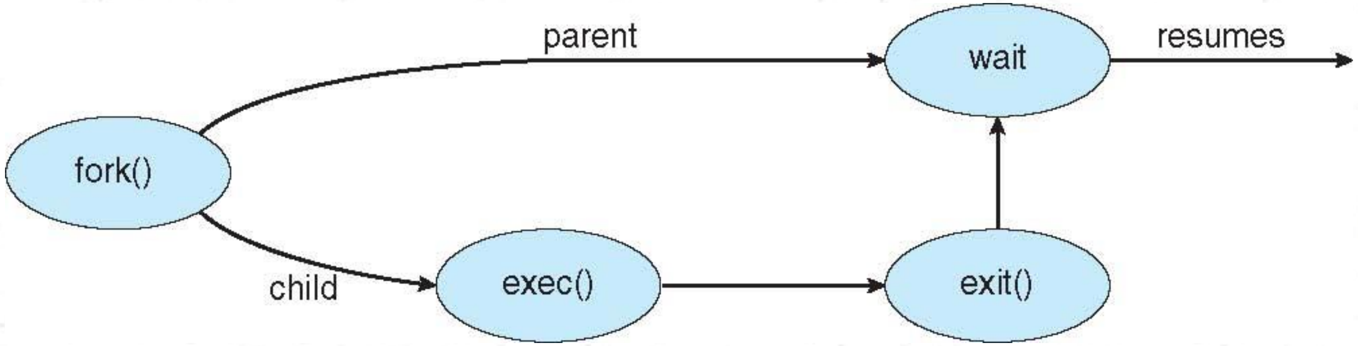
- System must provide mechanisms for:
  - process creation,
  - process termination,

Process Creation

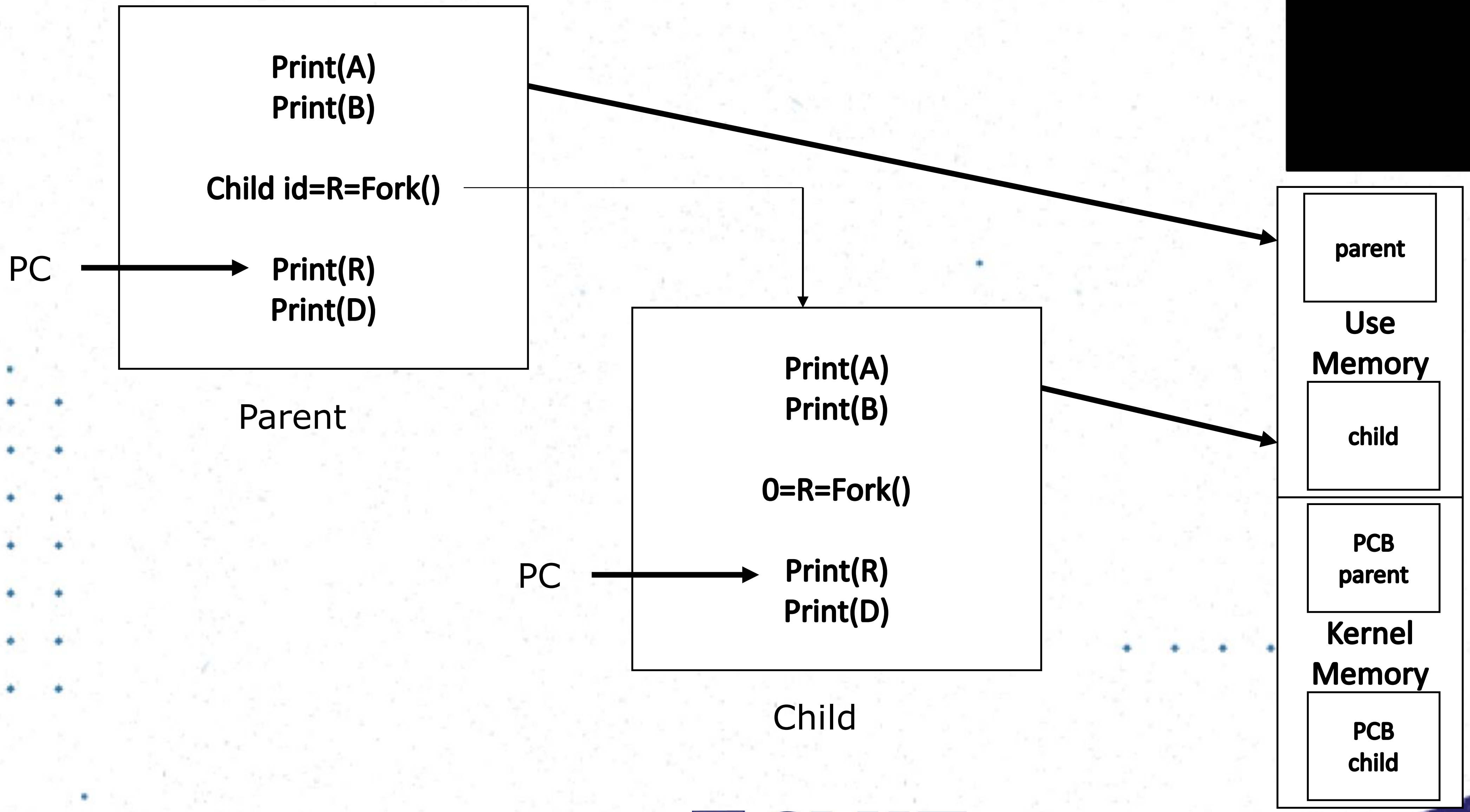
- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - 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



[redacted]


Print(A)
Print(B)
Child id=R=Fork()
PC
Print(R)
Print(D)
Parent
Print(A)
Print(B)
0=R=Fork()
PC
Print(R)
Print(D)
Child
parent
Use
Memory
child
PCB
parent
Kernel
Memory
PCB
child

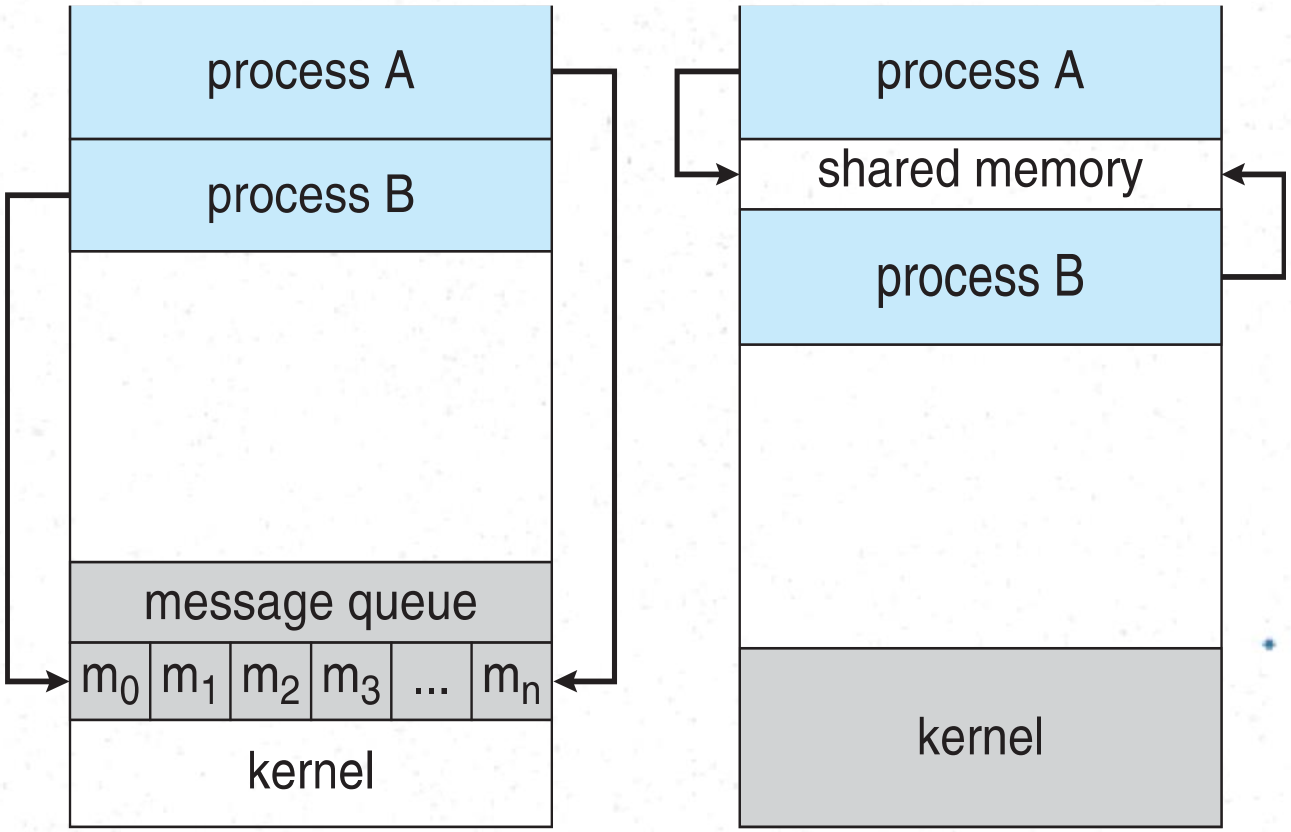
Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data

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

Communications Models

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



(a) (b)

IPC through shared memory

Inter Process Communication through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel

System call in SM

ftok(): is use to generate a unique key.

shmget(): int shmget(key_t,size_tsize,intshmflg); upon successful completion, shmget() returns an identifier for the shared memory segment.

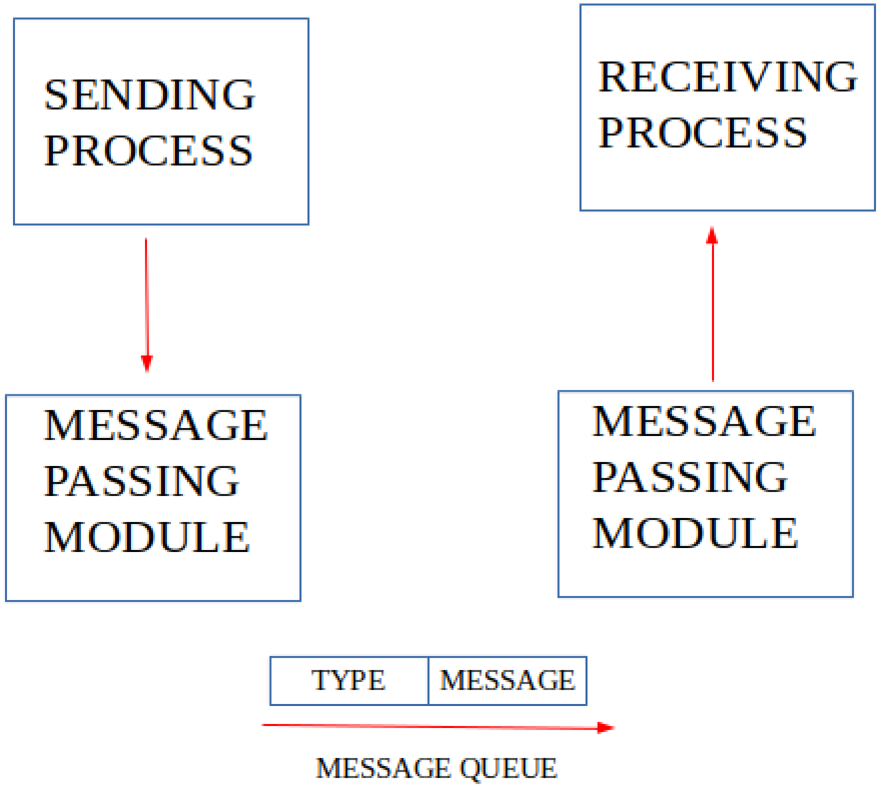
shmat(): Before you can use a shared memory segment, you have to attach yourself
to it using shmat(). void *shmat(int shmid ,void *shmaddr ,int shmflg);
shmid is shared memory id. shmaddr specifies specific address to use but we should set
it to zero and OS will automatically choose the address.

shmdt(): When you're done with the shared memory segment, your program should
detach itself from it using shmdt(). int shmdt(void *shmaddr);

shmctl(): when you detach from shared memory,it is not destroyed. So, to destroy
shmctl() is used. shmctl(int shmid,IPC_RMID,NULL);

IPC using Message Queues

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by msgget().



System cals in MQ

ftok(): is use to generate a unique key.

msgget(): either returns the message queue identifier for a newly created message
queue or returns the identifiers for a queue which exists with the same key value.

msgsnd(): Data is placed on to a message queue by calling msgsnd().

msgrcv(): messages are retrieved from a queue.

msgctl(): It performs various operations on a queue. Generally it is use to
destroy message queue.

Pipes

Oldest (and perhaps simplest) form of UNIX IPC
Half duplex.
The oldest mechanism for IPC in Unix is pipes.

Here's the prototype:

```
#include <unistd.h>

int pipe(int fildes[2]);
```

Pipe takes an array of two ints (two file descriptors) and, if the kernel succeeds in creating the pipe, it puts the file descriptor for the reading end of the pipe in the 0th entry

e.g. filedes[0], and it puts the file descriptor for the write end of the pipe in the 1st entry,

e.g. filedes[1]. Pipe returns 0 if successful and -1 otherwise.

Example

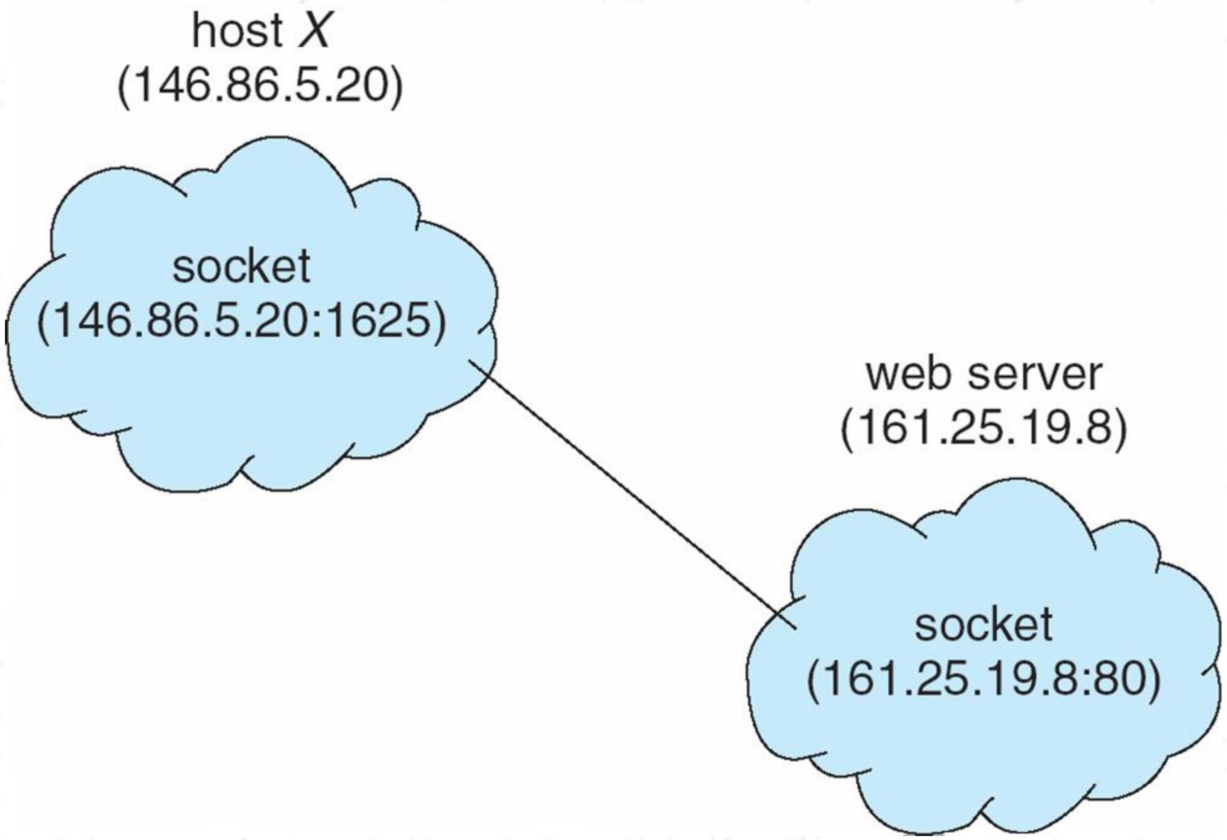
Let's do the example we just did with FIFOs with pipes.
Here it is:

```
#include <unistd.h>

int main()
{
  int pfd[2], fv;
  pipe(pfd);
  fv = fork();
  if (fv)
  {
    close(pfd[0]);
    dup2(pfd[1],STDOUT_FILENO);
    execlp("cat","cat",NULL);
  }
  else
  {
    close(pfd[1]);
    dup2(pfd[0],STDIN_FILENO);
    execlp("tr","tr"," ","x",NULL);
  }

  return 0;
}
```

Sockets

- A **socket** is defined as an endpoint for communication
- Concatenation of IP address and **port** – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are ***well known***, used for standard services
- Special IP address 127.0.0.1 (**loopback**) to refer to system on which process is running

Socket Communication



U.U.Samantha Rajapaksha
BSc. Eng. (Moratuwa), MSc in IT
Senior Lecturer
Sri Lanka Institute of Information Technology
New Kandy Road,
Malabe, Sri Lanka
Tel:0112-301904
email: samantha.r@sliit.lk
Web: www.sliit.lk