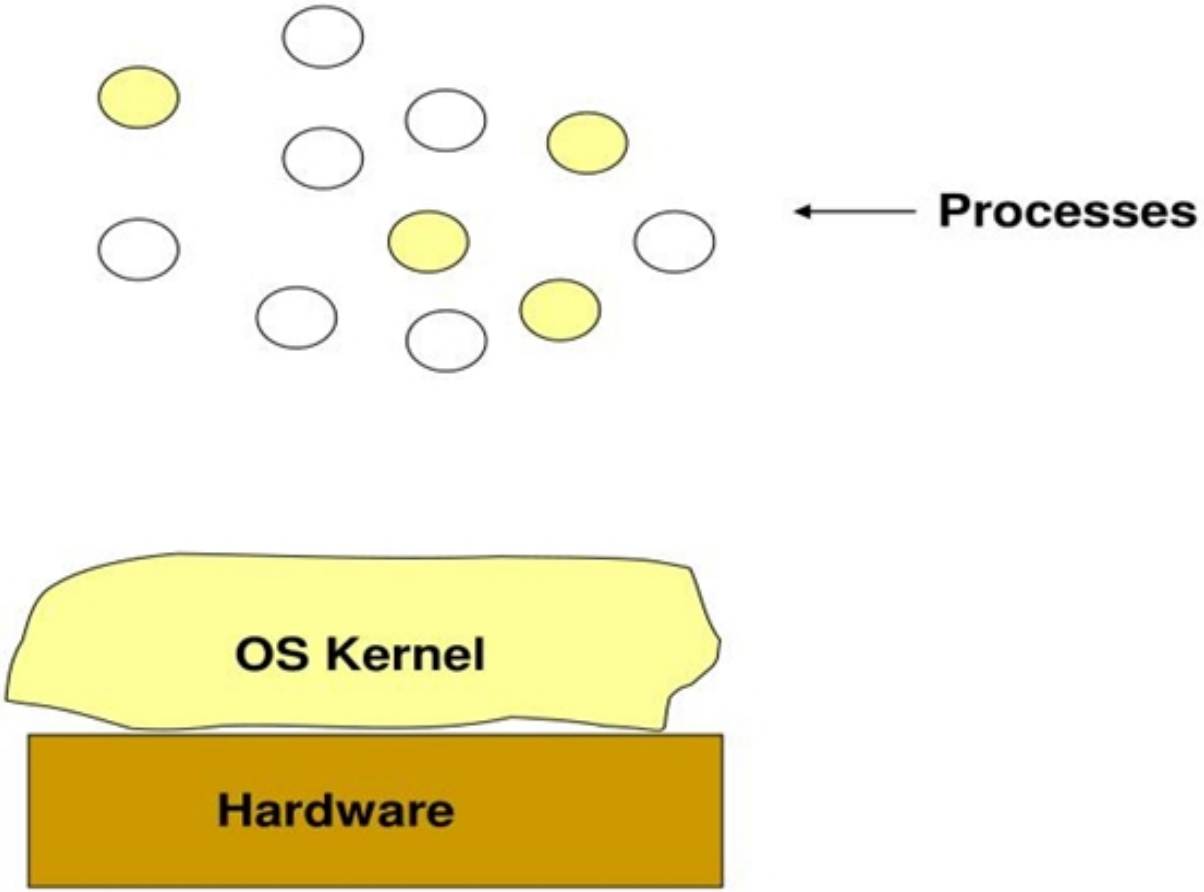
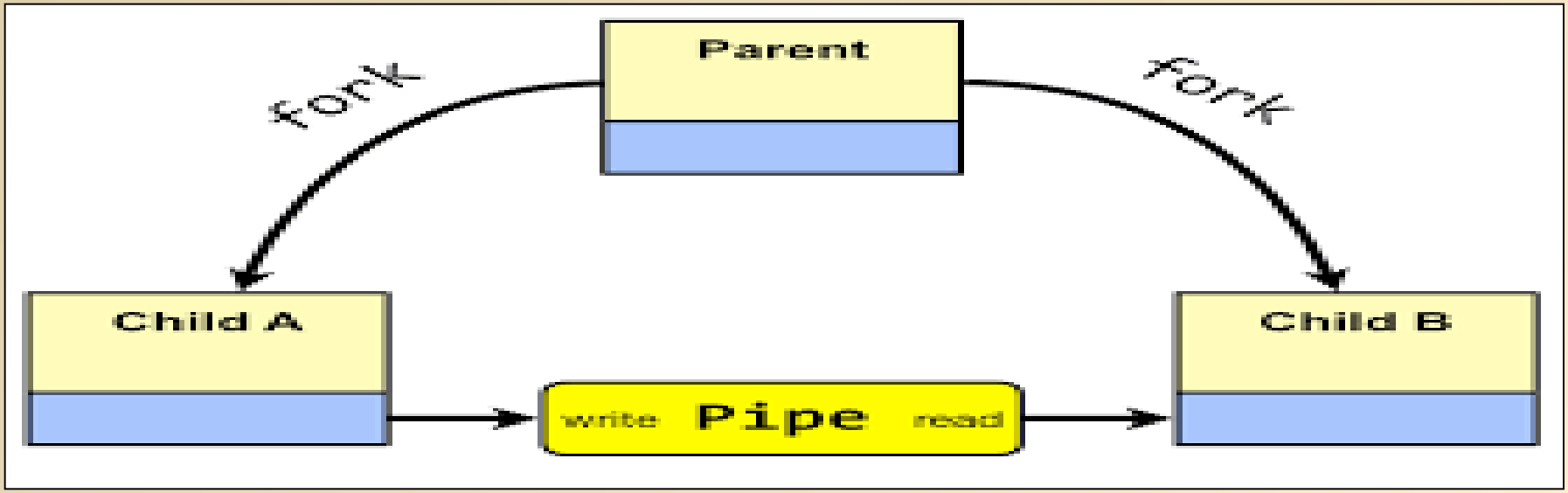
Operating System, Processes, Hardware



Processes

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



Processes

- Example: Special Processes in Unix
  - PID 0 – *Swapper* (i.e., the scheduler)
    - Kernel process
    - No program on disks correspond to this process
  - PID 1 – *init* responsible for bringing up a Unix system after the kernel has been bootstrapped. (/etc/rc* & init or /sbin/rc* & init)
    - User process with superuser privileges
  - PID 2 - pagedaemon responsible for paging
    - Kernel process

Processes

- Process
  - A Basic Unit of Work from the Viewpoint of OS
  - Types:
    - Sequential processes: an activity resulted from the execution of a program by a processor
    - Multi-thread processes
  - An Active Entity
    - Program Code – A Passive Entity
    - Stack and Data Segments
  - The Current Activity
    - PC, Registers , Contents in the Stack and Data Segments

Process Structure

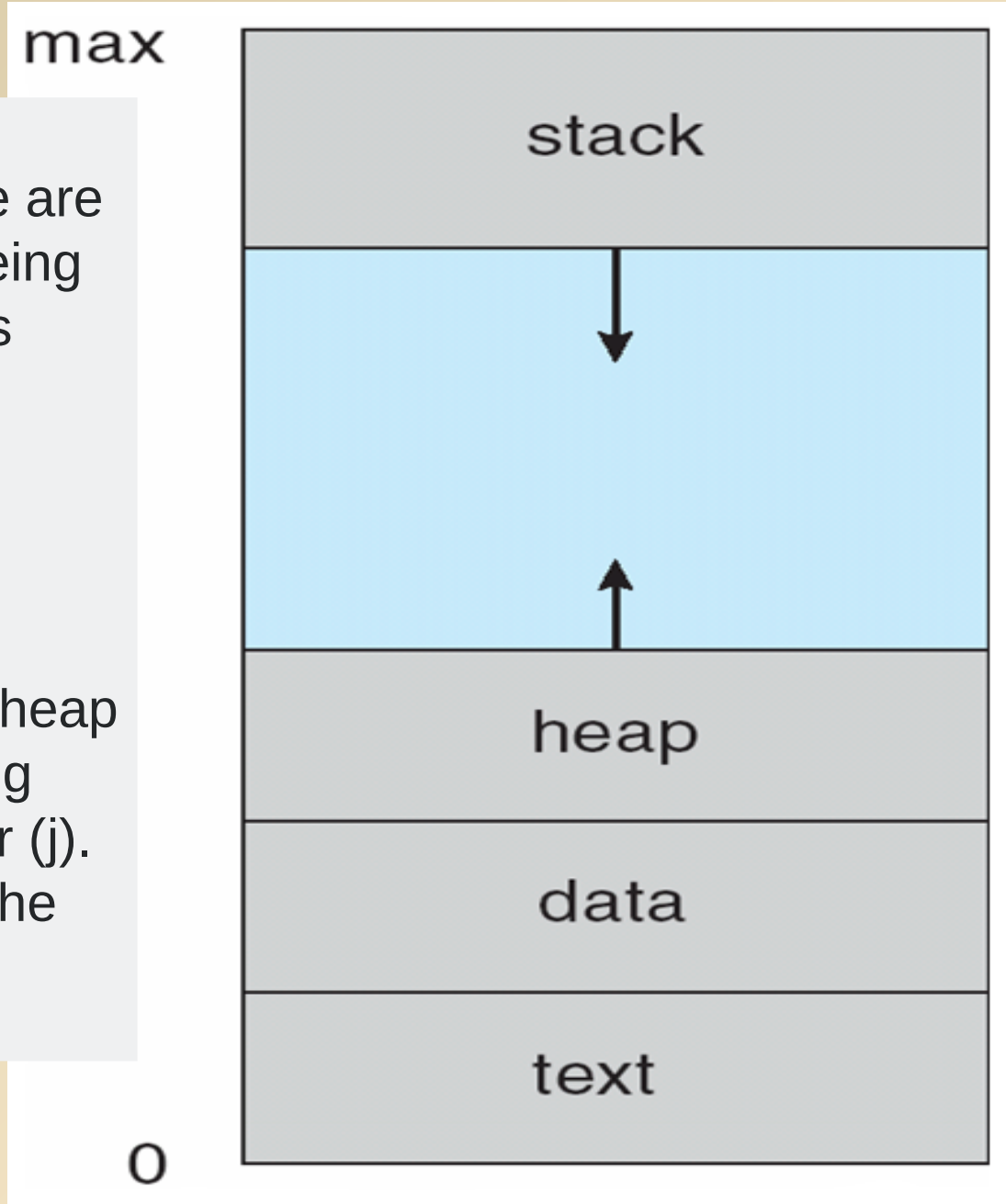
- A process is more than the program code, which is sometimes known as the **text** section.
- It also includes the current activity:
  - The value of the **program counter**
  - The contents of the **processor's registers**.
- It also includes the process **stack**, which contains temporary data (such as function parameters, return addresses, and local variables)
- It also includes the **data section**, which contains global variables.
- It may also include a **heap**, which is memory that is dynamically allocated during process run time.

Process in Memory

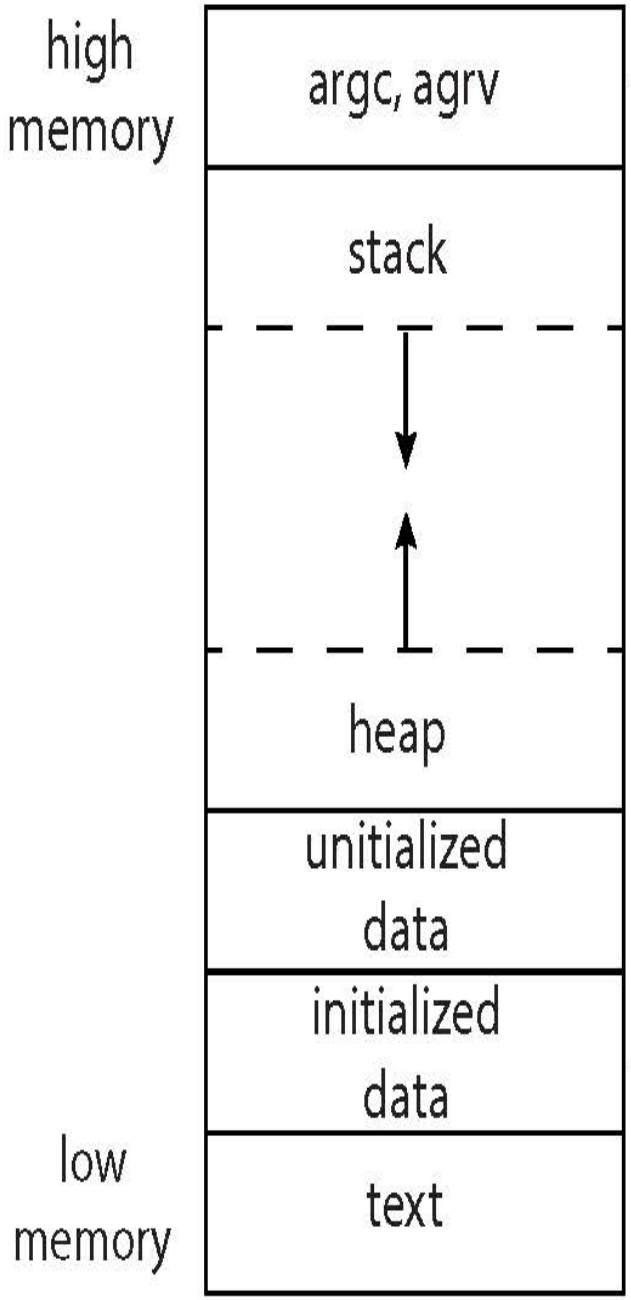
When I allocate something dynamically using malloc, there are actually ***TWO*** pieces of data being stored. The dynamic memory is allocated on the heap, and the pointer itself is allocated on the stack. So in this code:

```
int* j = malloc(sizeof(int));
```

This is allocating space on the heap for an integer. It's also allocating space on the stack for a pointer (j). The variable j's value is set to the address returned by malloc.



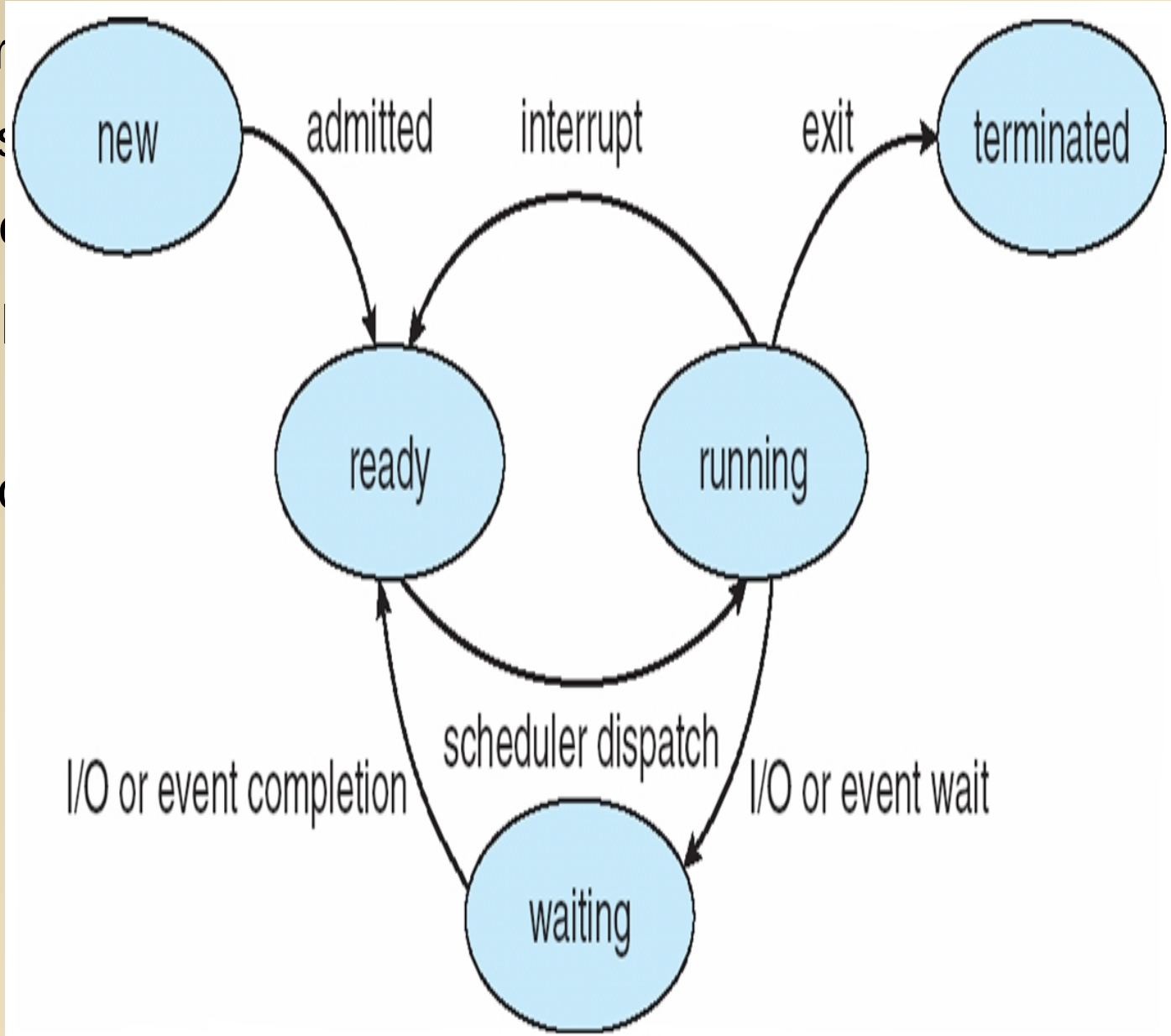
Memory Layout




```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

Process State

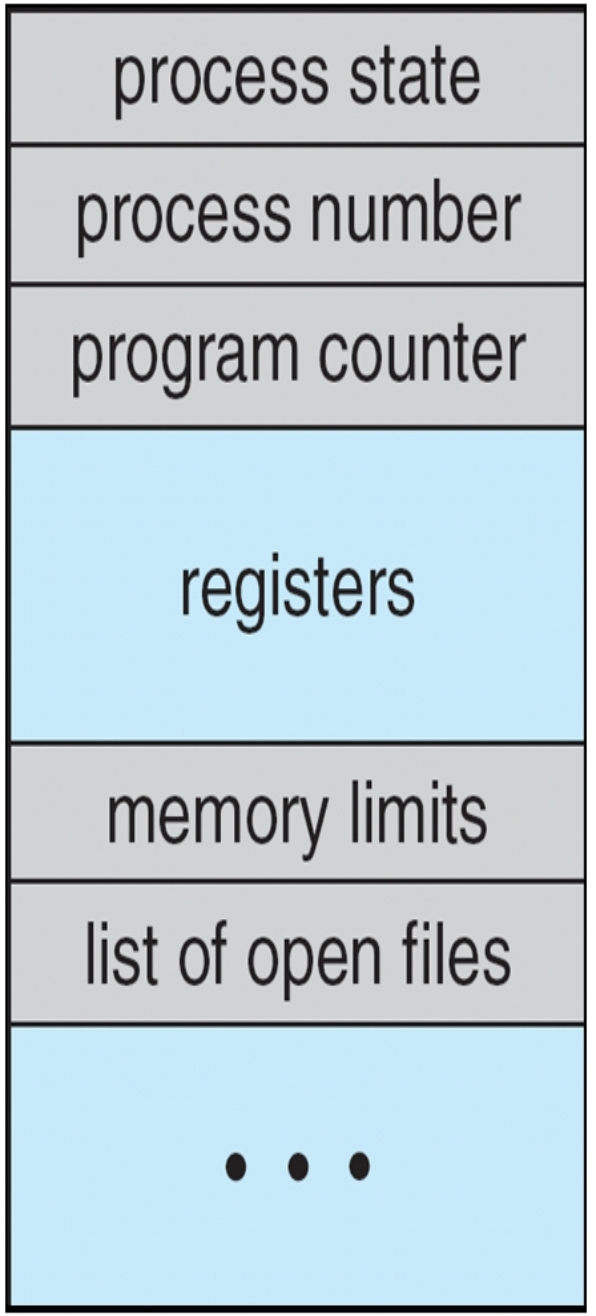
- As a process executes, it changes **state**
  - **new**: The pr
  - **running**: Ins
  - **waiting**: Th
  - **ready**: The
  - **terminated**:
- Diagram of Proc



Process Control Block (PCB)

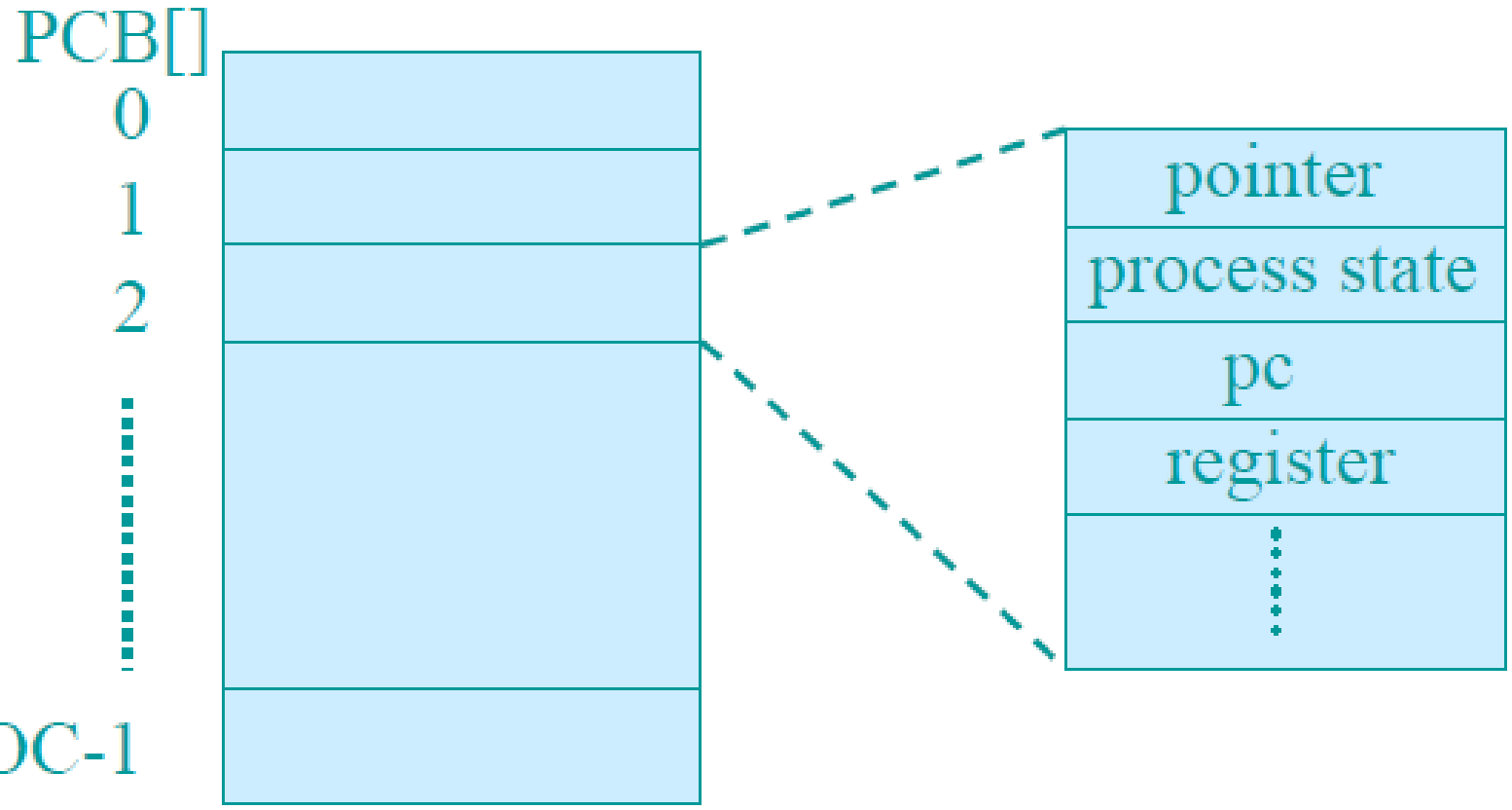
Information associated with each process (also called **task control block**)

- 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 info.-CPU used, clock time elapsed since start,time limits
- I/O status info.–I/O devices allocated to process,list of open files



Processes

- PCB: The repository for any information that may vary from process to process



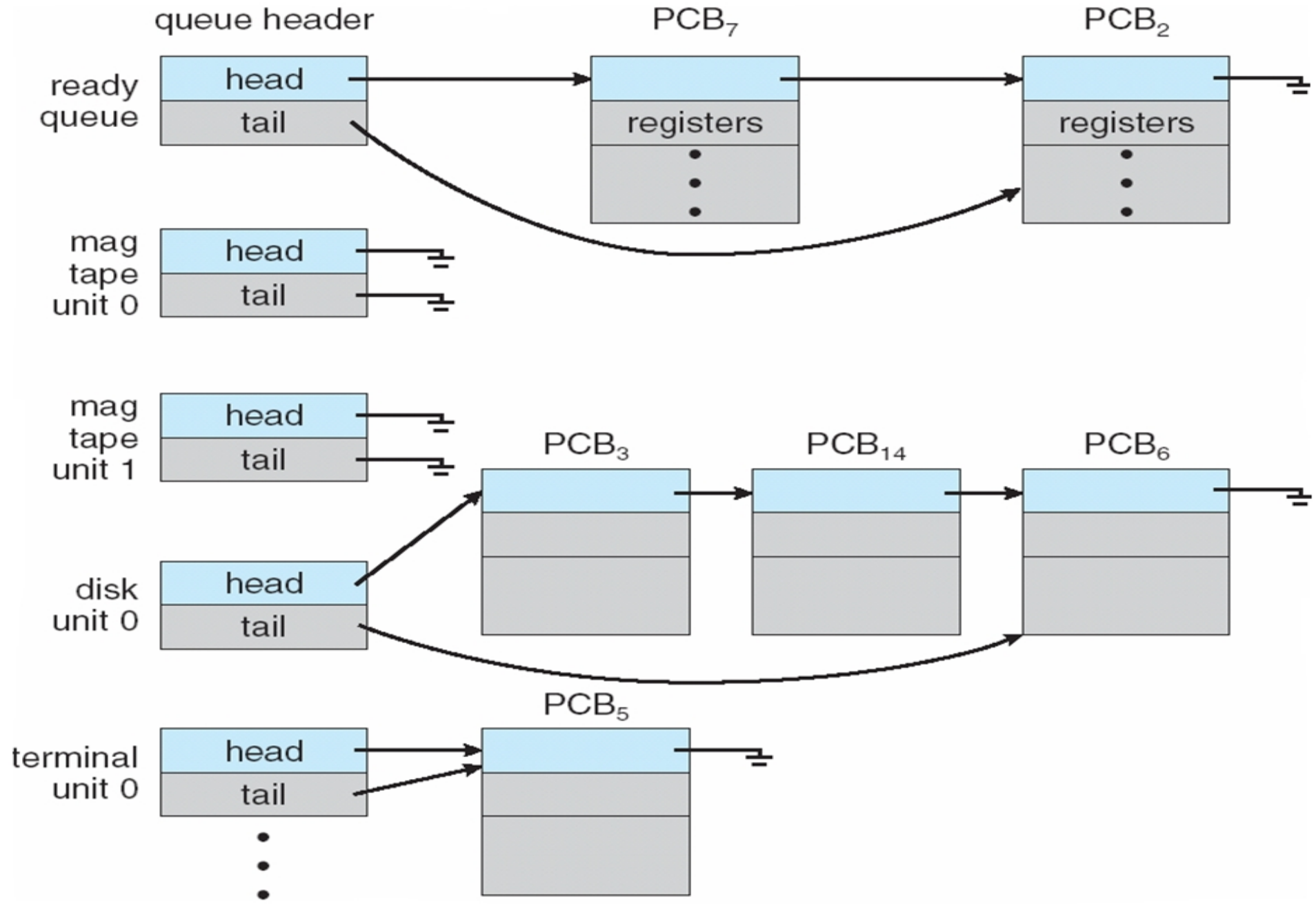
Process Scheduling

- Maximize CPU use
  - Quickly switch processes onto CPU for time sharing
- Process “gives” up then CPU under two conditions:
  - I/O request
  - After N units of time have elapsed (need a timer)
- Once a process gives up the CPU it is added to the “ready queue”
- **Process scheduler** selects among available processes in the ready queue for next execution on CPU

Scheduling Queues

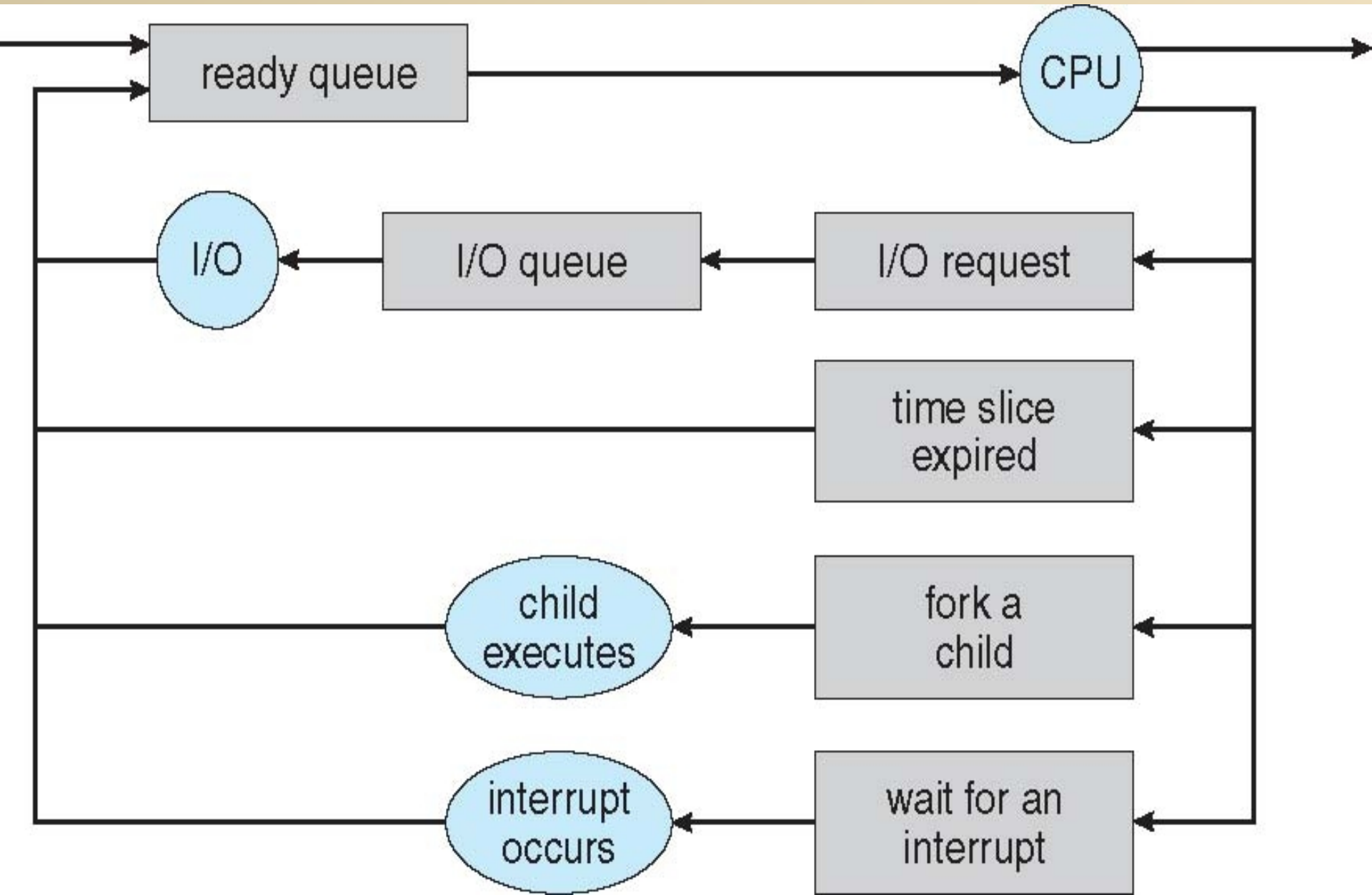
- OS Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the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

Ready Queue And Various I/O Device Queues

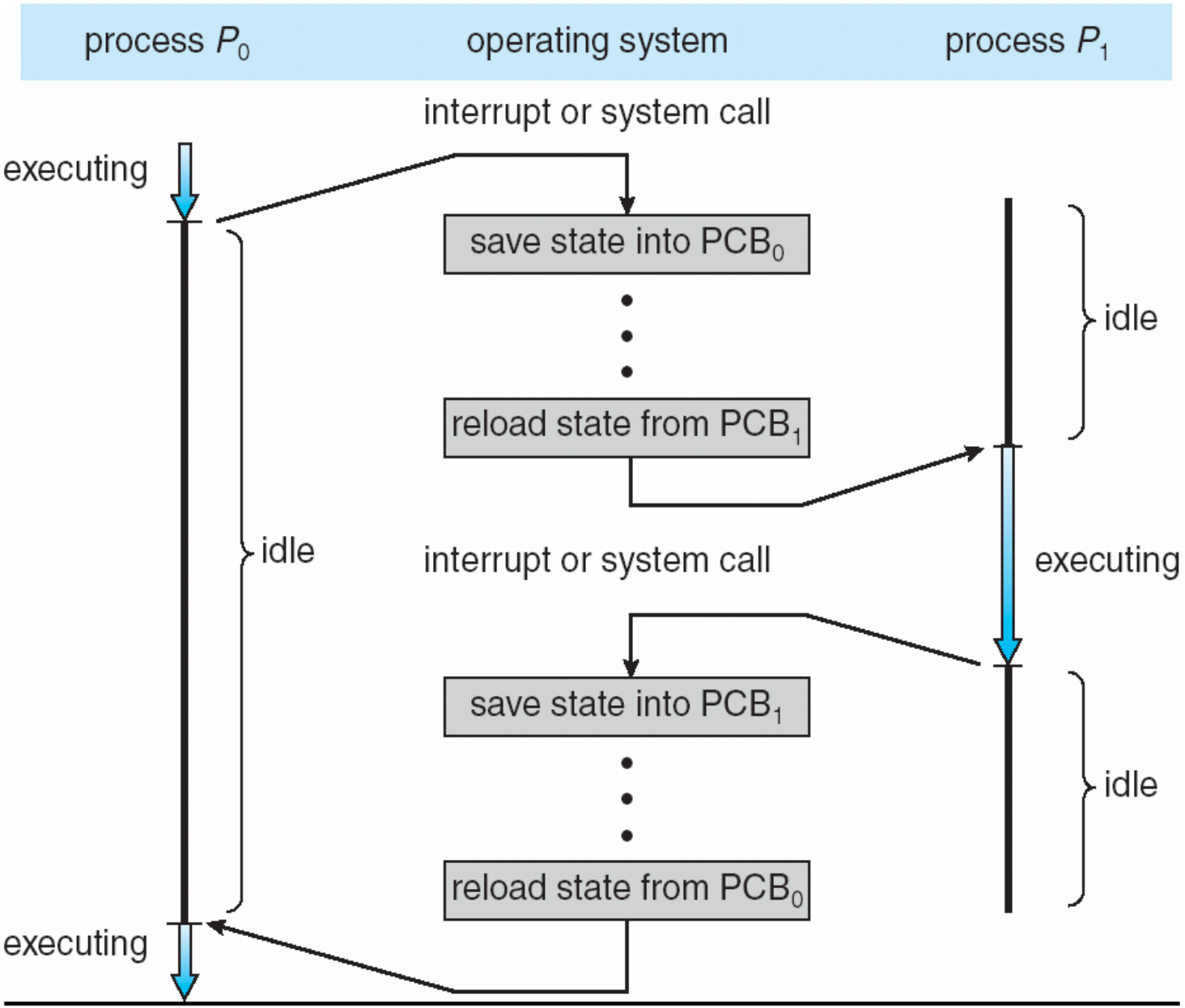


Representation of Process Scheduling

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



CPU Switch From Process to Process



Context Switch

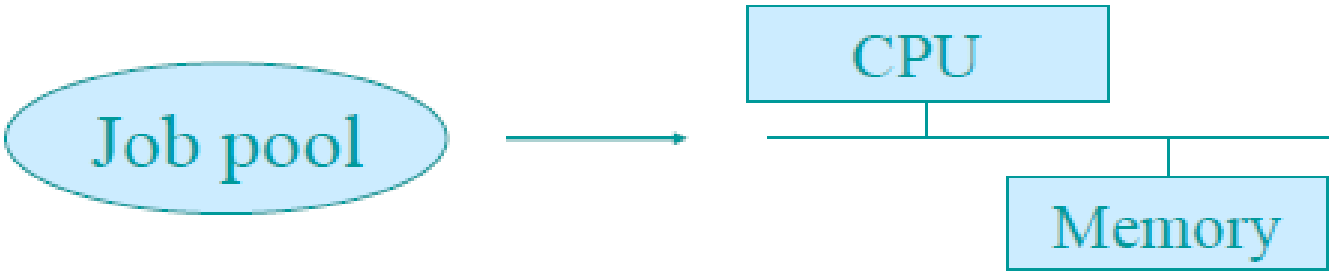
- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is pure overhead; the system does no useful work while switching
  - 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

Schedulers

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

Process Scheduling – Schedulers

- Long-Term (/Job) Scheduler



  - Goal: Select a good mix of I/O-bound and CPU-bound process
  - Remarks :
    1. Control the degree of multiprogramming
    2. Can take more time in selecting processes because of a longer interval between executions
    3. May not exist physically


START

Process Scheduling – Schedulers

- Short-Term (/CPU) Scheduler
  - Goal : Efficiently allocate the CPU to one of the ready processes according to some criteria.
- Mid-Term Scheduler
  - Swap processes in and out memory to control the degree of multiprogramming

Process Scheduling – Context Switches

- Context Switch ~ Pure Overheads
  - Save the state of the old process and load the state of the newly scheduled process.
    - The context of a process is usually reflected in PCB and others, e.g., .u in Unix.
- Issues :
  - The cost depends on hardware support
    - e.g. processes with multiple register sets or computers with advanced memory management.
  - Threads, i.e., light-weight process (LWP), are introduced to break this bottleneck !

Cooperating Processes

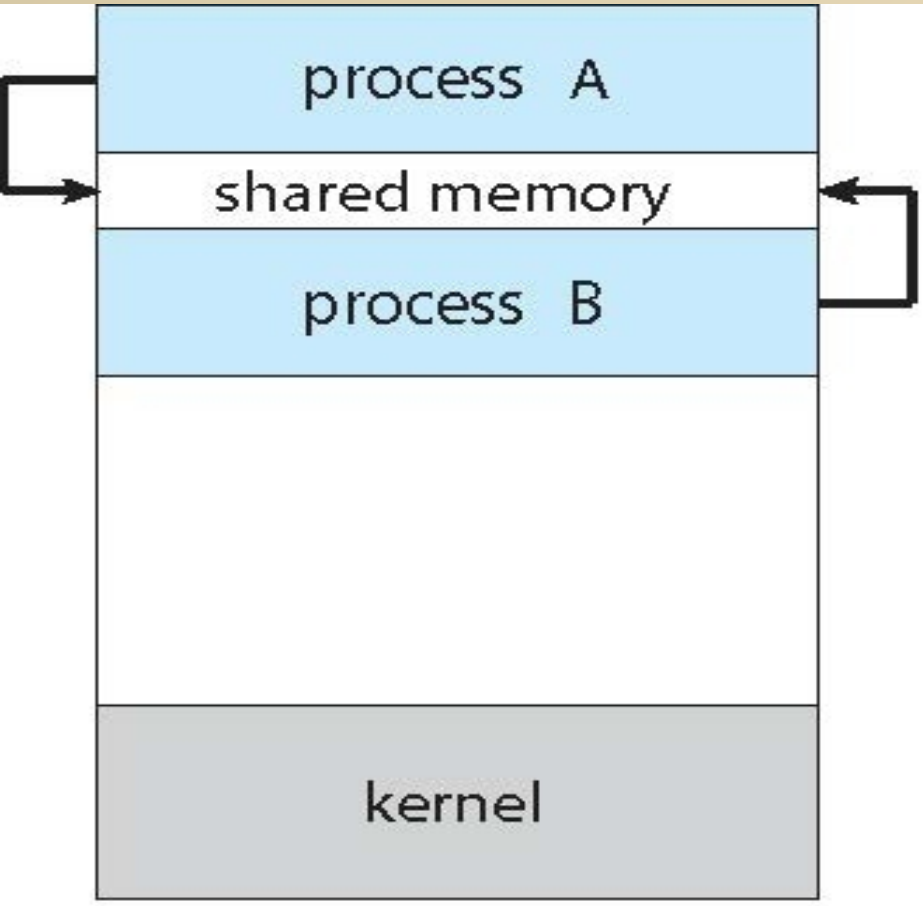
- Cooperating processes can affect or be affected by the other processes
  - Independent Processes
- Reasons:
  - Information Sharing, e.g., files
  - Computation Speedup, e.g., parallelism.
  - Modularity, e.g., functionality dividing
  - Convenience, e.g., multiple work

Interprocess Communication

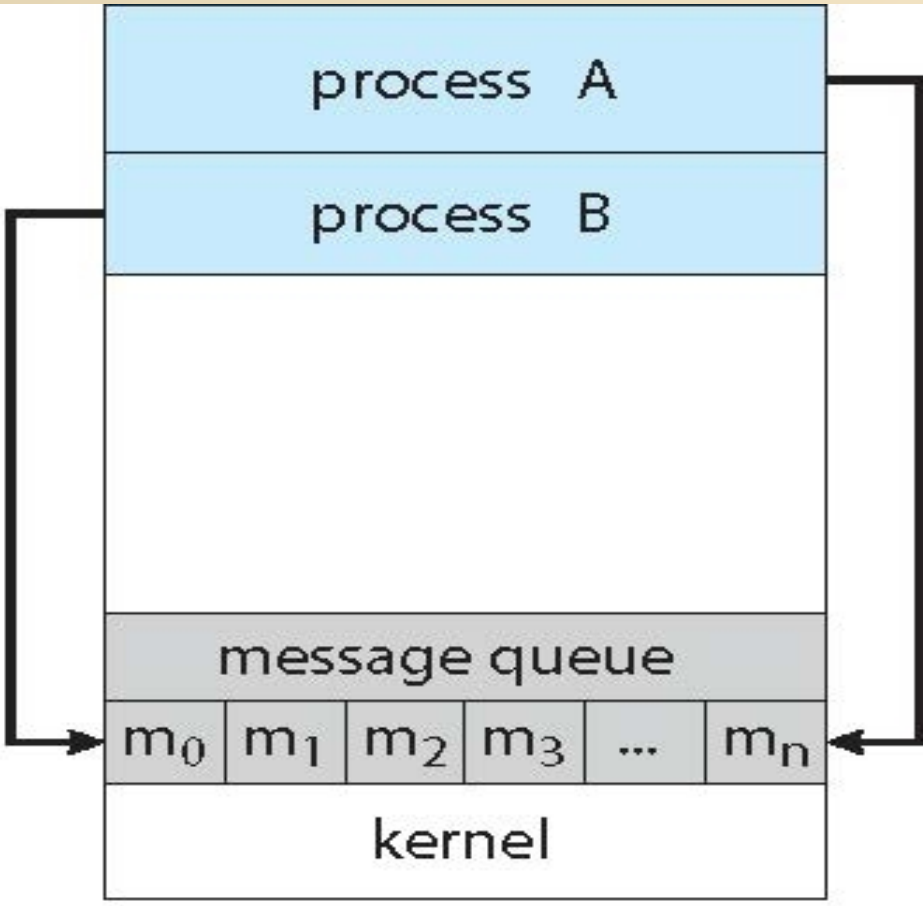
- Why Inter-Process Communication (IPC)?
  - Exchanging of Data and Control Information!
- Why Process Synchronization?
  - Protect critical sections!
  - Ensure the order of executions!

Communications Models

- Two models of IPC
  - **Shared memory**
  - **Message passing**



(a)



(b)

Example of IPC Systems

- There are four different IPC systems.
  - POSIX API for shared memory
  - Mach operating system, which uses message passing
  - Windows IPC, which uses shared memory as a mechanism for providing certain types of message passing.
  - Pipes, one of the earliest IPC mechanisms on UNIX systems.

Ordinary Pipes

- On UNIX systems, ordinary pipes are constructed using the function

  pipe (int fd[]) [pipe system calls….]

- This function creates a pipe that is accessed through the

  int fd[]

  file descriptors:

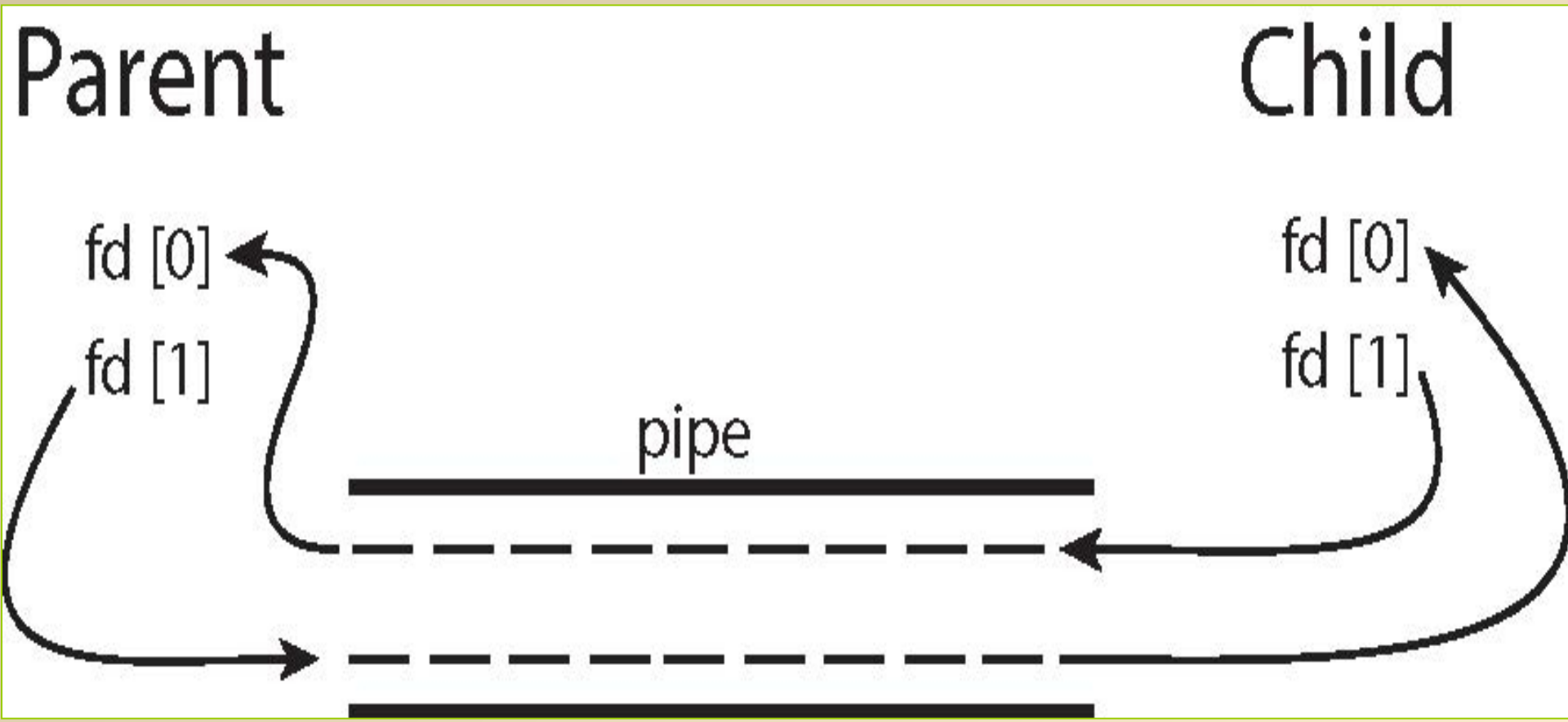
  fd[0] is the read-end of the pipe

  fd[1]} is the write-end of the pipe

- UNIX treats a pipe as a special type of file. Thus, pipes can be accessed using ordinary read() and write() system calls.

- Windows calls these **anonymous pipes**

Figure Pipe



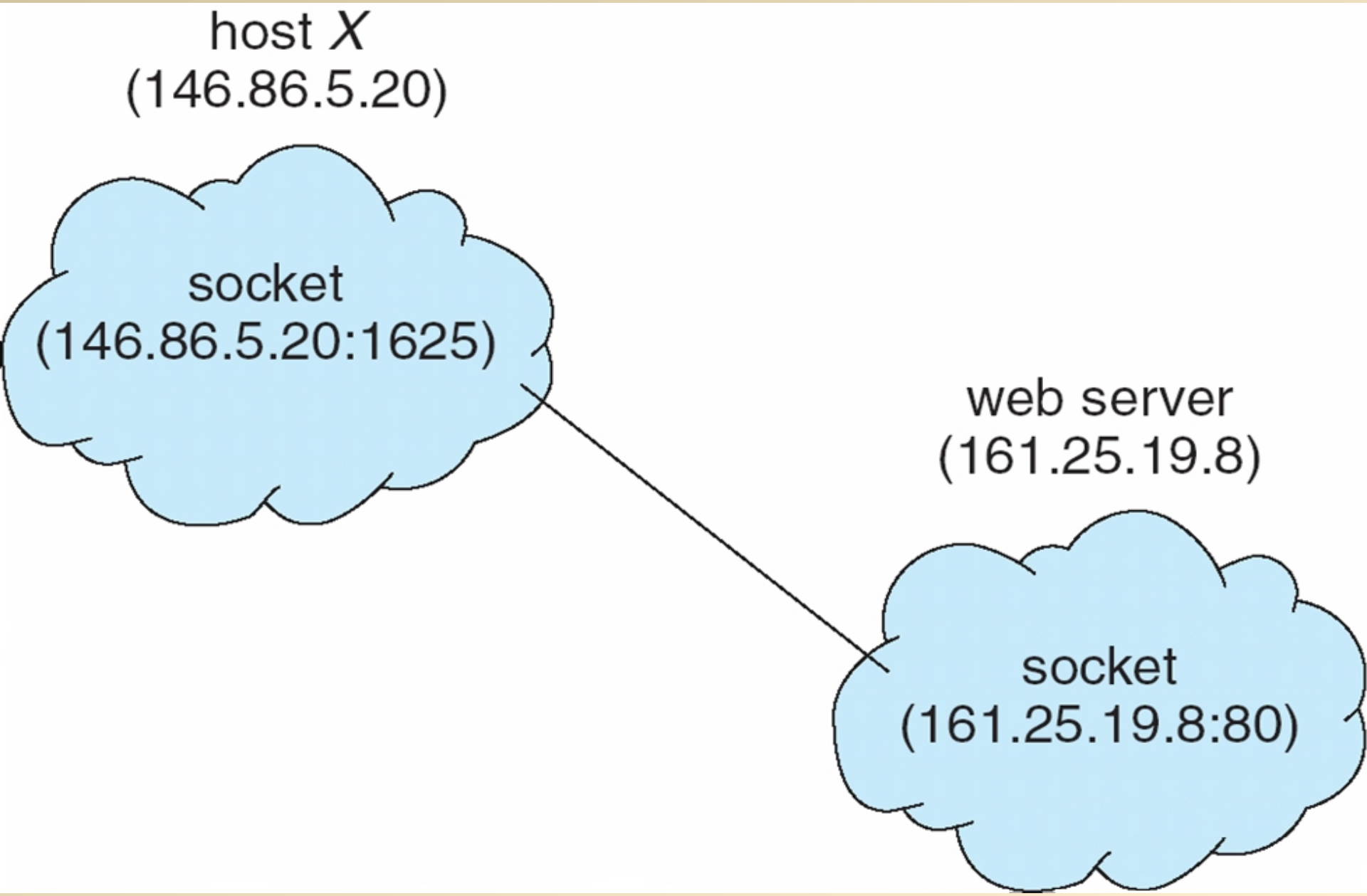
Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)

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**) is used to refer to system on which process is running. That is, when a computer refers to address 127.0.0.1, it is referring to itself.

Socket Communication



Sockets in Java

- Three types of sockets
  - **Connection-oriented** (**TCP**)
  - **Connectionless** (**UDP**)
  - **MulticastSocket class** – data can be sent to multiple recipients
- Consider this “Date” server:

```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            /* write the Date to the socket */
            pout.println(new java.util.Date().toString());

            /* close the socket and resume */
            /* listening for connections */
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Remote Procedure Calls

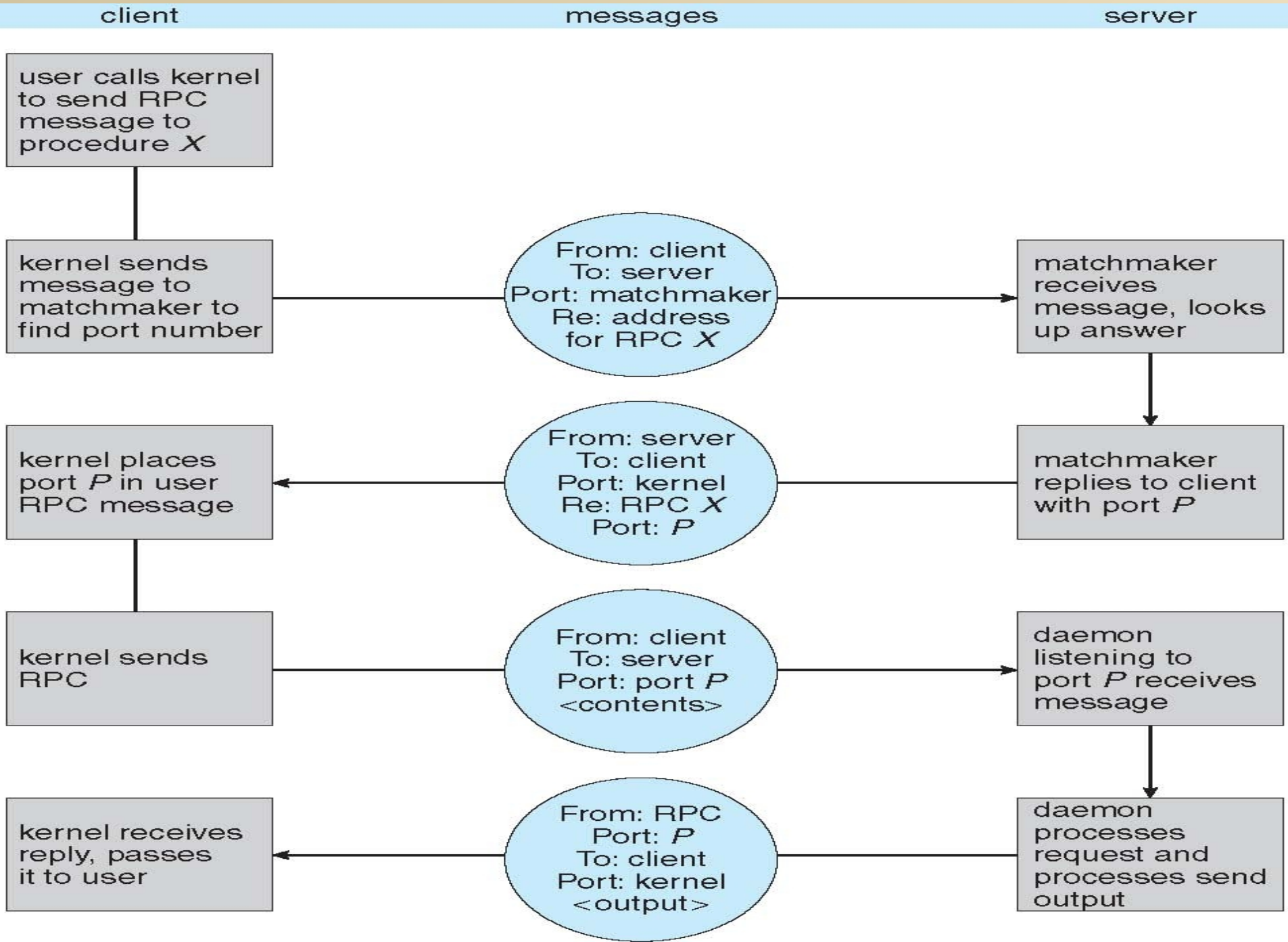
- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
  - Again uses ports for service differentiation
- **Stubs** – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and **marshalls** the parameters (marshalling involves packaging the parameters into a form that can be transmitted over a network).
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server
- On Windows, stub code compile from specification written in **Microsoft Interface Definition Language** (**MIDL**)

Remote Procedure Calls (Cont.)

- Data representation handled via **External Data Representation** (**XDL**) format to account for different architectures.
- Must be dealt with concerns differences in data representation on the client and server machines.
- Consider the representation of 32-bit integers.
  - **Big-endian.** Store the most significant byte first
  - **Little-endian.** Store the least significant byte first.

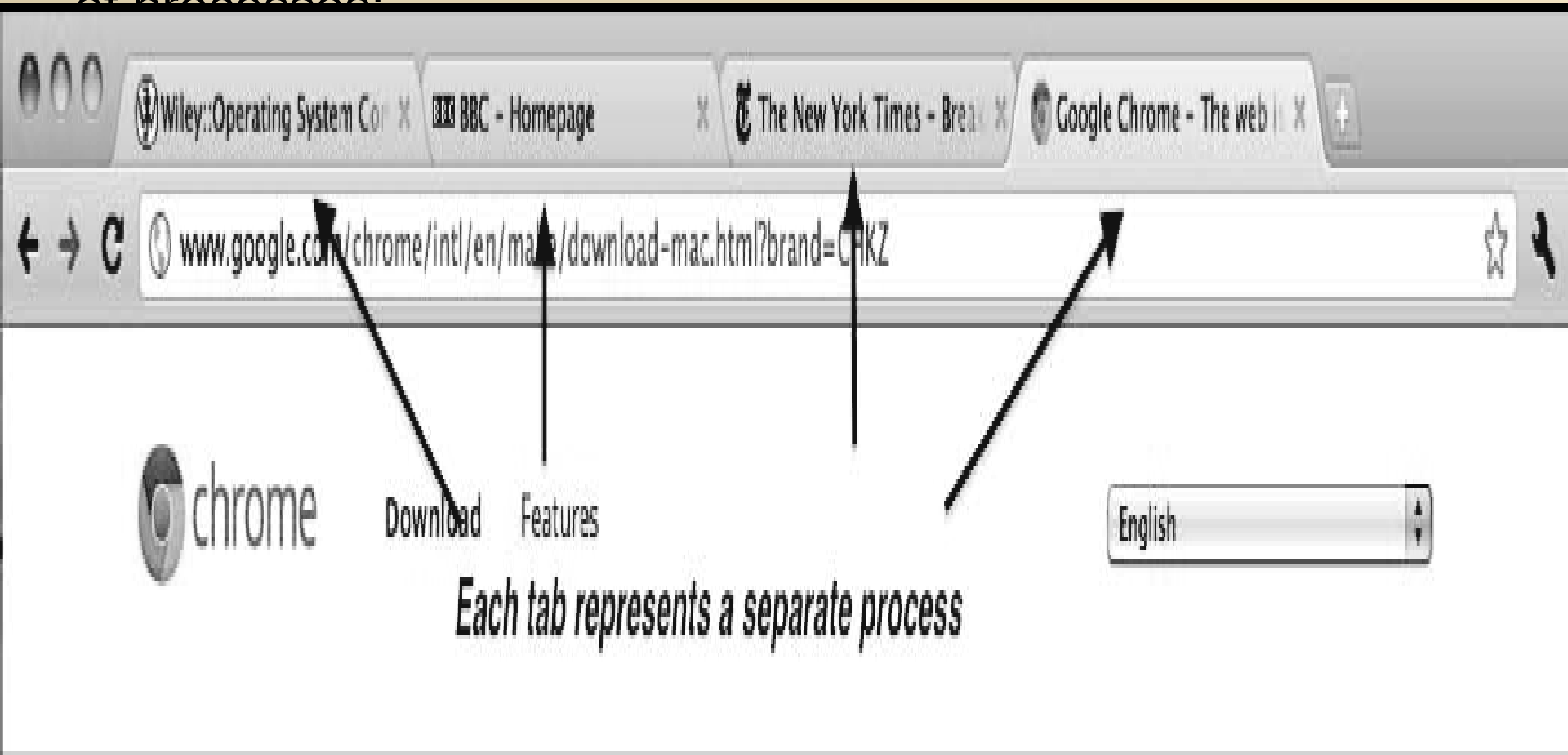
  Big-endian is the most common format in data networking . It is also referred to as **network byte order**.
- Remote communication has more failure scenarios than local
  - Messages can be delivered ***exactly once*** rather than ***at most once***
- OS typically provides a rendezvous (or **matchmaker**) service to connect client and server

Execution of RPC



Multiprocess Architecture – Chrome Browser

- Many web browsers ran as single process (some still do)
  - If one web site causes trouble, entire browser can hang or crash
- Google Chrome Browser is multiprocess with 3 different types of processes:



Let's relax a bit….

1. The systems which allows only one process execution at a time, are called:

A. uniprogramming systems

C. unitasking systems

Answer: Option A

Explanation:

Those systems which allows more than one process execution at a time, are called multiprogramming systems. Uniprocessing means only one processor.

2. In operating system, each process has its own:

A. address space and global variables **B.** open files

Answer: Option D

C. pending alarms, signals and signal handlers **D.** all of the mentioned

3. In Unix, Which system call creates the new process?

A. fork **B.** create

Answer: Option A

C. new **D.** none of the mentioned

Questions…

4. A process can be terminated due to:

A. normal exit

B. fatal error

Answer: Option D

C. killed by another process

D. all of the mentioned

5. What is the ready state of a process?

A. when process is scheduled to run after some execution

B. when process is unable to run until some

Answer: Option A
Explanation:
When process is unable to run until some task has been completed, the process is in blocked state and if process is using the CPU, it is in running state.

C. when process

6. What is interprocess communication?

Answer: Option B

A. communication within the process

B. communication between two process

C. communication between two threads of same process

D. none of the mentioned

Questions…

7. A set of processes is deadlock if:

A. each process is blocked and will remain so forever

B. each process is terminated

Answer: Option A

D. none of the mentioned

C. all processes are trying to kill each other

8. A process stack does not contain:

A. function parameters

B. local variables

Answer: Option D

C. return addresses

D. PID of child process

Process Stack in next slide:

9. Which system call returns the process identifier of a terminated child?

A. wait

B. exit

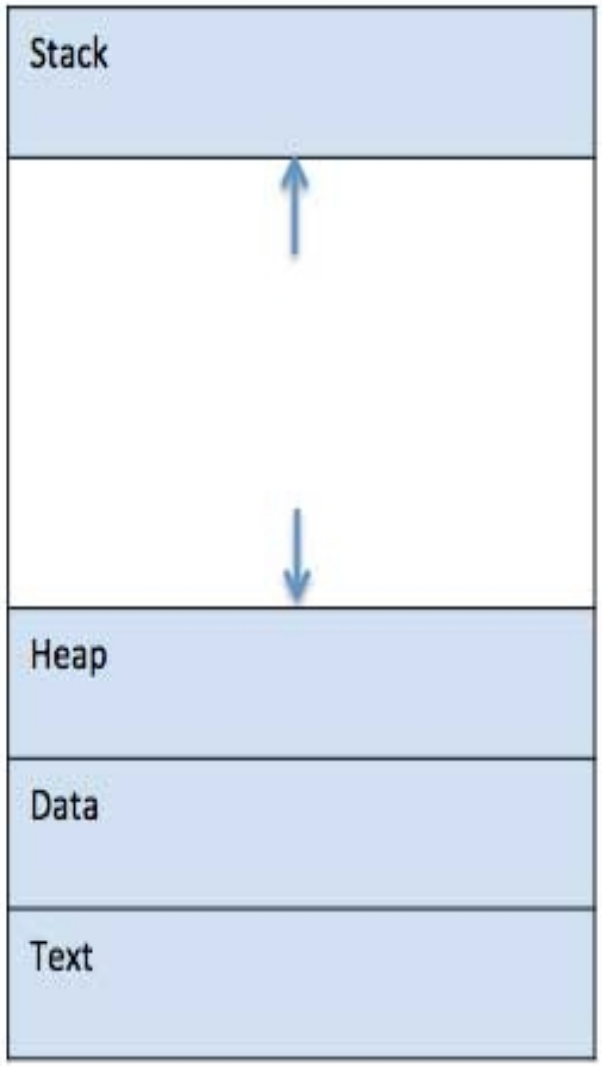
Answer: Option A

C. fork

D. get

Process stack

When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –



Seq	Component & Description
1	**Stack** **The process Stack contains the temporary data such as method/function parameters, return address and local variables.**
2	**Heap** **This is dynamically allocated memory to a process during its run time.**
3	**Text** **This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.**
4	**Data** **This section contains the global and static variables.**

Questions…

10. The address of the next instruction to be executed by the current process is provided by the:

A. CPU registers
B. program counter
C. process stack
D. pipe

Answer: Option B

11. A Process Control Block(PCB) does not contain which of the following :

A. code
B. stack
C. Process State
D. I/O status information
E. bootstrap program

Answer: Option E

12. The number of processes completed per unit time is known as _____.

A. output
B. Throughput
C. Efficiency
D. Capacity

Answer: Option B

Questions…

13. The state of a process is defined by : Answer: Option D

A. the final activity of the process
B. the activity just executed by the process
C. the activity to next be executed by the process
D. the current activity of the process

14. Which of the following is not the state of a process ?

A. new
B. old
C. Waiting
D. Running
E. Terminated

Answer: Option B

15. The Process Control Block is : Answer: Option B

A. Process type variable
B. Data Structure
C. a secondary storage section
D. a Block in memory

Questions…

16. The entry of all the PCBs of the current processes is in
A. Process Register
B. Program Counter
C. Process Table
D. Process Unit

Answer: Option C

17. The degree of multi-programming is :
A. the number of processes executed per unit time
B. the number of processes in the ready queue
C. the number of processes in the I/O queue
D. the number of processes in memory

Answer: Option D

18. A single thread of control allows the process to perform :
A. only one task at a time
B. multiple tasks at a time
C. All of these

Answer: Option A

19. When the process issues an I/O request :
A. It is placed in an I/O queue
B. It is placed in a waiting queue
C. It is placed in the ready queue
D. It is placed in the Job queue

Answer: Option A

Questions…

20. What is a long-term scheduler ?

Answer: Option A

A. It selects which process has to be brought into the ready queue
B. It selects which process has to be executed next and allocates CPU
C. It selects which process to remove from memory by swapping

22. If all processes I/O bound, the ready queue will almost always be _____, and the Short term Scheduler will have a _____ to do.

A. full,little
B. full,lot
C. empty,little
D. empty,lot

Answer: Option C

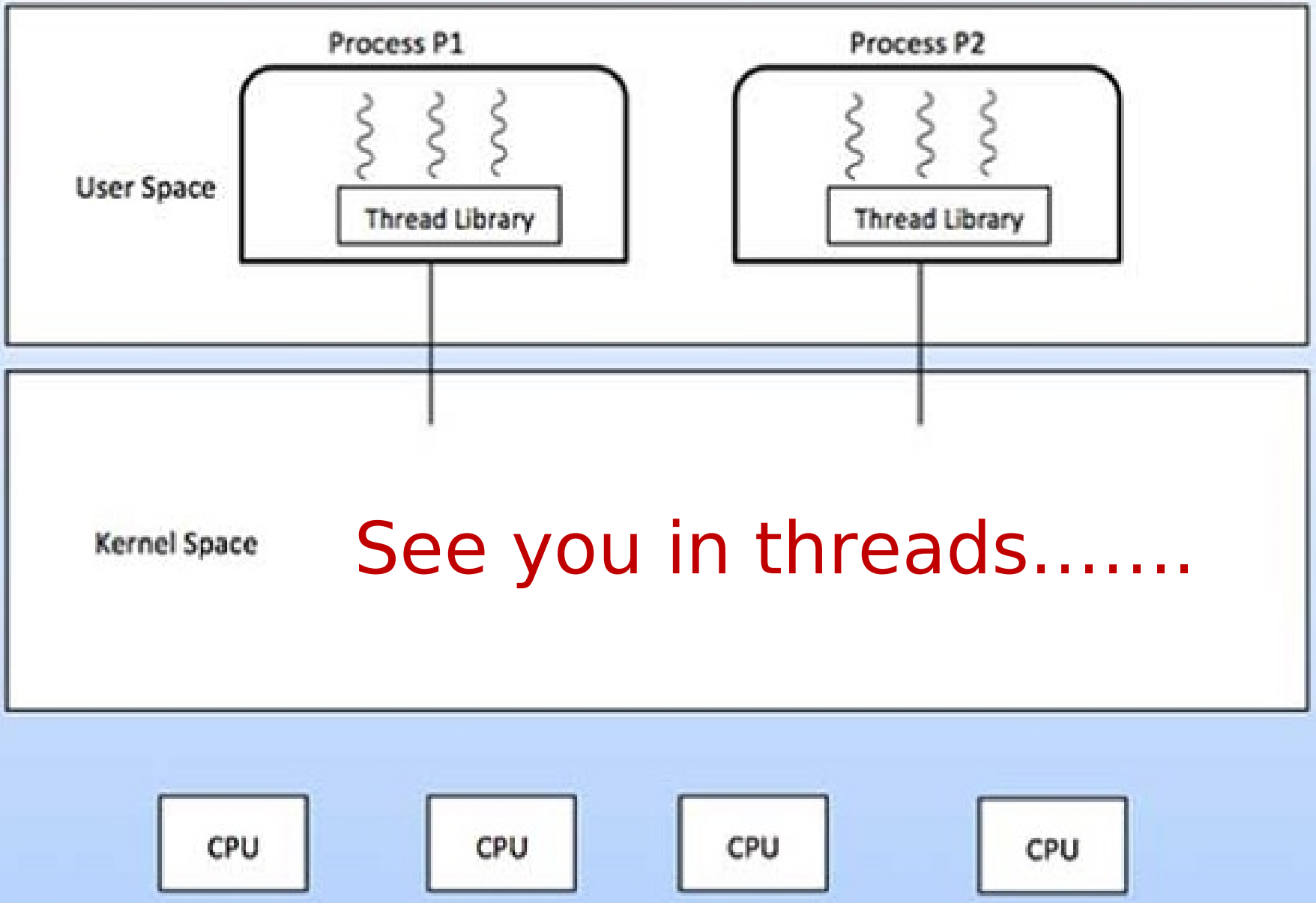
23. What is a medium-term scheduler ?

Answer: Option C

A. It selects which process has to be brought into the ready queue
B. It selects which process has to be executed next and allocates CPU
C. It selects which process to remove from memory by swapping

27. In a time-sharing operating system, when the time slot given to a process is completed, the process goes from the running state to the :**A.** Blocked state
B. Ready state
C. Suspended state
D. Terminated state

Answer: Option B


Process P1
Process P2
User Space
Thread Library
Thread Library
Kernel Space
See you in threads.....
CPU
CPU
CPU
CPU