USN | | | | | | | | | | |

Internal Assessment Test 1 – March 2018

Solutions

Sub:	Operating Systems					Sub Code:	15CS64	Branch:	ISE	
Date:	13/03/2018	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VI/ A, B			OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1 (a)	What is an operating system? What are its roles?	[04]	CO1	L1

An **operating system** is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

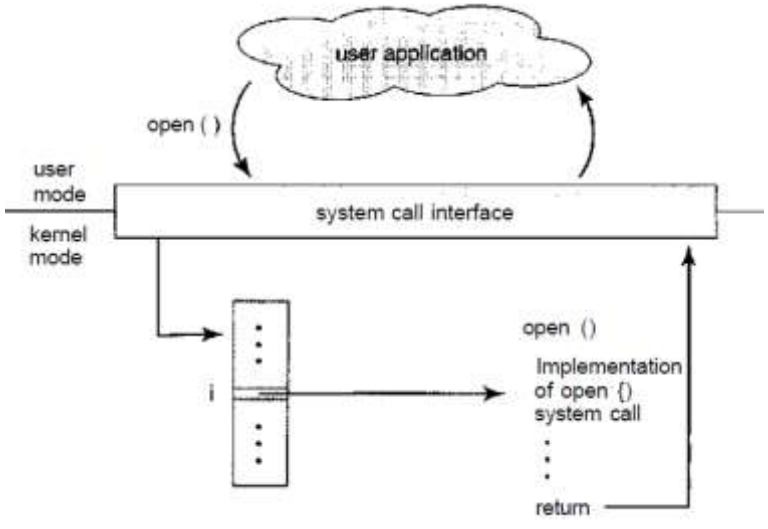
Roles:

User's view: The user's view of the computer varies according to the interface being used. The goal is to maximize the work that the user is performing. The OS is designed mostly for ease of **use,** with some attention paid to performance.

System's view: From the computer's point of view, the operating system is the program most intimately involved with the hardware. In this context, we view an operating system as a **resource allocator.**

(b)	What are system calls? With example explain different categories of system calls.	[06]	CO1	L2

- **System calls** provide an interface to the services made available by an operating system.
- These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly), may need to be written using assembly-language instructions.



Process control

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time

- wait event, signal event
- allocate and free memory

File management
- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

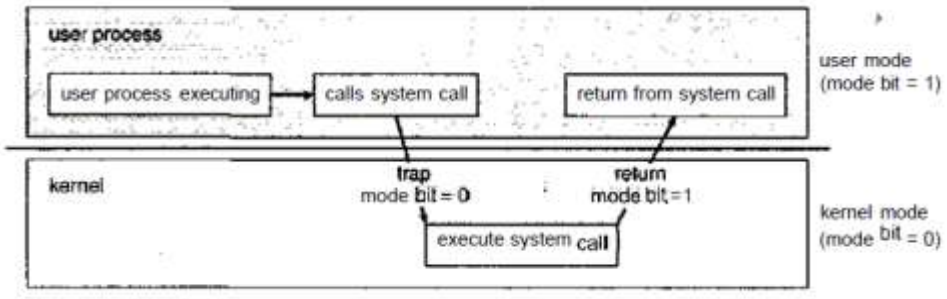
Device management
- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Information maintenance
- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

Communications
- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

2 (a) Explain dual mode operation in OS with a neat block diagram. [04] CO1 L2

- There are two separate **modes** of operation: **user mode** and **kernel mode** (also called **supervisor mode, system mode, or privileged mode).**
- A bit, called the **mode bit,** is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.
- When the computer system is executing on behalf of a user application, the system is in user mode.
- However, when a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfil the request.



- At system boot time, the hardware starts in kernel mode.
- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode

to kernel mode (that is, changes the state of the mode bit to 0).

- Whenever the operating system gains control of the computer, it is in kernel mode.
- The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.
- The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.
- The hardware allows privileged instructions to be executed only in kernel mode.
- If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.

(b) With a neat diagram explain the concept of VM. How is dual mode implemented in VM? Explain its advantages and disadvantages. [06] CO1 L2

- The layered approach is taken to its logical conclusion in the concept of a **virtual machine.**
- VMs abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

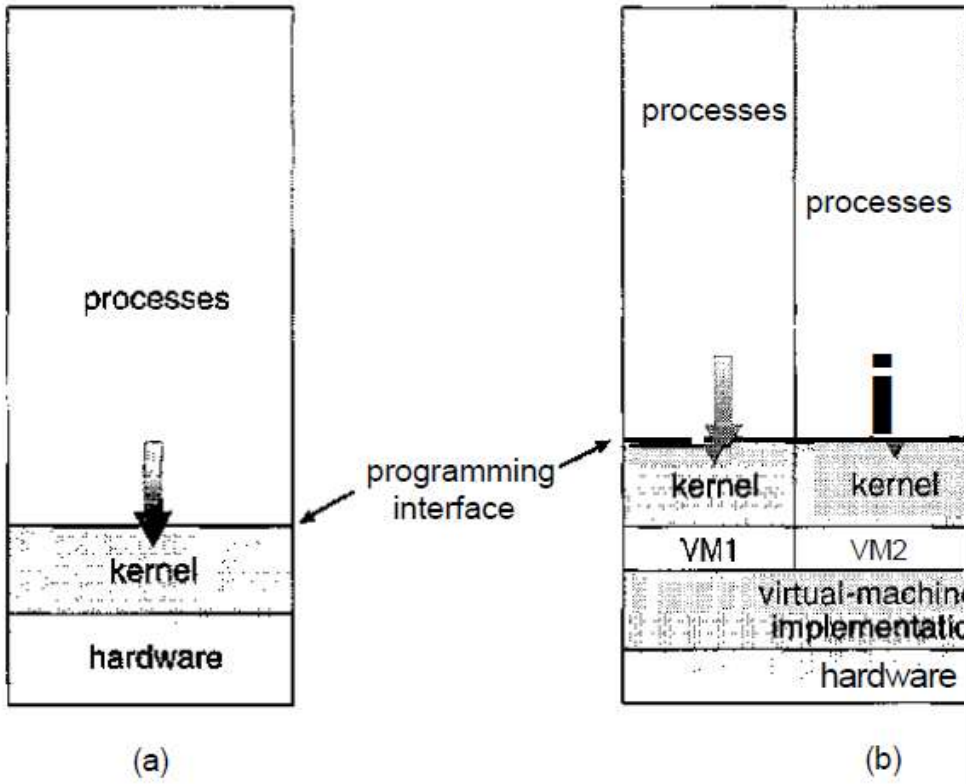


Figure 2.15 System models. (a) Nonvirtual machine. (b) Virtual

Implementation:

- The virtual-machine software can run in kernel mode, since it is the operating system. The virtual machine itself can execute in only user mode.
- Virtual Machine has virtual user mode and a virtual kernel mode, both of which run in a physical user mode. Those actions that cause a transfer from user mode to kernel mode on a real machine must also cause a transfer

from virtual user mode to virtual kernel mode on a virtual machine.

Such a transfer can be accomplished as follows:

- When a system call, for example, is made by a program running on a virtual machine in virtual user mode, it will cause a transfer to the virtual-machine monitor in the real machine.
- When the virtual-machine monitor gains control, it can change the register contents and program counter for the virtual machine to simulate the effect of the system call.
- It can then restart the virtual machine, noting that it is now in virtual kernel mode.

Advantages

- By using CPU scheduling and virtual-memory techniques, an operating system can create the illusion that a process has its own processor with its own (virtual) memory.

- The virtual-machine approach does not provide any such additional functionality, such as system calls and a file system, but rather provides an interface that is *identical* to the underlying bare hardware. Each process is provided with a (virtual) copy of the underlying computer.

Disadvantages:

- Although the virtual-machine concept is useful, it is difficult to implement.
- Much work is required to provide an *exact* duplicate of the underlying machine.
- Virtual machines are slower

3 (a) What is the need for multithreaded processes? Indicate the four major categories of benefits derived from multithreaded programming. [04] CO2 L2

Need for Multithreaded processes:

- In certain situations, a single application may be required to perform several similar tasks. For example, a web server accepts client requests for web pages, images, sound, and so forth. If the web server ran as a traditional single-threaded process, it would be able to service only one client at a time. The amount of time that a client might have to wait for its request to be serviced could be enormous.
- Process creation is time consuming and resource intensive, as was shown in the previous chapter. If the new process will perform the same tasks as the existing process, it is unnecessary to incur all the overhead.
- It is generally more efficient to use one process that contains multiple threads. The server would create a separate thread that would listen for client requests; when a request was made, rather than creating another process, the server would create another thread to service the request.
- Threads also play a vital role in remote procedure call (RPC) systems. RPC servers are multithreaded. When a server receives a message, it services the message using a separate thread. This allows the server to service several concurrent requests. E.g., Java's RMI.
- Finally, many operating system kernels are now multithreaded; several threads operate in the kernel, and each thread performs a specific task, such as managing devices or interrupt handling.

Benefits:

The benefits of multithreaded programming can be broken down into four major categories:

1. Responsiveness. Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. For instance, a multithreaded web browser could still allow user interaction in one thread while an image was being loaded in another thread.
2. Resource sharing. By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
3. Economy. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads than in a process. In Solaris, for example, creating a process is about thirty times slower than is creating a thread, and context switching is about five times slower.
4. Utilization of multiprocessor architectures. The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors.

(b) Distinguish between the following pairs of terms: [06] CO2,6 L2

(i) User vs kernel mode (ii) Process vs Thread (iii) User vs Kernel threads

User mode	**Kernel Mode**
User applications run in user mode	When user application requests service from OS it must transition to kernel mode
After system boots while starting user applications, system enters user mode	System boots in kernel mode
Privileged instructions can't be run in user mode	Privileged instructions run in kernel mode

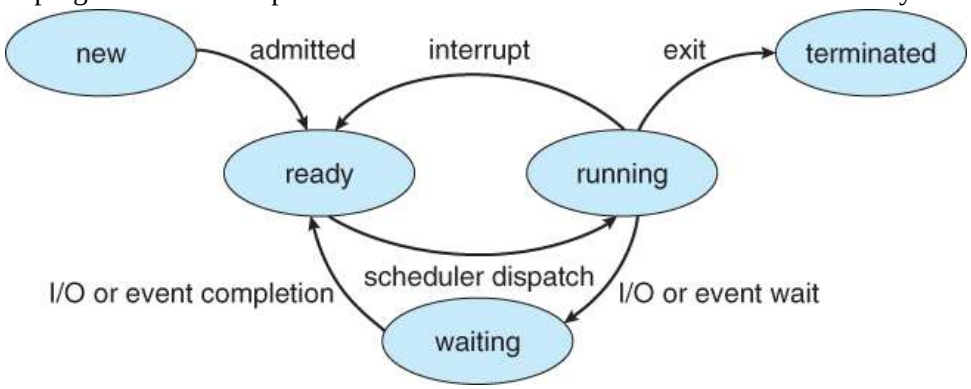
Process	**Threads**
An executing instance of a program is called a process.	A thread is a subset of the process.
It has its own copy of the data segment of the parent process.	It has direct access to the data segment of its process.
Processes must use inter-process communication to communicate with sibling processes.	Threads can directly communicate with other threads of its process.
Processes have considerable overhead.	Threads have almost no overhead.
Run in separate memory spaces.	Run in shared memory spaces.

User-level threads	**Kernel-level threads**

User thread are implemented by users/application programs.	kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris.

4 (a) What is a process? Illustrate the process state transition diagram. [04] CO1 L1

A process is a program in execution. Process is an *active* entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory.



(b) Explain multithreading models. [06] CO2 L2

User threads are supported above the kernel and are managed without kernel support.
Kernel threads are supported and managed directly by the operating system.

Many-to-One Model

- The many-to-one model maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient; but the entire process will block if a thread makes a blocking system call.
- Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.
- Example: Green threads—a thread library available for Solaris—uses this model, as does GNU Portable Threads.

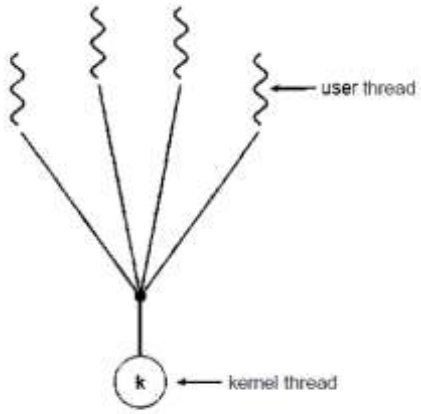


Figure 4.2 Many-to-one model.

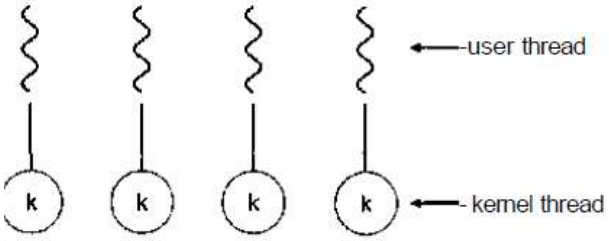


Figure 4.3 One-to-one model.

One-to-One Model

- The one-to-one model maps each user thread to a kernel thread.
- It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.
- Most implementations of this model restrict the number of threads supported by the system.
- Example: Linux, along with the family of Windows operating systems—including Windows 95, 98, NT, 2000, and XP implement the one-to-one model.

Many-to-Many Model

- The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads.
- In Many-to-one model true concurrency is not gained because the kernel can schedule only one thread at a time.
- The one-to-one model allows for greater concurrency, but the developer has to be careful not to create too many threads within an application.
- In many-to-many model developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
- When a thread performs a blocking system call, the kernel can schedule another thread for execution.
- Example: Java Threads on Solaris, Win32 Threads with ThreadFibre package

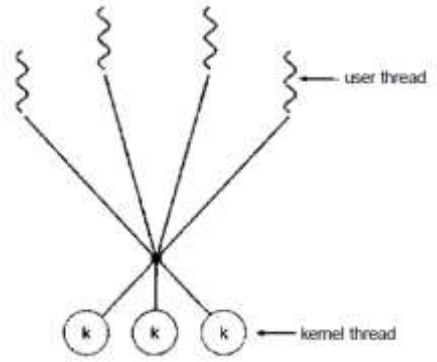


Figure 4.4 Many-to-many model.

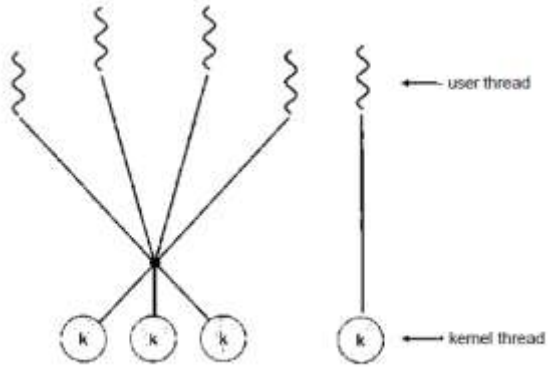


Figure 4.5 Two-level model.

Two-Level Model

- One popular variation on the many-to-many model still multiplexes many user-level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread.
- This variation, sometimes referred to as the *two-level model*, is supported by operating systems such as IRIX, HP-UX, and Tru64 UNIX.

5 (a) What is IPC? What are the merits of IPC? [04] CO1 L1

IPC, stands for interprocess communication, is a mechanism where two or more cooperating processes communicate for sharing data, resources or for synchronization.

Merits of IPC:

- Information sharing
- Computation speedup
- Modularity
- Convenience

(b) Explain direct and indirect communication with respect to message passing. [06] CO1 L2

Direct Communication:

Symmetric direct communication:

- This scheme exhibits *symmetry* in addressing; that is, both the sender process and the receiver process must name the other to communicate.
- Each process that wants to communicate must explicitly name the recipient or sender of the communication.
- In this scheme, the send.0 and receive() primitives are defined as:
  - send(P, message)—Send a message to process P.
  - receive (Q, message)—Receive a message from process Q
- A communication link in this scheme has the following properties:
  - A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.

- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link.

Asymmetric direct communication:

- This scheme employs *asymmetry* in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender.
- In this scheme, the send() and receive () primitives are defined as follows:
  - send(P, message)—Send a message to process P.
  - receive(id, message)—-Receive a message from any process; the variable *id* is set to the name of the process with which communication has taken place.
- The disadvantage in both of these schemes (symmetric and asymmetric) is the limited modularity of the resulting process definitions. Changing the identifier of a process may necessitate examining all other process definitions.

Indirect Communication:

- With indirect communication, the messages are sent to and received from mailboxes, or ports.
- A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification.
- Two processes can communicate only if the processes have a shared mailbox.
- The sendC) and receive () primitives are defined as follows:
  - send(A, message)—Send a message to mailbox A.
  - receive(A, message)—Receive a message from mailbox A.
- In this scheme, a communication link has the following properties:
  - A link is established between a pair of processes only if both members of the pair have a shared mailbox.
  - A link may be associated with more than two processes.
  - Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.

- Example: Suppose that processes *P1, P2,* and *P3* all share mailbox A, Process P1 sends a message to *A,* while both P_2 and P3 execute a receive() from A.
- Which process will receive the message sent by P1 depends on the following methods we choose:
  - Allow a link to be associated with two processes at most.
  - Allow at most one process at a time to execute a receive () operation.
  - Allow the system to select arbitrarily which process will receive the message (that is, either P_2 or P_3, but not both, will receive the message).
- A mailbox may be owned either by a process or by the operating system.
- If the mailbox is owned by a process, then we distinguish between the owner and the user. (who can only send messages to the mailbox). Since each mailbox has a unique owner, there can be no confusion about who should receive a message sent to this mailbox.
- The operating system then must provide a mechanism that allows a process to do the following:
  - Create a new mailbox.
  - Send and receive messages through the mailbox.
  - Delete a mailbox.
- When a process that owns a mailbox terminates, the mailbox disappears. Any process that subsequently sends a message to this mailbox must be notified that the mailbox no longer exists.

6 (a) List the services provided by an OS that are designed for: [04] CO1 L1

(i) user convenience (ii) efficient operation of the system.

(i) User convenience

- **User interface (UI).**
- **Program execution**
- **I/O operations**
- **File-system manipulation**
- **Communications**
- **Error detection**

(ii) Efficient operation of system

Resource allocation
Accounting
Protection and security

(b) Consider the following set of processes with arrival time. [06] CO2,6 L3

Processes	Burst Time (ms)	Arrival time (ms)
P1	10	0
P2	29	1
P3	3	2
P4	7	3

Draw Gantt chart using preemptive SJF, and RR with time slice of 10 ms quantum. Calculate average WT and TAT for each.

Preemptive SJF: (time in ms)

	P1=2	P3=3	P4=7	P1=8	P2=29
0	2	5	12	20	49

Process	WT (ms)	TAT (ms)
P1	12-2=10	20
P2	20-1=19	49-1=48
P3	2-2=0	5-2=3
P4	5-3=2	12-3=9
Avg	**31/4=7.75**	**80/4=20**

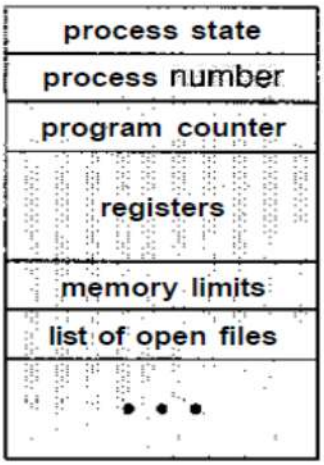
RR with time slice 10 ms. (time in ms)

	P1=10	P2=10	P3=3	P4=7	P2=10	P2=9
0	10	20	23	30	40	49

Process	WT (ms)	TAT (ms)
P1	0	10
P2	20-1=19	49-1=48
P3	20-2=18	23-2=21
P4	23-3=20	30-3=27
Avg	**57/4=14.25**	**96/4=26.5**

7 (a) What is a PCB? Illustrate the different components of a PCB. [04] CO1 L1

Each process is represented in the operating system by a **process control block (PCB)**—also called a *task control block.* PCB contains many pieces of information associated with a specific process.



(b) For the processes listed below, draw Gantt chart using preemptive and non-preemptive priority scheduling. Calculate the average WT and TAT. [06] CO2,6 L3

Note: In this case, A larger priority number has higher priority.

Processes	Arrival time (ms)	Burst Time (ms)	Priority
P1	0	6	4
P2	3	5	2
P3	3	3	6
P4	5	5	3

Non-preemptive Priority (time in ms)

	P1=6	P3=3	P4=5	P2=5
0	6	9	14	19

Process	WT (ms)	TAT (ms)
P1	0	6
P2	14-3=11	19-3=16
P3	6-3=3	9-3=6
P4	9-5=4	14-5=11
Avg	**18/4=4.5**	**37/4=9.25**

Preemptive Priority (time in ms)

	P1=3	P3=3	P1=3	P4=5	P2=5
0	3	6	9	14	19

Process	WT (ms)	TAT (ms)
P1	3	9
P2	14-3=11	19-3=16
P3	0	6-3=3
P4	9-5=4	14-5=11
Avg	**18/4=4.5**	**37/4=9.25**

8 (a) Why is it important for scheduler to distinguish CPU and I/O bound programs? [04] CO1 L2

- It is important that the long-term scheduler select a good **process mix** of I/O-bound and CPU-bound processes.
- In general, most processes can be described as either I/O bound or CPU bound.

- An **I/O-bound process** is one that spends more of its time doing I/O than it spends doing computations.
- A **CPU-bound process,** in contrast, generates I/O requests infrequently, using more of its time doing computations.
- If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do.
- If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced.
- The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes.

(b) Write a multithreaded program using pthread library to perform multiplication of two vectors. [C[i]=A[i]*B[i]; for all $0 \leq i < N$ (where N is a positive integer)]. [06] CO2 L3

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

int i=0,mul=0; /* this data is shared by the thread(s) */
int A[100], B[100], C[100];

void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
     pthread_t tid1,tid2; /* the thread identifier */
     pthread_attr_t attr; /* set of thread attributes */
     int N,j;

     if (argc != 2) {
          fprintf(stderr,"usage: a.out <integer value>\n");
          return -1;
     }

     N=atoi(argv[1]);
     if (atoi(argv[1]) < 0) {
          fprintf(stderr,"%d must be >= 0\n",N);
          return -1;
     }

     printf("Enter the values for Vector A[]: ");
     for (j=0;j<N;j++)
          scanf("%d",&A[j]);

     printf("Enter the values for Vector B[]: ");
     for (j=0;j<N;j++)
          scanf("%d",&B[j]);

     /* get the default attributes */
     pthread_attr_init(&attr);

     /* create the thread */
     pthread_create(&tid1,&attr,runner,argv[1]);
     pthread_create(&tid2,&attr,runner,argv[1]);
     /* wait for the thread to exit */
     pthread_join (tid1, NULL) ;
     pthread_join (tid2, NULL) ;

     printf("\nProduct Vector is: ");
     for (j=0;j<N;j++)
          printf("%d ",C[j]);
     printf("\n");
```

```
}

/* The thread will begin control in this function */
void *runner(void *param)
{
     int upper = atoi(param);

     pthread_t my_tid;
     my_tid = pthread_self();

     printf("Hi there! This is Thread %ld\n",my_tid);

     for (;i <= upper; i++)
          C[i]=A[i]*B[i];
     pthread_exit(0);
}
```

Output:

```
$ ./a.out 5

Enter the values for Vector A[]: 2 2 2 2 2
Enter the values for Vector B[]: 3 3 3 3 3
Hi there! This is Thread 6432
Hi there! This is Thread 3465

Product Vector is: 6 6 6 6 6
```

END