5675152

B.Tech. DEGREE EXAMINATION JANUARY 2023

Fifth Semester

Information Technology

OPERATING SYSTEMS

(2013 – 14 Regulations)

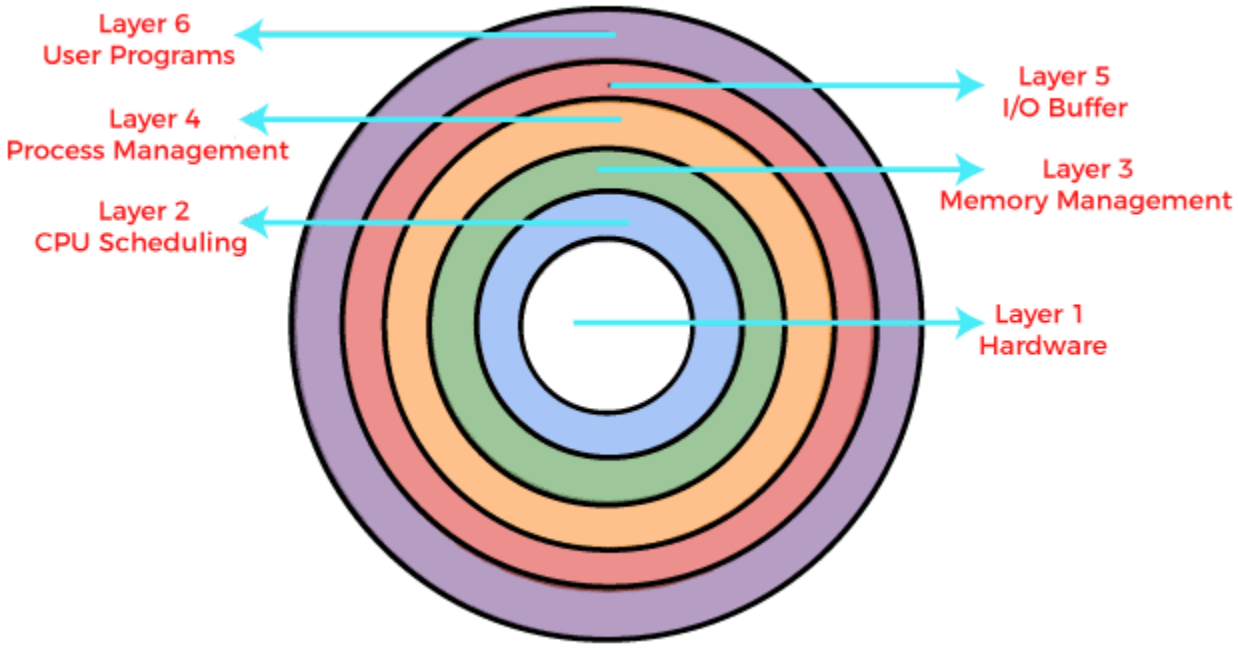
Time: Three hours **Maximum: 75 marks**

PART-A(10*2=20 marks)

Answer all Questions

1. **Show the layered structure of the operating system.**

   The layered structure approach breaks up the operating system into different layers and retains much more control on the system. The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers only. It simplifies the debugging process as if lower-level layers are debugged, and an error occurs during debugging. The error must be on that layer only as the lower-level layers have already been debugged.



2. **List out the three way implementation of queue in buffering.**

   Messages are passed via queues, which may have one of three capacity configurations:
   Zero capacity - Messages cannot be stored in the queue, so senders must block until receivers accept the messages.

Bounded capacity- There is a certain pre-determined finite capacity in the queue. Senders must block if the queue is full, until space becomes available in the queue, but may be either blocking or non-blocking otherwise.

Unbounded capacity - The queue has a theoretical infinite capacity, so senders are never forced to block.

3. **Compare process and program.**

A program is a passive entity that contains the set of codes required to perform a certain task. A process is an active instance of the program which is started when the program is executed. Once a program is executed, a process is started by the program. The process executes the instructions written in the program.

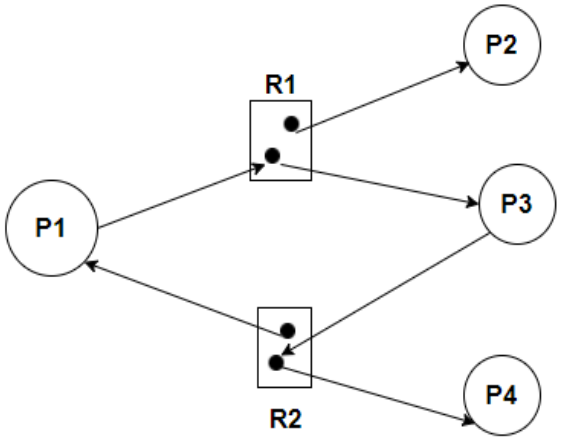
4. **Define Race Condition.**

A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

5. **Construct the resource allocation graph with cycle but no deadlock.**

- The Resource Allocation graph mainly consists of a **set of vertices V** and **a set of Edges E.**
- This graph mainly contains all the information related to the processes that are holding some resources and also contains the information of the processes that are waiting for some more resources in the system.
- Also, this graph contains all the information that is related to all the instances of the resources which means the information about available resources and the resources which are being used by the process.
- In this graph, the circle is used to represent the process, and the rectangle is used to represent the resource.



Multiple Instance Resource Allocation graph with a cycle but no deadlock

6. **Distinguish between logical and physical addresses.**

S. No.	Logical Address	Physical Address
1.	This address is generated by the CPU.	This address is a location in the memory unit.
2.	The address space consists of the set of all logical addresses.	This address is a set of all physical addresses that are mapped to the corresponding logical addresses.
3.	These addresses are generated by CPU with reference to a specific program.	It is computed using Memory Management Unit (MMU).
4.	The user has the ability to view the logical address of a program.	The user can’t view the physical address of program directly.
5.	The user can use the logical address in order to access the physical address.	The user can indirectly access the physical address.

7. **Mention the advantages and Disadvantages of Contiguous allocation of files.**

Advantages

1. It is simple to keep track of how many memory blocks are left, which determines how many more processes can be granted memory space.
2. The read performance of contiguous memory allocation is good because the complete file may be read from the disk in a single task.
3. The contiguous allocation is simple to set up and performs well.

Disadvantages

1. Fragmentation isn't a problem because every new file may be written to the end of the disk after the previous one.
2. When generating a new file, it must know its eventual size to select the appropriate hole size.
3. When the disk is filled up, it would be necessary to compress or reuse the spare space in the holes.

8. **List the types of access in file system interface.**

   The file contains the information but when it required to used this information can be access by the access methods and reads into the computer memory. Some system provides only one access method and some provide more than on access method to access the file,

1. Sequential Access Method

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one.

2. Direct or Random Access Methods

The disk is a direct access device which gives us the reliability to random access of any file block. In the file, there is a collection of physical blocks and the records of that blocks.

3. Index Access Method

An indexed file is a computer file with an index that allows easy random access to any record given its file key. The key is an attribute that uniquely identifies a record. We can say that If more than one index is present the other ones are alternate indexes. The creation of the indexes is done with the file but maintained by the system.

9. Define seek time and Rotational Latency.

Seek time is the time taken for a hard disk controller to locate a specific piece of stored data. Seek time can vary upon where the head is present when the read/write request is sent. The read/write head of the disc takes to move from one disk to another is called the **seek time**.

The time taken by the platter to rotate and position the data under the read-write head is called ***rotational latency***. This latency depends on the rotation speed of the spindle and is measured in ***milliseconds***.

10. Give short notes on indexed allocation method.

Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.

PART-B (5*11=55 marks)

Answer all Questions

11. List five services provided by an operating system. Explain how each provides convenience to the users. Also explain in which cases it would be impossible for user-level programs to provide these services.

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection
- Resource Allocation
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

12. a) Distinguish between tightly coupled system and loosely coupled system.

S.NO	Loosely Coupled	Tightly Coupled
1.	There is distributed memory in loosely coupled multiprocessor system.	There is shared memory, in tightly coupled multiprocessor system.
2.	Loosely Coupled Multiprocessor System has low data rate.	Tightly coupled multiprocessor system has high data rate.
3.	The cost of loosely coupled multiprocessor system is less.	Tightly coupled multiprocessor system is more costly.
4.	In loosely coupled multiprocessor system, modules are connected through **Message transfer system** network.	While there is PMIN, IOPIN and ISIN networks.
5.	In loosely coupled multiprocessor, Memory conflicts don't take place.	While tightly coupled multiprocessor system have memory conflicts.
6.	Loosely Coupled Multiprocessor system has low degree of interaction between tasks.	Tightly Coupled multiprocessor system has high degree of interaction between tasks.
7.	In loosely coupled multiprocessor, there is direct connection between processor and I/O devices.	While in tightly coupled multiprocessor, IOPIN helps connection between processor and I/O devices.
8.	Applications of loosely coupled multiprocessor are in distributed computing systems.	Applications of tightly coupled multiprocessor are in parallel processing systems.

b) Specify the five major activities of an operating system in regard to process management.

Five major process management activities of an operating system are

a. creation and deletion of user and system processes.

b. suspension and resumption of processes.

c. provision of mechanisms for process synchronization.

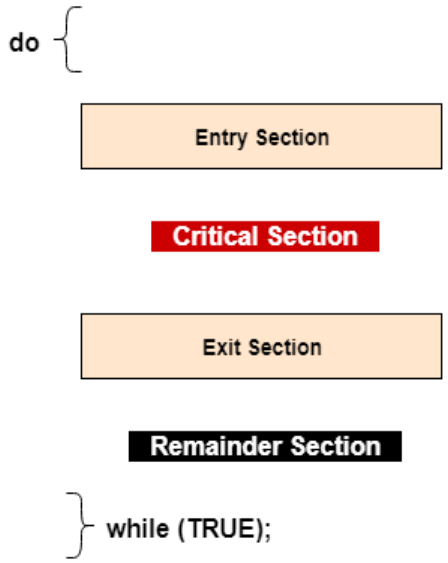
d. provision of mechanisms for process communication.

e. provision of mechanisms for deadlock handling.

13. Describe an algorithm which satisfies all the conditions of critical section problem and also solve the readers writers problems using semaphore.

The critical section is a code segment where the shared variables can be accessed. An atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

A diagram that demonstrates the critical section is as follows −



In the above diagram, the entry section handles the entry into the critical section. It acquires the resources needed for execution by the process. The exit section handles the exit from the critical section. It releases the resources and also informs the other processes that the critical section is free.

Solution to the Critical Section Problem

The critical section problem needs a solution to synchronize the different processes. The solution to the critical section problem must satisfy the following conditions −

- **Mutual Exclusion** Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

- **Progress** Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.
- **Bounded Waiting** Bounded waiting means that each process must have a limited waiting time. Itt should not wait endlessly to access the critical section.

Peterson's Solution

- Peterson's Solution is a classic software-based solution to the critical section problem. It is unfortunately not guaranteed to work on modern hardware, due to vagaries of load and store operations, but it illustrates a number of important concepts.
- Peterson's solution is based on two processes, P0 and P1, which alternate between their critical sections and remainder sections. For convenience of discussion, "this" process is Pi, and the "other" process is Pj. (I.e. j = 1 - i)
- Peterson's solution requires two shared data items:

  - **int turn -** Indicates whose turn it is to enter into the critical section. If turn = = i, then process i is allowed into their critical section.
  - **boolean flag[2] -** Indicates when a process ***wants to*** enter into their critical section. When process i wants to enter their critical section, it sets flag[i] to true.

- In the following diagram, the entry and exit sections are enclosed in boxes.

  - In the entry section, process i first raises a flag indicating a desire to enter the critical section.
  - Then turn is set to ***j*** to allow the ***other*** process to enter their critical section ***if process j so desires.***
  - The while loop is a busy loop (notice the semicolon at the end), which makes process i wait as long as process j has the turn and wants to enter the critical section.
  - Process i lowers the flag[i] in the exit section, allowing process j to continue if it has been waiting.

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = FALSE;

        remainder section

} while (TRUE);
```

Figure - The structure of process Pi in Peterson's solution.

READERS-WRITERS PROBLEM

In the readers-writers problem there are some processes (termed readers) who only read the shared data, and never change it, and there are other processes (termed writers) who may change the data in addition to or instead of reading it. There is no limit to how many readers can access the data simultaneously, but when a writer accesses the data, it needs exclusive access.

- There are several variations to the readers-writers problem, most centered around relative priorities of readers versus writers.
- The first readers-writers problem gives priority to readers. In this problem, if a reader wants access to the data, and there is not already a writer accessing it, then access is granted to the reader. A solution to this problem can lead to starvation of the writers, as there could always be more readers coming along to access the data. (A steady stream of readers will jump ahead of waiting writers as long as there is currently already another reader accessing the data, because the writer is forced to wait until the data is idle, which may never happen if there are enough readers.
- The second readers-writers problem gives priority to the writers. In this problem, when a writer wants access to the data it jumps to the head of the queue - All waiting readers are blocked, and the writer gets access to the data as soon as it becomes available. In this solution the readers may be starved by a steady stream of writers.

The following code is an example of the first readers-writers problem, and involves an important counter and two binary semaphores:

- readcount is used by the reader processes, to count the number of readers currently accessing the data.
- mutex is a semaphore used only by the readers for controlled access to readcount.

- rw_mutex is a semaphore used to block and release the writers. The first reader to access the data will set this lock and the last reader to exit will release it; The remaining readers do not touch rw_mutex. (Eighth edition called this variable wrt.)

```
do {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
} while (true);
```

Figure 5.11 The structure of a writer process.

```
do {
   wait(mutex);
   read_count++;
   if (read_count == 1)
      wait(rw_mutex);
   signal(mutex);
      . . .
   /* reading is performed */
      . . .
   wait(mutex);
   read_count--;
   if (read_count == 0)
      signal(rw_mutex);
   signal(mutex);
} while (true);
```

Figure 5.12 The structure of a reader process.

14. Draw the gantt charts of the processes using RR, non preemptive SJF scheduling with suitable examples.

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2

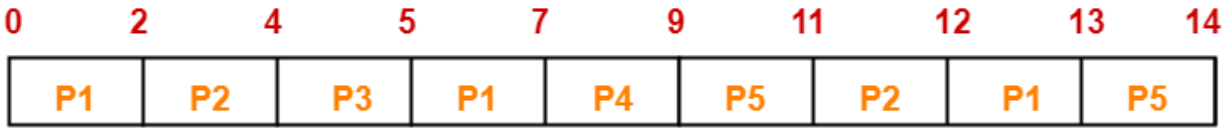
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

Gantt Chart-

Ready Queue-

P5, P1, P2, P5, P4, P1, P3, P2, P1



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

Now,

Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit

Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

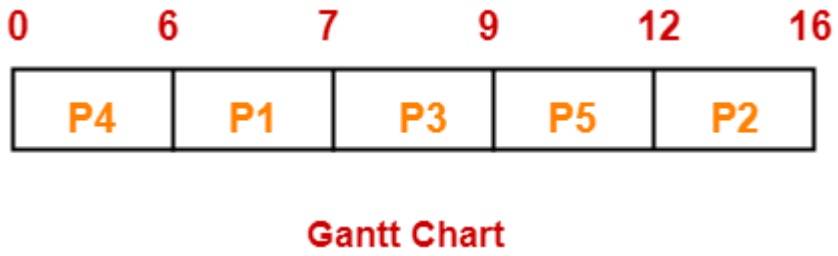
Non Preemptive SJF Scheduling

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turn around time.

Gantt Chart-



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 1 = 3

P2	16	16 – 1 = 15	15 – 4 = 11
P3	9	9 – 4 = 5	5 – 2 = 3
P4	6	6 – 0 = 6	6 – 6 = 0
P5	12	12 – 2 = 10	10 – 3 = 7

Now,

- Average Turn Around time = (4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8 unit
- Average waiting time = (3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8 unit

15. **Consider the following snapshot of a system:**

Process	**Allocation**	**Max**	**Available**
	R1 R2 R3	**R1 R2 R3**	**R1 R2 R3**
P1	**0 1 0**	**7 5 3**	**3 3 2**
P2	**2 0 0**	**3 2 2**	
P3	**3 0 2**	**9 0 2**	
P4	**2 1 1**	**2 2 2**	
P5	**0 0 2**	**4 3 3**	

Answer the following questions based on the banker's algorithm:

a) Find out the content of matrix need.

b) Is the system in a safe state.

Solution :-

① To calculate a Need Matrix

Need = Max − Allocation

Process	Need		
	R_1	R_2	R_3
P_1	7	4	3
P_2	1	2	2
P_3	6	0	0
P_4	0	1	1
P_5	4	3	1

Banker's Algorithm :

Work = Available

$Need_i \leq$ Work $\Rightarrow$ Work = Work + Allocation.

P_0 7 4 3 $\leq$ 3 3 2 ×

P_2 1 2 2 $\leq$ 3 3 2 ✓

Work: Work + Allocation

= 3 3 2 + 2 0 0 = 5 3 2

Now **Work = 5 3 2**

P_3 6 0 0 $\leq$ 5 3 2 ×

P_4 0 1 1 $\leq$ 5 3 2 ✓ Work: Work + Allocation

= 5 3 2 + 2 1 1

Now **Work = 7 4 3** = 7 4 3

P_5 4 3 1 $\leq$ 5 3 2 ✓ Work = Work + Allocation

= 7 4 3 + 0 0 2

Now **Work = 7 4 5** = 7 4 5

P_0 7 4 3 $\leq$ 7 4 5 ✓ Work = Work + Allocation

= 7 4 5 + 0 1 0

Now **Work = 7 5 5** = 7 5 5

P_3 6 0 0 $\leq$ 7 5 5 ✓ Work = Work + Allocation

= 7 5 5 + 3 0 2

Now **Work = 10 5 7** = 10 5 7

b) yes The process is in Safe state $\langle P_2, P_4, P_5, P_1, P_3 \rangle$

16. **Given memory Partitions of 200 kb, 600 kb, 300 kb, 400 kb, and 700 kb(in order), how would each of the first fit, best fit and worst fit algorithms place processes of 312kb, 517kb, 212kb and 526kb (in order)?**

Which algorithm makes the most efficient use of memory?

First-fit

The first-fit algorithm selects the first free partition that is large enough to accommodate the request.

First-fit would allocate in the following manner:

312 KB => 600 KB partition, leaves a 288 KB partition

517 KB =>700 KB partition, leaves a 183 KB partition

212 KB => 288 KB partition, leaves a 76 KB partition

526 KB would not be able to allocate, no partition large enough!

Best-fit

The best-fit algorithm selects the partition whose size is closest in size (and large enough) to the requested size.

Best-fit would allocate in the following manner:

312 KB => 400 KB, leaving a 88 KB partition

517 KB => 600 KB, leaving a 83 KB partition

212 KB => 300 KB, leaving a 88 KB partition

526 KB => 700 KB, leaving a 174 KB partition

Worst-fit

The worst-fit algorithm effectively selects the largest partition for each request.

Worst-fit would allocate in the following manner:

312 KB => 700 KB, leaving a 388 KB partition

517 KB => 600 KB, leaving a 83 KB partition

212 KB => 388 KB, leaving a 176 KB partition

526 KB would not be allowed to allocate as no partition is large enough!

Which algorithm makes the most efficient use of memory?

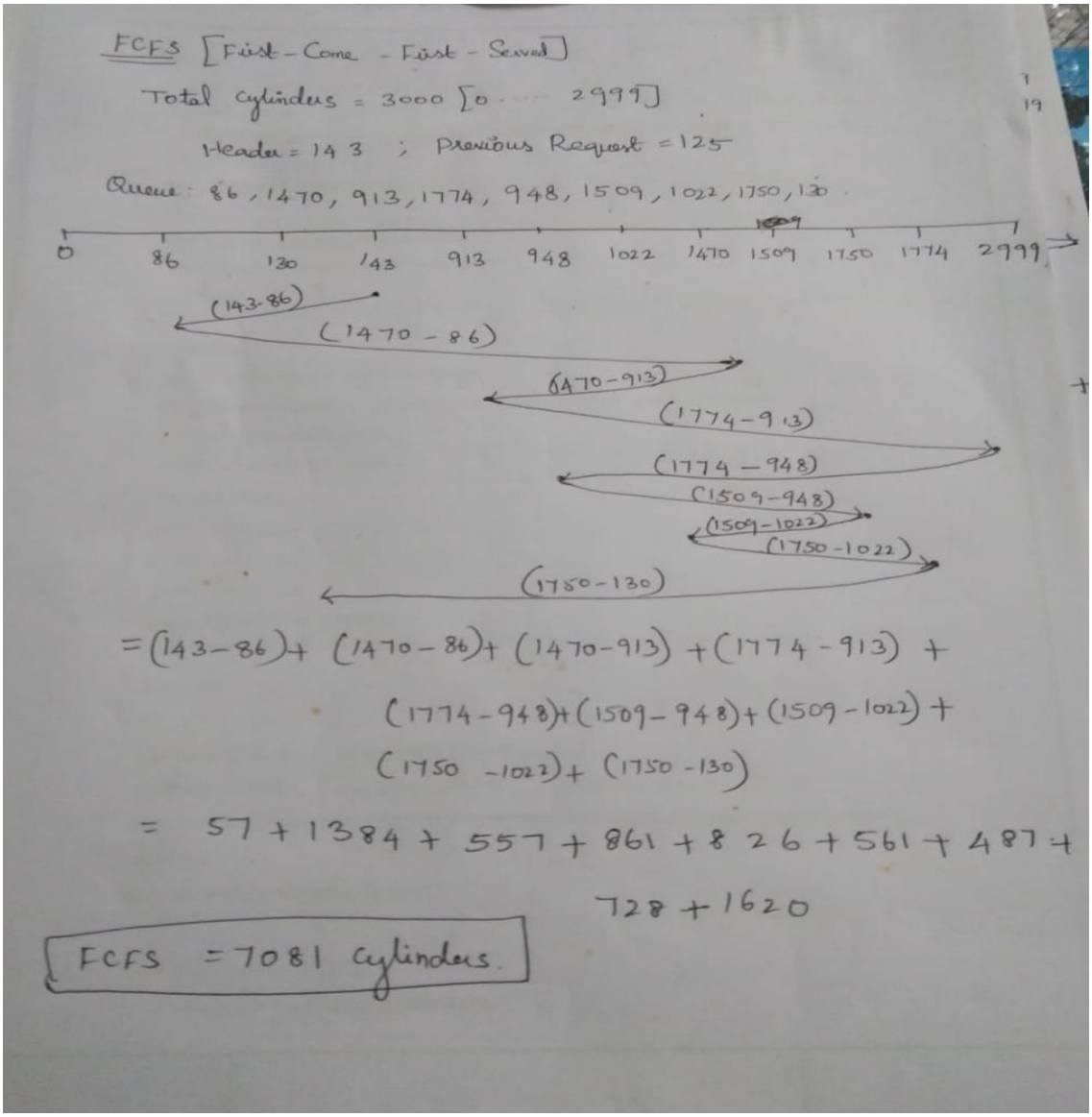
The best-fit algorithm performed the best of the three algorithms, as it was the only algorithm to meet all the memory requests.

17. **Suppose that a disk drive has 3000 cylinders. Numbered 0 to 2999 the drive is currently serving a request at cylinder 143. And the previous request was at 125. The queue of pending requests in FIFO order is 86,1470, 913,1774,948,1509,1022,1750,130. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms?**

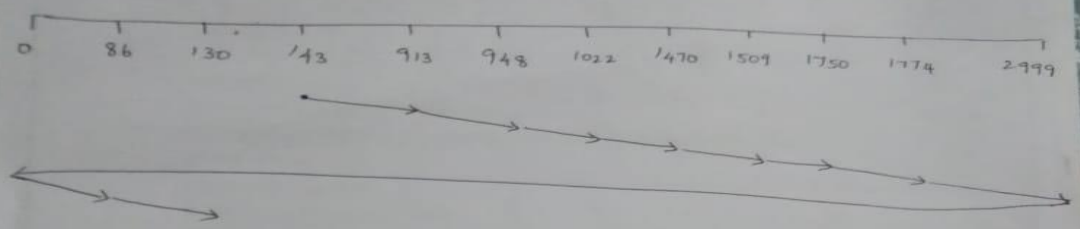
a) FCFS

b) C-SCAN

c) C-LOOK



C-SCAN:



$= (913-143) + (948-913) + (1022-948) + (1470-1022) + (1509-1470) +$

$(1750-1509) + (1774-1750) + (2999-1774) + (2999-0) +$

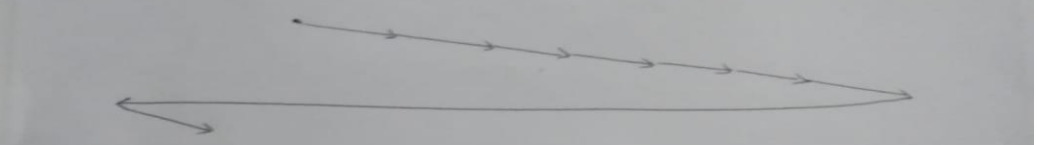
$(86-0) + (130-86)$

$= 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1225 +$

$2999 + 86 + 44$

C-SCAN = 5985 cylinders

C-LOOK



$= (913-143) + (948-913) + (1022-948) + (1470-1022) + (1509-1$

$(1750-1509) + (1774-1750) + (1774-86) + (130-86)$

$= 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1688 + 44$

C-LOOK = 3363 cylinders

18. a) list out the file types in file system interface

The following is the types of the files. A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts-a name and an extension, usually separated by a period character (Figure (a))

FILE TYPE	USUAL EXTENSION	FUNCTION
Executable	Exe,com,bin,or none	Read to run machine language program
Object	Obj,o	Compiled,machine language,not linked
Source code	C,cc,java,pas,asm,a	Source code in various language
Batch	Bat,sh	Commands to the command interpreter
Text	Txt,doc	Texual data,documents
Word processor	Wp,tex,rrf,doc	Various word processor format
Library	Lib,a,so,dill,mpeg,mov,rm	Libraries of the routines for programmers
Print or view	Arc,zip,tar	ASCII or binary file in a format for printing or viewing
Archive	Arc,zip,tar	Related files grouped into A one file sometimes compressed for archiveing or storage
Multimedia	Mpeg,mov,rm	Binary file containing audio or A/V information

b)Enumerate the directory implementation in file system

The selection of directory-allocation and directory-management algorithms has a large effect on the efficiency, performance, and reliability of the file system. Two algorithms are linear and hash table algorithm.

Linear List

The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks. A linear list of directory entries requires a linear search to find a particular entry. This method is simple to program but time-consuming to execute. To create a new file, we must first search the directory to be sure that no existing

file has the same name. Then, we add a new entry at the end of the directory. To delete a file, we search the directory for the named file, then release the space allocated to it.

- To reuse the directory Entry: mark the entry as unused (by assigning it a special name, such as an all-blank name, or with a used-unused bit in each entry)
- Attach it to a list of free directory entries.
- Copy the last entry in the directory into the freed location, and to decrease the length of the directory. A linked list can also be used to decrease the time to delete a file.

Disadvantage:

Through linear search it takes time to search a file.

Directory information is used frequently, and users would notice a slow implementation of access to it.Many operating systems implement a software cache to store the most recently used directory information. A cache hit avoids constantly rereading the information from disk. A sorted list allows a binary search and decreases the average search time.

Hash Table

Another data structure that has been used for a file directory is a hash table. In this method, a linear list stores the directory entries, but a hash data structure is also used. The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time. Insertion and deletion are also fairly straightforward, although some provision must be made for collisions-situations where two file names hash to the same location. The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.

For example, assume that we make a linear-probing hash table that holds 64 entries. The hash function converts file names into integers from 0 to 63, for instance, by using the remainder of a division by 64. If we later try to create a 65th file, we must enlarge the directory hash table-say, to 128 entries. As a result, we need a new hash function that must map file names to the range 0 to 127, and we must reorganize the existing directory entries to reflect their new hash-function values. Alternately, a chained overflow hash table can be used.

19. Illustrate I/O schedule and buffering in kernel I/O schedule.

Kernels provide many services related to I/O. Several services-scheduling, buffering, caching, spooling, device reservation, and error handling-are provided by the kernel's I/O subsystem and build on the hardware and device driver infrastructure.

I/0 Scheduling

To schedule a set of I/O requests means to determine a good order in which to execute them. The order in which applications issue system calls rarely is the best choice. Scheduling can improve overall system performance, can share device access fairly among processes, and can reduce the average waiting time for 1/0 to complete. Here is a simple example to illustrate Application 1 requests a block near the end of the disk, application 2 requests one near the beginning, and application 3 requests one in the middle of the disk. The operating system can reduce the distance that the disk arm travels by serving the applications in order 2, 3, 1. Rearranging the order of service in this way is the essence of I/O scheduling. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

Buffering

A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons. One reason is to cope with a speed mismatch between the producer and consumer of a data stream. Suppose, for example, that a file is being received via modem for storage on the hard disk. The modem is about a thousand times slower than the hard disk. So a buffer is created in main memory to accumulate the bytes received from the modem. When an entire buffer of data has arrived, the buffer can be written to disk in a single operation. Since the disk write is not instantaneous and the modem still needs a place to store additional incoming data, two buffers are used. After the modem fills the first buffer, the disk write is requested. The modem then starts to fill the second buffer while the first buffer is written to disk. By the time the modem has filled the second buffer, the disk write from the first one should have completed, so the modem can switch back to the first buffer while the disk writes the second one. This double buffering decouples the producer of data from the consumer, thus relaxing timing requirements between them.

A second use of buffering is to adapt between devices that have different data-transfer sizes. Such disparities are especially common in computer networking, where buffers are used widely for fragmentation and reassembly of messages. At the sending side, a large message is fragmented into small network packets. The packets are sent over the network, and the receiving side places them in a reassembly buffer to form an image of the source data.

Caching

A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. For instance, the instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, just holds a copy on faster storage of an item that resides elsewhere. Caching and buffering are distinct functions, but sometimes a region of memory can be used for both

purposes. For instance, to preserve copy semantics and to enable efficient scheduling of disk I/O, the operating system uses buffers in main memory to hold disk data. These buffers are also used as a cache, to improve the I/O efficiency for files that are shared by applications or that are being written and reread rapidly. When the kernel receives a file I/O request, the kernel first accesses the buffer cache to see whether that region of the file is already available in main memory. If so, a physical disk I/O can be avoided or deferred.

Spooling and Device Reservation

A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each application's output is spooled to a separate disk file. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in-kernel thread. In either case, the operating system provides a control interface that enables users and system administrators to display the queue, to remove unwanted jobs before those jobs print, to suspend printing while the printer is serviced, and so on.

Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors, so that a complete system failure is not the usual result. Devices and 1/0 transfers can fail in many ways, either for transient reasons, such as a network becoming overloaded, or for "permanent" reasons, such as a disk controller becoming defective. Operating systems can often compensate effectively for transient failures. For instance, a disk read() failure results in a read() retry, and a network send () error results in a resend (), if the protocol so specifies. As a general rule, an I/O system call will return 1 bit of information about the status of the call, signifying either success or failure. In the UNIX operating system, an additional integer variable named errno is used to return an error code one of about 100 values-indicating the general nature of the failure. For instance, a failure of a SCSI device is reported by the SCSI protocol in terms of a sense key that identifies the general nature of the failure, such as a hardware error or an illegal request; an additional sense code that states the category of failure, such as a bad command parameter or a self-test failure; and an additional sense-code qualifier that gives even more detail, such as which command parameter was in error, or which hardware subsystem failed its self-test.

20. Explain the kernel architecture of a linux system in detail with a neat diagram.

Linux resembles any other traditional, non microkernel UNIX implementation. It is a multiuser, multitasking system with a full set of UNIX-compatible tools. Linux's file system adheres to traditional UNIX semantics, and the standard UNIX networking model is implemented fully.

Components of a Linux System

The Linux system is composed of three main bodies of code, in line with most traditional UNIX implementations:

1. Kernel. The kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.

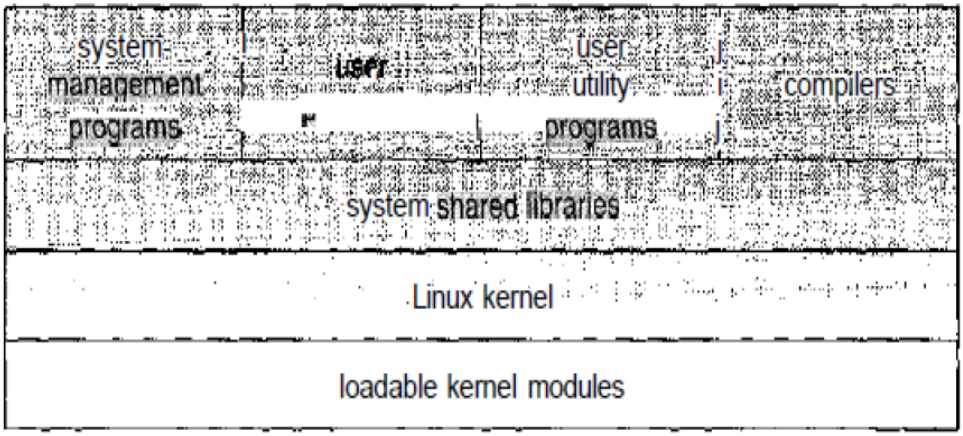
2. System libraries. The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality that does not need the full privileges of kernel code.

3. System utilities. The system utilities are programs that perform individual, specialized management tasks. Some system utilities may be invoked just once to initialize and configure some aspect of the system;

others—known as *daemons* in UNIX terminology—may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals, and updating log files.

Figure illustrates the various components that make up a full Linux system. The most important distinction here is between the kernel and everything else. All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer.

Linux refers to this privileged mode as **kernel mode.** Under Linux, no user-mode code is built into the kernel. Any operating-system-support code that does not need to run in kernel mode is placed into the system libraries instead.



Kernel Modules

The Linux kernel has the ability to load and unload arbitrary sections of kernel code on demand. These loadable kernel modules run in privileged kernel mode and as a consequence have full access to all the hardware capabilities of the machine on which they run. In theory, there is no restriction on what a kernel module is allowed to do; typically, a module might implement a device driver, a file system, or a networking protocol.

Kernel modules are convenient for several reasons. Linux's source code is free, so anybody wanting to write kernel code is able to compile a modified kernel and to reboot to load that new functionality; however, recompiling, relinking, and reloading the entire kernel is a cumbersome cycle to undertake when you are developing a new driver. The module support under Linux has three components:

1. The **module management** allows modules to be loaded into memory and to talk to the rest of the kernel.

2. The **driver registration** allows modules to tell the rest of the kernel that a new driver has become available.

3. A **conflict-resolution mechanism** allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

PROCESS MANAGEMENT

A process is the basic context within which all user-requested activity is serviced within the operating system.

The fork() and exec() Process Model

The basic principle of UNIX process management is to separate two operations: the creation of a process and the running of a new program. A new process is created by the fork() system call, and a new program is run after a call to exec().

Process Identity

A process identity consists mainly of the following items:

- Process ID (PID). Each process has a unique identifier. PIDs are used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process. Additional identifiers associate the process with a process group (typically, a tree of processes forked by a single user command) and login session.
- Credentials. Each process must have an associated user ID and one or more group IDs that determine the rights of a process to access system resources and files.
- **Personality.** Process personalities are not traditionally found on/UNIX systems, but under Linux each process has an associated personality identifier that can modify slightly the semantics of certain system calls.

Process Environment

A process's environment is inherited from its parent and is composed of two null-terminated vectors: the argument vector and the environment vector. The **argument vector** simply lists the command-line arguments used to invoke the running program; it conventionally starts with the name of the program itself.

The **environment vector** is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values. The environment is not held in kernel memory but is stored in the process's own user-mode address space as the first datum at the top of the process's stack.

Process Context

process context is the state of the running program at any one time; it changes constantly.

Processes and Threads

flag	meaning'
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Scheduling

Scheduling is the job of allocating CPU time to different tasks within an operating system.

Process Scheduling

Linux has two separate process-scheduling algorithms. One is a time-sharing algorithm for fair, preemptive scheduling among multiple processes; the other is designed for real-time tasks, where absolute priorities are more important than fairness.

MEMORY MANAGEMENT

Memory management under Linux has two components. The first deals with allocating and freeing physical memory—pages, groups of pages, and small blocks of memory. The second handles virtual memory, which is memory mapped into the address space of running processes.

FILE SYSTEMS

The Linux kernel handles all these types of file by hiding the implementation details of any single file type behind a layer of software, the virtual file system (VFS). Here, we first cover the virtual file system and then discuss the standard Linux file system—ext2fs.

The VFS defines four main object types:

- An **inode object** represents an individual file.
- A **file object** represents an open file.
- A **superblock object** represents an entire file system.
- A**dentry object** represents an individual directory entry.