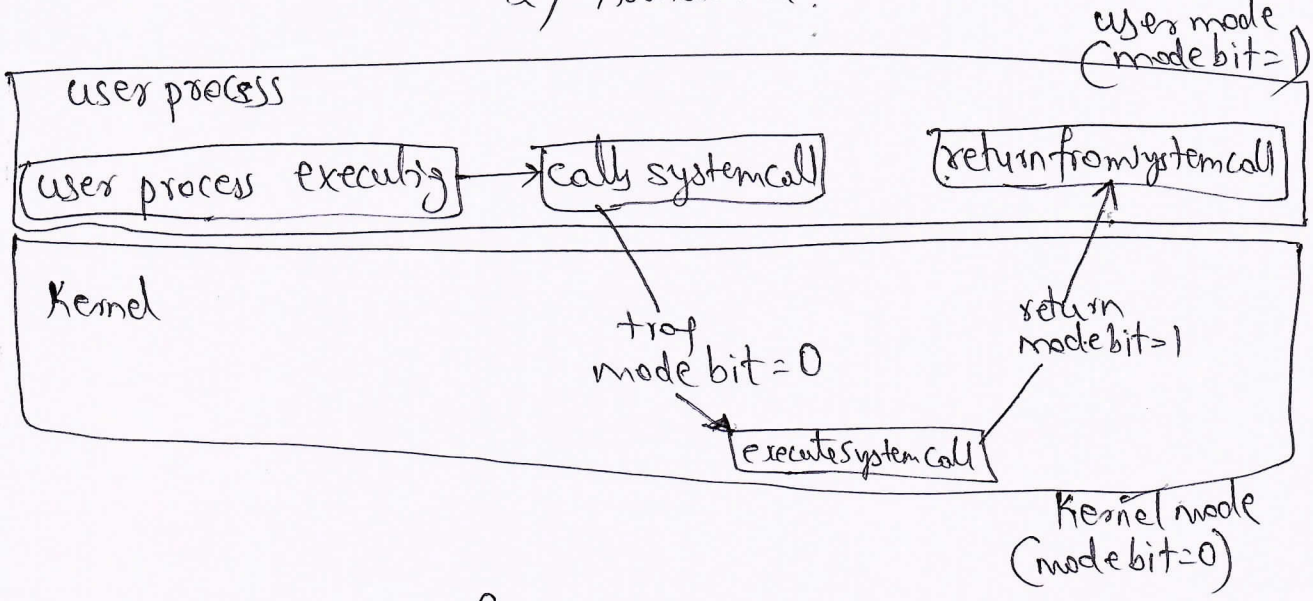
Dr. Naveenkumar T. Rudrapp

Operating System solutions

1) a) Operating system is an interface between the computer hardware and the user. O.S is designed for convienience, efficiency or for both.

Dual Mode of OS with diagram.

There are 2 modes 1) User Mode
2) Kernel Mode.



OS requests service from the user application via system call.

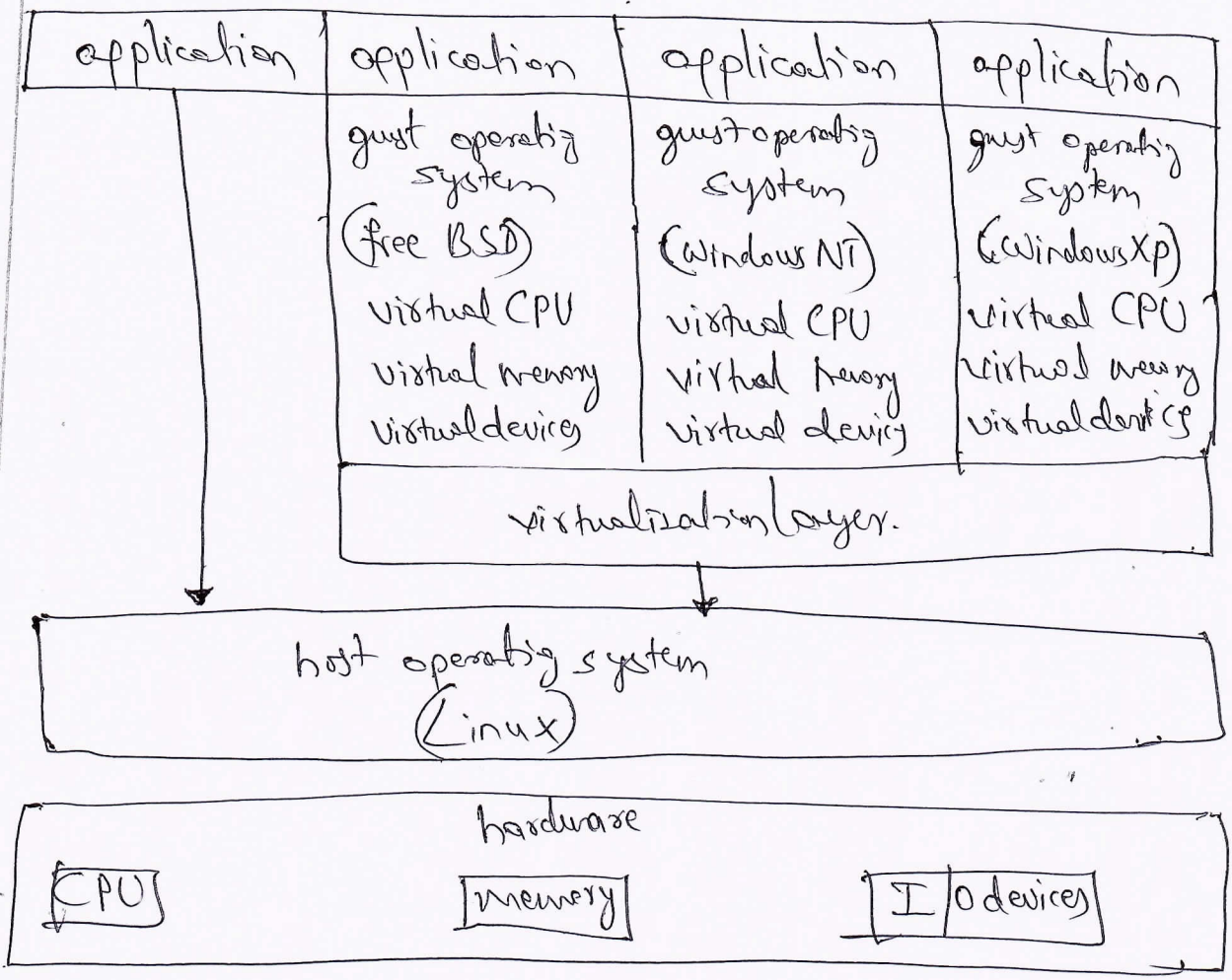
System boot is the Kernel mode called the hardware mode where it switches to O.S & starts user applications in user mode System switches to User mode before passing the control by setting the mode bit to 1 to user program.

Purpose of dual mode 1) Protecting O.S from errant users.
2) Errant users from one another.

H/W executes privileged instructions in user mode.

Switching to Kernel mode is privileged instruction. It consists of I/O control, timer management and interrupt management.

1) c) V-MWare architecture



Writing virtualization tool to run in usermode as an application on top of the Operating System.
Vm running in this tool believe that they are running on bare hardware but infact are running inside a user-level application.
VM ware workstation abstracts Intel X86 and compatible hardware into isolated virtual machines.
VM ware runs as an application ona host O.S such as Windows or linux and allows host system to concurrently run several different guest O.S as independent virtual machines.

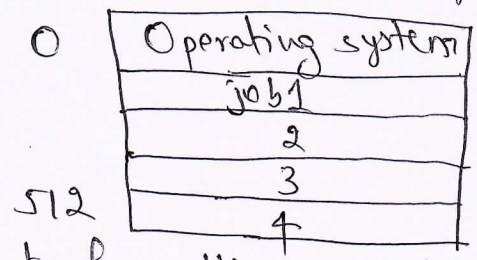
The physical disk the guest owns and manages is a file within file system of host O.S.
Scenario's show how virtualization can improve efficiency of system administration & system resources

1) b) Distinguish between

(i) Multiprogramming and Multitasking.

Single users frequently have multiple programs running.

Multiprogramming increases CPU utilization by organizing jobs (code + data) so that CPU always has one to execute.



Memory layout of multiprogramming systems.

Jobs are kept on the disk in the job pool.

Job pool consists of all processes residing on the disk awaiting allocation of Main memory.

Main memory is a subset of Job pool.

OS picks 1 Job from main memory & executes. Job waits for some task, I/O operation. OS picks 2nd Job for execution

It does not provide for user interaction.

Eventually Job1 completes I/O operation & gets CPU back.

Conclusion: CPU is never idle.

It provides an environment for effective CPU utilization like CPU, Memory and Peripheral devices.

Provides an environment for job switching.

Time sharing systems provide a file system & hence needs disk management. Time sharing systems are also known as Multitasking systems.

It is a logical extension of multiprogramming.

CPU executes multiple jobs by switching among them.

It provides for user interaction.

During switching users can interact with each program while it is running. called the hands-on computer system.

It provides direct communication between the user and the system.

ii) Multiprocessor system and clustered System.

Multiprocessor system

Have 2/more Processors in close communication by sharing 1) Computer bus
2) clock.
3) Memory
4) Peripheral devices.

Advantages

1) Increased throughput
Speedup ratio with N processors is not N but $< N$
(∵ some overhead incurred in keeping all the processors working correctly)

2) Economy of scale.
Cost effective < when compared to multiple sigle processors.
(∵ sharing of peripherals, mass storage, power supplies).

3) Increased reliability.
Functionality distributed properly among several processors, failure of 1 processor does not halt the system but slows it down.
Graceful degradation: Continue providing service proportional to level of surviving hardware.
Fault tolerant: Can suffer failure of any single component.

Clustered System.

Clustered systems
Gather together multiple CPU's to accomplish computational work.
Clustered computers share storage and are closely linked via LAN.
Clustering used to provide high availability service.
Each node can monitor 1/more of others over the LAN.
If the monitored machine fails, monitoring machine can take ownership of its storage.
Users see only brief interruption of the service

Asymmetric clustering
1 machine is in hot standby mode, others running applications, if server fails hot standby machine becomes the server.

2/more machines are running applications & are monitoring each other. Uses all available hardware.

2 b) Different computing environments:-

1) Traditional computing.

Computers connected to network services, print services where remote access was lagging.

Terminals were attached to mainframes with less remote access and portability options.

Company portals provide web accessibility to internal users.

Network computers understand web based computing.

Handheld computers/PDA's can connect to wireless networks to access company's web portal.

2) Client-Server Computing

Type 1: Compute Servers

Interface to which client can send a request to perform an action.

Ex READ data. Server executes action, sends back result to the client. Ex DB system.

Type 2: File servers

where clients can create/update/read/delete files.

Ex Web server that delivers files to clients running web browsers. Server down whole [E-S] down.

3) Peer-to-peer Computing

No client server concepts. All machines connected to n/w act as clients/act as servers. All nodes are called peers.

Client requesting server responding.

4) Web-based computing

P.C's workstations handheld PDA's, cell phones provide access.

Enhanced emphasis on networking, Networked devices are wired/wireless. Faster n/w connectivity is achieved through

2)a) Operating system services

1) User interface:

Command line interface which uses text commands & a method for entering them.
Batch interface, commands and directives to control those commands are entered into files, and those files are executed. commonly used is graphical user interface.

2) Program execution:-

~~Program~~ System must be able to load a program into memory and to run that program. Program should be able to terminate normally or abnormally.

3) I/O operations

A running program may require I/O, which may involve file or an I/O device. Special functions may be desired for efficiency & protection.

4) Filesystem manipulation

Programs need to read and write files and directories. They also need to read and write files to directories/ information from one source to another. Various operations are create and delete them. Some programs include permission management to allow or deny access to files or directories based on file ownership.

5) Communications:

1 process needs to exchange information with another process. There are 2 types of communication Shared memory & message passing.

6) Error detection

OS should be constantly aware of possible errors. Errors may occur in CPU and memory hardware in I/O devices. Sharing computer resources like

1) Resource allocation 2) Accounting 3) Protection & Security.

2) c) System calls

They act as the interface between the OS & the underlying services.

System calls high level tasks are written in C routines and C++ and Low level tasks are directly accessed through hardware.

There may be a copy system call to read data from file F_1 and copy data to file F_2. In addition the system calls provide error detection & prevention mechanisms for smooth functionality.

Systems execute 1000's of system calls / second.

API → Application programming interfaces like POSIX API, win32 API, Java API for JVM.

They provide run time support system for PL.

Conclusion: Most of the details of the OS interface are hidden from the programmer by API + managed by run time support library.

Types of system calls.

1) Process control. System calls for normal and abnormal termination. Dump written to disk & examined by the debugger. OS should transfer control to the command interpreter.

2) File management: Create file, delete file. File to be created, should be opened & used to read, write, reposition the cursor and close the file. For directories file attributes are filename, file type, protection codes, accounting information, get file attributes, set file attributes, calls for file move, file copy

3) Device Management

Process needs several resources, main memory, disk drives, and access to files. These resources available can be granted and control returned to user process. if not available wait till resources are available. Physical devices, disk drives, virtual devices, shared devices, multiple users need to request / release device. after completion of the task. UI can make files + devices appear similar, even though underlying system calls are dissimilar.

4) Information maintainance

System calls exist for transferring information from user process to operating system and other information about the system ~~calls~~ like:
1) No of current users 2) Version of OS 3) Amount of free memory 4) Disk space 5) System calls for debugging program, 6) Dump memory.
Program trace lists each system call as it is executed. Trap is executed by the CPU after every instruction. Time profile requires tracing facility at regular time ~~intervals~~ intervals value of program counter is recorded. OS keeps information about all processes & system calls used to access information.

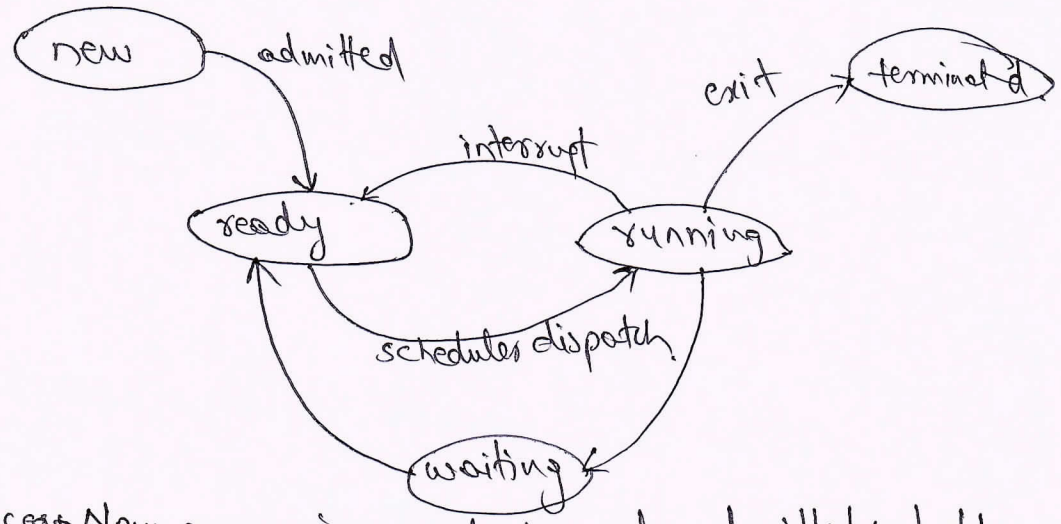
5) Communication

Message passing model: Exchange messages directly/indirectly → the connection is opened. Source address & destination address to be known as to check whether the process resides on other system or on the same system.

6) Protection

Controlling access to resources. Multiprogrammed computer system with several users for set permission, get permission, allow user and deny user.

9) 3.a) A program in execution is called a process.
Processes are of 2 types 1) OS processes executing system code.
2) User processes executing user code.



~~Process~~ New process is created and admitted into the ready queue. Scheduler dispatches it to the cpu for running. If an interrupt occurs, the process is again put back into the ready queue.
The process enters the waiting state for some event to occur like the I/O completion or recieving of a signal.
~~After~~ the process waits in the ready queue to be assigned to a processor. The process has completed execution & terminates normally.

3) b) A process is co-operating if it can affect or be affected by other processes executing in the system. Co-operating processes require an interprocess communication mechanism that will allow them to exchange data and information.
Process A and process B share the address space to synchronize actions. It is useful in distributed environment where process A and B reside on different computers.

Message passing facility:-

It requires 2 operation i.e send(message) and recieve(message)

Messages are of 2 types:

Fixed size where system level implementation is simple but programming is more difficult.

Variable size where system level implementation is complex but programming is simple.

Direct Communication

1) send (P, message) ⟶ send a message to process P
2) receive (Q, message) ⟶ reciere a message from process Q.

Communication link.

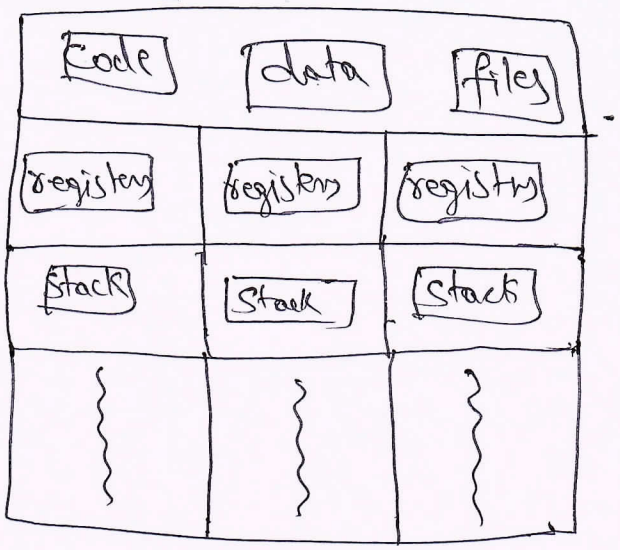
1) link established automatically between pair of processes.
2) link is exactly between 2 processes.
3) 1 link between each pair of processes.

This is symmetric addressing where both sender & reciever should name each other to communicate.

3) c) ## Context switching

Interrupts cause the O.S to change a CPU from its current task. and to run a Kernel routine. When an interrupt occurs the system needs to save the current context. of the process running on the CPU. so that it can restore the context. when its processing is done. Context includes value of the CPU registers, the process state and memory management information. Switching CPU to another process requires performing a state save of the current state of the current process and a state restore of a different process. This is context switch. In

4.a) Multi threaded Process



Process creation is more time consuming. Hence it is more efficient to use one process that contains multiple threads. The server creates separate thread that listens for client requests. When a request is made rather than creating another process, the server will create a new thread to service the request and resume listening for additional requests.

Benefits of multithreaded programming.

1) Responsiveness: Multithreading an interactive application may allow a program to continue running even if part of it is blocked thereby increasing responsiveness to the user.

2) Resource sharing: Process may only share resources through techniques such as shared memory or message passing. Such techniques should be arranged by the programmer.

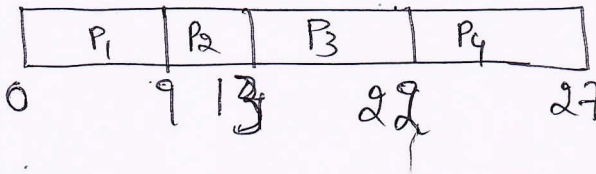
3) Economy: Threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.

4) Scalability: In a multiprocessor architecture where threads may be running in parallel on different processors. Multithreading on a Multi CPU machine increases parallelism.

4) b) Gant chart.

Process	Arrival	Burst time	Priority
P1	0	9	3
P2	1	4	2
P3	2	9	1
P4	3	5	4

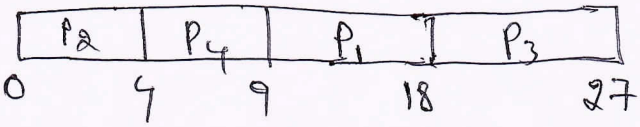
FCFS



$P_1 = 0$
$P_2 = 9$
$P_3 = 13$
$P_4 = 22$

Avg Waiting Time $= \frac{44}{4} = 11$ ms

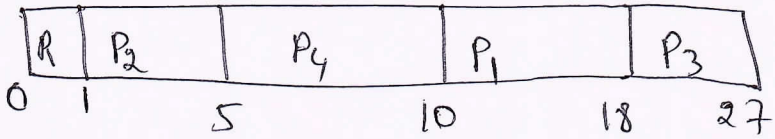
SJF non preemptive.



$P_2 = 0$
$P_4 = 4$
$P_1 = 9$
$P_3 = 18$

Avg Waiting Time $= \frac{31}{4} = 7.75$ ms

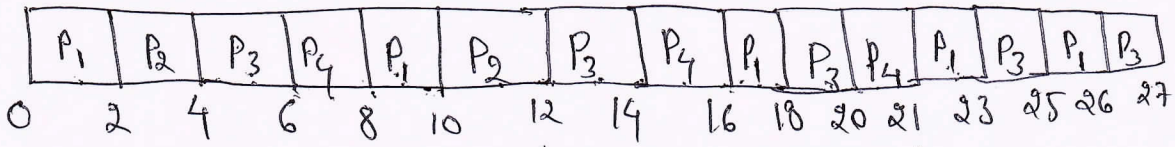
SRTF



$P_1 = 9$
$P_2 = 1 - 1 = 0$
$P_3 = 18 - 2 = 16$
$P_4 = 5 - 3 = 2$

AWT $= \frac{27}{4} = 6.75$

RRSA $q = 2$ ms.



$P_1 = 6 + 6 + 5 + 2 = 19$

$P_2 = 2 + 6 = 8$

$P_3 = 4 + 6 + 4 + 3 + 3 = 20$

$P_4 = 6 + 6 + 4 = 16$

AWT $= \frac{19 + 8 + 20 + 16}{4}$

$= 15.75$

5)a) Critical section

System consisting of n processes $\{P_0, P_1, \ldots P_{n-1}\}$. Each process has a segment of code, called a critical section in which the process may be changing common variables, updating a table, writing a file and so on. Feature is that when one process is executing in its critical section, no 2 processes are executing in their critical sections at the same time.

```
do {
    entry section
        critical section
    exit section
        remainder section.
} while (TRUE)
```

Requirements for the solution to critical section problem.

1) Mutual exclusion: If process P_i is executing in its critical section, then no other processes can be executing in their critical sections

2) Progress: Only those processes not executing in their remainder sections can participate in deciding which will enter its critical section next. this selection cannot be postponed.

3) Bounded waiting :- Limit the no of times other processes are allowed to enter their critical section.

Peaterson's solution

It does not guarantee that it works correctly on load & store instructions. Peterson's solution is restricted to 2 processes that alternate execution between their critical sections & remainder sections. The processes are numbered P_0 and P_1. For convenience, when presenting P_i we use P_j to denote the other process, i.e j equals to 1-i

P.S stores 2 data items

```
int turn;
boolean flag[2];
```

turn → whose turn it is to enter critical section.

turn == i P_i enters critical section.

flag[2] → indicates P ready to enter critical section.

if flg[i] = true.

1) Mutual exclusion: P_i enters critical section only iff

flag[j] == false or

turn == i

if P_i & P_j executing in critical section at the same time then flag[0] == flag[1] == true.

P_i does not change value of the variable turn during execution.

P_i enters critical section (Tracking progress) after atmost one entry by P_j (bounded waiting).

5) b) Reader's-Writer's problem using semaphores.

Some process wants to only read value → READERS.
Some process wants to read & write value → Writers.
2 Readers reading the same value. No adverse effect.
Problem arises. if the processes are RW, WR, WW.?

To overcome this problem writers have exclusive access to the shared DB only while writing. called RW problem.

Case 1: First-readers-writers problem

No reader kept waiting unless writer has permission to write shared object. Multiple readers permitted to read parallely.

Case 2: Second readers-writers problem.

Writer ready — Performs Write as soon as possible
== Writer waiting to access shared object
No readers not permitted to read.

Case 1 or case 2 ⟶ May lead to Starvation.

Case 1 : Disadvantage ⟶ Writers Starve.

Case 2 : Disadvantage ⟶ Readers Starve.

Readers process share Data structures

① semaphore mutex, wrt to

② int read count;

① ⟶ initialized to 1.

② ⟶ initialized to 0.

```
do {
  wait (wrt)
//writing is performed.
  ...
signal (wrt);
} while (TRUE);
```

Writer process.

```
do {
 wait (mutex);
 read count ++;
 if (readcount == 1)
    wait (wrt);
 signal mutex;
  ...
// reading is performed
  ...
wait (mutex);
read count --;
if (read count == 0)
signal (wrt);
signal (mutex);
} while TRUE
```

Reader process

Reader-Writer Locking ⟶ specify mode of the lock.

Read Lock ⟶ mode 1.

Write Lock ⟶ mode 2.

Readers ⟶ need to read shared data requests for reader-writer lock in READ mode.

6a) Deadlock.

A set of processes are in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. The events with which we are mainly concerned are resource acquisition & release. The resources may be either physical resources or logical resources.

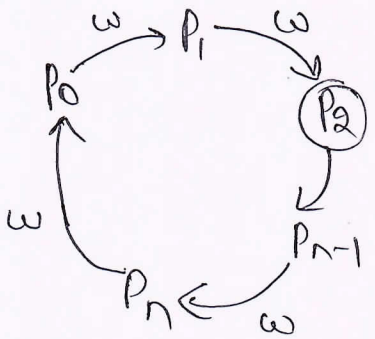
Necessary condition for a deadlock to occur are:

1) Mutual Exclusion: Only 1 P at a time can use resource 2nd proces requests. Requesting process delayed until resource released.

2) Hold and wait: P holding atleast 1 Resource waiting to acquire other resources held by $P_2, P_3 \ldots P_n$.

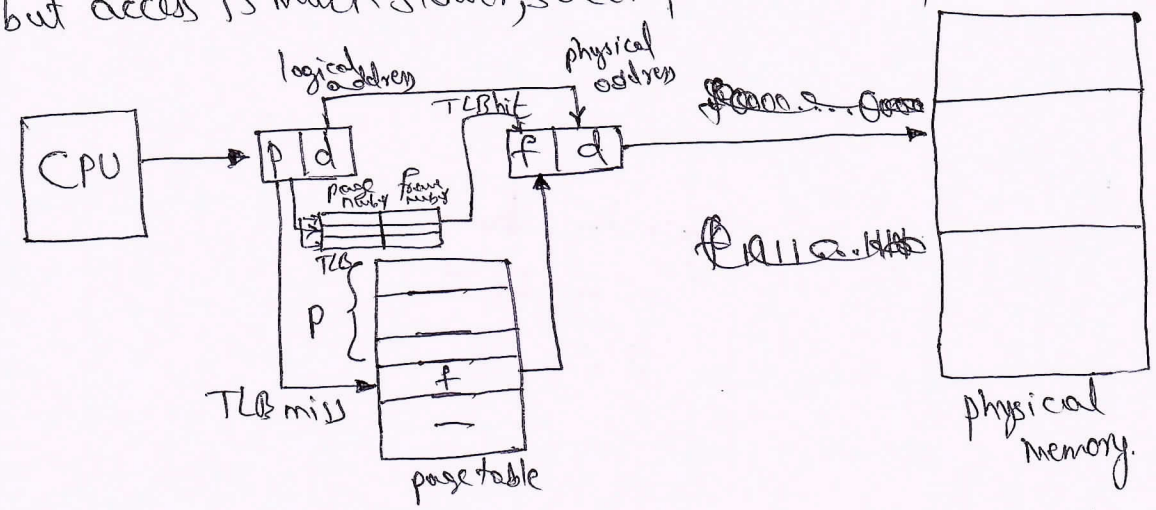
3) No preemption: Resource released volantarily by process holding it. After proces has completed task.

4) Circular wait: set $= \{P_0, P_1, \ldots P_n\}$.



For a deadlock to occur all the 4 conditions must hold.

7)a) Paging is a memory-management scheme that permits the physical address space of a process to be noncontiguous. Paging avoids external fragmentation and the need for compaction. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store, most memory management schemes used before the introduction of paging suffered from this problem. The problem arises because, when some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store. The backing store has the same fragmentation problems discussed as that of main memory but access is much slower, so compaction is impossible.



The page table is implemented as a set of dedicated registers. These registers should be built with very high speed logic to make the paging address translation efficient.

CPU dispatcher reloads these registers, just as it reloads other registers.

The address consists of 16 bits and the page size is 8 Kb. The page table thus consists of 8 entries that are kept in fast registers.

Page table is kept in memory and a page-table base register (PTBR) points to the page table.

changing page table requires changing only one register thereby reducing context-switch time.

A special, small, fast-lookup hardware cache called a translation look-aside buffer (TLB). Each entry in TLB consists of two parts: a key and a value. When the associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. Search is fast but hardware is expensive. Number of entries in a TLB is small with numbering between 64 and 1024.

The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. ~~The TLB is associative high-speed memory~~ If the page number is found, its frame number is immediately available and used to access memory.

If the page number is not in TLB (Known as TLB miss), a memory reference to the page table must be made. When the frame number is obtained. we can use it to access memory. In addition, we add the page number & the frame number to the TLB so that they will be found quickly on the next reference. Certain entries cannot be removed from the ~~table~~ TLB Some TLB's store address-space identifiers in each TLB entry. Percentage of times that a particular page number is found in the TLB is called the Hit ratio.

7)b) Simplest method for allocating memory is to divide memory into several fixed size partitions. Each partition may contain exactly one process. The degree of multiprogramig is bound by the number of partitions.

In case of multiple partition method, when a partition is free, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for another process.

In case of variable partition scheme, the O.S Keeps a table indicating which parts of memory are available and which are occupied. All memory is available for user processes and is considered as one large block of available memory, a hole. Memory contains a set of holes of various sizes. As processes enter the system they are put into an input queue. The O.S takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory. When a process is allocated space, it is loaded into memory and it can then compete for CPU time. When a process terminates it releases its memory which the OS may then fill with another process from the input queue.

Commonly used strategies to select a free hole from the set of available holes

1) First fit: Allocate the first hole i.e big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. Stop searching as soon as we find a free hole i.e large enough.

2) Best fit: Allocate the smallest hole i.e big enough. We must search the entire list, unless the list is ordered by size. This produces the smallest left over hole.

3) Worst fit: Allocate the largest hole. Again we must search the entire list, unless it is sorted by size. This strategy produces the largest left over hole, which may be more useful than the smaller left over hole from a best fit approach.

7c) Fragmentation

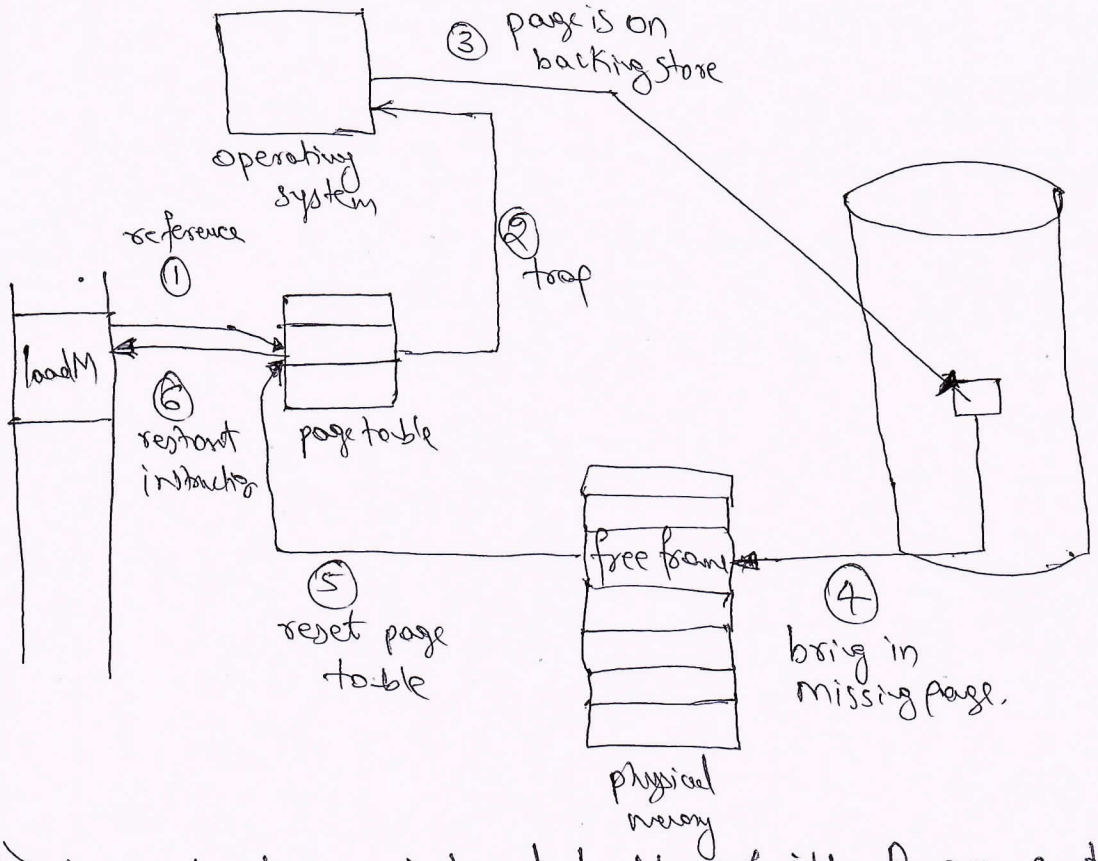
First-fit and best-fit strategies suffer from external fragmentation. As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous, storage is fragmented into a large number of small holes. In the worst case we could have a block of free memory between every 2 process.

First fit and best fit strategy can affect the amount of fragmentation. The left over piece, is end of a free block is allocated by the one on the top or the one on the bottom.

Statistical analysis of first fit, for instance reveals that even with some optimization given N allocated blocks. another .5 N blocks will be lost to fragmentation. This property is known as fifty percent rule.

Overhead to keep track of the holes which are left over will be larger than the hole itself. With this approach memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation.

8a) Steps in handling the page fault:-



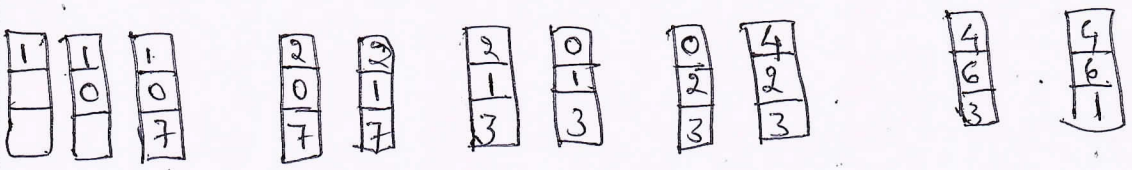
1) We check an internal table with Process control block to determine whether the reference was a valid or an invalid memory access.
2) If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.
3) We find a free frame by taking one from the free frame list.
4) We schedule a disk operation to read the desired page into the newly allocated frame.
5) When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6) We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

8)b) Consider the page reference string:
1, 0, 7, 1, 0 2, 1, 2, 3, 0, 3, 2, 4, 0 3, 6, 2, 1 memory with 3 frames.

Page faults = 10

FIFO

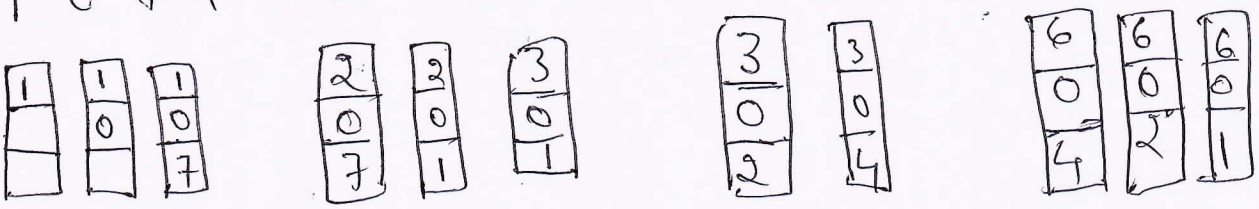
1 0 7 1 0 2 1 2 3 0 3 2 4 0 3 6 2 1



Page fault = 9

Optimal,

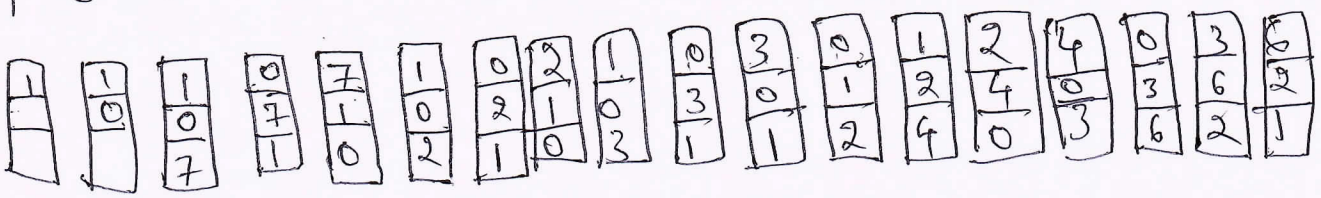
1 0 7 1 0 2 1 2 3 0 3 2 4 0 3 6 2 1



Page faults = 13

LRU

1 0 7 1 0 2 1 2 3 0 3 2 4 0 3 6 2 1



9)a) File.

Operating system provides a uniform, logical view of information storage. It abstracts the physical properties of its storage devices to define a logical storage unit, the file. OS maps files onto physical media and accesses these files via storage devices. A file name is usually a string of characters. Some systems differentiate between uppercase & lower case characters in names.

Different file attributes are:

1) Name: The symbolic file name is the only information kept in human readable form.
2) Identifier: The unique tag, a number that identifies the file within the file system, non human readable name for the file.
3) Type: Information needed for systems that support different types of files.
4) Location: Information is a pointer to a device and to the location of the file on that device.
5) Size: Current size of the file and maximum allowed size are included in this attribute.
6) Protection: Access-control information determines who can do reading, writing, executing and so on.
7) Time, date and user identification: This information may be kept for creation, last modification & last use.

File operations:

1) Creating a file: Step 1: Space in the file system must be found for the file.
   Step 2: An entry for the new file must be made in the directory.
2) Writing a file: To write a file, we make a system call specifying both the name of the file and the information to be written

to the file. System must keep a write pointer to the location in the file where the next write is to take place. Write pointer updated whenever a write occurs.

3) Reading a file: To read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put. System needs to keep a read pointer to the location in the file where the next read is to take place. Current operation location can be kept as a per process current-file-position pointer. Both read and write operations use this same pointer, saving space and reducing system complexity.

4) Repositing within a file: Directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. This operation is known as a file seek.

5) Deleting a file: To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files and erase the directory entry.

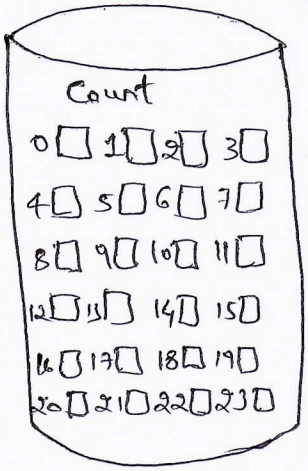
6) Truncating a file: User may want to erase the contents of a file, but keep its attributes. This function allows all attributes to remain unchanged except for file length but lets the file be reset to length zero and its file space released.

9)b) Different allocation methods are

1) Contiguous allocation:

Requires each file to occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk, accessing block b+1 after block b normally requires no head movement. When head movement is needed, head moves from one track to the next. The no of disk seeks required for accessing contiguously allocated files is minimal as the seek time is finally needed.

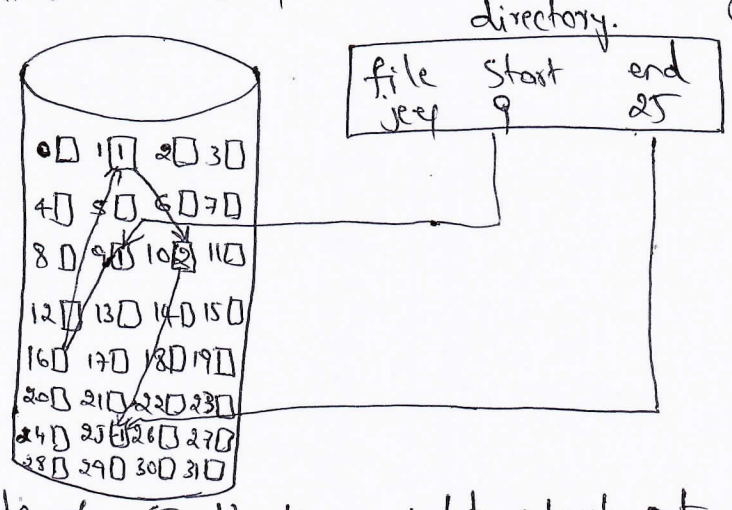
Contiguous allocation of a file is defined by the disk address and length of first block. If the files is n blocks long and starts at location b, then it occupies block b, b+1, b+2, ... b+n-1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file. For sequential access, the file system remembers the disk of the last block referenced and when necessary, reads the next block. For direct access to block i of a file that starts at block b we access block b+i. Both sequential and direct access can be supported by contiguous allocation.



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

2) Linked Allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.



A file of 5 blocks might start at block 9 and continue at block 16, then block 10, and finally block 25. Each block contains a pointer to the next block. These pointers are not made available to the user. Thus if each block is 512 bytes in size & a disk address requires 4 bytes then the user sees blocks of 508 bytes. Size of the file not declared when that file is created. A file can continue to grow as long as free blocks are available.

Disadvantages: It can be used effectively only for sequential access files. To find the i^{th} block of the file, we must start at the beginning of that file and follow the pointers until we get to the i^{th} block.

Space required for pointers. If a pointer requires 4 bytes out of a 512 byte block, then .78 percent of the disk is being used for pointers rather than information. Usual solution is to collect blocks into multiples called clusters, and to allocate clusters rather than blocks.

Files are linked together through pointers. scattered all over the disk, and consider what would happen if a pointer were lost or damaged. A bug in OS may require picking up a wrong pointer

10)a) Model of protection can be viewed abstractly as a matrix, called an access matrix.

Rows of access matrix represent domains and columns represent objects. Each entry in the matrix consists of set of access rights. Because the column defines objects explicitly we omit object name from the access right. Entry access(i,j) defines the set of operations that a process executing in domain D_i can invoke on object O_j.

domain \ object	F_1	F_2	F_3	printer.
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write.	

4 domains D_1, D_2, D_3, D_4

4 objects F_1, F_2, F_3, Laser printr.

Process executing in domain D_1 can read files F_1 and F_3

Processes executing only in domain D_2 can print files on printer.

D_4 can rw file F_1, F_3.

D_3 can read F_2 & execute F_3.

In general process executing in domain D_i can access only those objects specified in row i, as allowed by access matrix entries.

Access matrix can implement policy decisions concerning protection.

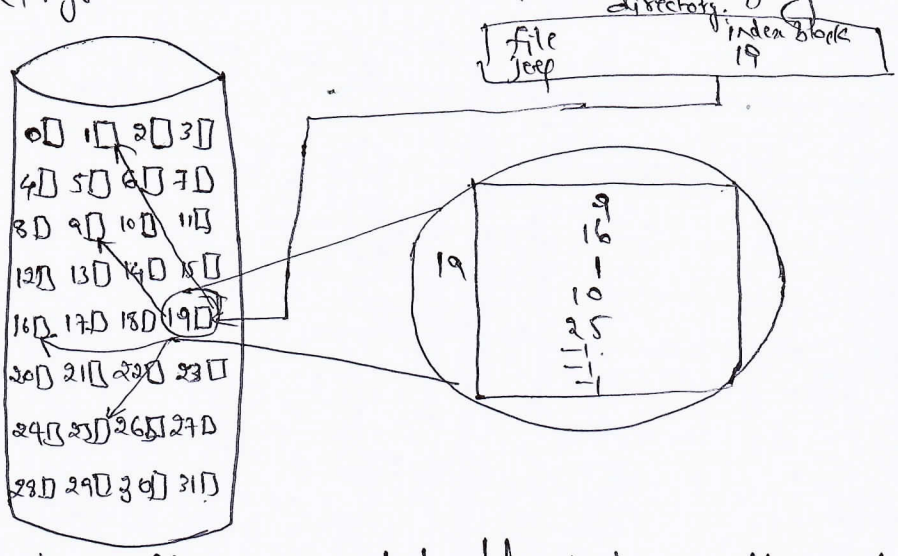
domain \ object	F_1	F_2	F_3	laser printr	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch	~~switch~~		

Process should be able to switch from one domain to another. Switching from domain D_i to domain D_j is allowed iff the access right switch $\in$ access(i,j)

Process executing in D_2 can switch to domain D_3 or to D_4. Process in domain D_4

3) Indexed allocation:

brings all the pointers together into one location called the index block. Each file has its own index block which is an array of disk block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. The directory contains the address of the index block. To find and read the i^{th} block, we use pointer in the i^{th} index-block entry. This scheme is similar to paging scheme.



When the file is created, all pointers in the index block are set to Nil. When the i^{th} block is first written, a block is obtained from the free space manager and its address is put in the i^{th} index block entry.

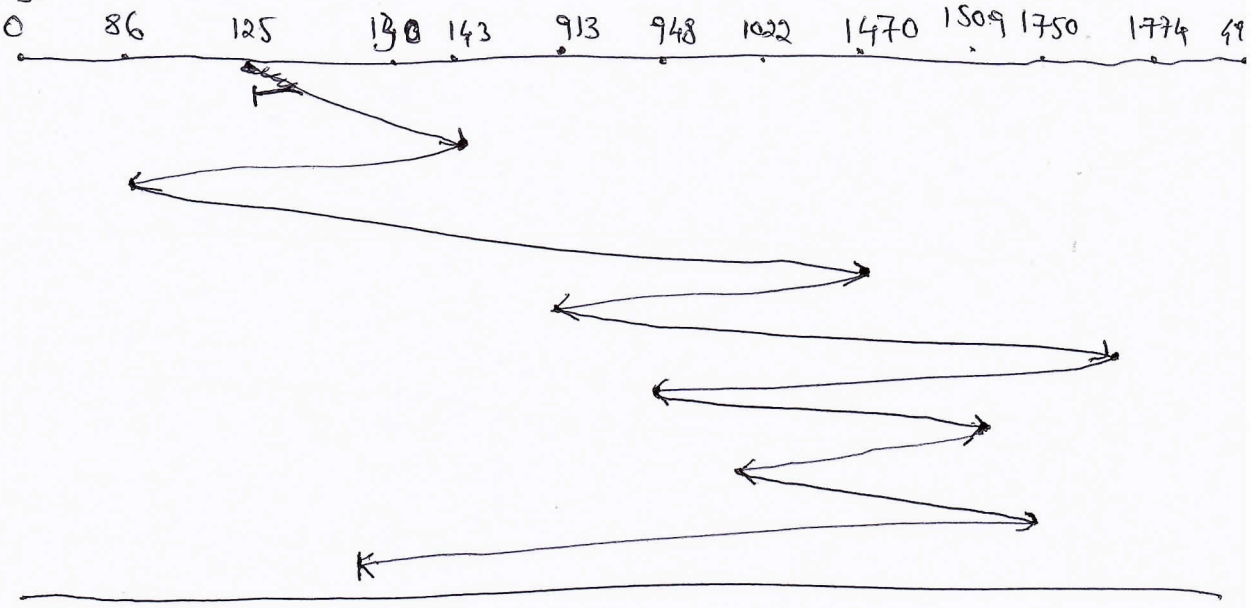
Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.

Indexed allocation suffers from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

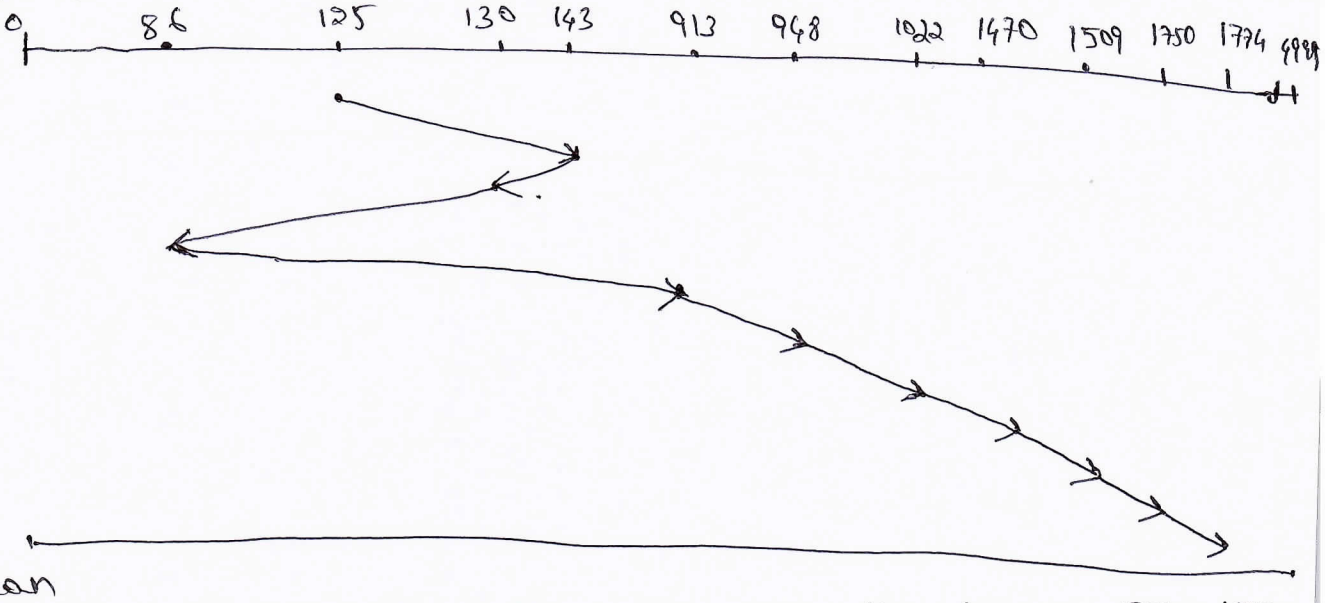
A file consisting of only one or 2 blocks, with linked allocation we lose the space of only one pointer per block. With indexed allocation, an entire index block must be allocated even if only 1/2 pointers will be non-nil.

10)b) Drive has 5000 cylinders numbered from 0 to 4999.
Drive currently serving request 143, previously serviced request at 125. FIFO order 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

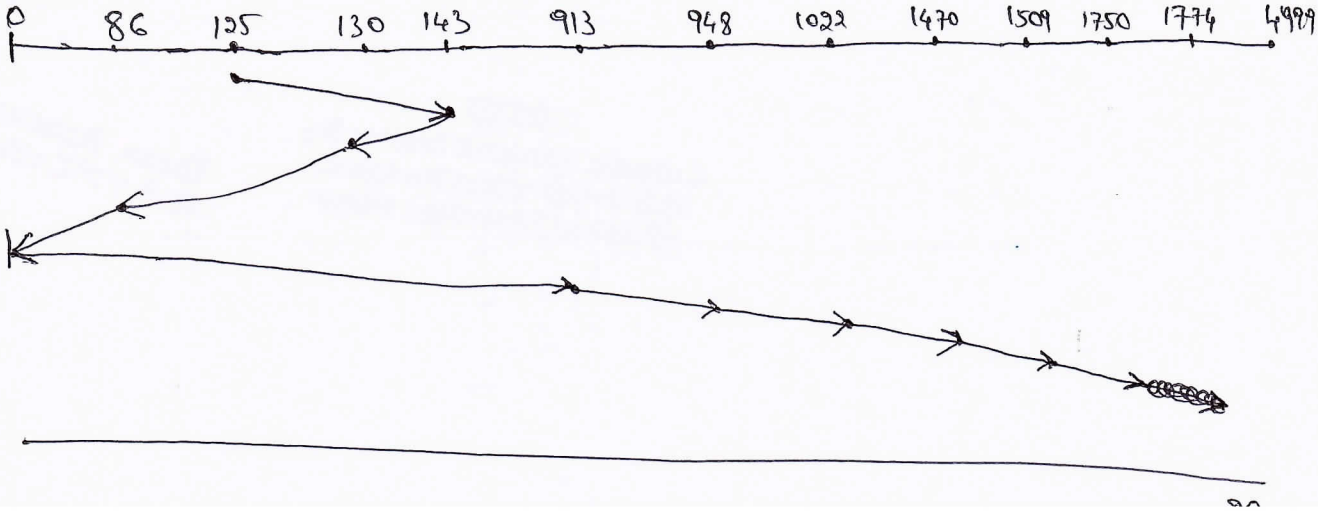
FCFS.



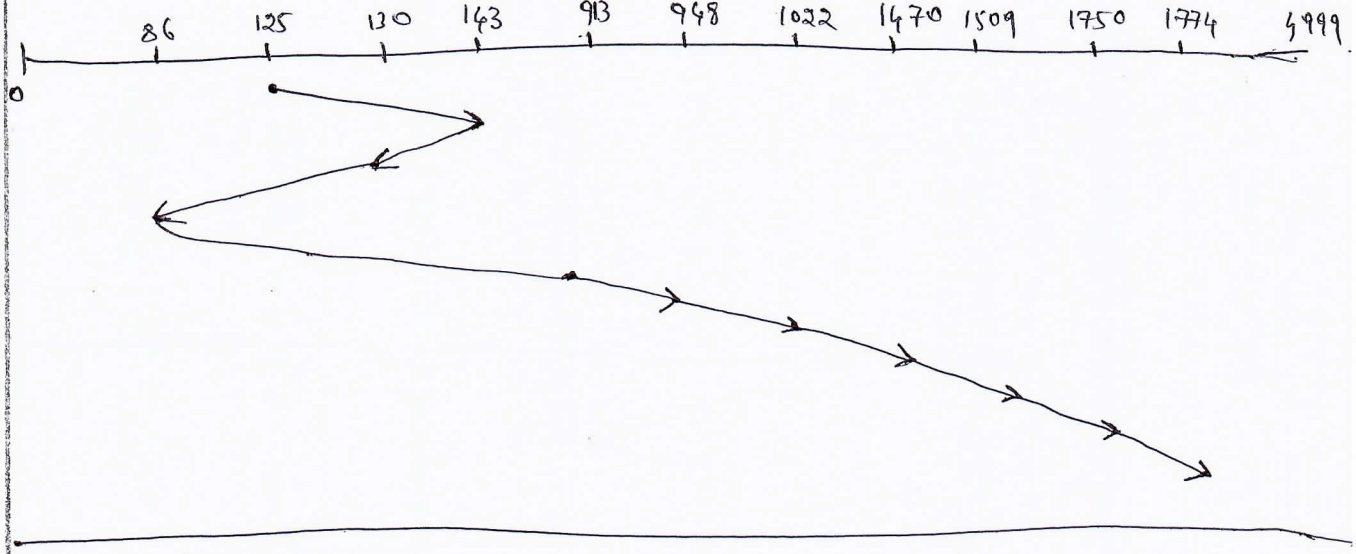
SSTF



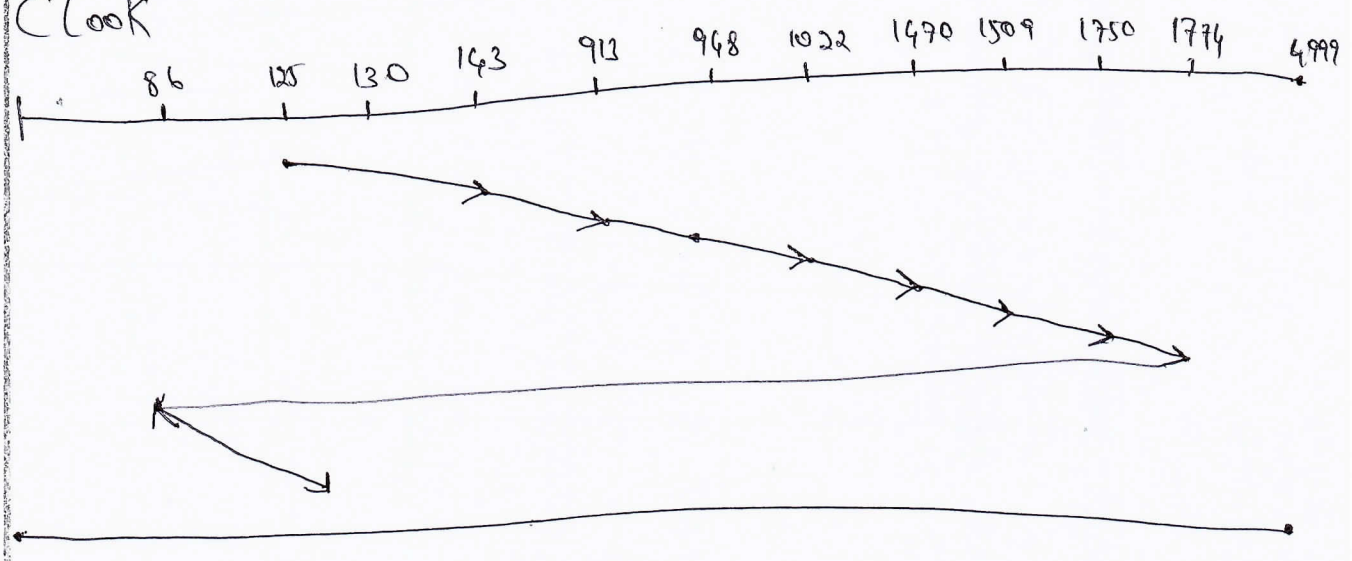
Scan



Look



CLook



29/04/2024

HOD
Computer Science & Engineering
KLS Vishwanathrao Deshpande
Institute of Technology, Haliyal

29/4/24
Dean, Academics
KLS VDIT, HALIYAL

6) b)

Process	Allocation A B C D	Max A B C D	Available A B C D
P_0	2 0 0 1	4 2 1 2	3 3 2 1
P_1	3 1 2 1	5 2 5 2	
P_2	2 1 0 3	2 3 1 6	
P_3	1 3 1 2	1 4 2 4	
P_4	1 4 3 2	3 6 6 5	
	9 9 6 9		

Need Matrix = Max − Allocation

P_0	2	2	1	1
P_1	2	1	3	1
P_2	0	2	1	3
P_3	0	1	1	2
P_4	2	2	3	3

Initial Work = available = (1, 5, 2, 0)

		Work vector
P_0	False	
P_1	False	
P_2	False	3 3 2 1
P_3	False	
P_4	False	

Need P_0 2 2 1 1 $\leq$ 3 3 2 1

Finish matrix: P_0 True

P_0 releases: 2 0 0 1

Work = Work + Allocated

= 3321 + 2001

= 5 3 2 2

P_1 2 1 3 1 $\leq$ 5 3 2 2 Condition False P_1 not executed.

P_2 0 2 1 3 $\leq$ 5 3 2 2 Condition False P_2 not executed

P_3 0 1 1 2 $\leq$ 5 3 2 2 P_3 will be executed.

P_2 requests 0, 1, 1, 3 covered with 3321 cannot be granted because we do not have enough available instances of resource 3

P_3 – True P_3 Allocated 1312

Work = Work + Allocated P_3

= 5322 + 1312

= 6634

Work vector

6	6	3	4

Need P_4 2233 ≤ 6634 P_4 Executed.

P_4 P_4 Allocated 1432

= 6634 + 1432

= 7066

Finish Matrix

P_0 – 1 True

P_1 False

P_2 False

P_3 – 2 True

P_4 – 3 True.

Need P_1 2131 ≤ 71066

P_0 – 1 True P_1 Allocate 3121

P_1 – 4 True Work = Work + Allocated P_1

= 71066 + 3121

= 10 11 8 7

Need P_2 0213 ≤ 10 11 8 7 P_2 Allocate 2103

P_2 – 5 True. Work = 10 11 8 7 + 2103

= 12 12 8 10

System is in safe state Process will be executed in the order

P_0, P_3, P_4, P_1, P_2