UNIT #1.

- SYSTEM CALLS

- Interface b/w the operating system and its application programs.

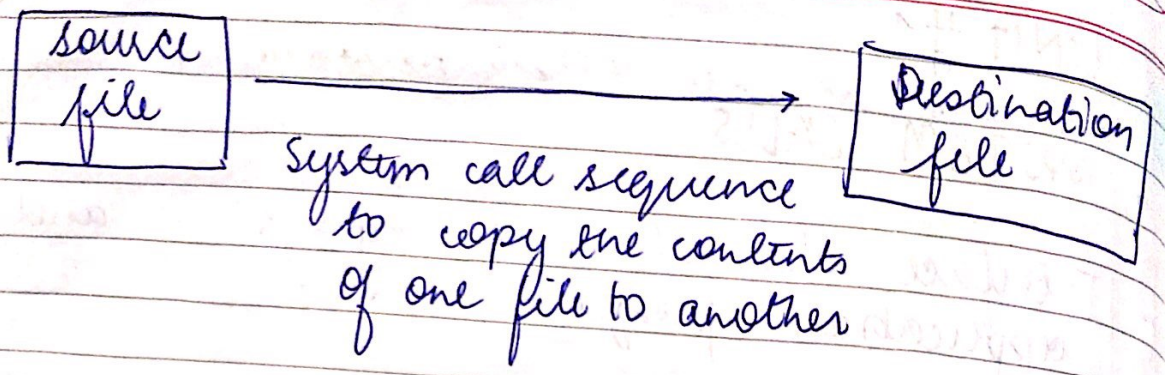
- If some processing is running in user mode and needs a system service, it has to execute a system call instruction to transfer control to operating system.
- Operating system figures out what the calling process wants by the parameters.

- Carries out the system call and then returns control to the instruction following the system call

- Only system calls enter Kernel or other operating system components.

Steps → System call mechanism.
→ Specifying the file
→ Specifying the buffer
→ Specifying the bytes to read

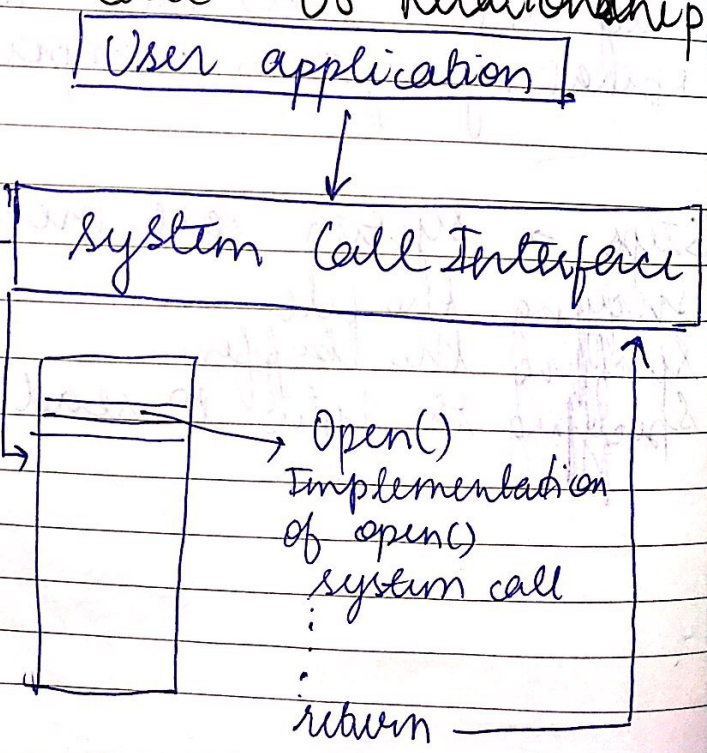
Scanned with CamScanner



System call implementation

- Intially, a number associated with each system call.
- It's interface maintains a table indexed according to these number.
- It just needs to obey API and understand what OS will do as a result call.

API - System Call - OS Relationship.



System Call parameter Passing -

Methods to pass parameters

- Simplest - pass parameter in registers
- Parameters stored in a block or table in memory, and address of block passed as a parameter in register
- Parameter placed, or pushed, onto the stack by the program and popped off the stack by the operating system.

TYPES

1. Process Control
- end, abort
- load, execute
- create process
- terminate process
- get process attributes
- set process attributes
- wait for time
- wait event
- signal event
- allocate & free memory

2. File Management
- create file, delete file
- open, close
- read, write, reposition
- get file attributes
- set file attributes

3. Device Management
- request device
- release device
- read, write, reposition
- logically attach
- detach devices.
- set device attributes
- get device attributes

Scanned with CamScanner

4. Information Maintainence
- get time or date, set time or data
- get system data, set system data.
- get process, file, device attributes
- set process, file, device attributes

5. Communication
- create, delete, communication connection
- send, recieve messages.
- transfer status information
- attach or detach remote device

6. Protection.
- set permission
- get permission
- allow user
- deny user

Scanned with CamScanner

What is an operating system?
- A program that acts as an interface between the user ~~and the~~ and the hardware of the computer.

Goals -
- Execute programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

COMPUTER-SYSTEM ~~concepts~~ COMPONENTS -

1. Hardware.
- Provides basic computing resources. (CPU, memory, I/O devices)

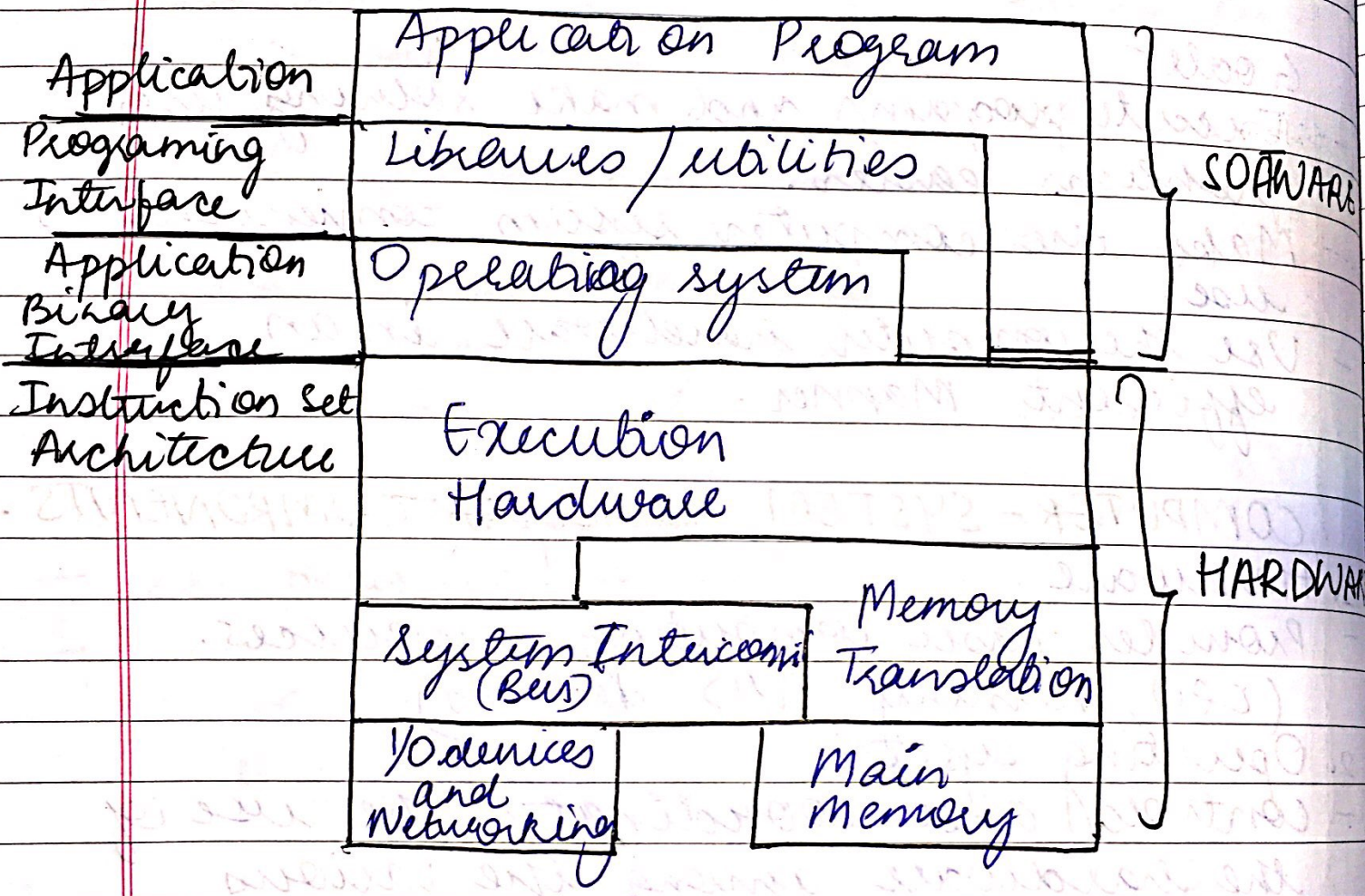
2. Operating System
- controls and coordinates the use of the hardware among the various application programs for various user.

3. Application programs -
- Defins the ways in which the system resources are used to solve the computing problems of the users.
- Compilers, Database systems, video games.

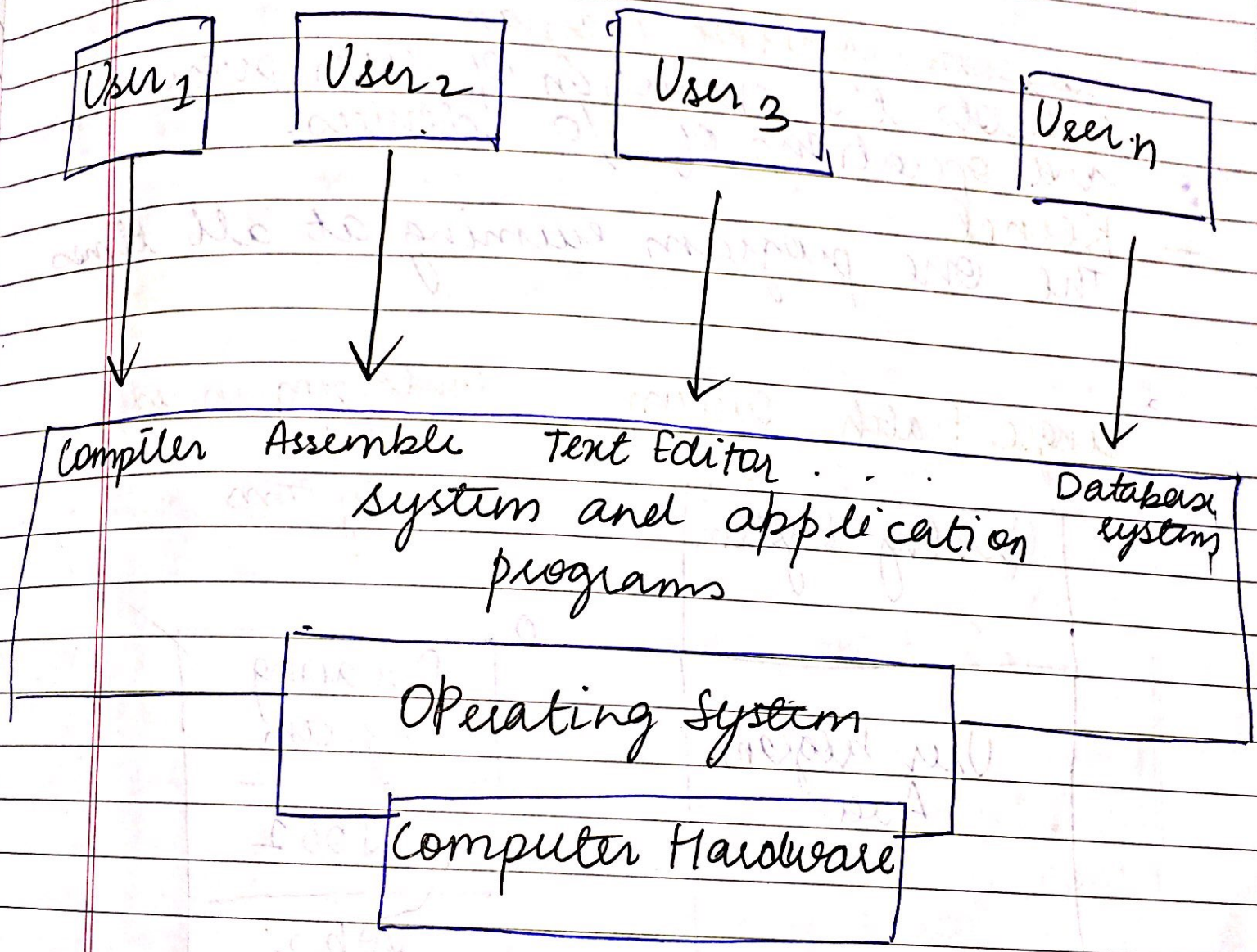
4. Users
- People, machines, other computers.

Scanned with CamScanner

Computer Hardware and Software Infrastructure.



Abstract View of system components



Scanned with CamScanner

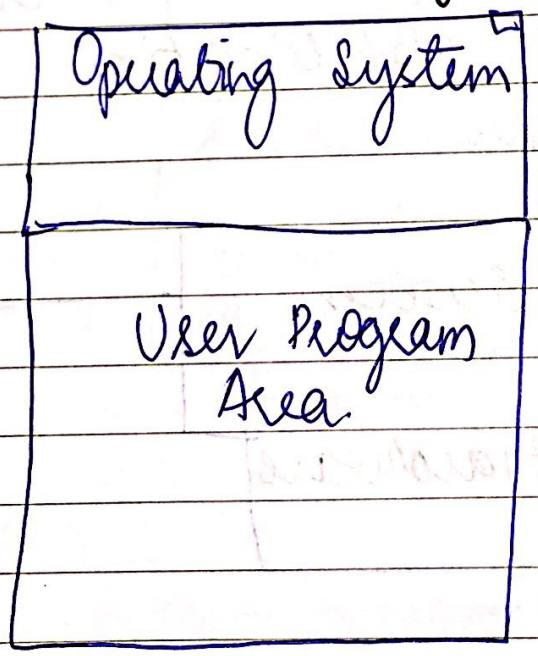
Operating System Definations.

- Resource allocator
  Manges and allocates the resources

- ~~Program~~ Control Program
  controls the execution of user program and operations of I/O devices.

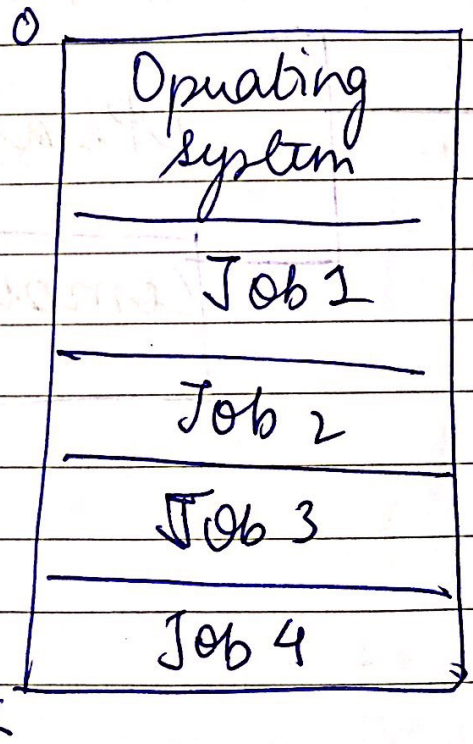
- Kernel
  The one program running at all times

Single Batch System



Memory layout

Multiprogrammed Batch Systems



Scanned with CamScanner

OS features Needed for Multiprogramming:

- I/O routine supplied by the system.
- Memory management
  system must allocate memory to ~~several~~ several jobs.
- Allocation of devices
- CPU scheduling – The system must choose among several jobs ready to run.

REAL-TIME SYSTEMS

- Used as a control device in a dedicated application such as controlling experiments, medical imaging systems, industrial control system and some display systems
- Well defined fixed-time constraints

Hard real-time system

- secondary storage limited or absent,
- Data stored in short-term memory (ROM)
- Conflicts with time sharing systems
- Not supported by general purpose operating system.

Soft real-time system

- Limited utility in industrial control & robotics.
- Useful in applications (multimedia, virtual reality) requiring advanced operating system features.

Scanned with CamScanner

DISTRIBUTED SYSTEMS

- Distribute the computation among several physical processors.
- Loosely coupled System
  + Each processor has its own local memory
  + processors communicate with one another through various communication lines, such as high speed buses or telephone lines.

Advantages of distributed system
+ Resource sharing
+ Computation speed up - load sharing
+ Reliability
+ Communications.

PARALLEL SYSTEMS

- Multiprocessor systems with more than one CPU in close communication.
- Tightly coupled system - Processors share memory and a clock.
  + Communication usually takes place through the shared memory.

Advantages
- Increased throughput.
- Economical
- Increased reliability
  + graceful degration
  + Fail-system safe

UNIT #2

PROCESS MANAGEMENT

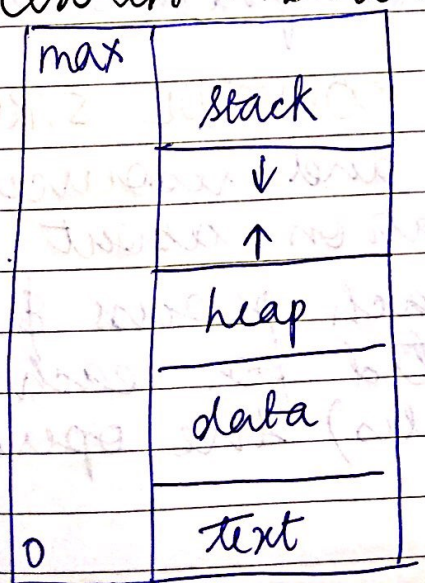
- An operating system executes a variety of programs

Batch system - Jobs
Time shared systems - User programs or tasks.

- Process - a program in execution
  - process execution must progress in sequential manner.

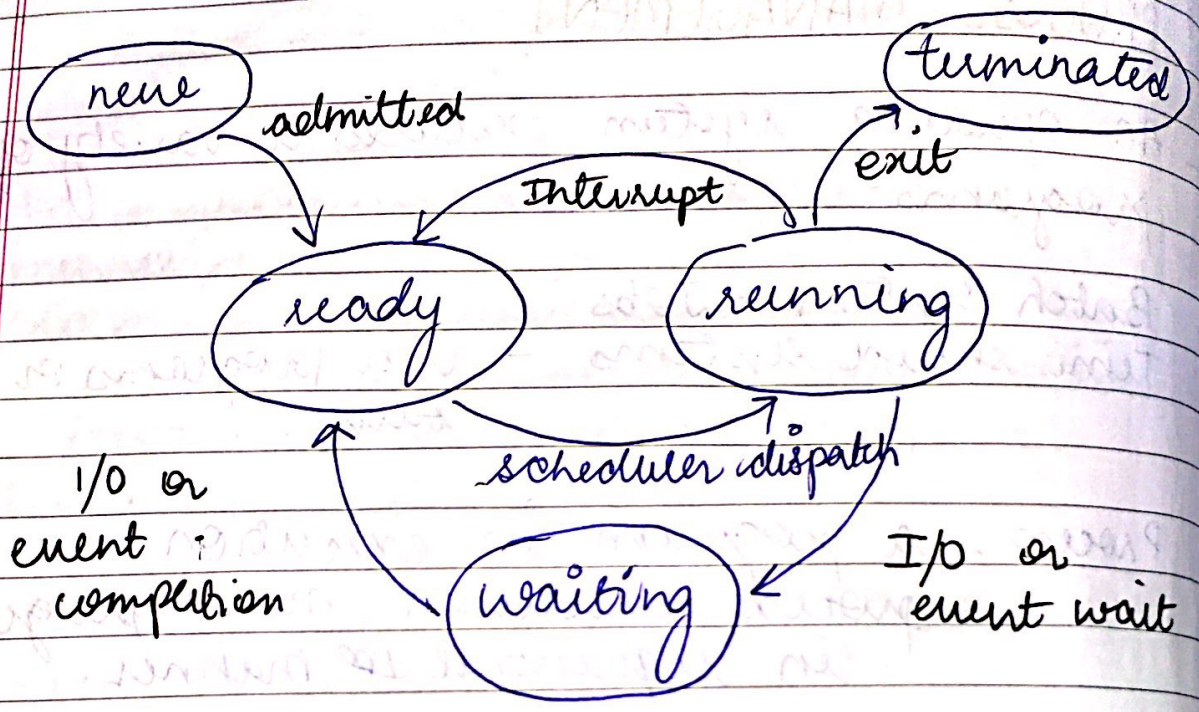
- A process includes
  + Program counter
  + Stack
  + Data section
  + Heap.

Process in Memory



Scanned with CamScanner

PROCESS STATE



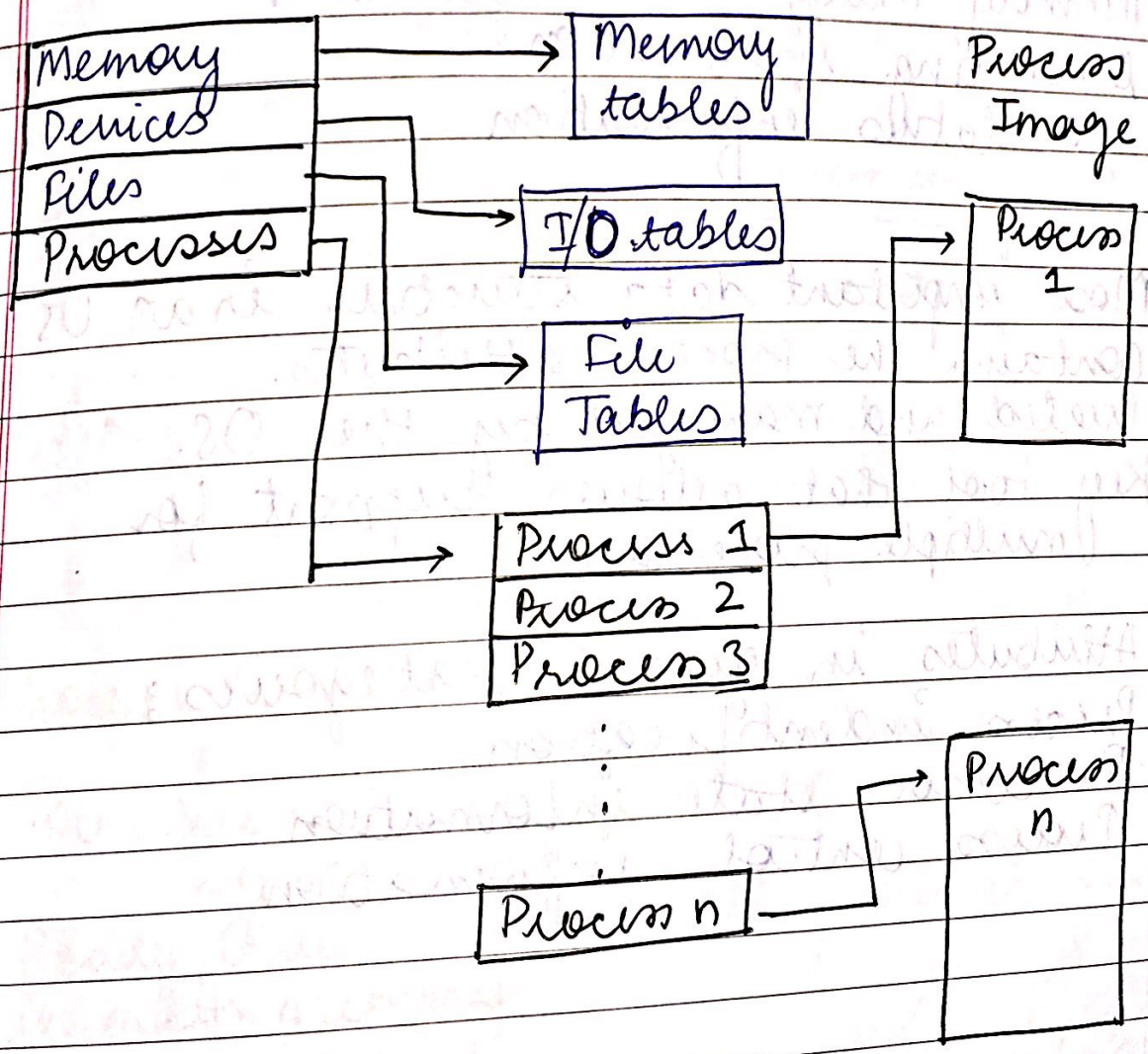
- As a process executes it changes state:
- New : Process is being created
- Running : Instruction are being executed
- Waiting : Process is waiting for some event to
- Ready : Process is waiting to be assigned to a processor.
- Terminated : Process has finished execution

OPERATING SYSTEM CONTROL STRUCTURE

- To manage processes and resources, it must have information about the current status of each process & resou
- Tables are constructed for each entity (Memory, I/O and files) the operating system manages

- These tables must be linked or cross-referenced.
- Memory, I/O and files are managed on behalf of processes.

OS control tables



Scanned with CamScanner

Process Control Block (PCB)

- Information associated with each process

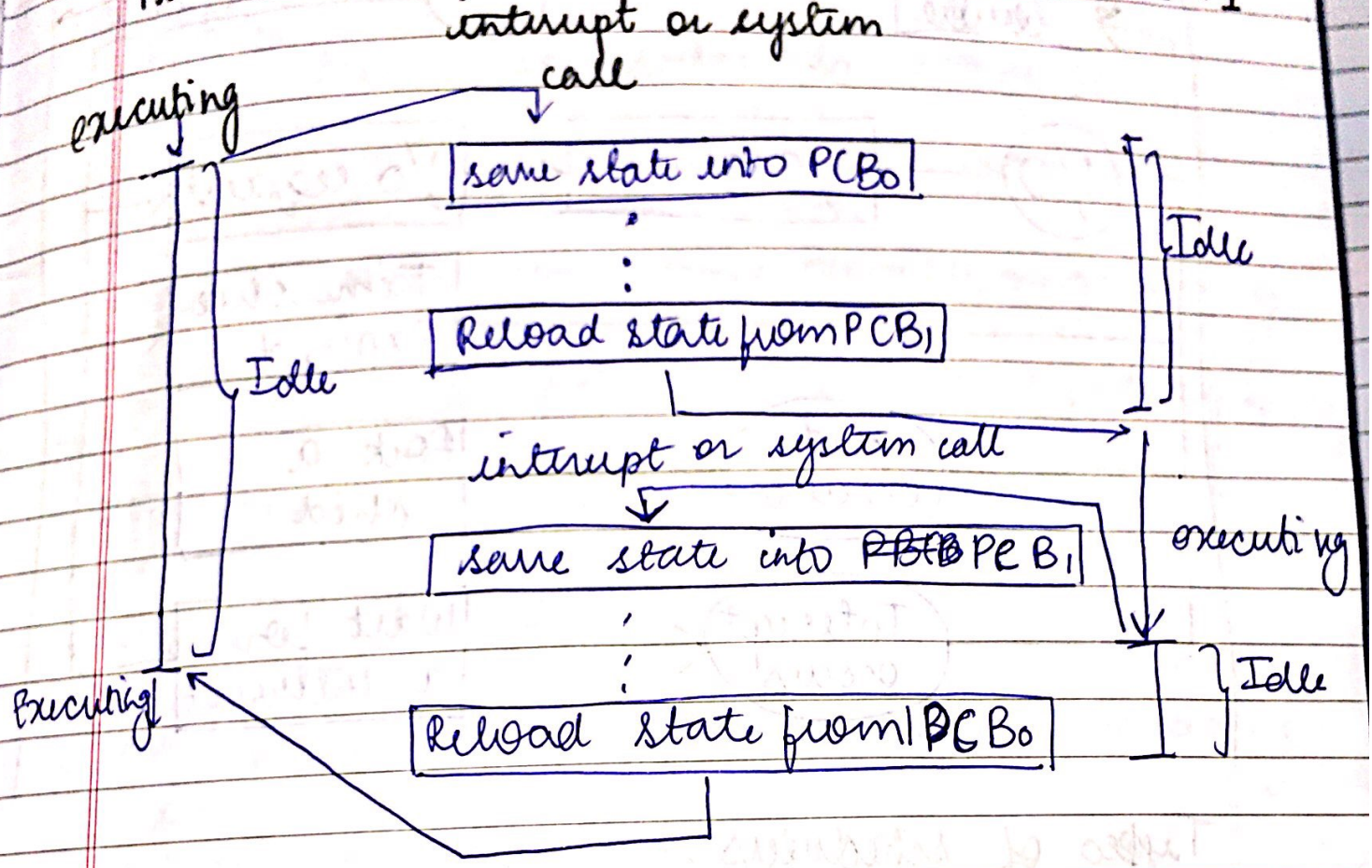
+ Process State
+ Program Counter
+ P. CPU registers
+ CPU scheduling information
+ Memory Management Information
+ Accounting information
+ I/O status information

- Most important data structure in an OS.
- Contains the process attributes.
- Created and managed by the OS.
- Key tool that allows support for multiple processes.

- Attributes in general categories:
+ Process identification
+ Processor state information
+ Process control information

Scanned with CamScanner

CPU switch from Process to Process



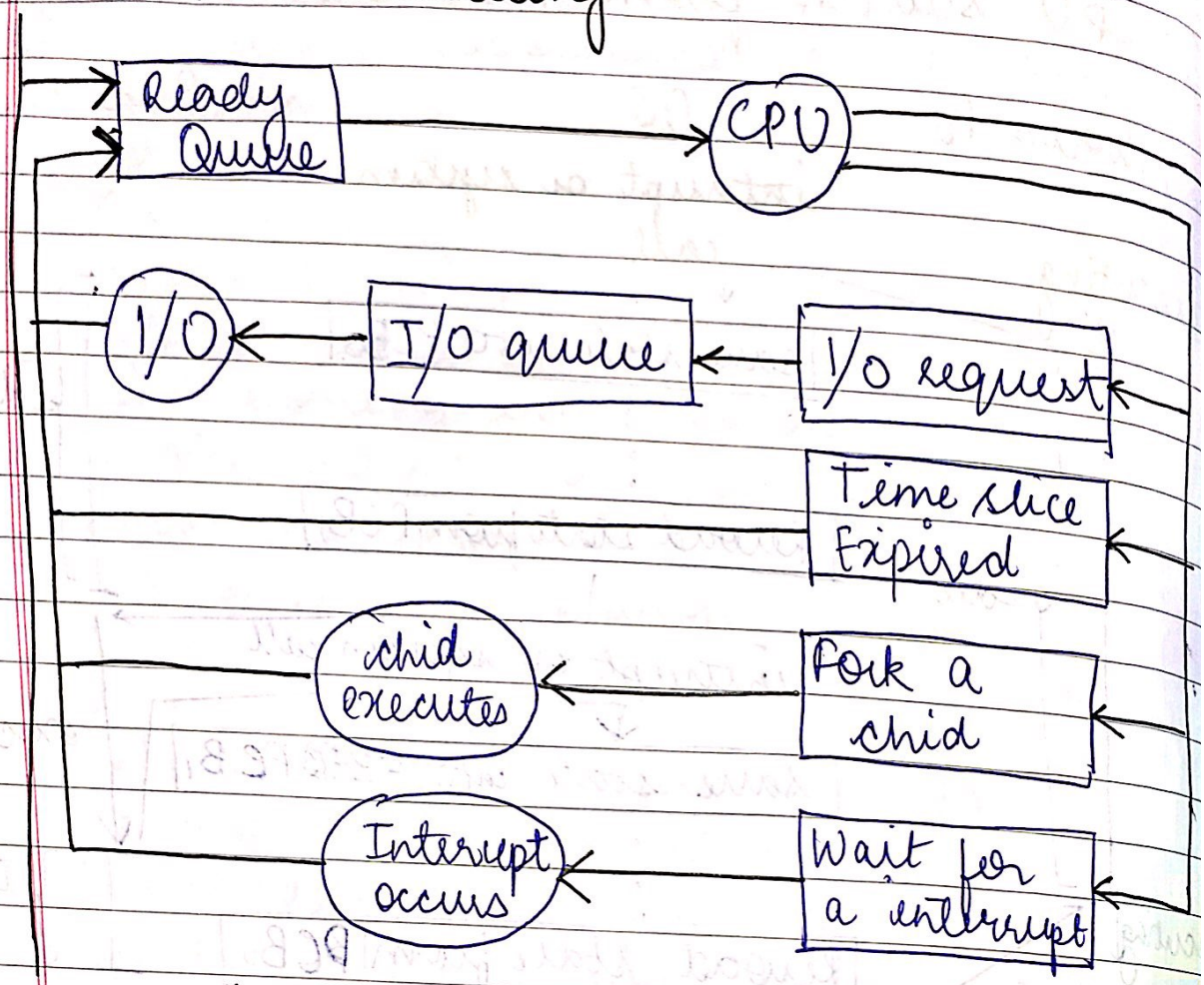
Process Scheduling Queues

- Job Queue - Set of all processes in a system
- Ready Queue - Set of all processes residing in main memory
- Device Queue - Set of processes waiting for an I/O device.

Processes migrate among the various queues.

Scanned with CamScanner

Process scheduling



Types of schedulers:

- Long term scheduler - JOB SCHEDULER
  + Selects which processes should be brought into ready queue
- Short term scheduler - CPU SCHEDULER
  + Selects which processes should be executed next and allocates CPU

SCHEDULERS

- short-term scheduler is invoked very frequently. (ms) — Must be fast
- long-term scheduler is invoked very infrequently ~~and invoked~~ (mins, s) — May be slow.

+ It controls the degree of multiprogramming.

- Processes can be ~~either~~ described either:

+ I/O bound process - Spends more time doing I/O than computations.
  ↳ Many short CPU bursts.

+ CPU bound process - Spends more time doing computations
  ↳ Fewer very long CPU bursts.

CONTEXT-SWITCH

- When CPU switches to another process, the system must save of the old process and load the saved ~~for~~ state for the new process.
- Context-switch time is overhead, the system does no useful work while switching
- Time dependent on hardware support.

Scanned with CamScanner

PROCESS CREATION (fork)

- Parent process creates children processes, which in turn create other processes, forming a tree of processes.

Resource sharing options:-
- Parent and children share all resources.
- Children share subset of parent's resources.
- Parent and child share no resources

Execution:
- Parent and children executes concurrently
- Parent waits until children ~~terminates~~ terminates

Address space:
- Child duplicate of parent
- Child has a program loaded into it.

PROCESS TERMINATION:

- Process executes last statement and (exit) ask the operating system to delete it (wait) Output data from parent to child.
  + Process resources are deallocated by OS

- Parent may terminate execution of (abort) children processes.
  + child has exceeded allocated resources
  + Task assigned is no longer required

Synchronization:

- Message passing may be either blocking or non-blocking

Blocking - SYNCHRONOUS

- Blocking send has the sender block until the is recieved.
- Blocking recieve has the reciever block until the message is available

Non-Blocking - ASYNCHRONOUS

- NB ~~to~~ send has the sender send the message and continue.
- NB recieve has the reciever recieve a valid message or null.

Buffering

- Queue of messages attached to the link, implemented in one of the three ways.
- Zero capacity - 0 messages
  Sender must wait for reciever
- Bounded capacity - Finite length of n message
  Send must wait if link is full
- Unbounded capacity - Infinite length.
  Sender never waits.

Scanned with CamScanner

MULTILEVEL QUEUE

- Ready queue is partioned in seperate queues.
  Foreground (Interactive)
  Background (Batch)

- Each queue has its own scheduling algo
  Foreground - Round Robin
  Background - First come First serve.

- Scheduling must be done b/w the queues.

+ Fixed-priority scheduling
  - serve all from foreground then from background.
  - Possibility of starvation.

+ Time slice - each queue gets a certain amount of CPU time which it can schedule amongst its processes
  - 80% to foreground in Round Robin
  - 20% to background in FCFS.

Multilevel Queue scheduling

highest priority -
→ system processes ⟷ FCFS
→ interactive processes ⟷ SJF
→ interactive editing processes →
→ batch processes → RR
→ student processes → Priority
lowest priority -

MULTILEVEL FEEDBACK QUEUE

- A process can move b/w the various queues.

Multilevel-feedback-queues scheduler can be defined by following parameters:

- ~~A process can move~~

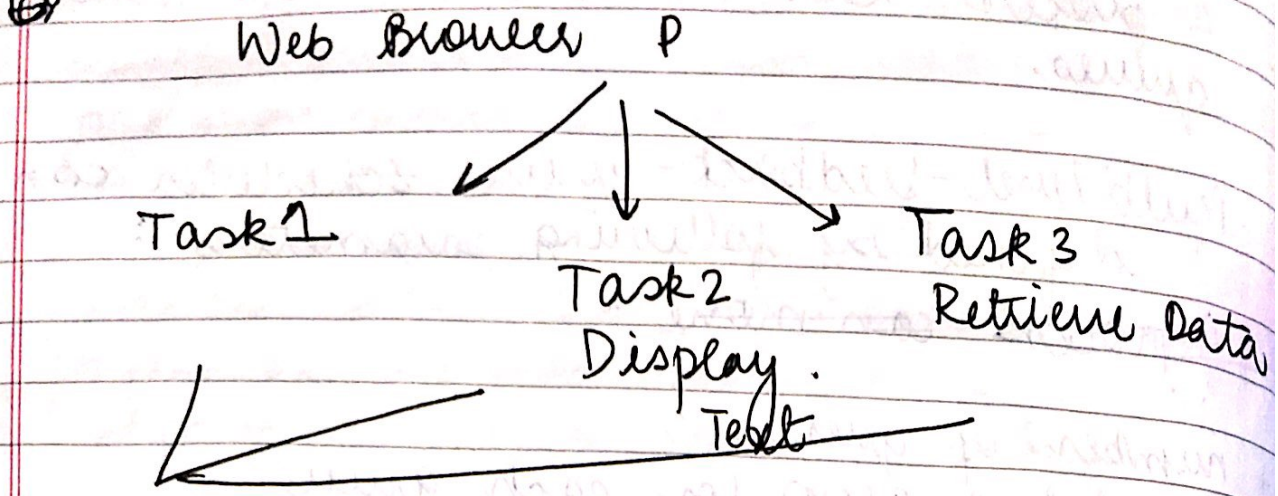
+ number of queues
+ scheduling algo for each queue.
+ method used to determine when to upgrade a process.
+ method used to determine when to demote a process
+ method used to determine which queue a process will enter when that process needs service.

EXAMPLE:

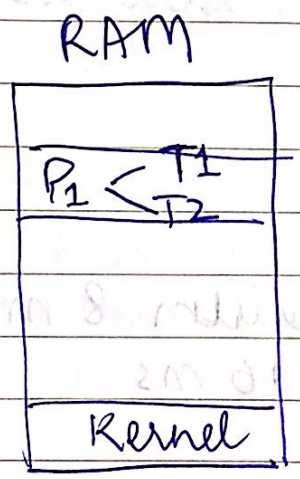
- Three queues:
  - Q_0 - RR with time quantum 8 ms.
  - Q_1 - RR time quantum 16 ms
  - Q_2 - FCFS

- scheduling
  + A new job enters Q_0 → FCFS. When it gains CPU, job recieves 8 ms. If it does not finish it in 8 ms, job is moved to queue Q_1.
  + At Q_1 job is again served FCFS, and recieves additional 16 ms. If does completes it is preempted and moved to Q_2.

THREADS



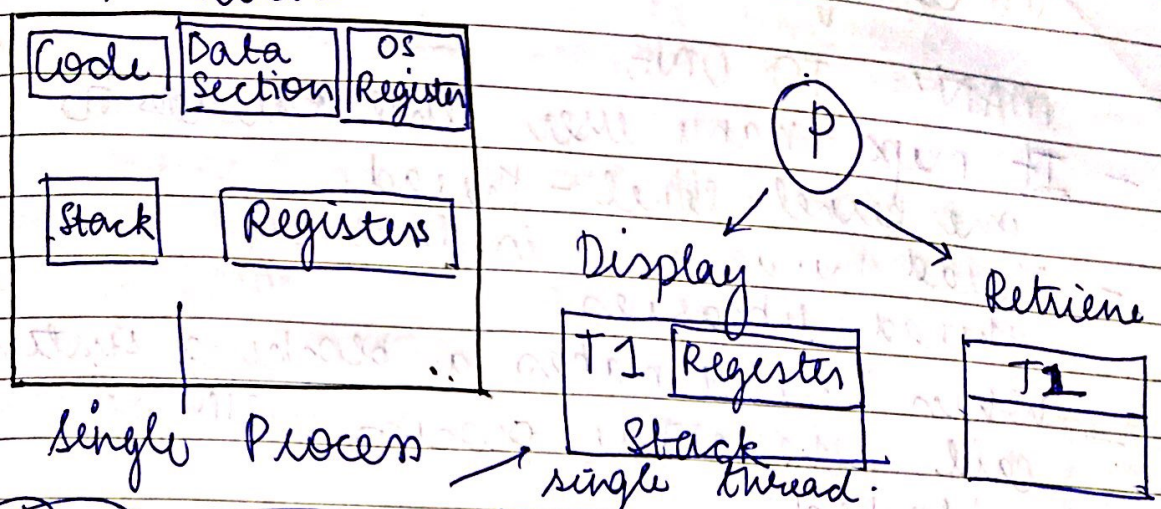
- Threads are light weighted process which are basically used to increase the CPU utilization.
- Threads contain
  + Thread counter
  + Thread ID
  + Thread State
  + General Purpose register ~~etc~~
  + Stack.



- Threads ~~space~~ shared address space of the process
- It also shares code section of the proc
- Shares data section + Resources of the process.

SINGLE THREADING

P → Process



Single Process → single thread.

~~Only one process is having threads~~

Single process with single thread

MULTI THREADING

Processes P

Code	Data	Files
Register	Register	Register
Stack	Stack	Stack
↓	↓	↓
T_0	T_1	T_2

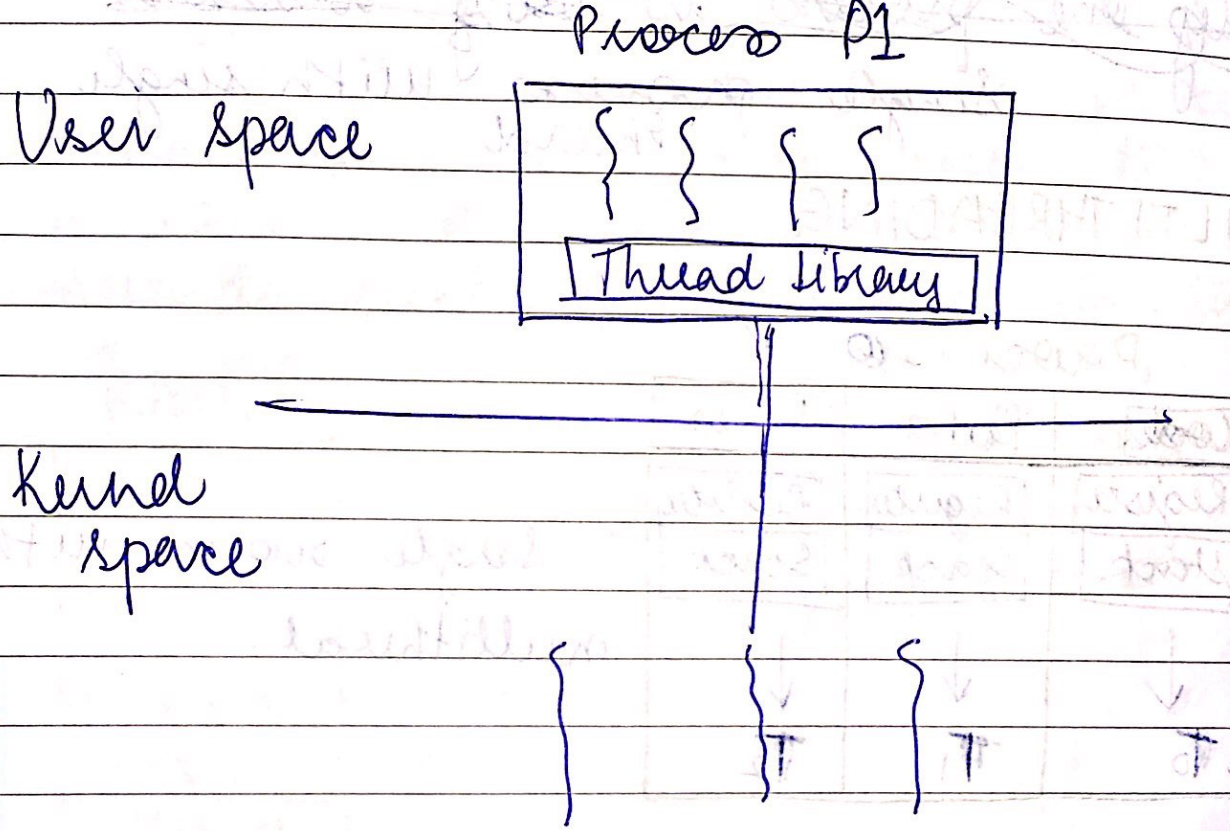
Single process with multithread.

- ~~Process creation is time consuming and~~ Benefits of multithreading is
- Responsiveness – If one thread blocked, other can continue.
- Resource sharing
- Economical
- Scalability - Increase Parallelism by scheduling the task to different core.

Models of Multithreading

MANY TO ONE

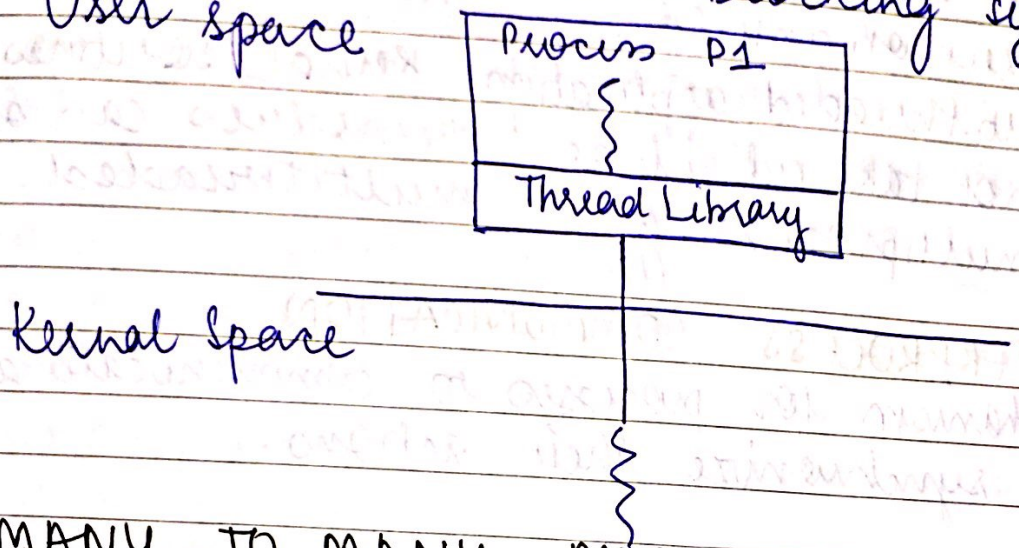
- It maps many user level threads to one kernel-level-thread.
- Thread management is done by thread libraries.
- When thread makes a blocking system call, the entire process will be blocked



Scanned with CamScanner

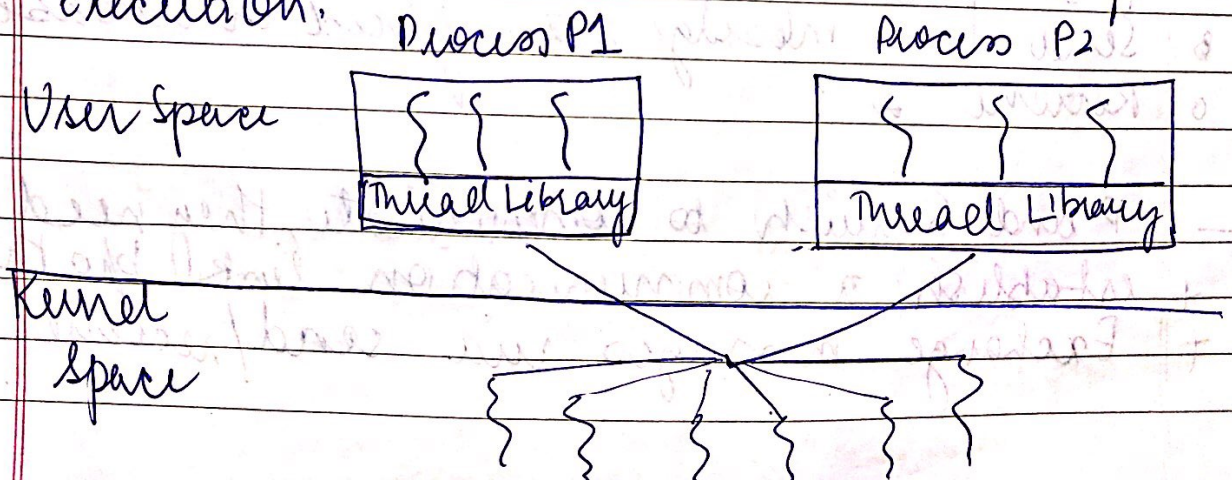
ONE TO ONE MODEL

- There is one-to-one relationship of user level thread to the kernel level thread.
- More concurrency than many-to-one
- Supports multiple threads to execute in parallel on microprocessors.
- ~~Provides~~ Allows one thread to run when a ~~read~~ thread makes a blocking system call



MANY-TO-MANY MODEL

- Many number of user threads onto an equal or smaller number of kernel threads
- Best accuracy on concurrency
- When a thread performs a blocking system call, the kernel can schedule another thread for execution.



Scanned with CamScanner

User-Level-Threads	Kernel-Level-Threads
— ULT are faster to create and manage.	KLT are slower to create and manage
— Implementation is by a thread library at user level.	Operating system supports creation of Kernel level threads.
— ULT is generic and can run on any OS.	KLT is specific to the OS.
— Multithreaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

INTERPROCESS COMMUNICATION

— Mechanism for processes to communicate and to synchronize their actions.

— Message system - Processes communicate with each other without resorting to shared variables i.e without sharing address space.

— IPC facility provides two operations.
- Send - message size fixed or variable
- Recieve "

— P and Q wish to communicate, they need to
+ establish a communication link b/w them
+ Exchange messages via send/recieve.

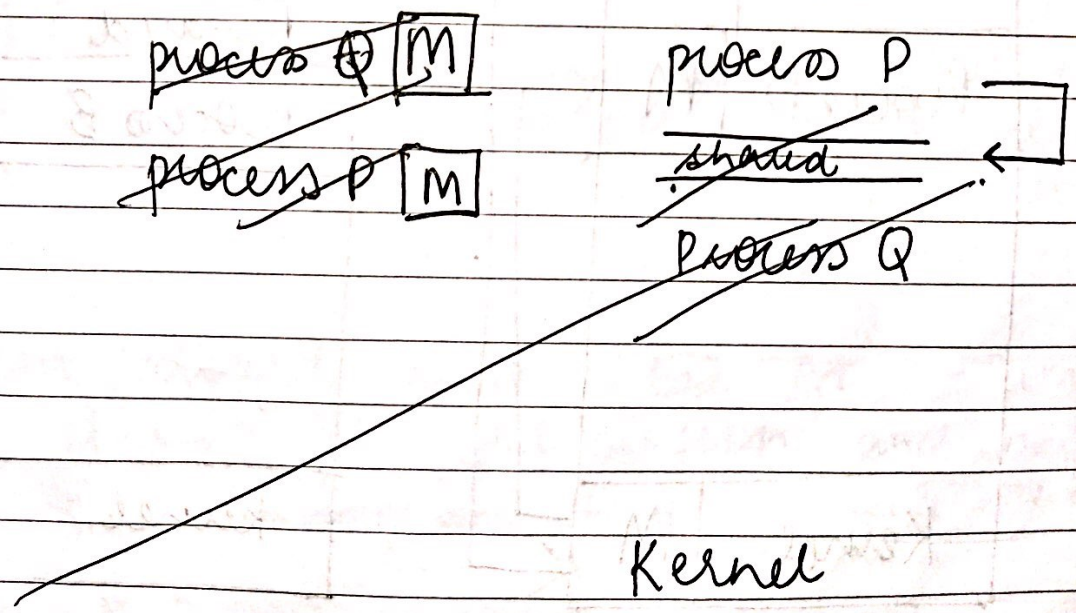
Scanned with CamScanner

- Implementation of communication link.
  + PHYSICAL - shared memory, hardware bus.
  + LOGICAL - Logical Properties.

Direct communication:

- Processes must name each other explicitly
  + send (P, message) - send a message to P
  + recieve (Q, message) - recieve a message from process Q.

- Properties of communication link.
  + Links are established automatically.
  + Link is associated with exactly one pair of communicating processes.
  + Between each pair there exists exactly one link.
  + Link may be unidirectional, but is usually bidirectional.

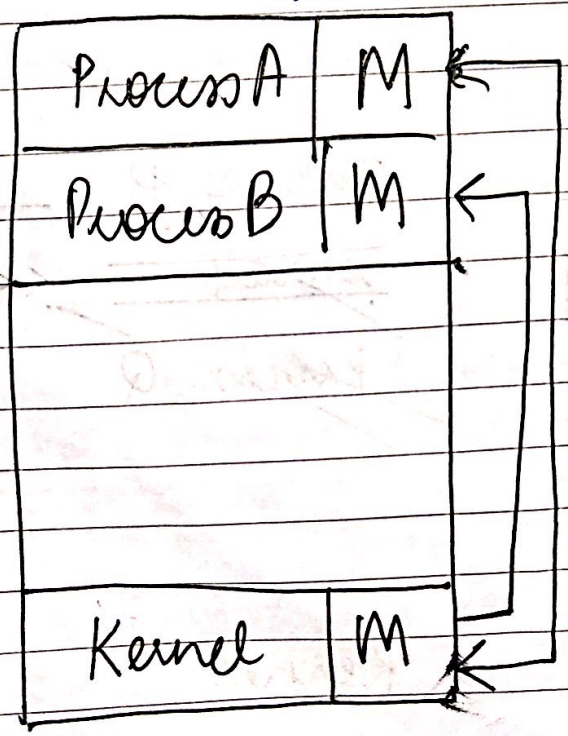


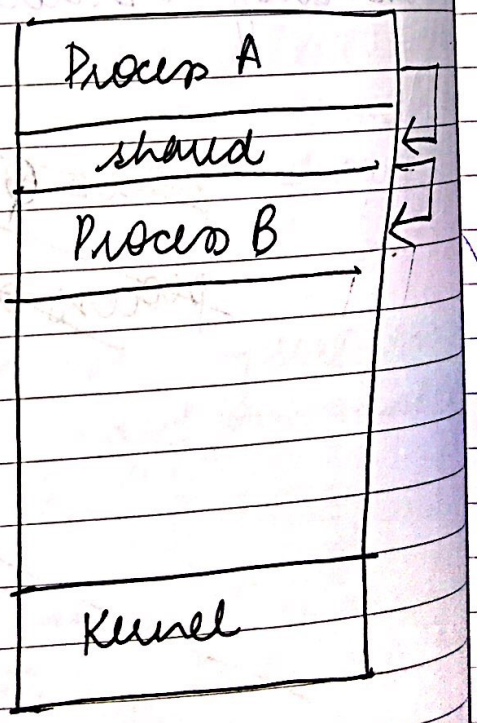
⇒ Message passing
⇒ Shared Memory

Indirect communication

- Messages are directed and recieved from mailboxes
  + Each mailbox has a unique Id
  + Processes can communicate only if they share a mail box.

- Properties of communication channel
  + Link is established only if they share a mailbox
  + A link may be associated with many processes.
  + Each pair of processes may share several communication links.
  + Links may be unidirectional or bidirectional





CPU scheduling.

- To achieve multiprogramming.
- CPU displays its processing in terms of clock pulses.
- Sequences of these clock pulses in discrete amount of time is known as CPU bursts.

Scheduling

- Preemptive - Processor is deallocated.
- Non-Preemptive - Processor keeps running until a job assigned isn't finished.
- Dispatch Latency - Delay in time. Time it takes for the dispatcher to stop ~~at~~ one process and start another running.
- Throughput = $\frac{\text{Number of processes completed}}{\text{Total time taken.}}$
- Dispatcher - module in OS
  - ~~Root~~ Provides control to the ~~process~~ CPU to ~~a~~ select which process is going to be executed next.
- Turn around time (~~tat~~ TAT) - Time elapsed b/w submission and completion of a task. / Time taken to execute a particular process.
- Wait time = Response time - Arrival time
- TAT = Burst time + Wait time.

Scanned with CamScanner

Waiting time - time spent in ready queue.

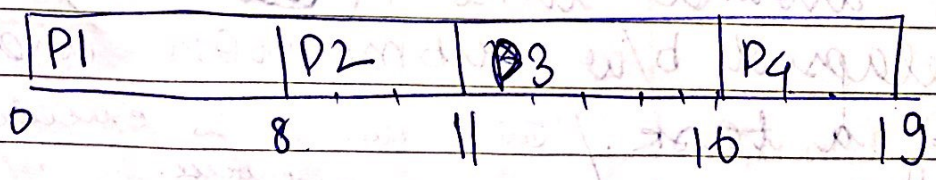
Response time - time slice b/w the first response and when it arrived in the ready queue.

FCFS - First come First serve.

- Implemented by maintaining ready queue as FIFO.
- Convoy Effect - Is slowing down of whole operating system because of few slow processes.
- Non preemptive in nature.

TAT = BT + WT

	BT	AT	WT	RT	TAT
P1	8	0	0-0	0	8+0 = 8
P2	3	0	8-0	8	8+3 = 11
P3	5	0	11-0	11	11+5 = 16
P4	3	0	16-0	16	16+3 = 19
			$\frac{35}{4}$		



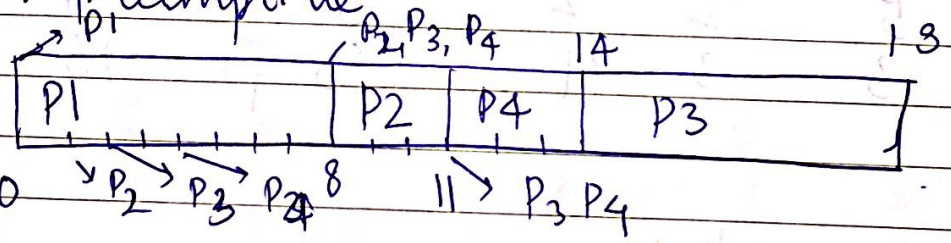
Scanned with CamScanner

SJF - Shortest Job First

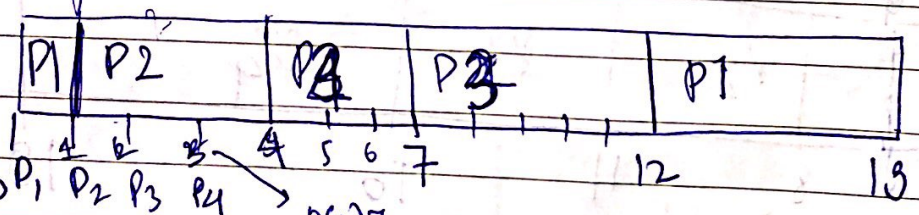
- Processor is allocated to the job in ready queue which has got shortest CPU burst
- Non-Preemptive - If a job is allocated to the processor ~~it will continue in~~ the processor won't be allocated to ~~the~~ another job until the job is done.
- Preemptive - SHORTEST REMAINING TIME FIRST.
- If there is a tie in CPU burst, it is resolved using FCFS.

	BT	AT	WT_P NP		TAT NP	P	RT NP
P_1	8	0	12-1=11	0	8	19	12	0
P_2	3	1	1-1=0	8-1=7	10	3	1	8
P_3	5	2	4-2=5	14-2=12	17	10	7	14
P_4	3	3	4-3=1	11-3=8	11	4	4	11

Non-preemptive



Preemptive



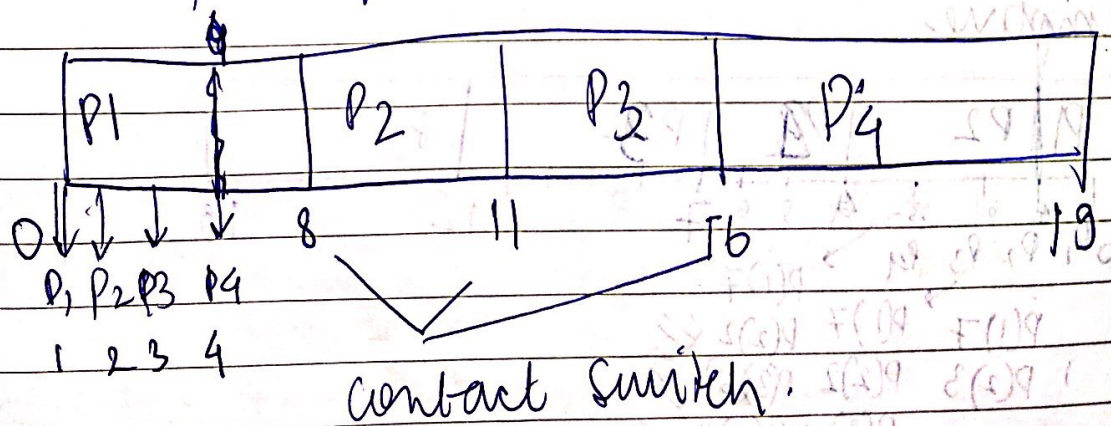
PRIORITY SCHEDULING

- Each process associated with a priority number
- Allocated CPU according to their priority number.
- Priority - can be preemptive
  can be non-preemptive

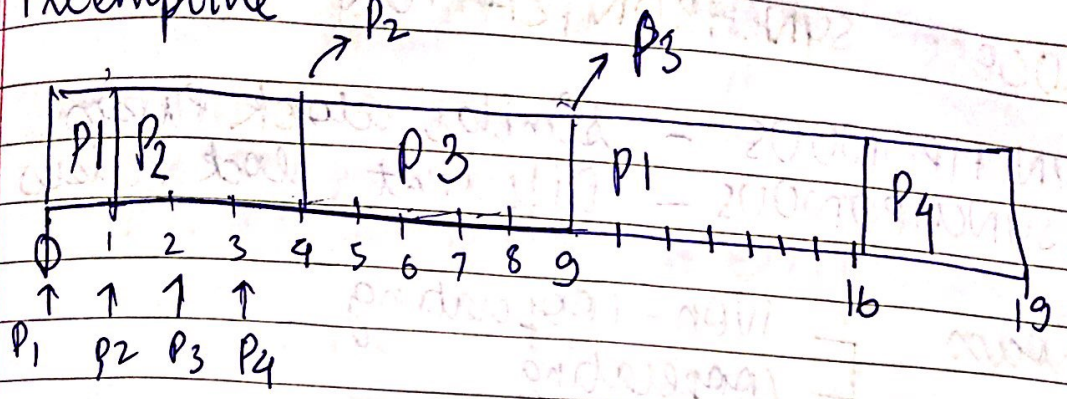
- SJF special case of priority
- STARVATION - A process is waiting for CPU to be scheduled.
  - When a process of higher priority keeps on coming and executing itself, and the process of lower priority has to wait.

	BT	AT	Pn	WT NP	WT P	TAT NP	TAT P
P_1	8	0	3	0	8	8	16
P_2	3	1	1	7	0	10	3
P_3	5	2	2	9	2	14	7
P_4	3	3	4	13	13	16	16

Non-preemptive



Preemptive

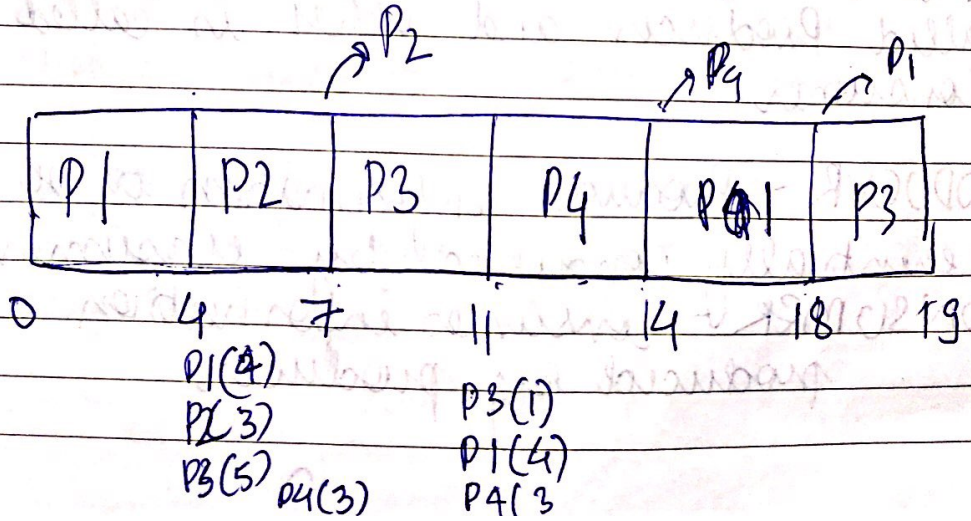


ROUND ROBIN

- FCFS + PRE-EMPTION Time slice
- Time Quantum (Q) – CPU will be allocated
- Ready queue is implemented as FIFO.
- Either – Process completes
  - Time slice expires
- Q is large – FCFS
- Q is small – More contact switches
  - Inadvertant delay.

Q = 4

	BT	AT	WT	TAT
P_1	8	0	10	18
P_2	5	1	3	8
P_3	5	2	5+11 = 16	21
P_4	3	3	8	11



PROCESS SYNCHRONIZATION

SYNCHRONOUS - Similar clock Pulses
ASYNCHRONOUS - Different clock Pulses

Process [Non-Cooperating / Cooperating]

- Non-Cooperating Processes
  + Resources belonging to non-cooperating processes are disjoint sets.
  + Execution of one process is not going to affect execution of any other process.
- Cooperating Processes
  + Resources that are held by cooperating processes are going to overlap each other.
  + Execution of one process is going to affect execution of any other process.

COOPERATING PROCESS -

- For a two process scenario, one process is called Producer and other is called Consumer.

- PRODUCER - Produces information to be eventually consumed by consumer
- CONSUMER - Consumes information produced by producer

Scanned with CamScanner

Shared Buffer Availability

- Unbounded Buffer
  - Producer produces unlimited buffer data.
  - Consumer waits if ~~is~~ buffer is empty.

- Bounded Buffer
  Producer waits if buffer is full
  Consumer waits if buffer is empty

ALGORITHM FOR PRODUCER/CONSUMER

```
#define buffer size 10          // size of bound
typedef struct
{                    } Producer / consumer.

} item;              // item
item buffer [buffer size];       // Buffer
int in;                          // points to next free position
int out;             // points to next occupied position
int counter          // increment when producer executes
                     // decument when consumer executes.
```

Producer Process

```
while(1)
{   while (counter == buffer_size)
        ;
    buffer [in] = next Produced;
    in = (in + 1) % buffer size;
    counter ++;
}
```

Scanned with CamScanner

consumer Process

```
while(1)
{   while (counter == 0)
        ;
    next consumer = buffer [out];
    out = (out + 1) % buffersize;
    counter --;
}
```

RACE CONDITION - Explanation

counter = 5

counter ++	P reg_1 = 5
reg_1 = counter	Producer P reg_1 = 6
reg_1 = reg_1 + 1	
counter = reg_1	
counter --	C reg_2 = 5
reg_2 = counter	C reg_2 = 4
reg_2 = reg_2 - 1	P counter = 6.
counter = reg_2	P register = 4

Race condition - Imposed on a shared resource

'CRITICAL SECTION' PROBLEM

Race condition

Let's assume ∃ 'n' process system
$P = \{P0, P1, P2 \ldots Pn\}$ are cooperating in nature.

∀ P, ∃ a section know as CRITICAL SECTION
- Modifies shared resources, utilize these shared ~~sea~~ resources.
Will have these commonly held resources.

Significance – If one process is executing in its critical section, no other process can enter its critical section.

counter ++ → CS

STRUCTURE OF AN IDEAL PROCESS

while (1)
{ Entry section — Where a process is being made to enter in its other process a valid entry
CRITICAL — Performs the task on the shared Resource.
Exit section – Inform certain instruction which will inform the other process to enter in the entry section
Remainder section
}

Solution to address Critical section fail

MUTUAL EXCLUSION

- Belongs to one process at a time.
- If one process is being executed in its critical section no other process can come in its critical section.

PROGRESS

- If no process is executing in its critical section and some process wise to enter into its critical section, then the process which is not executing in their remainder section can participate in the decision as to which process enters its critical section.

BOUNDED WAIT

- When a process has to wait unnecessarily There has to be a bound on how many times a process can enter into its critical section and how many times a process can enter into its entry section.

ALGORITHM 1

Two processes P_i and P_j
shared variable : int turn

Mutual Exclusion ✓
Progress
Bounded Wait ✓

Scanned with CamScanner

```
Pi                                  Pj
while(1)                            while(1)
{                                   {
  while (turn ==j) ENTRY              while(turn ==i)
    ;                                   ;
  Critical Section                    Critical section
  turn = j;        EXIT               turn = i;
  Remainder Section                   Remainder Section
}                                   }
```

ALGORITHM 2

Two processes P_i and P_j

T_0 = flag [i] = true

T_1 = flag [i] = true

shared variable: Boolean flag [2]

flag [i] = flag [j] = false.

```
      Pi                                  Pj
while(1)                            while(1)
{                                   {
  flag [i] = true   ENTRY             flag [j] = true;
while(flag[j] = true)               while(flag[i] = true)
  ;                                   ;
  Critical Section                    Critical section
  flag [i] = false; EXIT              flag [j] = false
  Remainder Section                   Remainder Section
}                                   }
```

Scanned with CamScanner

ALGORITHM 3 (PETERSON'S SOLUTION)

shared variable: int turn, boolean flag[]

flag [i] = flag [j] = false

Pi	Pj
while (1)	while (1)
{T_0: flag [i] = true	{T_1: flag [j] = true
turn = j;	turn = i;
while (flag [j] == true)	while (flag [i] == true)
; // Lock.	;
Critical section	Critical section
flag [i] = false;	flag [j] = false;
Remainder section	Remainder section
}	}

Mutual Exclusion ✓

Bounded Wait ✓

Progress ✓ Due to T_0 & T_1

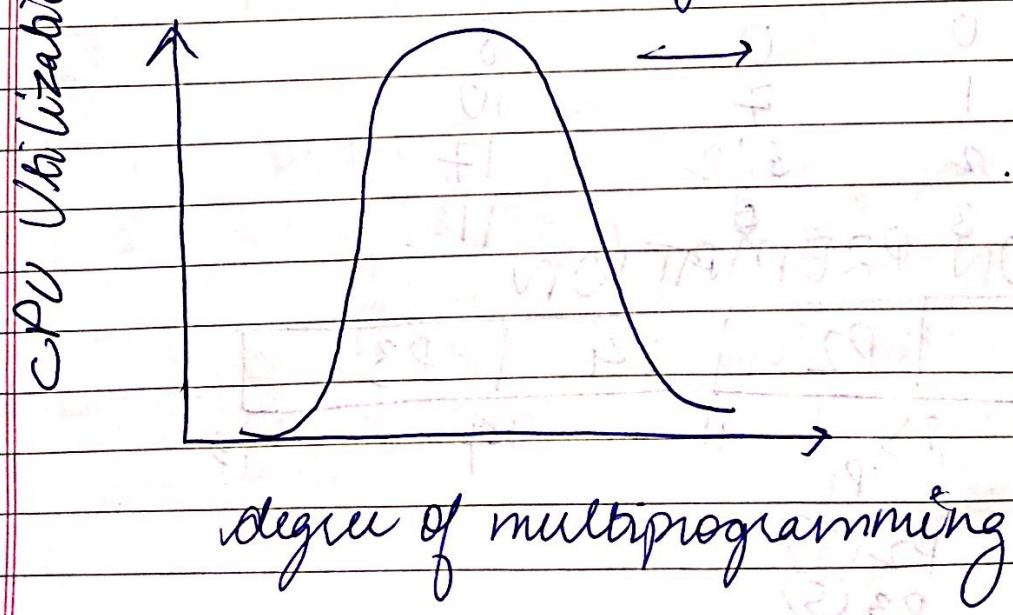
THRASHING: If the process does not have the no. of frames it needs to support pages in active use, it is a page-fault.

At this point it must replace some page. However, since all its pages are in active use, it must replace a page that will be required right away. Consequently, it will quickly fault again and again replacing pages that it must be bringing back in immediately.

This high paging activity is called THRASHING.

A process thrashes if is spending more time paging than executing.

Cause of thrashing:



FCFS

RT = WT − AT

TAT = BT + WT

= RT − AT

	BT	AT	WT	RT	TAT
P1	8	0	0		8
P2	3	0	8		11
P3	5	0	11		16
P4	3	0	16		19
			35		54

P1 P2 P3 P4

P1	P2	P3	P4

0 8 11 16 19

$\frac{35}{4} = \frac{54}{4}$

SJF RT_AT

	BT	AT	WT	TAT
P1	8	0	0	8
P2	3	1	7	10
P3	5	2	12	17
P4	3	3	8	11

NON PREMPTION

P1		P2	P4	P3

0 8 11 14 19

P1

P2 (3)

P3 (5)

P4 (3)

PREEMTION

	BT	AT	WT	TAT
P1	8	0	11	19
P2	3	1	0	3
P3	5	2	5	10
P4	3	3	1	4

P1	P2	P2	P4	P3	P1

0 1 2 3 4 7 8 12 19

P2(2) P3(5) P4(3) P1(7)

P2(3) P3(5)

ROUND ROBIN

Q = 2

	BT	AT	WT
P1	4	0	0+4
P2	5	1	1+7+4
P3	2	2	2
P4	1	3	5
P5	6	4	5+8
P6	3	6	7+3

P1	P2	P3	P1	P4	P5	P2	P6	P5	P2	P6	P5

0 2 4 5 6 8 9 11 13 15 17 18 19 21

P2(5) P3(2) P1(2)

P3 P1 P4 P5 P2(3)

P1 P4 P5 P2 P6

P4 P5 P2 P6

P5 P2 P6

P2 P6 P5(4)

P5(4) P2(1) P6(1) P5(2)

P2(1)

Effective Access Time

p = page fault rate

$$EAT = p * (\text{page fault service time}) + (1-p) * \text{Memory access time}$$

PFST = 10 ms
MAT = 20 ns
$p = 1/10^6$

$$EAT = \frac{1}{\cancel{10^6}} \times 10 \times \cancel{10^6} + \left(1 - \frac{1}{10^6}\right) 20$$

$$= 10 + \frac{\cancel{10^6} \times 20}{\cancel{10^6}} - \frac{20}{10^6}$$

$$= 10 + 20 - \frac{20}{10^6} = \frac{30 \times 10^6 - 20}{10^6}$$

TLB Ratio & Effective Access Time.

$$EAT = h(\text{TLB time} + \text{memory time}) + (1-h)(\text{TLB time} + 2\,\text{memory time})$$

h = 0.90
memory = 50 ns
TLB time = 10 ns

$$EAT = 0.90(60) + 0.10(10 + 100)$$

$$EAT = 0.90 \times 60 + 0.10 \times 110$$

$$= 54 + 11$$

$$= 65 \text{ ns.}$$

Scanned with CamScanner

1K = 1024

1M = 1024K

Que.1 Assume that we have 1M memory. O/S is alloted 256K starting from 0. Memory is partioned into fixed size memory chunks of 64K.

Calculate:

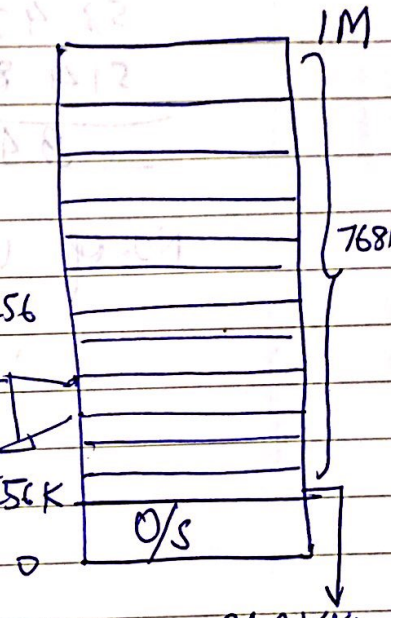
(a) No. of Memory chunks

(b) Starting physical address of first memory partition.

(c) No. of words in each memory partition.

(d) No. of words in each memory

1 chunk = 64K.



P1: BR 448K LR: 56056

P2: BR 640 LR: 63000

Available Memory = 1024 − 256

= 768K.

514808 = 56056 ← 458752 ← P1

(b) 1024×256

= 262,144

(c) No. of words in 1 partition = 64×1024

=

(d) Base address + (n−1) × size of one.

$448 = 256 + (n-1) \times 64$

$\frac{192}{64} = n-1$

$4 = n$

(e) Determine the trap when P1 generates the following address.

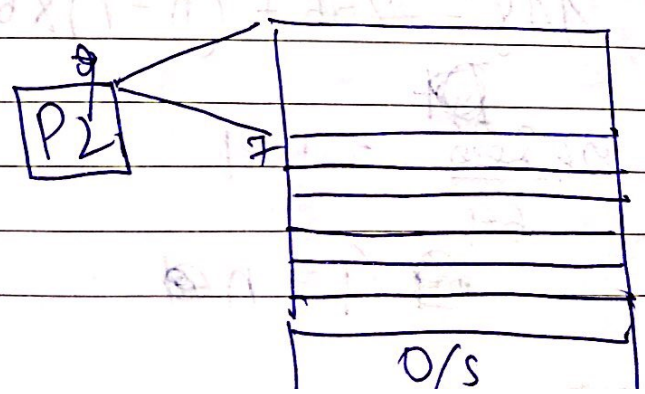
(i) 458752
(ii) 524288
(iii) 45200

$$\begin{array}{r} 524288 \\ 514808 \\ \hline 9480 \end{array}$$

No. of words wasted in P1 = 9480

% wasted in internal fragmentation $= \frac{9480}{65536} \times$
$= 14.4\%$

$640 = 256 + (n-1)64$
$640 - 256 = (n-1)64$
$\frac{384}{64} = n-1$
$6 + 1 = n$
$7 = n$



2	640	0
2	320	0
2	160	0
2	80	0
2	40	
	20	

$64 = 2^6 \times 10$.

(e) (i) $458752 \geq$ Base address (458752).
< 514808

VALID

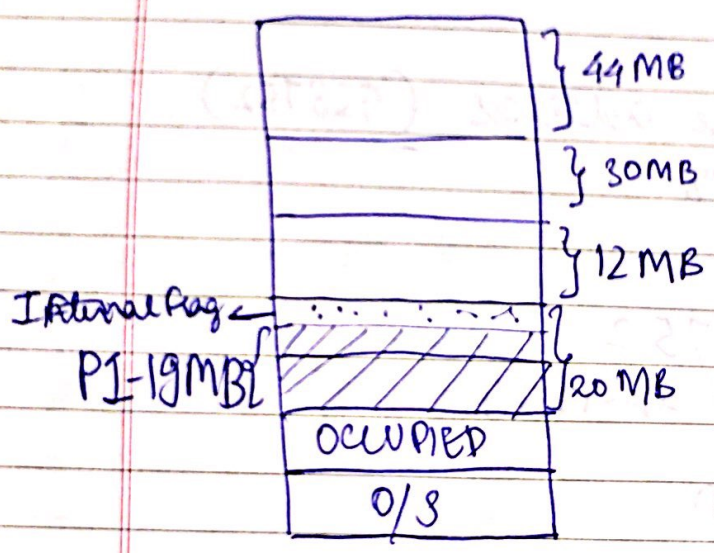
(ii) $524288 \geq 458752$
$\therefore 524288 > 514808$

$\therefore$ INVALID

(iii) 452000
✓ VALID.

Scanned with CamScanner

Que 2. Consider variable size partitioning technique where holes are 20MB, 30MB, 44MB & 12MB of size. A process of size 19MB is allocated as per First Fit, Best Fit and Worst Fit algos.



First-Fit : Allocate hole (first) that is big enough.

Best Fit : Allocate the.

- Segmentation
- Page size & Frame size
- Features of segmentation
- STBR, STLR
- Block diagram of segmentation process

- Fixed Block Allocation scheme - paging
  - contiguous
  - non-contiguous

Lookup → Paging Table

page size == frame size.

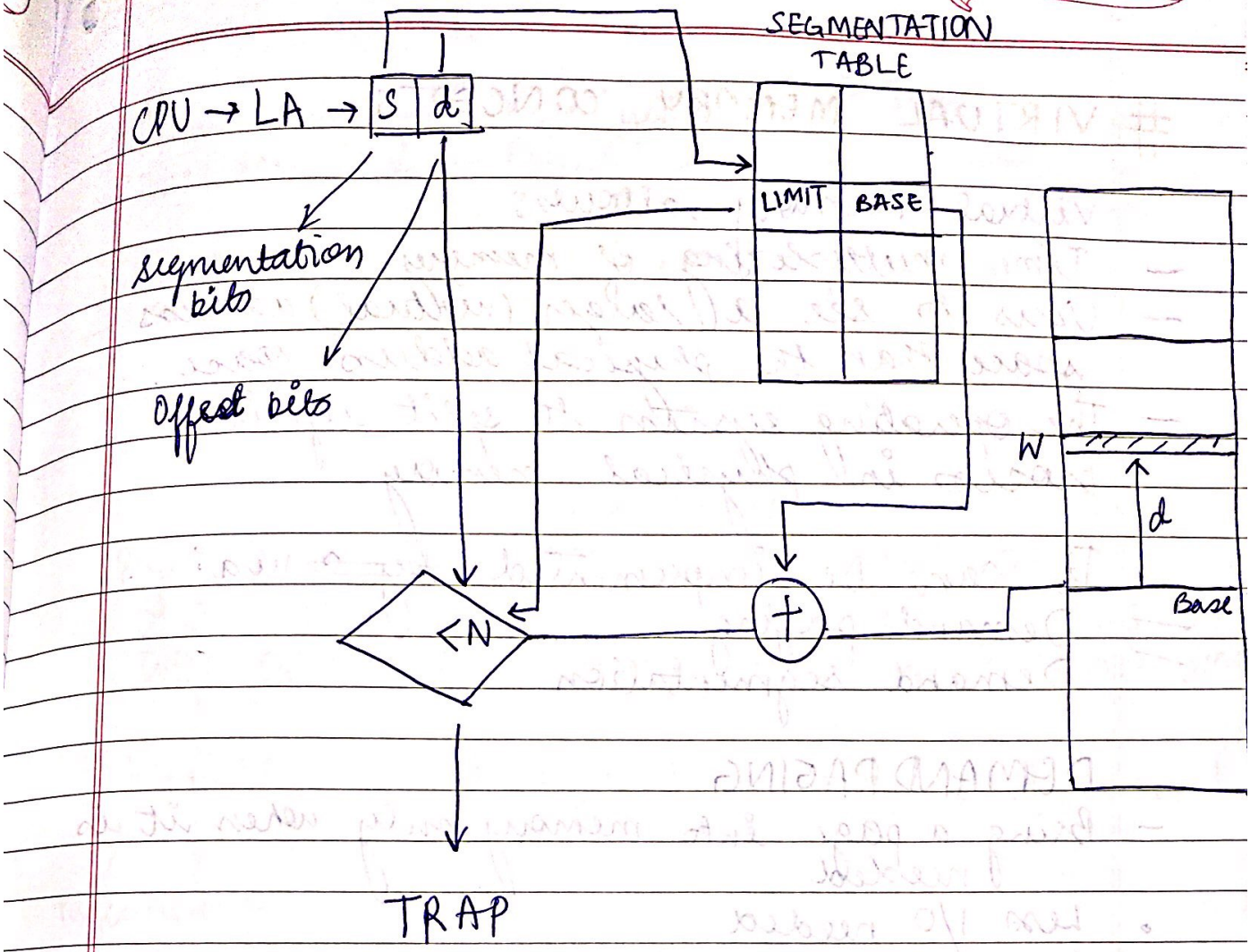
segmentation → page size ≤ frame size
Variable block allocation scheme.
→ Lookup - segmentation table

Features -
- Variable block allocation.
- Process is broken into segments.
- Segments can be of equal / unequal sizes.

SEGMENTATION - Memory Resident Data Stru
STBR — Segmentation Base Register.

STLR - contains length of the register as content.

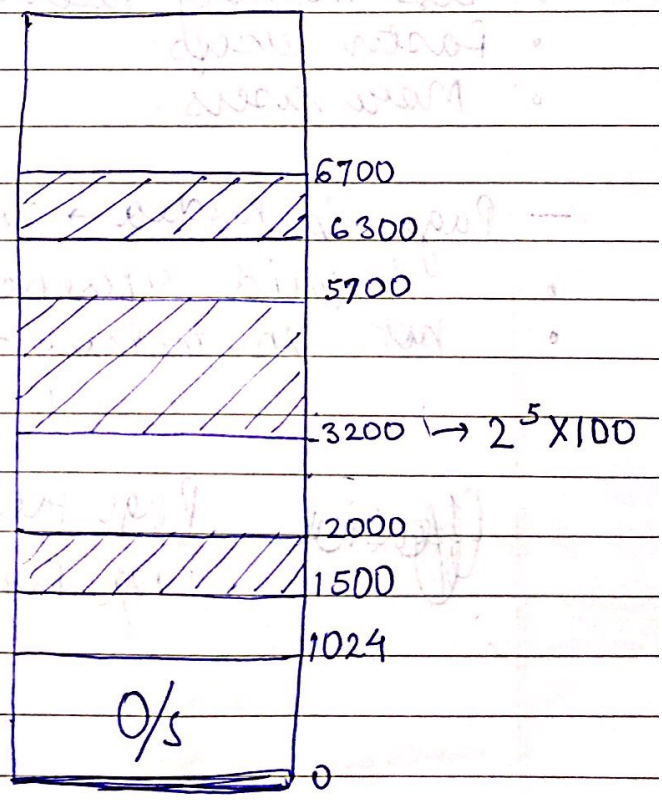
Scanned with CamScanner



Que.

LIMIT	BASE
500	1500
400	6300
2500	3200

	9	13
00	111110100	1001110111000



VIRTUAL MEMORY CONCEPTS

Virtual Memory allows:

- Time multiplexing of memory
- Users to see a larger (virtual) address space than the physical address space.
- The operating system to split up in a process in physical memory.

It can be implemented ~~by a~~ via:

- Demand paging
- Demand segmentation.

DEMAND PAGING

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster users
  - More users.

- Page is needed → reference to it
  - invalid reference → abort
  - not in memory → bring to memory.

~~Effective~~ Page Fault → ~~Decision~~ ~~diagram~~
Page Replacement → Flow chart.

- Demand Paging — Page fault
- Two aspects — Frame allocation
  — Page Replacement
- Demands on Page Replacement Algorithm
- FIFO/OPTIMAL.

↓ Minimization of page faults.

Page size - 100B

Page references — 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0105

1, 4, 1, 6, 1, 1, 1, 1, 6, 1, 1, 1, 1, 6, 1, 1, 1, 1, 6, 1

↑ Page number.

This have we convert the logical address into page IDs.

Page ID - 7 0 1 2 0 3 0 4 2 3 0 3 2 0
1 7 0 1

FIFO.

3 frames available for this process.

Paging. Numericals

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7
7	7	7	2	N	2	2	4	4	3				2				7
	0	0	0	0	3	3	3	2	2				1				2
		1	1	1	1	0	0	0	0				0				1
									4	0			0	0			7
									2	2			1	1			1
									3	3			3	2			2

$$\frac{15}{20} \times 100 = 75\%$$

OPTIMAL – Replace the page that won't be used for longest period of time.

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

→ Generate a data set to demonstrate Belady's anamoly.

$$\frac{9}{20} \times 100 = 45\%$$

~~Backward~~ BELADY'S ~~ANAMOLY~~ ANOMALY
– Occurs in FIFO algo.

	MAX			ALLOC			NEED		
	A	B	C	A	B	C	A	B	C
P1	8	6	9	4	1	1	4	5	8
P2	3	3	3	0	3	2	3	0	1
P3	5	2	7	3	2	5	2	0	2
P4	2	2	1	1	0	0	1	2	1

AVIAL = [2, 4, 2]

MAX [3, 2] = 2

↓ P3 ↓ B

K instances needed by P_i of R_j.

↓ P3 ↓ B

Scanned with CamScanner

BANKER' ALGORITHM

[Safety Algo
Resource Request Algo.

MAX [n x m]
ALLOC [n x m]

NEED = MAX − ALLOC

Safety Algorithm

Work [1 ... m] → Available
Finish [1 ... n], Boolean.

Initialize work = Available

Set Finish [i] = false, ∀ 1 ≤ i ≤ n

1 Find an i s.t
Need$_i$ ≤ Work && Finish [i] = false
If no such i, go to step 3.

2 Work = Work + Allocation;
Finish [i] = True;
Goto step 1

3 If finish [i] = true
∀ 1 ≤ i ≤ n, system is in

safe state
else
unsafe state.

→ Work = 2, 4, 2

1 4, 5, 8 ≤ 2, 4, 2, N
3, 0, 1 ≤ 2, 4, 2, N
2, 0, 2 ≤ 2, 4, 2,

Work = 2, 4, 2 + 3, 2, 5
= 5, 6, 7

Finish [3] = True.

2. 4, 5, 8 ≤ 5, 6, 7, N
3, 0, 1 ≤ 5, 6, 7, ✓

Work = 5, 6, 7 + 0, 3, 2 = 5, 9, 9
Finish [2] = True

3. 4, 5, 8 ≤ 5, 9, 9 ✓

Work = 5, 9, 9 + 4, 1, 1

= 9, 10, 10

Finish [4] = True.

Scanned with CamScanner

4. 2, 2, 1 $\leq$ 9, 10, 10 ✓

Work = 1, 0, 0 + 9, 10, 10

Work = 10, 10, 10

Finish [4] = True.

Resource Request Algo [RRA]

→ Request cannot be more than the need

R_i is the request vector for P_i

1. ? R_i > $Need_i$ INVALID REQUES
   go to step 7

2. ? R_i > Available, P_i waits

3. // Make temporary changes

$Allocation_i = Allocation_i + R_i$

$Available = Available - R_i$

$Need = Need - R_i$

Scanned with CamScanner

4 Call Safety Algo.

5 If safe state in safety algo, go to step 7

6 If unsafe state in safety algo.

// undo the temporary changes.

$Allocation_i = Allocation_i - R_i$

$Available = Available + R_i$

$Need = Need + R_i$

7 EXIT

- If P_i issues a request $\langle 1, 0, 1 \rangle$ R_i

MAX - ALLOCATION

(7.3)

	ALLOCATION A B C D	MAX A B C D	(i) NEED A B C D
P_0	0 0 1 2	0 0 1 2	0 0 0 0
P_1	1 0 0 0	1 7 5 0	0 7 5 0
P_2	1 3 5 4	2 3 5 6	1 0 0 2
P_3	0 6 3 2	0 6 5 2	0 0 2 0
P_4	0 0 1 4	0 6 5 6	0 6 4 2

(ii) Available → [1 5 2 0]

↓

Work

Work → [1, 5, 2, 0]

P_0 0000 $\leq$ 1520

Work = Work + Allocation

= 1532.

Finish [0] = True

[P_0] = True.

P_1 0,7,5,0 $\leq$ 1,5,3,2

~~Work = 1532 + 1000~~

~~= 2532~~

~~Finish [P1] = True~~

P_2 [1, 0, 0, 2] $\leq$ [1, 5, 3, 2]

Finish [P_2] = false. ✓

$\therefore$ Work = 1,5,3,2 + 1,3,5,4

Work = 2, 8, 8, 6.

Finish [P_2] = True.

P_3 [0, 0, 2, 0] $\leq$ [2, 8, 8, 6]

Finish [P_3] = False ✓

Work = 2, 8, 8, 6 + 0, 6, 3, 2

= 2, 14, 11, 8

Finish [P3] = True.

P_4 [0, 6, 4, 2] $\leq$ [2, 14, 11, 8]

Work = 2, 14, 11, 8 + 0, 0, 1, 4

Finish(P_4) = True

= 2, 14, 12, 12.

Scanned with CamScanner

(iii) If a request from process P_1 arrives for (0, 4, 20) can the request be granted immediately.

$R_1 - [0, 4, 2, 0]$

$R_i < [0, 7, 50]$

$R_i < [1, 5, 2, 0]$

$Allocation_1 = Allocation_1 + R_1$

$= 1, 0, 0, 0 + 0, 4, 2, 0.$

$= 1, 4, 2, 0$

$Available = 1, 5, 2, 0 - 0, 4, 2, 0$

$= 1, 1, 0, 0.$

$Need_1 = Need_1 - R_1$

$= 0, 7, 5, 0 - 0, 4, 2, 0.$

$= 0, 3, 3, 0.$

Run safety algo.

~~P_0 P_2 P_3 P_1 aka P_4~~

New sequence will be created

If unsafe undo changes of the request

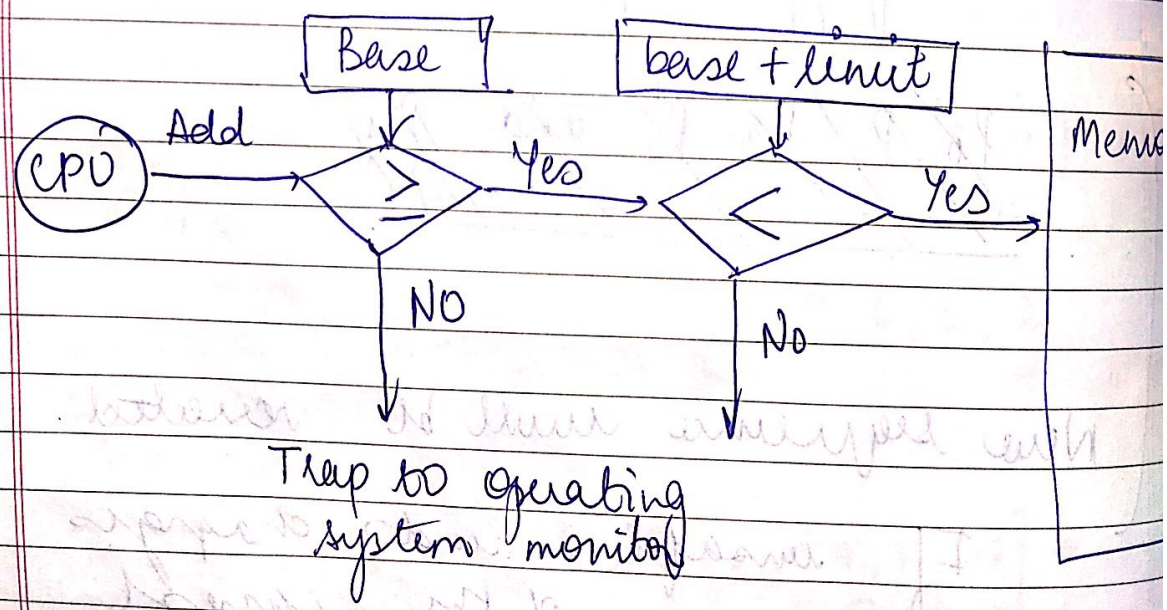
UNIT #3

MEMORY MANAGEMENT

- A program must brought into memory and placed with a process for it to be run
- Main memory and registers are only storage CPU can access directly.
- Main memory can take many cycles, causing a stall.
- Cache sits between main memory and CPU reg

- Base and limit registers
- A pair of base and limit registers define the logical address space.
- CPU must check every address access granted in user mode to be sure it is between base and limit registers.

Hardware address protection



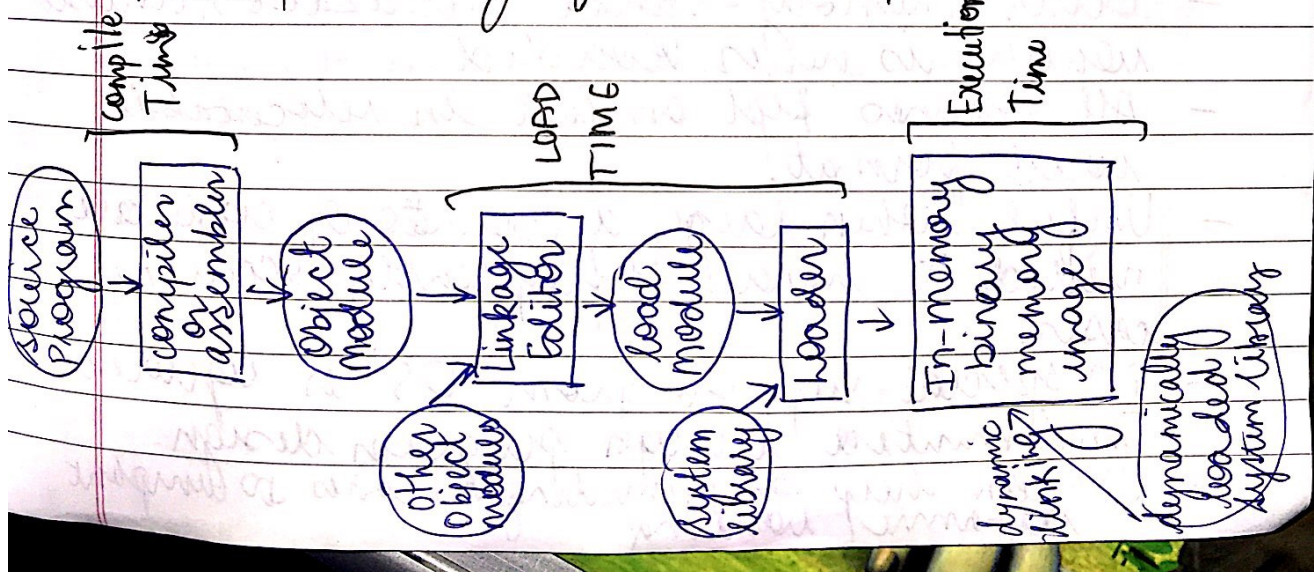
ADDRESS BINDING

- compiled code addresses bind to relocatable addresses
- Linker or loader will bind relocatable addresses to absolute addresses.
- Each binding maps one address space to another

Binding of instructions and data to memory:

- COMPILE TIME: If we know where the process will reside in memory, the absolute code can be generated, must recompile if starting location changes
- LOAD TIME: Must generate relocatable code if memory location is ~~not~~ known at compile time
- EXECUTION TIME: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Needs hardware support for address maps. (Base & Limit Registers)

Multiprocessing of a user program.



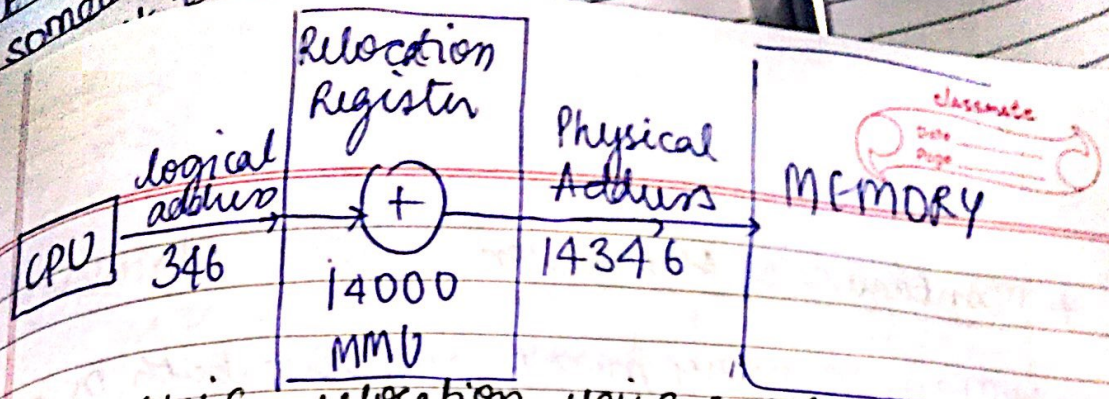
LOGICAL V/S PHYSICAL ADDRESS SPACE

– Generated by CPU, also referred to as virtual address	– Address seen by the memory unit
– Same in compile time and load-time address-binding schemes.	
– Both differ in execution-time address binding scheme.	
– Set of all logical addresses generated by a program	Set of all physical addresses generated by a program

– An address generated by a CPU is referred to as a <u>logical address</u>, whereas an address seen by the memory unit, the one loaded into memory-address register of the memory is <u>the physical address</u>.

Dynamic allocation using a relocation register

– Routine is not loaded until it is called.
– Better memory-space utilization, unused routine is never loaded.
– All routines kept on disk in relocatable load format.
– Useful when large amounts of code are needed to handle infrequently occuring cases.
– No special support from OS is required
+ Implemented through program design.
+ OS can help by providing libraries to implement dynamic loading.



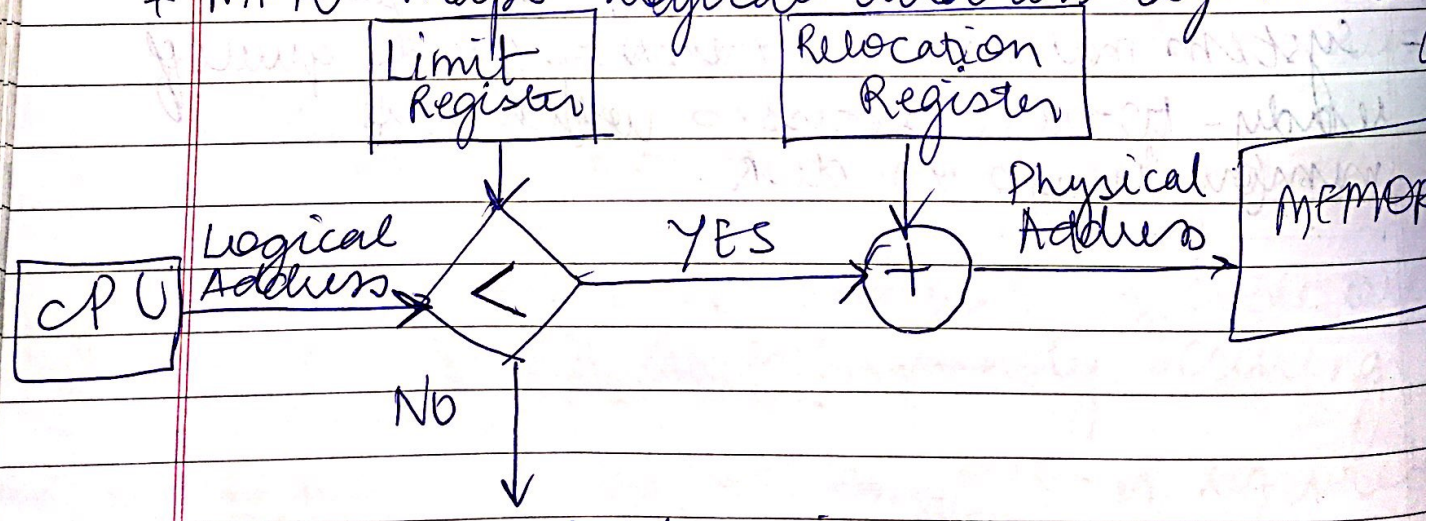
Dynamic relocation using a relocation register

SWAPPING

- A process can be swapped temporarily out of memory to a backing store, then brought back into memory for continued execution.
- Total physical memory space of processes can exceed physical memory.
- Backing Store - fast disk large enough to accomodate copies of all memory images for all users, must provide direct access to these memory images.
- Roll out, Roll in - swapping variant used for priority-based scheduling algorithms.
- Lower priority process is swapped out so higher priority process can be loaded and executed.
- Major part of swap time is transfer time.
- Total transfer time is directly proportional to the amount of memory swapped.
- System maintains a ~~memory~~ ready queue of ready-to-run processes which have memory images on disk.

+ Contiguous Allocation

- Main memory must support both OS and user processes.
- Limited resources, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two partitions:
  + Resident OS, usually held in low memory w/ interrupt vector.
  + User process then held in high memory.
  + Each process contained in single contiguous section of memory
  + Relocation registers used to protect user processes from each other and from changing OS code and data.
  + ~~Base~~ Limit register contains range of logical addresses - each logical address must be less than the limit register.
  + Base register contains value of smallest physical address.
  + MMU maps logical address dynamically



DYNAMIC STORAGE-ALLOCATION PROBLEM

n size from a list of free holes.

- FIRST FIT: Allocate the first hole that is big enough
- BEST FIT: Allocate the smallest hole that is big enough, must search entire list, unless ordered by size, produces smallest leftover hole.
- WORST FIT: Allocate the largest hole, must search entire list.

MULTIPLE PARTITION ALLOCATION

- Degree of multiprogramming limited by number of partitions.
- Variable-partition sizes for efficiency
- Hole - block of available memory, various sized holes are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accomodate it.
- Process exiting frees it partition, adjacent free partitions combined.

OS maintains info about:

- Allocated partitions
- Free partitions - holes

FRAGMENTATION

- External fragmentation - Total memory space exists to satisfy a request, but it is not contiguous.

- Internal fragmentation - Allocated memory may be larger than requested memory. This size difference is memory internal to a partition.

- REDUCE external fragmentation by compaction.
  + Shuffle memory contents to place all free memory together in one large block.
  + compaction is possible only if relocation is dynamic, and is done at execution time
  + I/O problem
    Latch job in memory while it is involved in I/O.
    DO I/O only into OS buffers.
- Backing store has same fragmentation problems.

SEGMENTATION - Memory managment scheme that supports user view of memory.
- A program is a collection of segments.
- A segment is a logical unit such as: main program, procedure, method function.

PAGING

- Physical address space of a process can be non-contiguous; process is allocated physical memory whenever the latter is available
- Avoids external fragmentation.
- Avoids problem of varying sized memory chunks.

* Segmentation permits the physical address space of a process to be noncontiguous. Paging is another memory management scheme.

- Divide physical memory into fixed sized blocks called FRAMES
  + Size is power of 2, b/w 512 Bytes & 16 Mbytes

- Divide logical memory into blocks of same size called PAGES.
- Keeps track of all free frames.

- To run a program of size N pages, need to find N free frames and load program.

- set up a page table to translate logical to physical addresses.

- Backing store likewise split into pages

- Still have internal fragmentation

Address Translation Scheme.

— Address generated by CPU is divided into :

PAGE NUMBER (p) - used as an index into a page table which contains base address of each page in physical memory.

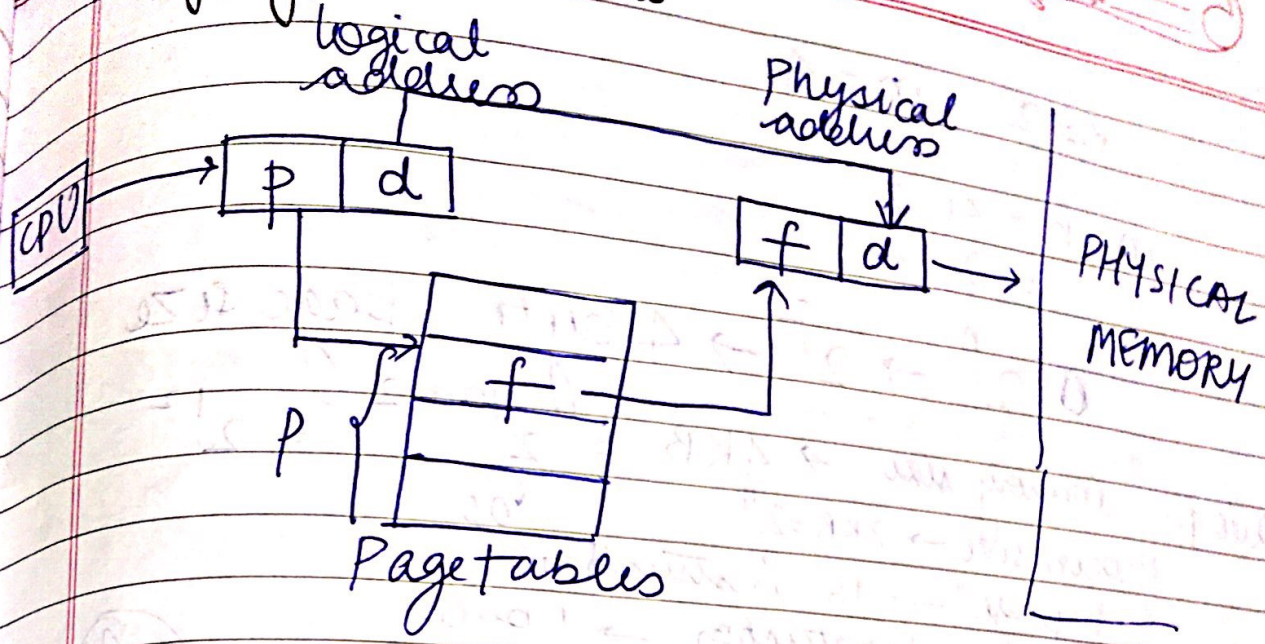
PAGE OFFSET (d) - combined with base address to define the physical memory address that is sent to the memory unit.

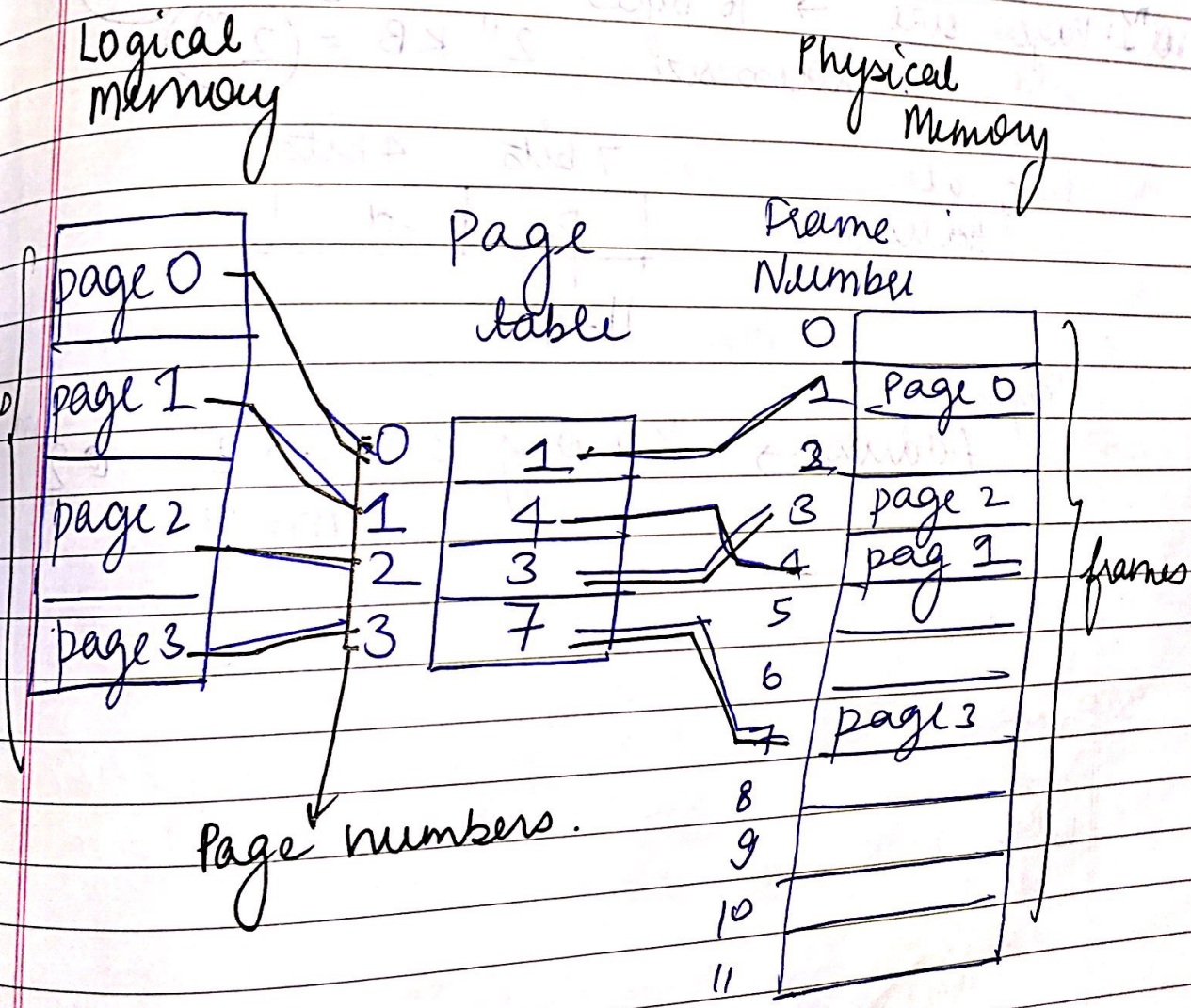
— For a given logical address space 2^m and page size 2^n.

Page number	Page Offset
p	d
m − n	n

Scanned with CamScanner

Paging Hardware





$n = 2 \quad m = 4$

$m - n = 4 - 2$
$= 2 \rightarrow$

$2^n \rightarrow 2^2 \rightarrow$ 4 byte page size

Ques 1: Memory size $\rightarrow$ 4KB $= 2^{10+2} = 2^{12}$
Process size $\rightarrow$ 2KB $= 2^{11}$
$\downarrow$ OS.
1 page $\rightarrow$ 16 instructions
1 instruction $\rightarrow$ 1 bytes

Sol^n 1: Page size $\rightarrow$ 16 bytes $= 2^4$ B $= 2^n$
Process size $= 2^{11}$ KB $= 2^m$

Logical Address $\rightarrow$

7 bits	4 bits
p	d

11-4

Physical Address $\rightarrow$ Memory size $\rightarrow 2^{12} = 2^m$
$m = 12$

Page Table

- Page table base register PTBR points to the page table.
- Page table length register PTLR indicates size of the page table
- Every instruction / data access requires two memory accesses - one for page table and one for data/instruction

TLBs - TRANSLATION LOOK-ASIDE BUFFERS

- Two memory access can be solved by the use of a special fast-lookup hardware ~~called~~ cache called associative memory or TLBs.
- Some TLBs store address-space identifier in each TLB entry. - uniquely identifies each process to provide address space protection for that process.

ASSOCIATIVE MEMORY - parallel search

page #	frame #

Address translation (p, d)

- If p is in associative register, get frame # out.
- Get frame # from page table in memory.

Scanned with CamScanner

~~Effective Access Time~~

MEMORY PROTECTION

- memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.

- VALID - INVALID bit attached to each entry in the page table:

VALID – Associated page is in the process' logical address space

INVALID – Indicates that the page is not in the process' logical address space

INVERTED PAGE TABLE

- Track all physical pages than having a page table and keeping track of all logical pages.
- One entry for each real page of memory.

- Entry consists of the virtual address of the page stored in that real memory location with information about the process that owns the page.

- Decreases memory needed to store each page table, but increases time needed to search the table when a page ~~tab~~ reference occurs

Scanned with CamScanner

VIRTUAL MEMORY - separation of user logical memory from physical memory.

- Only part of the program needs to be in memory for execution.
- Logical address space needs to be in memory for execution.
- Allows address space to be shared by several processes.
- Allows for more efficient process creation
- More programs running concurrently
- Less I/O needed to load or swap processes.

Virtual Address space: logical view of how process is stored in memory.

- Starts at address 0, contiguous adresses until end of space
- Physical memory is meanwhile organized in page frames.
- Memory Management Unit must map logical to physical.
- Can be implemented by demand paging & demand segmentation.

Scanned with CamScanner

Date
Page

DEMAND PAGING

- Entire process into memory at load time
- Bring a page into memory only when it is needed.

+ Less I/O needed, no necessary I/O.
+ Less memory needed
+ Faster Response
+ More users.

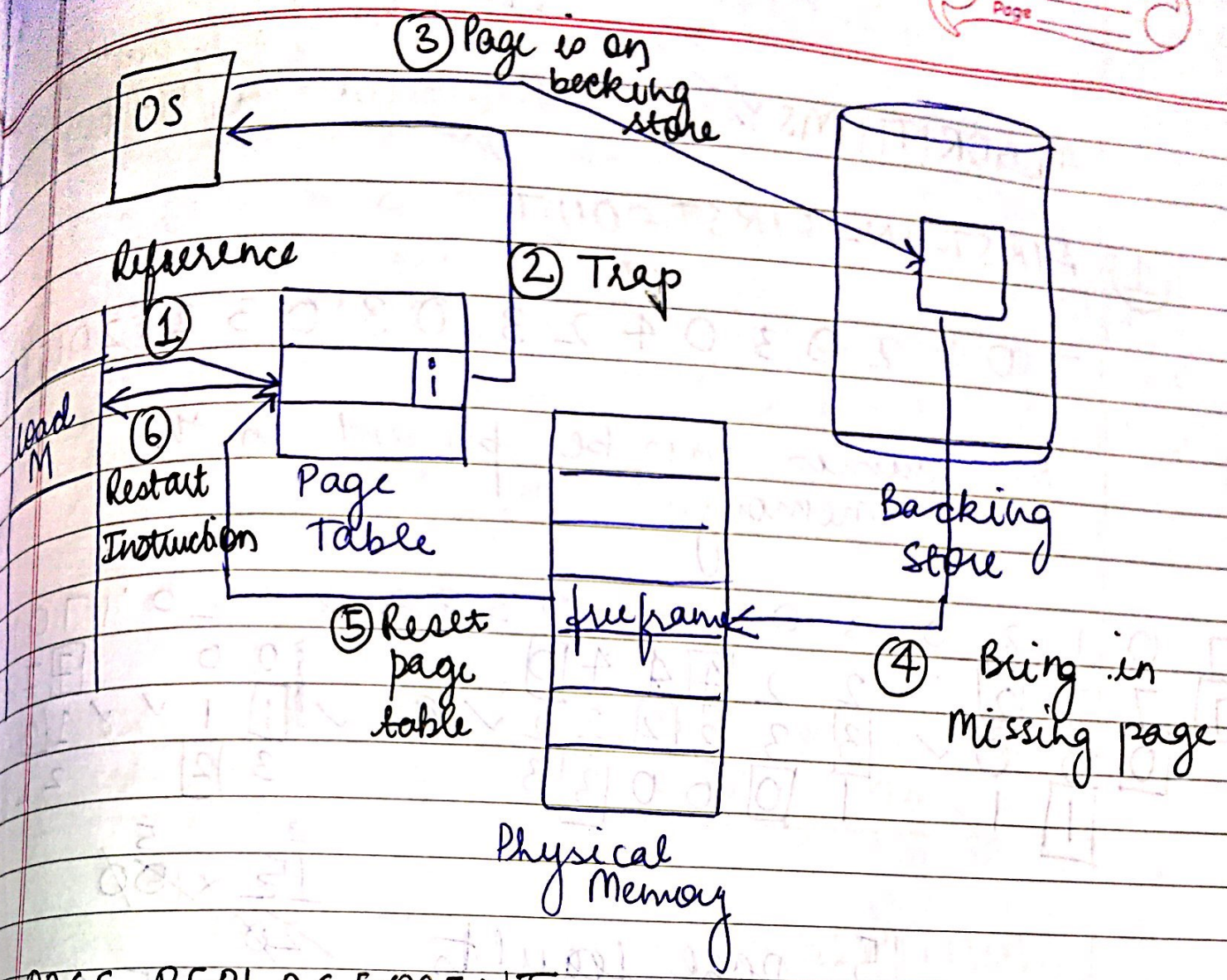
- Similar to paging system while swapping
- Page is needed - reference to it.
  - Invalid reference → abort
  - Not-in-memory → bring to memory.

PAGE FAULT

If there is a reference to a page, first reference to that page will ~~tra~~ trap to operating system: PAGE FAULT

1. OS looks at another table to decide:
   - Invalid reference → abort
   - Just not in memory.
2. Find free frame.
3. Swap page into frame via scheduled disk operation.
4. Reset tables to indicate page now in memory. Set validation bit → 'v'.
5. Restart the instruction that caused the page fault

Scanned with CamScanner



PAGE REPLACEMENT

1. Find the location of desired page on disk.
2. Find a free frame, use a page replacement algorithm to select a victim frame.
3. Bring the desired page into the free frame, update the page and frame tables.
4. Continue the process by restarting the instruction that caused the trap.

Page Rep Current Algo -
Fault Rate?
No. of Faults
check for Belady Anomaly

ALGORITHMS

① FIRST-IN-FIRST-OUT

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

3 - frames can be present in the memory.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0
7	7	7	2		2	2	4	4	4	0			0	0			7	7
	0	0	0	✓	3	3	3	2	2	2	✓	✓	1	1	✓	✓	1	0
		1	1		1	0	0	0	3	3			3	2			2	2

$\frac{15}{20} \times 100$

15 page faults

= 75%

→ Belady's Anomaly
Increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns.

② OPTIMAL ALGORITHM

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

9 page faults.

$$\frac{9}{20} \times 100 = 45\%$$

- Replace the page that will not be used for longest period of time.

③ LEAST RECENTLY USED ALGORITHM

- Replace the page that has not been used in the most amount of time.

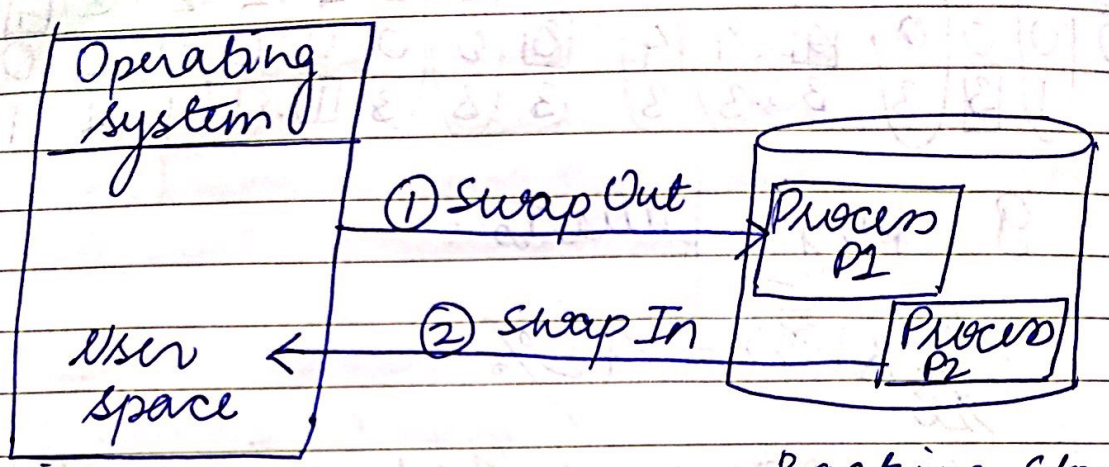
	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2	2	2	2	4	4	4	0			1		1		1		
		0	0	0	✓	0	✓	0	0	3	3	✓	✓	3	✓	0	✓	0	✓	✓
			1	1		3		3	2	2	2			2		2		7		

12 page faults

$$\frac{12}{20} \times 100 \rightarrow 60\%$$

Scanned with CamScanner

swapping of two processes using a disk as a backing store.



Main Memory Backing Store

SEGMENTATION

- MM scheme that supports the user view of memory.
- A logical address space is a collection of segments.
- Each segment has a name and a length.
- Logical address consists of two tuple:

  <segment-number, Offset>

- Segment table maps two dimensional physical addresses - each table has:

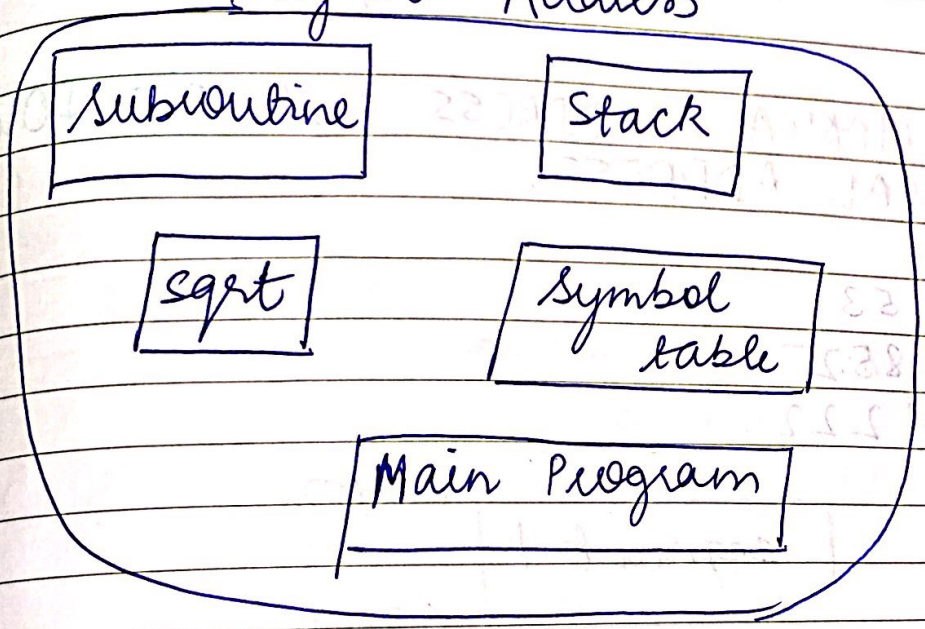
Base: Contains the starting physical address where the segments reside the memory.

Limit: Specifies the length of the segment.

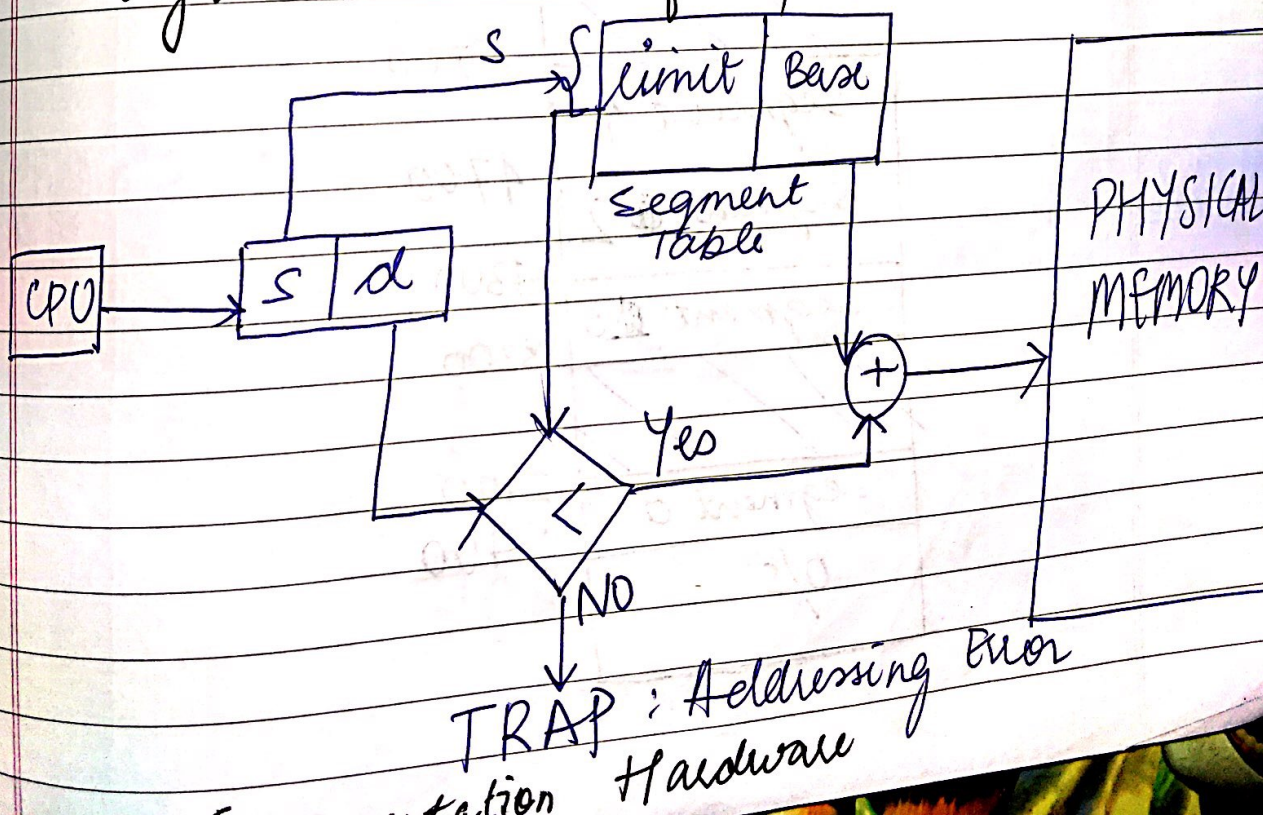
Scanned with CamScanner

- Segment table base register (STBR) points to the segment table's location in the memory.
- segment table length register (STLR) indicates number of segments used by a program.
- Segment number is legal if **s < STLR**.

Logical Address



Programmers view of a program.

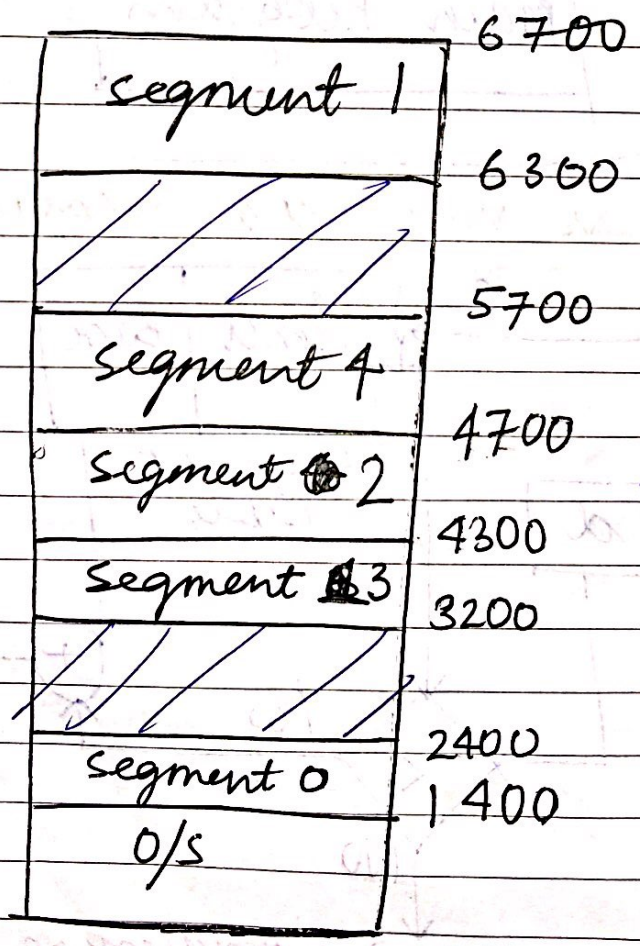


NUMERICALS :

segment No	limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

— Find PHYSICAL ADDRESS FOR THE FOLLOWING LOGICAL ADDRESS.

1. 2, 53
2. 3, 852
3. 0, 1222.



Scanned with CamScanner

$\langle s, d \rangle$

$s \rightarrow$ segment

$d \rightarrow$ offset.

$d \leq l \rightarrow$ limit

$d \rightarrow$ offset

(i) 2, 53

segment 2.

$d \leq l$.

$53 \leq 400$. ✓

$\therefore$

$$\begin{array}{r} 4300 \quad b \\ +\ 53 \quad \text{offset} \\ \hline 4353 \\ \hline \end{array}$$

(ii) 3, 852.

segment 3.

$852 \leq 1100$.

$$\begin{array}{r} 3200 \quad b \\ +852 \\ \hline 4052 \\ \hline \end{array}$$

4052 ✓

(iii) D: 1222

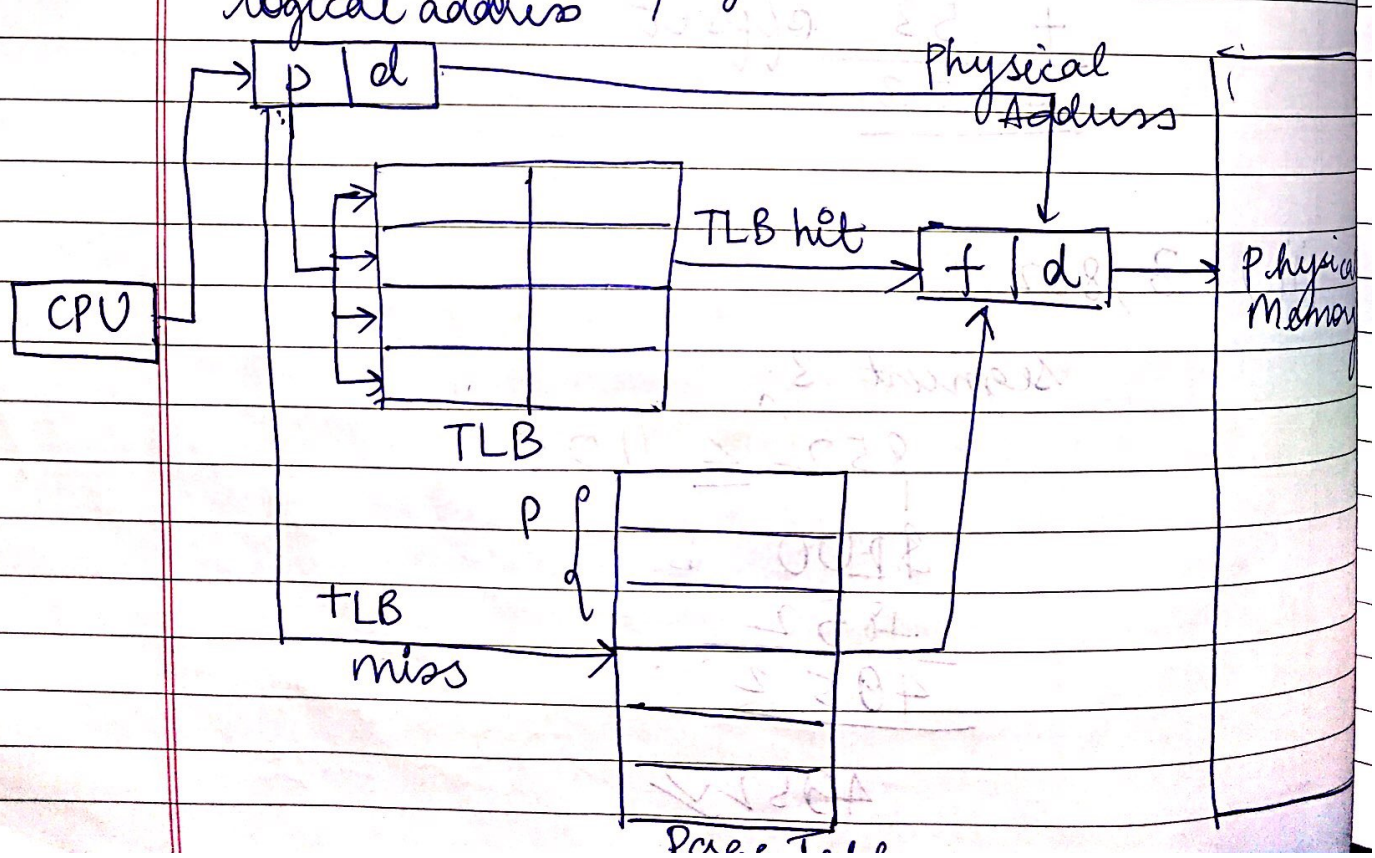
$d \leq l$.

$1222 \not\leq 1000$

* TRANSLATION LOOK ASIDE BUFFERS

Implementation of a page is done in the following ways:

- Page table is kept in main memory
- Page table Base register (PTBR) points to page table.
- Page table length register (PTLR) points the size of the page table.



- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or TRANSLATION LOOK-ASIDE BUFFERS (TLBs)
- A set of associative register is built of high-speed memory where each register consists of two parts: a key and a value.
- When the associative registers are presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is the output.
- A typical TLB table entry consists of page # and frame # when a logical address is generated by the CPU, its page number is presented to a set of associative registers that contain page number along with their corresponding frame numbers.

Page number found in associative register → Frame number available & used to access memory

Page number not found → memory reference made.

Frame number found

LOGICAL AND PHYSICAL MEMORY ADDRESS

- Logical address is the address of the instruction or data word as used by a program.
- Includes use of index, base or segment register
- A physical address is the effective memory address of an instruction or data word.
- The set of physical addresses generated during operating of system. The compile time and the load time address binding schemes result in environment where the logical and physical address are same, whereas during execution time addresses differ.

Effective Access Time

Scanned with CamScanner

UNIT #4

FILE SYSTEM

- File system provide efficient and convenient access to the disk by allocating data to be stored, located and retrieved easily
- The file system itself is generally composed of many different levels.

Application Program
↓
Logical File System
↓
File - Organization Module
↓
Basic file System
↓
I/O control
↓
Devices.

Layered File System

ACCESS METHODS

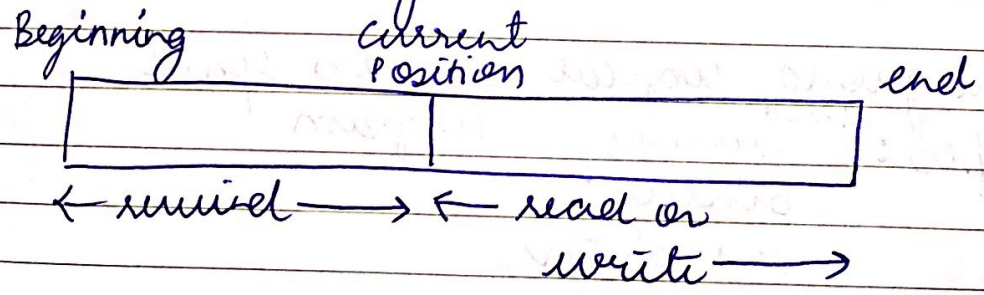
Files are used to store information. The information stored must be accessed and read into the computer memory.

Scanned with CamScanner

There are different ways in which we can access files

<u>Sequential Access</u> : The simplest access method. Information in the file is processed one record after another.
Eg: Editors and compilers access files using this method.

Read and write make up the bulk of the operations.
- A read operation - read_next ()
  - reads the file in the sequential manner and advances the file pointer, which tracks the I/O location.
- The write operation - write_next()
  - writes in the append mode i.e it writes in a file at the end of the newly written material.



<u>Direct Access</u> : File is made up of fixed-length logical records that allows programs to be read or write in no particular order.
- File is viewed as a numbered sequence of records.

We can read block 14 then read block 54 and then write block 7.

- Can be of great help for immediate access to information.
- read (n) - n is the block number.
  write (n) - n is the block number.

  n → or the block number is used as a parameter.
- Block no → relative block number.

Sequential	Direct
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

FILE CONCEPT

- Contiguous logical address space.
- Types: Data: numeric, binary, character; Program

Reso

DIRECTORY STRUCTURE

- Collection of nodes in which information of all the files are stored.
- Directory structure and files reside on disk.
- Backups are kept on tapes

ALLOCATION METHODS

Direct-access nature of disks gives us flexibility in the implementation of files. How to allocate space to such files so that that disk space is utilized. 3 major methods of allocating of disk space.

Contiguous Allocation:

- defined by the disk address and length (in blocks units) of the first block.
- b, b+1, b+2 ..... b+n-1.
  if the file is n blocks long and starts at b.
- Directory entry for each file indicates the address of the starting block and the length of the area allocated.

File	Start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- Suffers from external fragmentation

- As we delete and allocate space to files, the space is broken into pieces.
- When free space is broken into chunks and a large contiguous chuck is insufficient for a request.

Linked Allocation

- Solves all the problems of contiguous allocation with linked list allocation each file is a linked list of disk blocks, the disk block may be scattered anywhere on the disk.
- Directory contains the pointer to the first block and last block of the file.
- Can only be used efficiently only for sequential-access files.
- Space required for a pointer.
  pointer requires 4 bytes out of 512 byte block, ~~then~~ 0.78 % of block
- Requires slightly more space that it would otherwise.

File	start	end
1	9	25

→ DIRECTORY

Indexed Allocation.

- Linked allocation solves external-fragmentation and size-declaration problem of contiguous allocation.
- Indexed allocation ~~so~~ brings all the pointers together into one location: the index block.

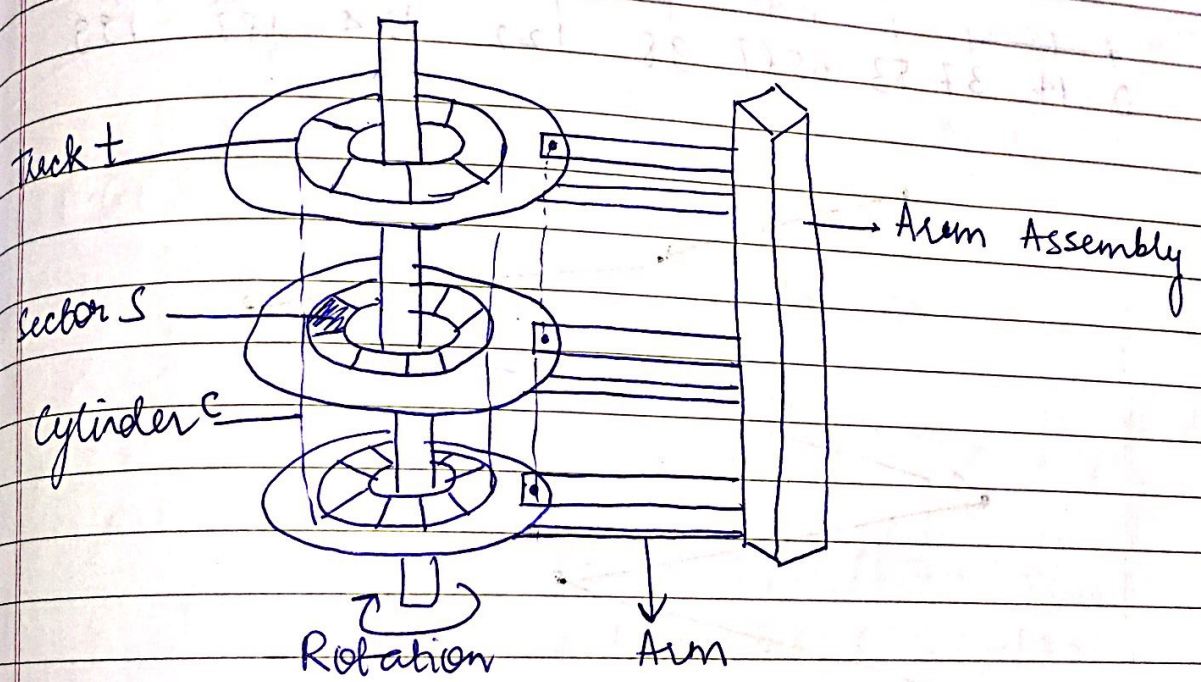
Scanned with CamScanner

Each file has its own indexed block which is an array of disk - block addresses.
- ith entry in the index block points to ith block of the file. Directory contains the address of the index block.

File	Index block
1	19

19	9	-1
	16	-1
	01	-1
	10	
	25	

DISK STRUCTURE



Magnetic Disks – It provide the bulk of secondary storage for modern computer system. Conceptually disks are relatively simple. Each disk platter has a flat circular shape like O, C O.

Magnetic Tapes – It was used as an early secondary storage medium. Although, it is relatively permanent and can hold and can hold large quantities

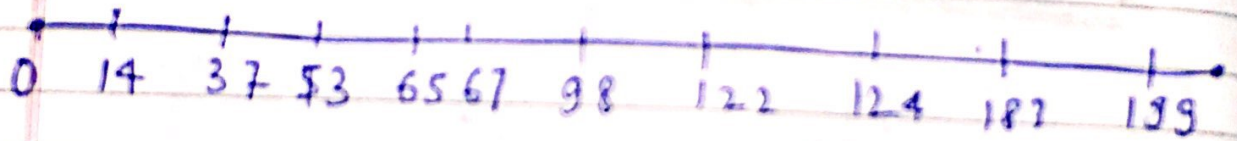
of data, its access time is slow compared with that of main memory and magnetic disk.

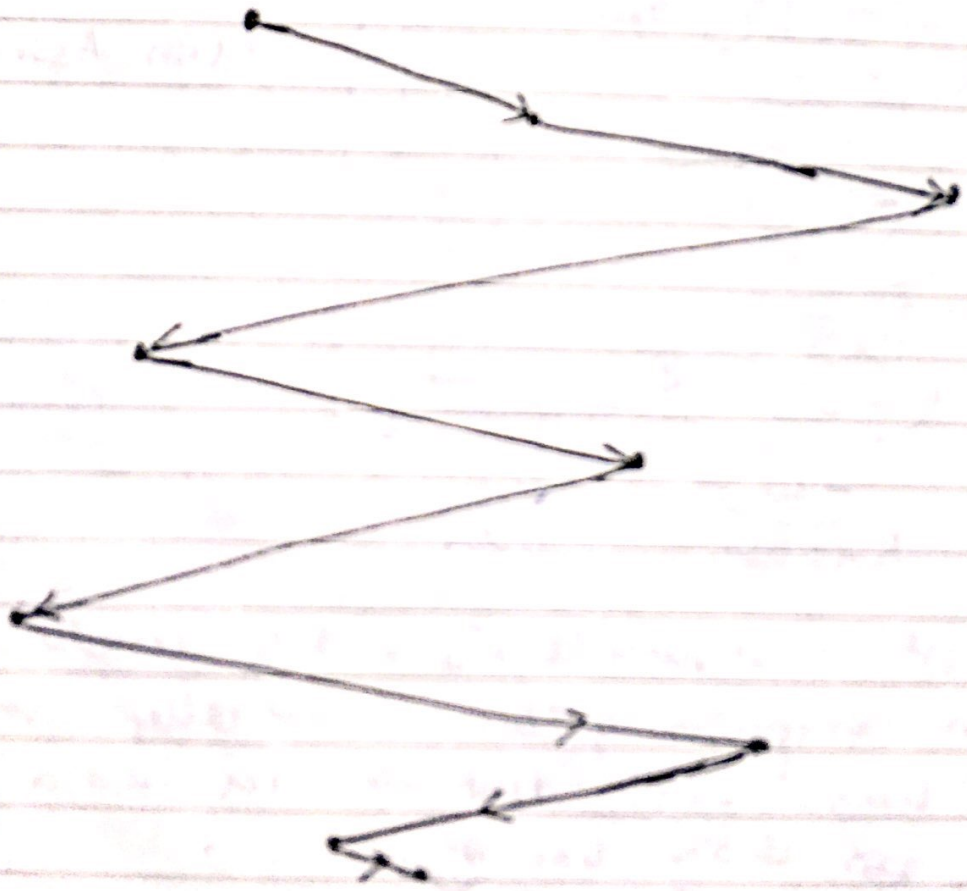
DISK SCHEDULING

- FCFS scheduling.

queue : 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



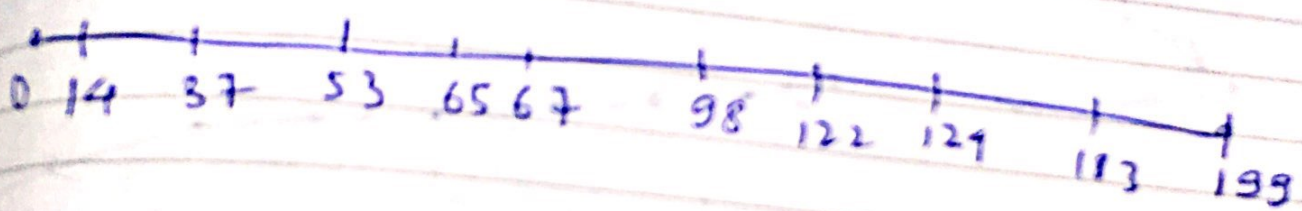


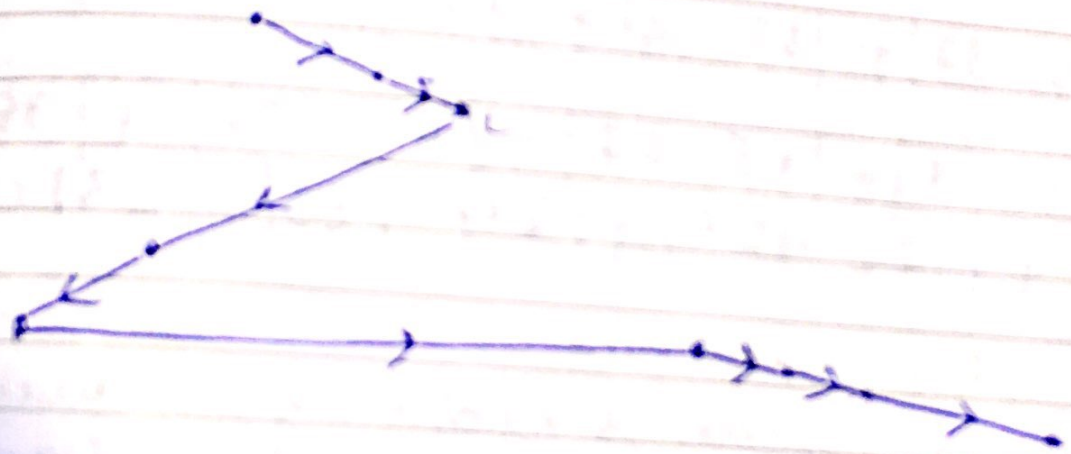
Scanned with CamScanner

$|53-98| + |98-183| + |183-37| + |37-122|$
$+ |122-14| + |14-124| + |124-65| +$
$|65-67| = 640$ cylinders.

shortest-seek-time-first (SSTF)

queue - 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53





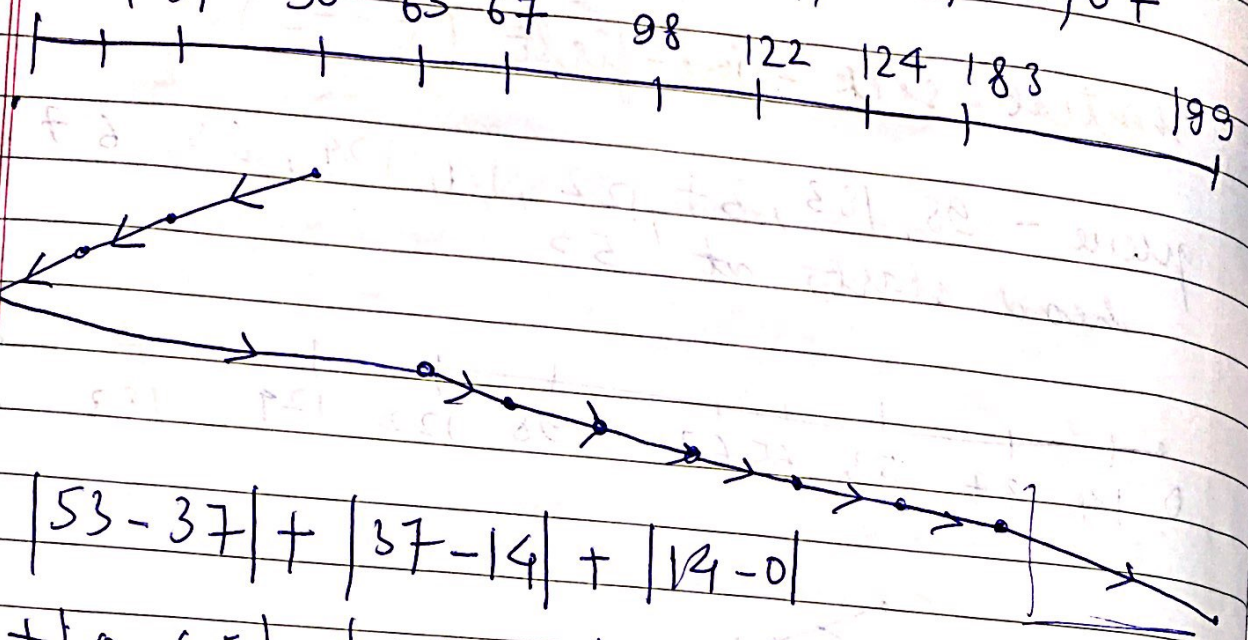
$|53-65| + |65-67| + |67-37|$
$+ |37-14| + |14-98| + |98-122|$
$+ |122-124| + |124-183| = 236$

— SCAN scheduling

— Assume head is moving towards zero.

queue → 98, 183, 37, 122, 14, 124, 65, 67

head starts → 53



$|53-37| + |37-14| + |14-0|$

$+ |0-65| + |65-67| + |67-98| + |98-122|$

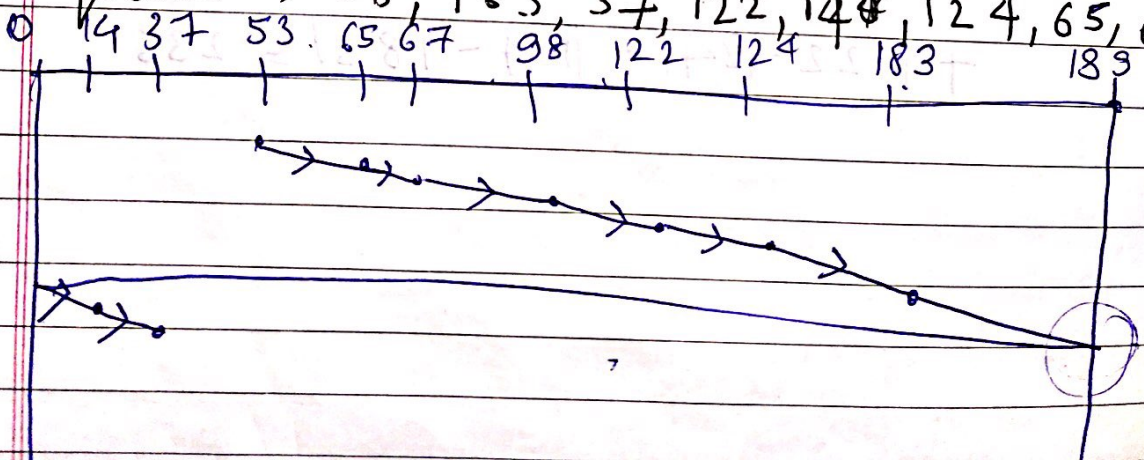
$+ |122-124| + |124-183| = 231$ cylinder

— C - SCAN Scheduling

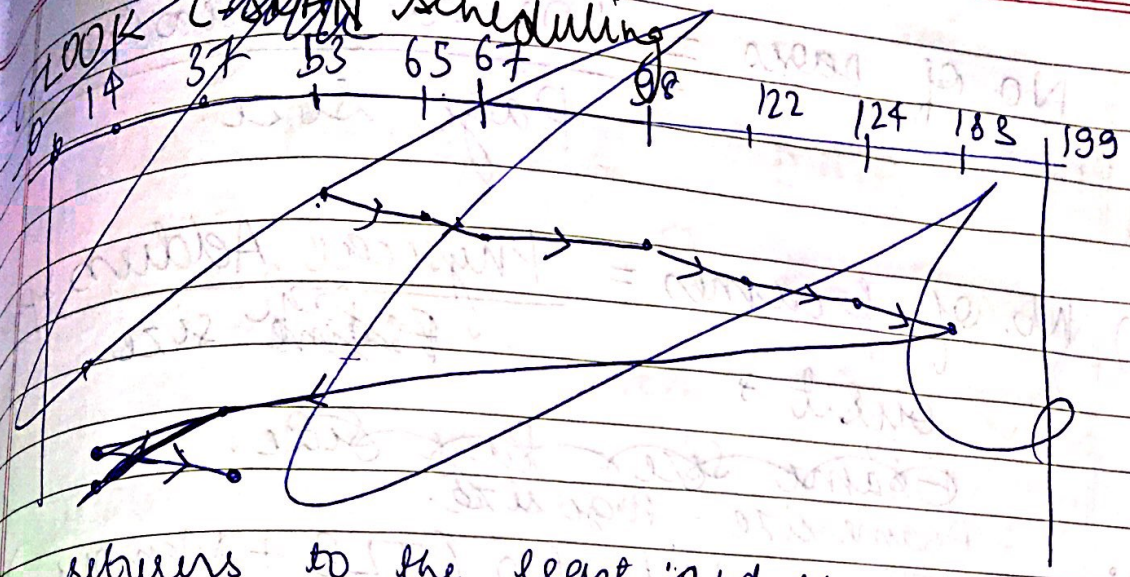
— Circular scan scheduling is variant of scan designed to provide a uniform wait time.

head starts → 53

queue → 98, 183, 37, 122, 14, 124, 65, 67

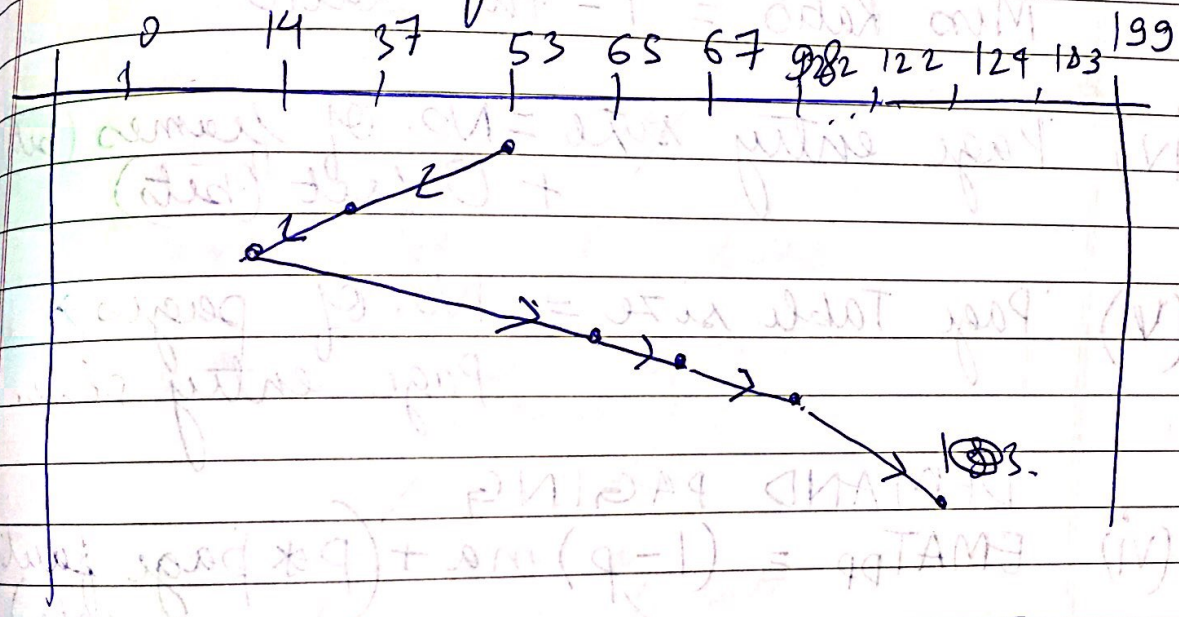
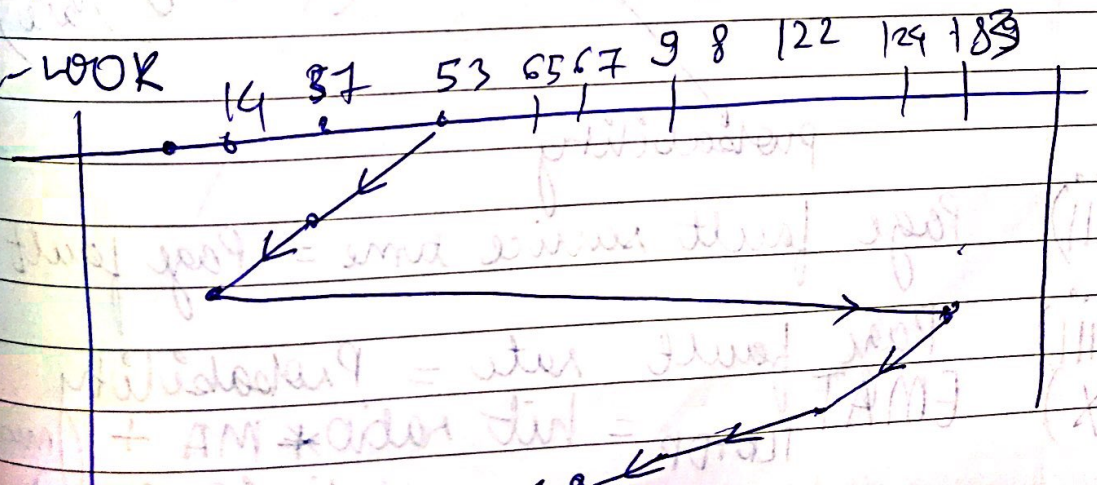


~~C-LOOK C-SCAN scheduling~~



returns to the least and then starts again.

LOOK scheduling



C-LOOK