JSPM

Page No. 1

Q.1 what is the scheduling criteria explain any two primitive scheduling algorithm with example.

In multiprogramming enviironment there may be the situation where 2 or more processes are simultaneously in ready state & if only one CPU is available, then a choice has to be made as to which process should execute next, The part of OS that makes this choice is called scheduler & the algorithms it use are called scheduling algorithms

A major issues related to scheduling when to make scheduling decisions there are vecieties of situations where scheduling is required.

1) when a new process is created, then decision regarding whether parent or child process should be executed.

2) when a process exits then some other process must be selected from set of ready states. If no process is in ready state a system supplied idle process is execute.

3) when a process blocks on Input/output on a semaphore or for some other reason another process has to be selected to run

4) when an I/O intrupt occurs a scheduling decision may be made.

scheduling algorithms can be divided into two catageries, depending on how they deal with clock intrrup

1) A non-preemptive

2) A preemptive

* non preemptive :-

A non preemptive scheduling algorithm pi

process to run & then just lets it run until it blocks or until voluntarily releases its CPU In other no scheduling decisions are made during clock intrrupts after clock intrrupt processing has been completed the process that was running before the interrupt it always resumed.

2) A preemptive :-

A preemptive scheduling algorithm picks a process to run & lets it run for a maximum of some fixed time. If it is still running at the end of time interval it is suspended & the scheduler picks another process to run

Different scheduling algorithm are needed in different environment & different application areas. Three distinguishable environment are -

1) Batch system
2) Interactive system
3) Real time system.

1) Batch system :-

No user are impatiently waiting for quickly response. This approch reduces process switches thus improving performance

2) Interactive system :-

Here preemption is essential to keep one process from hogging the CPU & denying service to others. one process might shut out all other indefinately. Hence preemption is needed to prevent this behaviour

3) Real time system :-

Here preemption is sometimes not needed since the processes know that they may not run for long periods of time & usually to their work quickly.

* Goal of scheduling algorithms

1) All systems:
   - Fairness: giving each process a fair share of CPU
   - Policy enforcement: seeing that stated policy is carried out
   - Balance: keeping all parts of system busy

2) Batch system:
   - Throughput: maximum jobs per hour
   - Turn around time: minimum time between submission & termination
   - CPU utilization - keep CPU busy all the time

3) Interactive system:
   - Response time: respond to request quickly
   - proportionality: meet user expectations.

4) Real time systems:
   - meeting deadlines: avoid losing data
   - predictability: avoid quality degradation in multimedia systems.

* scheduling criteria:

Different scheduling algorithms have different properties & are applicable in different situations selecting which algorithm to use in which situation.

ii) CPU

is to keep CPU as busy as possible. It may ranges from 0 to 100 %.

iii) throughput:

is the number of processes completed per time unit It may range from 10 processes 1 second upto 1 process

iii) turnaround time:

is the time interval between submission of a process & completion of a process It is the sum of the periods spent waiting to get into memory. waiting in ready queue executing or CPU & doing I/O

iv) waiting time:

is the amount of time a process spends waiting in ready queue It is the sum of the periods spent waiting in ready queue.

v) Response time:

It is the time form submission of a request until first repo response is produced. It is the amount of time taken to start responding & not the time taken to output the response.

vi) We want to maximum CPU utilization & throughput & to minimize turnaround waiting & response time for interactive systems minimizing the variance in response time is important then minimizing average response time.

* scheduling algorithm:

CPU scheduling deals with problems of deciding which of the processes in ready queue is to be allocated the CPU.

There are many different CPU scheduling algorithms

1) First-come, first-served scheduling
2) Shortest job First scheduling
3) priority scheduling
4) Round Robin scheduling
5) Fair share scheduling.

1) First-come First served scheduling-
- It is a non preemptive algorithm
- In FCFS scheme, the process that request the CPU first is allocated the CPU
- The FIFO queue is used to the FCFS implementation.
- There is a single queue of ready processes & process are assigned to the CPU in the order they request it.
- The code of FCFS scheduling is simple to write & understand.
- The average waiting time under the FCFS policy is often quite long

2) Shortest job First scheduling:
- The SJF scheduling algorithm associated with each process the length of the processes next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU brust.
- If the next CPU bursts of two processes are the same FCFS scheduling is used to break the tie.
- Thus insted of SJF most approprite term is shortest-next-CPU-burst algorithm because scheduling depends on the length of the next CPU burst of a process rather than its total length.

- This approach takes into account a
re scheduling it
- Each user is allocated some
as in such a way as to
- Thus if a

- It is non preemptive as well as preemptive algorithm
- optimal when all the jobs are available simultaneously.
- shortest jobs do not have to wait for lengthy jobs hence increasing
average turn around time & reducing average waiting time

7. priority scheduling:-
- priority scheduling algorithm is associated with SJF.
- A priority is associated with each process & the CPU is allocated to the process with the highest priority equal priority process are scheduled in FCFS order.
- An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the next CPU burst. The larger the CPU burst the lower the priority & vice versa

8. Round Robin algorithm / scheduling
- The RR scheduling algo is designed especially for time sharing system.
- It is similar to FCFS scheduling. but preemption is added to swit ch between processes.
- A small unit of time called a time quantum or a time slice is defined. A time quantum is generally from 10 to 100 millisecond.
- The ready is queue is treated as a circular queue
- To implement RR scheduling the OS keep the ready queue as a FIFO queue of process.
- Each process is assigned a time interval called quantum, in which it can run. If process is still running at the end of the quan tum, the CPU is preempted & given to another process
- If process has blocked or finished before quantum has elapsed CPU switches the next process in queue.

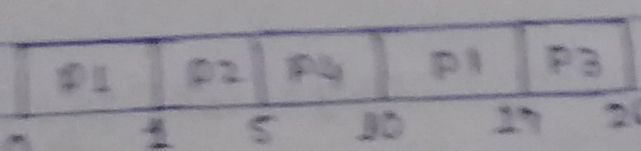
9. Fair share scheduling:-

10. Real time System:-

- This approach takes into account who owns a process before scheduling it
- Each user is allocated some fraction of CPU & scheduler picks & proces so in such a way as to enforce it.
- Thus if 4 users have each between promised 25% of CPU, will each get that, irrespective of number of process owned by each one of them

* non preemptive scheduling algorithm.

| Process | Arrival time | Bus time |
|---|---|---|
| P1 | 0 | 8 7 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |




WAT = start time - Arrival time + new start time - old finish time

P1 = 0 - 0 + 10 - 1 = 0 + 9 = 9
P2 = 1 - 1 = 0
P3 = 17 - 2 = 15
P4 = 5 - 3 = 2

avg = $\frac{9+0+15+2}{4} = \frac{26}{4}$ 6.5

6.5

TT = (Finish time - Arrival time) + (new finish time - old finish time) + old time

P1 = 1 - 0 + 17 - 1 + 8 = 1 + 16 + 8 = 25
P2 = 5 - 1 = 4
P3 = 26 - 2 = 24
P4 = 10 - 3 = 7

avg = $\frac{25+4+24+7}{4} = \frac{60}{4} = 15$

preemptive priority scheduling algorithm.

| process | bus time | Arrival time | priority |
|---|---|---|---|
| P1 | 8 | 0 | 4 |
| P2 | 6 | 1 | 6 highest |
| P3 | 7 | 3 | 3 |
| P4 | 9 | 3 | 1 |

| P1 | P2 | P3 | P3 | P4 |
|---|---|---|---|---|
| 0 | 1 | 7 | 14 | 21 | 30 |

AWT = (ST-AT) + nST + oFT

P1 = 0 - 0 + 7 - 1 = 6
P2 = 1 - 1 = 0
P3 = 14 - 3 = 11
P4 = 21 - 3 = 18

avg = $\frac{18+11+0+6}{4} = \frac{35}{4}$ 8.7

TT = (FT - AT) + (nST - oFT) + bt

P1 = (0-0) + 7 - 1 + 8 = 0 + 6 + 8 = 14
P2 = 7 - 1 = 6
P3 = 21 - 3 = 18
P4 = 30 - 3 = 27

avg = $\frac{27+18+6+14}{4} = \frac{65}{4} = 16.22$

Q 2 what is deadlock explain deadlock avoidance

concept :- in a multiple environment several process may request finite no of resources A process request resources if the resource are not available at that time, the process enters a wait state within processes may never again change state because the resources they have requested are held by other waiting processes this situations is called deadlock deadlock occure when some process makes a request that cannot be granted immediately.

deadlock may involve some or different resource types. there are various features that characterised deadlock i.e a deadlock function can arise if following 4 conditions hold simultaneously in a system.

1) mutual Exclusion:-

At least one resources must be held in a non sharable mode i.e only one process at a time can use the resource if another process requests that resource if another process request that must be delayed until the resource has been released.

2) circular wait:-

To impose total ordering of all resources type & to require that each process request resources in an increasing order enumeration whenever a process request an instance of resource type with higher order it has to release resource to lower order if acquired.

* Resource allocation-

A dired edges from resource type $j \rightarrow P$ called as assignment edges. signifies that an instances of resour-

ce type Rj has been allocated to process pi. A directed edge pi→Rj called request edges. In addition we introduce new type edges called claim edge. Ii→Rj which indicates that process pi may request resource Rj at some time in future. this edge resembles a request edge in direction but is represented by dashed line.

suppose that pi request Rj the request can be granted only if converting the request edge Ii→Rj to an assignment edge Rj→Pi does not result in formation of cycle in resource allocation graph IF no cycle in resource allocation graph.

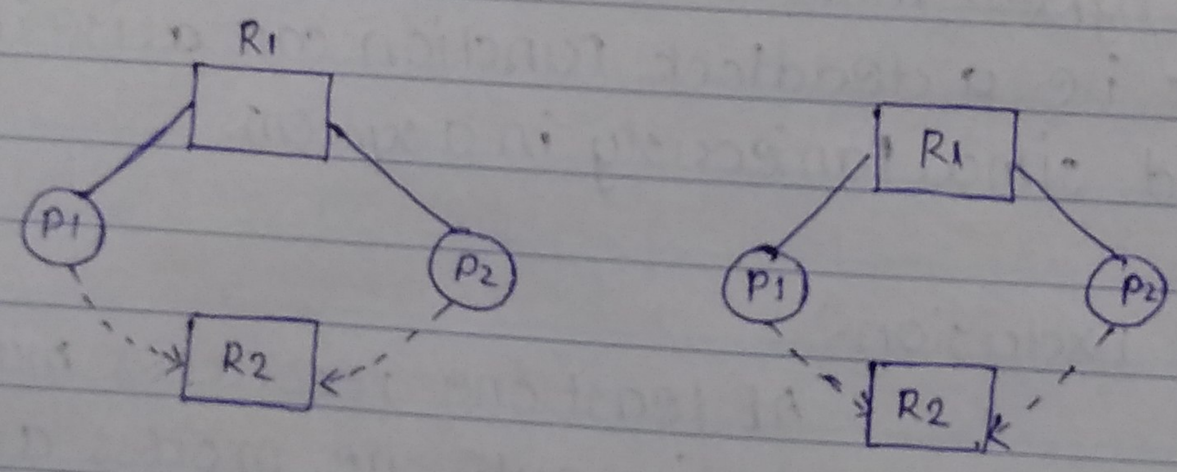



Fig:- Resource allocation graph
deadlock avoidance

1) Hold & wait:

A process must be handling at least one resource waiting to acquire additional resource that are currently being hold by other processes

2) No preemption:-

Resource can not be preempted i.e resource can be released only voluntarily by the process holding it after that process has completed its task.

3) circular wait:-

A set of {p0, p1 ..... pn} of waiting processes must exist such that p0 is waiting for a resource that is held by p1, p1 is waiting for resources that is held by p0

we can deal with deadlock problems in one of the following 3 ways

1) we can use a protocols to prevent or avoid deadlock ensuring that the system will never enter a deadlock state
2) we can allow the system to enter a deadlock state & detect & it recovery
3) we can ignore the problem altogether & pretend that deadlock never occur in the system.

* Banker's Algorithms-

The resource allocation graph algorithm is not applicable to a resource allocation system with the multiple instances of each resource type Hence bankers algorithm was introduced. the name was choosed as these algorithm could be used in a banking system to ensure that the bank never allocates its available cash such that it can be longer satisfy the needs of all its customers

when a new process enters the system, it must declared the maximum number of instances of each resources type that it my need. this number nfy not exceed the total number of resource in the system. when a user request a set of resources the system must determine whether the allocation of these resource will leave the system in safe state

Let, n = no of process in system
m = no of resource type

* we needs following data structure :-

1) Available :-

Indicate no of available resource of each type if available[j]=k there are k instances of resource type Rj avail

2) max:- define maximum demand of each process if max[i,j]=k then process pi may request at most k instances of resource type Rj

3) Allocation:- defines the number of resources of each types currently allocated k instances of Resource type Rj:

4) Need:- indicates remaining resource need of each process if need[i,j]=k then process p may need k more instances of resource type.

* safe state:-

A safe state is there if system can allocate resources to each process in some order & still avoid a deadlock. If the resource that process pi needs is not immediately available then pi can wait until all pi have finished when they have finished pi can obtain all of its needed resource complete its designed task, return it allocate resources & terminate when pi terminated pi+1 can obtain its needed resources & so on, If no such sequence exists then system state is said to be unsafe

A safestate is not a deadlock state. deadlock state is unsafe state but not all unsafe state are deadlock. Initially a system is in safe state whenever process request a resource that is currently available that system must decide whether the resource can be allocated immediately or whether the process must wait. the request is generated only if the allocation leaves the system in a safe state.

6 | 3 Explain segmentation in memory management

concept:-

we can impriace CPU utilization & speed of computers is response to its users by allocating various process to share CPU But for this we must be able to share memory as well selection of memory management method for specific system depends on many factors especially on hardware design of system.

memory is central of the operation of modern computer system It consist of large array of words or bytes each with its own address. CPU fetches instructions from memory according to the value of program counter. The instruction is decoded & may cause operands to be fetch from memory after the instruction has been executed on the operands result may be stored back in memory the memory units see only a stream of memory address does not know how they are generated

The normal procedure is to select one of the processes in the ready queue & to load that process is memory as the process is executed it access instruction & data from memory Eventually the process terminates & memory space is declared available

* Contiguous & non-contiguous memory allocation.

The main memory has to accommodate both os & various user processes os may be placed either in low memory high memory generally interrupt vector & os are placed in low memory.

various user process resides in memory at the same time the issues is how to allocate available memory to the process in ready queue In contiguous

memory allocation each process in contained in single contiguous section of memory.

The simplest method of memory allocation is to divided memory into several fixed sized partition where each partition may contain exactly one process. A generalization to fixed partition scheme MVT is generally used in batch environment Initially all memory ~~is in batch environment~~ available for user processes & considered as one large enough for this process If we find such we allocate only has much memory has needed keeping the rest available satisfy to future request

As any given time we have a list of available block size & ready queue. os can order the ready queue according to some scheduling algorithm generally a set of holes of various size is scattered throughout the memory at any given time. If the hole is too large it is split into 2 one part is allocated to arriving process other the return to be set of holes. when a process terminates. it releases its block of memory which then placed block in the set of holes. If the new holes is adjecent to other holes. these adjucent holes are merged to form one large hole. the system can use dynamic storage allocation methods -firstfit, best fit or worst fit for allocating memory. but these algorithm suffer from external fragmentation as process are loaded & removed from memory free memory space is broken into little pices.

Internal fragmentation occurs when memory is allocated in unit of blocks & if process gets a block slightly larger than request memory. this difference is called internal fragmentation memory that is internal to a partition but is not being used The solution is external fragmentation is compaction. compaction is possible only if relocation is dyno-

Segmentation is memory management scheme that support this user view of memory a logical address space is a collection of segments. Each segment has a name & a length the addresses specify both the segment name & the offset within the segment.

The user therefore specifies each address by two quantities a segment name & offset. For simplicity of implementation segments are numbered & are referred to by a segment number rather than by a segment name thus a logical address consist of a two tuple

<segment number, offset>

Normally the user program is complied & the compiler automatically constructs segment reflecting the input program.

A C compiler might create seprate segment for the following

1) The code
2) Global variables
3) The heap from which memory is allocate
4) The stack used by each thread
5) The standard c library.

* Hardware:

We can refer to object in the program by a two dimensional address in the actual physical memory is still of course a one dimensional sequence of bytes. thus we must define an implementation to map two dimentional physical address.

The mapping is effected by the segment tables. Each entry in the segment table has a segment base & segment units. The segment base contains starting physical address where the segment resides in memory whereas the segment limits specify the

it's done as the execution time.

* segmentation.

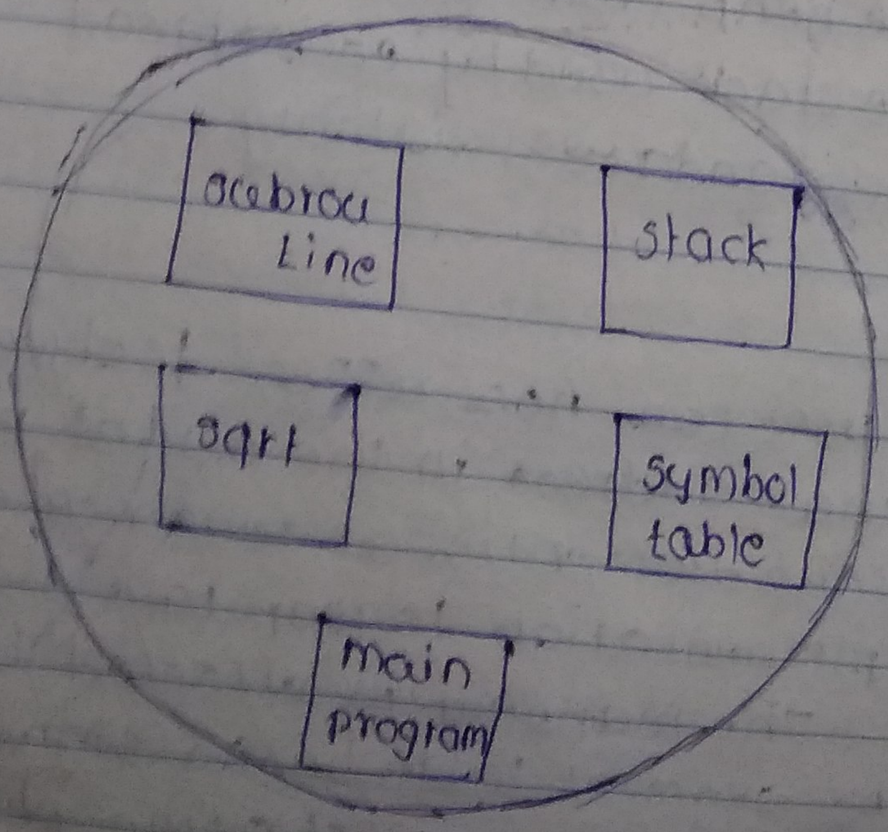
An important aspect of paging is the seprati on of the user view of memory & actual physical memory The user view is mapped into physical memory. This mapping allows differentiation between logical & memory & physical memory.

* Basic Methods.

users view memory as the allocation of the collection of variable-sized segment with no necessary ordering among segments.

* users view of programs.

user view of program you think of it as main program with a set of methods procedures or function it may also include various data structures object, array, stack, variable etc



length of the segments.
- A logical address consist of the two parts. a segment number s & an offset into a segment
- The offset d is logical address must between 0 & segmit limit.
- If it is not we trap to the operating system when an offset is legal it is added to the segment base & produce an address in physical memory of the desired bytes. the segment table is thus essentially an array of base limit register pairs.

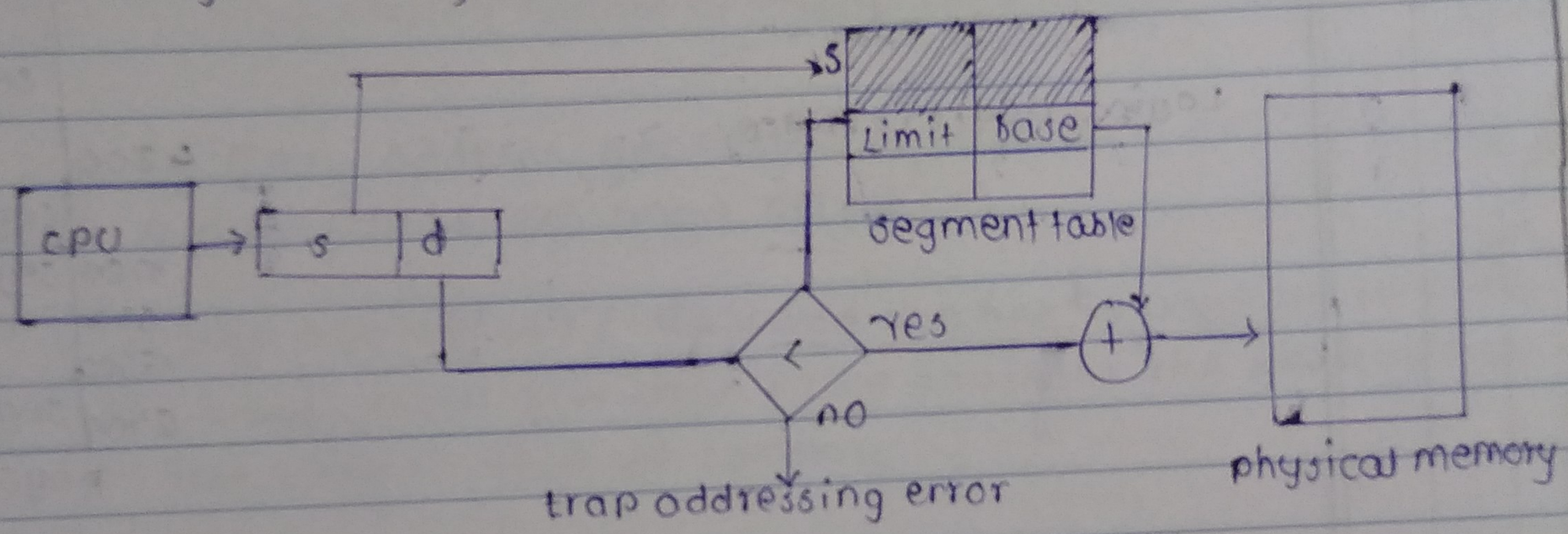
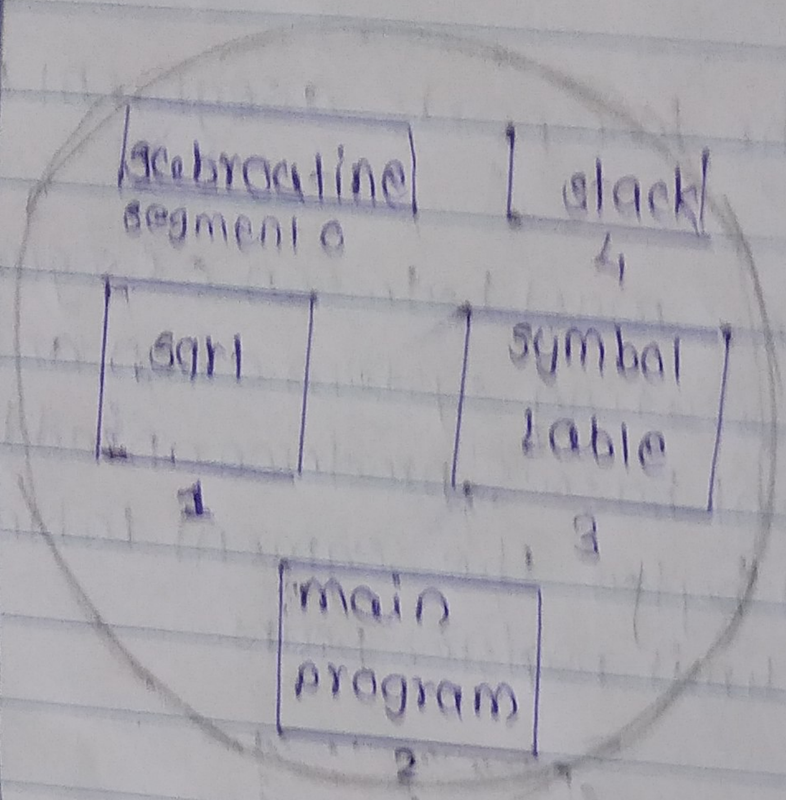


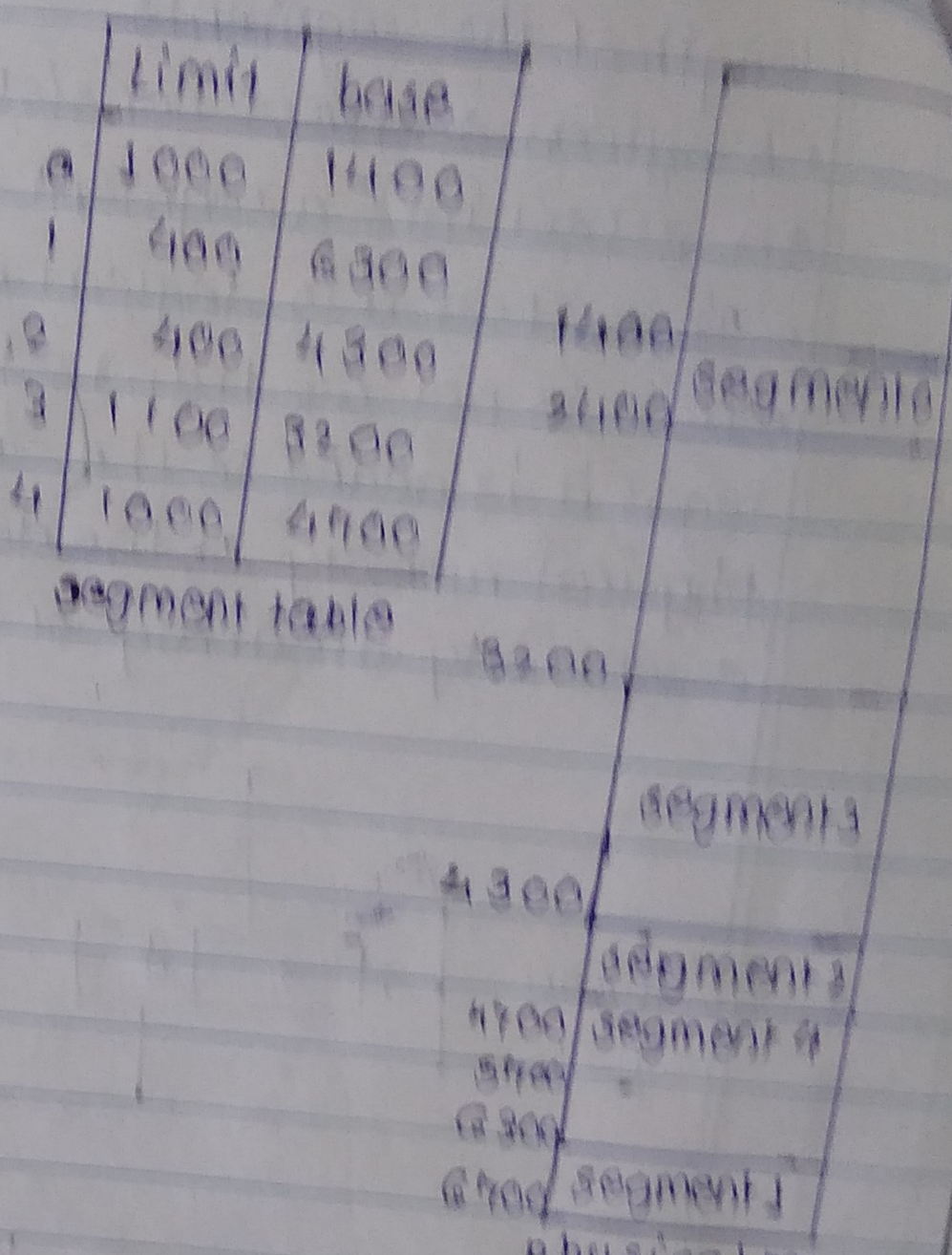

Fig:- segmentation hardware

* consider the situation show the following figure
- we have segment numbered from 0 through
- The segment are stored in physical memory
- The segment table has a seprate entry for each segments in physical memory (base) & the length of that segment (limit)

example:-
segment 2 is 400 bytes long & begins at location 4300 thus a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353 a reference to segment 3 byte 852 is mapped of 3200 + 852 = 4052 a reference to byte 1222 of segment 0 would result in a trap to the operating system as this segment is only 1000 byte long




Logical address space




* Paging:

paging is the memory management scheme that permits a physical address space of a process to be non-contiguous.

* Basic Method.

physical memory is broken into fixed size blocks called frames logical memory is also broken into block of same size called pages. when a process is to be executed its pages are loaded into any available memory frames from the backing store. the backing store is divide into fixed size blocks that are of the same size as the memory frames.

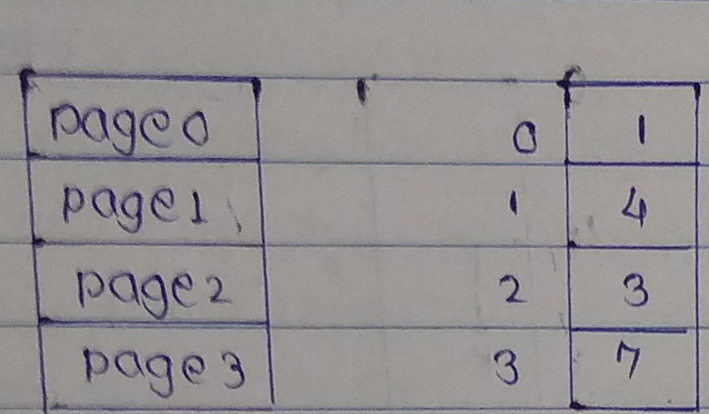



Fig:- paging model of logical & physical memory.

Every address generated by the CPU is divided into two parts 1) page number(p)2) page offset (d)

1) page-number(p) :- The page number is used as an index into page table. The page table contains the base address of each page in physical memory

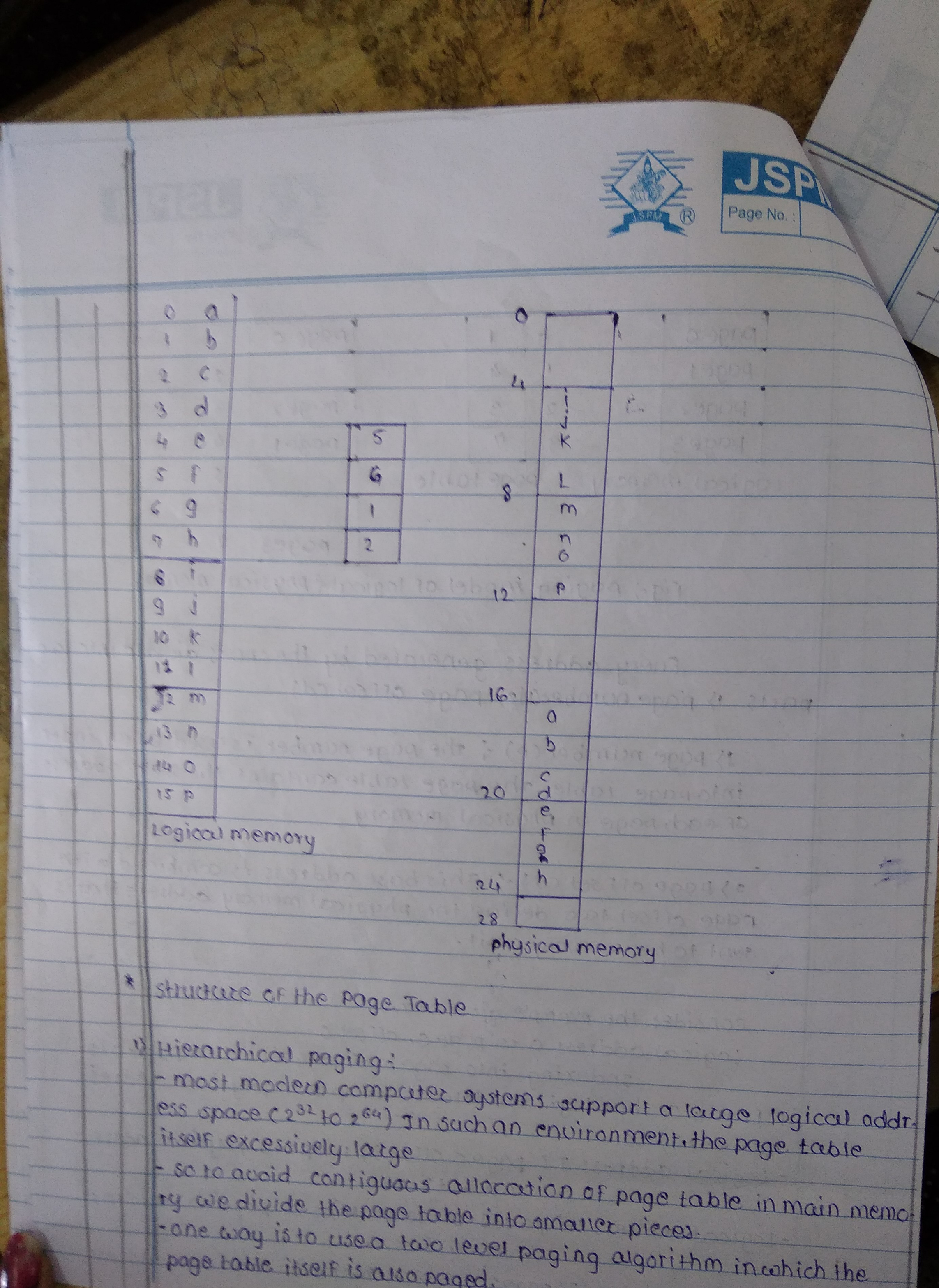
2) page offset (d) :- This base address is combined with page offset two define the physical memory address that is sent to be memory unit.

consider the example given below.

Logical address 0 is page 0, offset 0

Indexing into page table we find that page 0 is in frame 5 thus logical address 0 map 5 to physical address 20 = (5×4) to

- Logical address 3 ( page 0 offset 3) maps to physical address (5×4)+3=23
- Logical address 4 ( page 1 offset 0) maps to physical address (6×4)+0=24
- Logical address 13 (page 3. offset 1) maps to physical address (2×4)+1=9




* structure of the page Table

1) Hierarchical paging:
- most modern computer systems support a large logical address space ($2^{32}$ to $2^{64}$) In such an environment, the page table itself excessively large
- so to avoid contiguous allocation of page table in main memory we divide the page table into smaller pieces.
- one way is to use a two level paging algorithm in which the page table itself is also paged.

2) Invented page tables-
- Usually each process has an associated page table. the page table has one entry for each page that the process is using
- This table representation is a natural one since process refrance pages through the pages vertical address.
- the operating system must be translate the refference into physical memory address
- starting the address space identifier ensures that logical page for a pecticalar process is mapped to the corresponding physical page frame

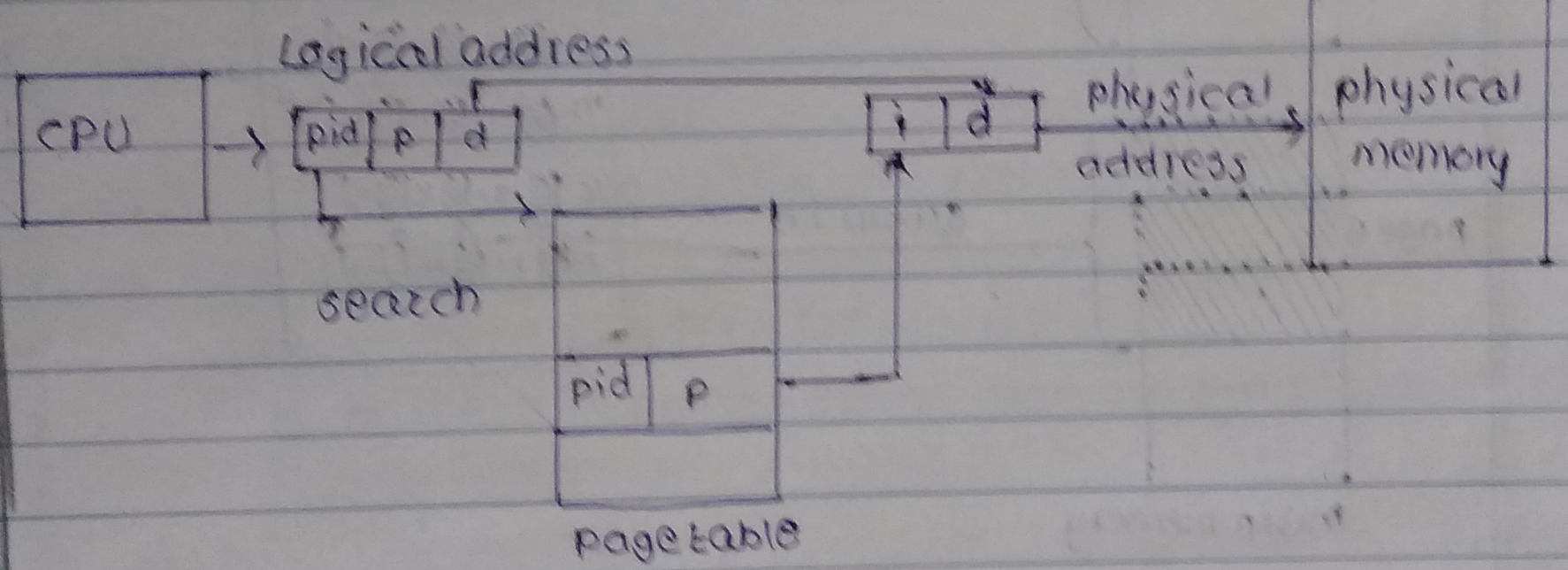



fig - Invented page tables.

Q.4 explain demand pagging with page facult with details.

An executable might be loaded from disk into physical memory at program execution time
- But user selects option available in lists this result in the loading the executable code for all options regardless of option is ultimately selected by the user or not.
- An alternative stratigy is to initially load page only as they are needed this technique is known as demand pagging & is

commonly used in vertual memory system.
- With demand paged vertual memory pages are only loaded when they are demand during program execution pages that are never accessed are thus never loaded in physical memory
- A demand paging system is similer to a pagging system with swapping where process reside in secondary memory.

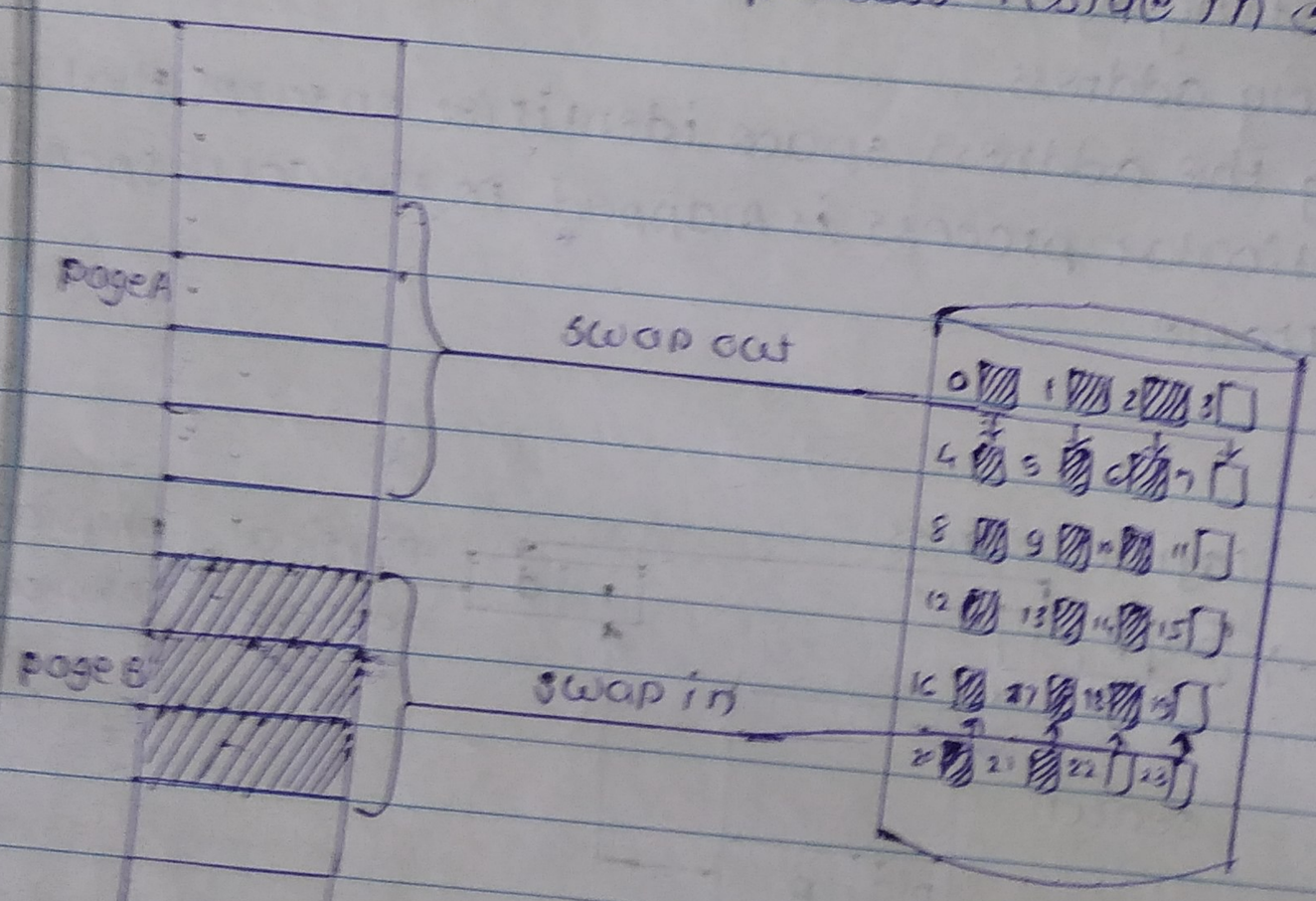



fig:- transfer of a paged memory to contiguous disk space

* Page Replacement:

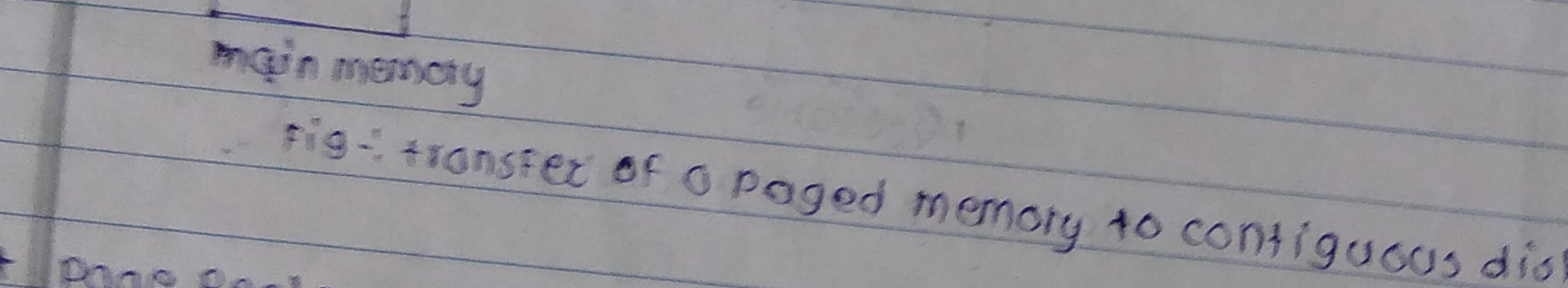



fig:- page Replacement

If we increase our degree of multiprogramming we are over allocating memory itself as follows. while a user process is executing a page fault occurs. The os determines where the desired page is residing on the disk but them finds that there are no free frames. on the free format list. all memory is in use.
- The operating system has several option at this point is could terminate the user process or uses the demand page but it is not the best choice
- The os could insted swap out process freeing all its frames & reducing the level of multiprogramming. but page Replacement is most common solutions.

* Basic Page Replacement.
The page fault service routine to include page replacement
1) Find the location of the desired page on the disk
2) Find a free frame.
   a) IF there is a free frame, use it.
   b) IF there is no free frame use a page replacement algorithm to select victime frame
3) Read the desired page into the newly freed frame chang the page & frame tables
4) Restart the user process.
- page Replacement is basic to demand page it compeates the sepration between logical & physical memory
- If we have multiple process in memory we must decide how many frame to allocate to each process.
- For implementing demand paging we must develop a frame allocation algorithm & a page replacement algorithm

2) FIFO page Replacement:
- The simplest page replacement algorithm is first in first out (FIFO) algo

rithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- when a page must be replaced the oldest page is choosen.
- we can create a FIFO queue to hold all page in memory. we replace the page at the head of the queue.
- when a page is brought into memory we insert it at the queue

3) Optimal page Replacement:
- The optimal page replacement algorithm has the lowest page fault rate of all algorithm & will never suffer from belay's anomaly
- such as algorithm does exist & has been called OPT or MIN It is simply this.
- Replace the page that will not be used for the longest period of time
- use of this page replacement algorithm guarantees the lowest possible page fault rate a fixed number of frame

4) LRU Page Replacement:
- IF the optimal algorithm is not feasiable an approximation of the optimal algorithm is possiable
- This approch is the least-recently used (LRU) algorithm.
- The LRU policy is often used as a page replaced algorithm & is considered to be guid.
- The LRU algorithm is somewhat difficult to implement the problem is determine an order for the frames defined by the time of last use Two of implementation are feasiable

1) counters:
In the simplest case, we associated with each page table entry a time of use field & add to the CPU logical or counter. The clock is incremented for every memory refferences.
- In this way, we always have the time of the last reference each page

-we replace the page with the smallest time.

2) Stack :-

Another approch to implementing LRU replacement is to keep a stack of page numbers.

- Removing a page & putting it on the top of the stack then requires changing six pointers at worst.
- LRU replacement does not suffer from belady's anomaly.

6) LRU - Approximation page Replacement.

Many system provides a refference bit to help to page replacement & set by the hlw & those bits are associated with each page Initially, all bits are cleared by o.s As a user process executes the bit associated with each page.

* page Fault :-

processes resides & secondary memory when we want to execcutes a process will swap it into memory Rather than swapping entire process into memory. we use a lazy swapper never swap a page into memory unless that page will be needed. thus it avoid reading into memory pages that will not be used any way decreasing the swap time & the amount of physical memory needed. To distingwish between page in memory. & those on disk valid & invalid bit scheme can be used when this bit is set to valid. the associated page in both legal in memory.

This trap is result of the opertating system failures to bring the desired page into memory. the procedure for handling this page Fault is follows.

1) we check an internal table for this process to determine whether the refference was a valid or invalid memory access

2) If the refference was invalid we terminate process if it was

valid but are have not yet brought in that page we now page
a) we find a free frame
b) we schedule a disk operation to read the desired page into the newly allocated frame.
c) when the disk read is complete we modify the internal table kept with the process & the page table to indicate that the page is now in memory.
d) we restart the instruction that was interrupts by the trap. the process can now access page as through it has always in memory

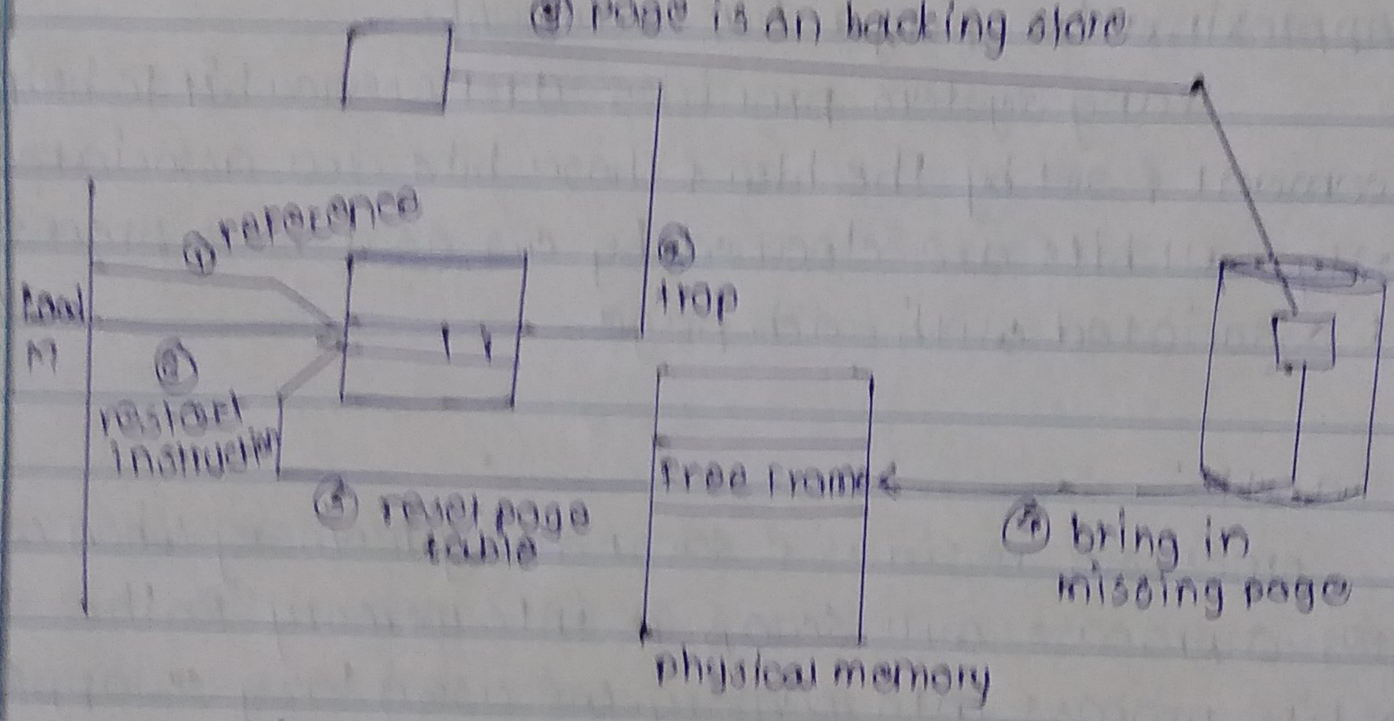



fig - step in handling a page fault

The hardware to support demand paging is the how same as the hardware for paging & swapping

1) page table : this table has the ability to mark an entry invalid through invalid. invalid bit or special value of protection bit

2) secondary memory : this memory holds those page that are not present in main memory. the secondary memory is usually a high speed disk. It is known as the swap device & the section of disk used for this purpose is known as swap space

Q 5 write short note

1) Deadlock Recovery

There are 2 options for breaking a deadlock

1) Abort one or more processes to break the circular wait.
2) To preempt some resources from one or more of the deadlocked processes.

1) Process Termination:-

one of the following 2 methods is used in which the system reclaims all resources allocated to the terminated processes.

a) Abort all deadlocked process:-

It clearly breaks the deadlock but a great overhead. the processes may have computers for a long term time & result of these partial computations must be declared & probibly recomputed later this may also cause database or file to be an inconsistent state

b) Abort one process at a time until deadlock cycle is eliminated.

It incurs less overheads since after each process is discarded, deadlock detection algorithm is invoked to determine whether any process are still deadlocked

Here we have to decide which processes must be terminated. the answer is termination of process which will incur minimum cost the factor to be considered while taking this dicision are.

1) what the priority of process is
2) How long the process has compated & how much longer the process will compute before completing its designated task.

a) How many & what type of resources the process has used
b) How many processes will needed to be terminating
c) whether the process is interactive or batch.

2) Resource Preemption:
The idea is the preemption some resources processes & given these resources to other processes until deadlock cycle is broken
A 3 issues need to be address if preemption is required to dead with deadlock

3) Selecting a victim:
which process & resource are to be preempted are must determine order of preemption to minimize cost. cost factor include parameters such as number of resources a deadlock process is holding & the amount of time a deadlocked process has consumed during its execution.

4) Rollback:
are must rollback the process to some safe state & restart it from that state are can either rollback a process totally or only as far as necessary to break the deadlock.

5) Starvation:
It many happen that the same process is always selected is a victim. As a result this process never completes its designated task. i.e starve are we must ensure that a process can be picked up as a victim only finite number of times. The most common solution is to include number of times the most rollback in the cost factor.

2) Resource Allocation Graph:-
A directed edges from resource type I→P called as assignment edges signifies that an instance of resource type Rj has been allocated to process Pi. A directed edge Pi→Rj is called request edge. In addition we use new type of edges called claim edge. Pi→Rj which indicates that process Pi may request resource Rj at some time in future this edge resemble a request edge in direction but is represented by dashed line.

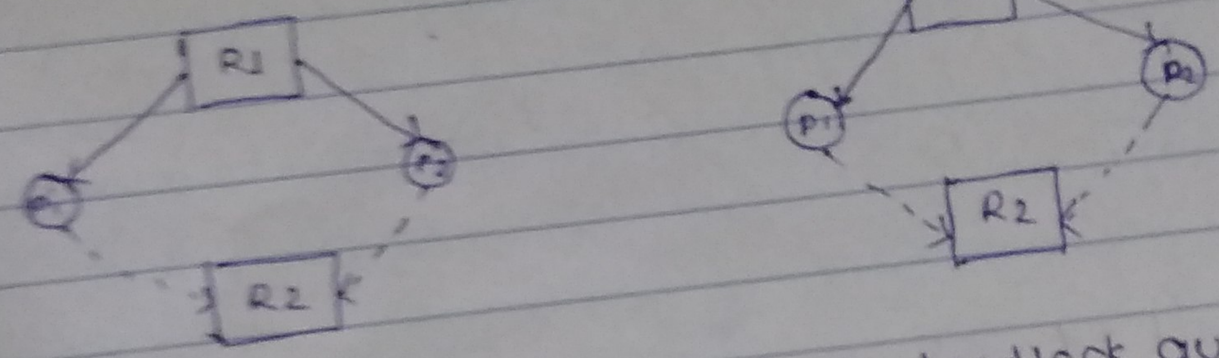



Fig - resource allocation graph deadlock avoidance

4) Hold & wait:- A proposed must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes

5) No preemption:- Resource can not be preempted i.e resource can be released only voluntarily by the process holding it after that process has completed its task

3) circular wait:- A set [P0, P1, ... Pn] of waiting process must be exist that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2. P0.

We can deal with deadlock problem in one of the following 3 ways.
1) we can use a protocols to prevent or avoid deadlock ensuring that the system will never enter a deadlock state
2) we can allow the system to enter a deadlock state & recover
3) we can ignore the problem altogether & pretend that deadlocks never

occur in the system.

fragmentation.