Thread

1. Why is a thread called Light Weight process? Explain the kernel-level thread architecture.

   Ans:

   Threads are sometimes called lightweight processes because they have their own stack but can access shared data. Because threads share the same address space as the process and other threads within the process, the operational cost of communication between the threads is low, which is an advantage.

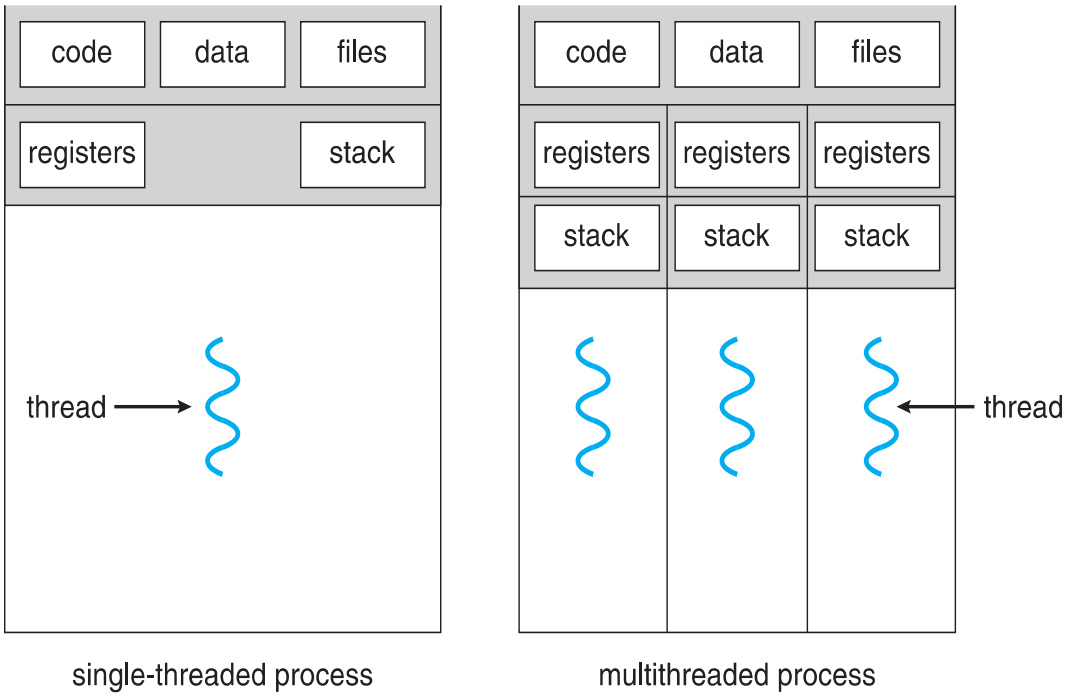


Figure: Kernel level thread architecture

Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel. The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads.

2. Difference between User level thread and Kernel level thread?

Ans:

User level thread	Kernel level thread
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
User level threads are designed as dependent threads.	Kernel level threads are designed as independent threads.
Example : Java thread, POSIX threads.	Example : Window Solaris.

3. When should we create thread instead of duplicating a process?

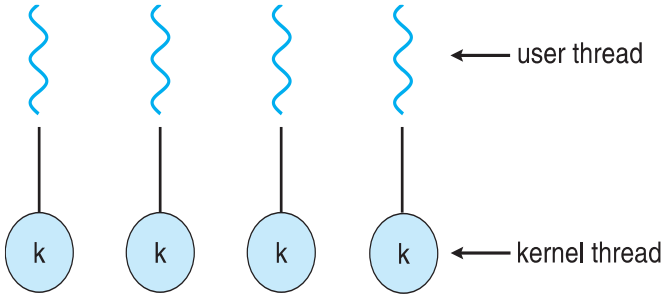
Ans:

- downloading a video while playing it at the same time
- Formatting a MS-Word file while typing at the same time

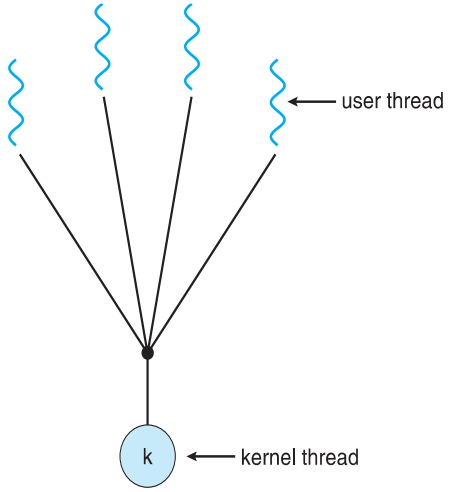
Process	Thread
Process means a program is in execution.	Thread means a segment of a process.
The process is not Lightweight.	Threads are Lightweight.
The process takes more time to terminate.	The thread takes less time to terminate.
It takes more time for creation.	It takes less time for creation.
Communication between processes needs more time compared to thread.	Communication between threads requires less time compared to processes.
It takes more time for context switching.	It takes less time for context switching.
Process consume more resources.	Thread consume fewer resources.

4. Multithreading Models

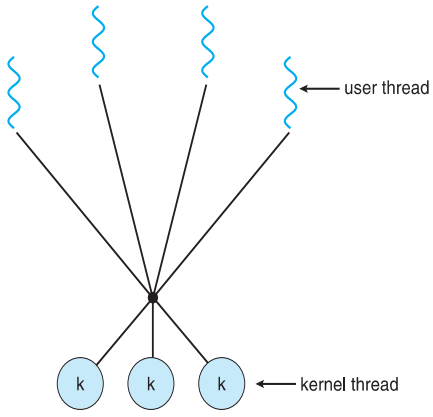
One to One:



Many to One:



Many to Many:



Deadlock

1. What are the probable options for **recovering** from a deadlock?

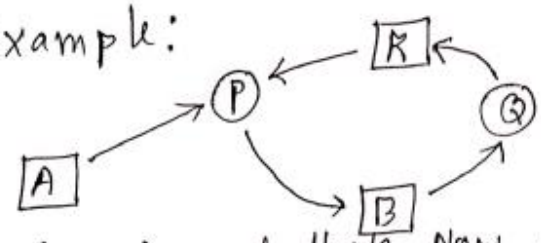
Ans:

(b)

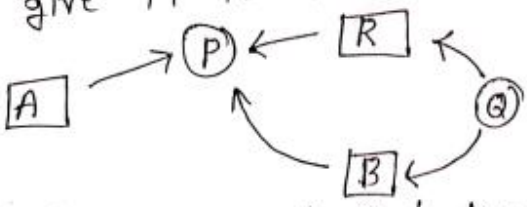
Following are the probable options for recovering from a deadlock:

i) Resourse Pre-emption: we can take back a resource from any process and give that resource to any other process. This is called resource pre-emption. we can recover from a deadlock by this.

For example:

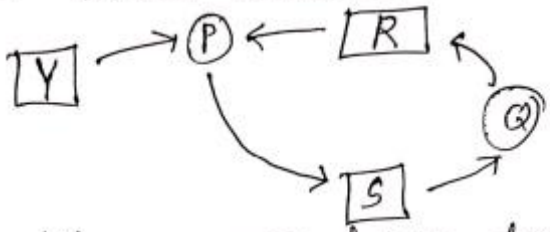


This is in deadlock. Now we can recover from the deadlock by taking back B from Q and give it to P.

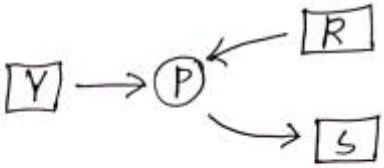


Deadlock is recovered. But the problem is, the Q's work will be lost. when Q will get the resource again, it should start its work from the very begining.

ii) Process killing: we can kill a process to recover from deadlock.



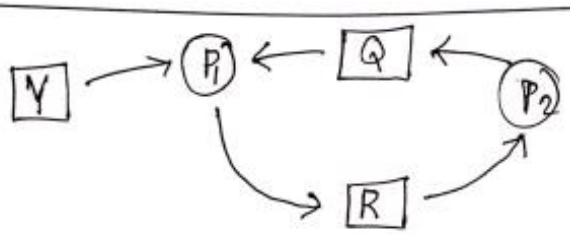
Now we will recover from deadlock by killing process Q.



But the same problem, loss of work, happen here. There is also other problem like taking decision about which process to kill. we can kill processes having -

a) low priority
b) less connection
c) less time running
d) killing common node.

iii) Rollback: we can rollback one step or more than one step to recover from deadlock. In case of rollback, all work will not be lost. only the works of those steps rolled back will be lost. This is an advantage compared to other ways.



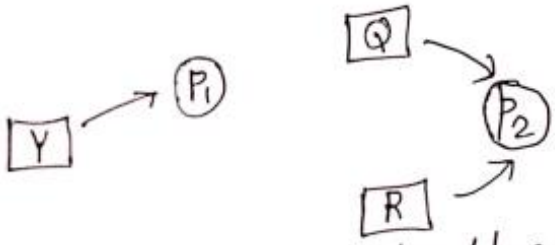
P_1 : { Y

~~Q~~

Req: ~~R~~

}

2 steps rolled back.



So, now there is no deadlock.

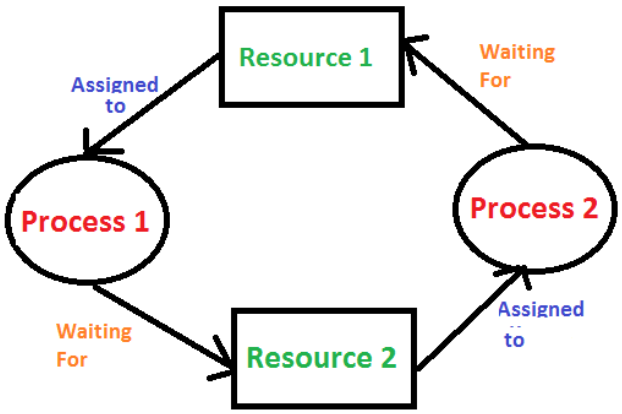
Thus we can recover deadlocks.

2. Explain the causes where a set of processes can be trapped into a deadlock situation. How can a deadlock be **prevented**?

Ans:

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock can arise if the following **four conditions** hold simultaneously (Necessary Conditions)

Mutual Exclusion: Two or more resources are non-shareable (Only one process can use at a time)

Hold and Wait: A process is holding at least one resource and waiting for resources.

No Preemption: A resource cannot be taken from a process unless the process releases the resource.

Circular Wait: A set of processes are waiting for each other in circular form.

Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four conditions.

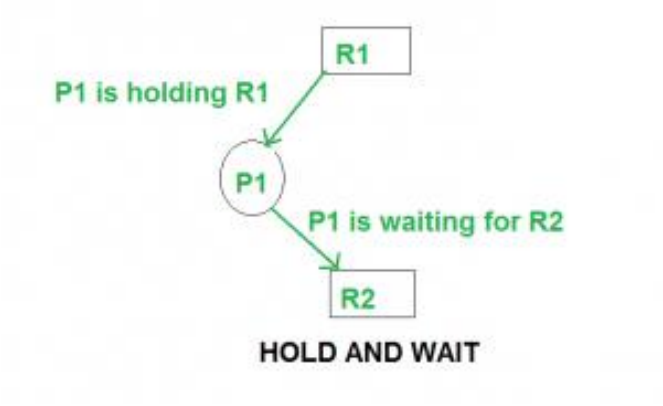
Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Eliminate Hold and wait

1. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.

2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate No Preemption

Preempt resources from the process when resources are required by other high-priority processes.

Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.
For Example, if the P1 process is allocated R5 resources, now next time if P1 asks for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

3. What are the ways an Operating System has to deal with deadlocks in the system?

Ans:

Methods for Handling Deadlocks

- Strategies for dealing with deadlocks:
  - 1. Just ignore the problem. (Ostrich Algorithm)
  - 2. Detection and recovery. Let deadlocks occur, detect them, take action.
  - 3. Dynamic avoidance by careful resource allocation.
  - 4. Prevention, by structurally negating one of the four required conditions.

4. What is meant by a safe state?

Ans:

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes, then the state of the system is called unsafe.

- If a system is in safe state ⇒ no deadlocks
- If a system is in unsafe state ⇒ possibility of deadlock
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

5. Show the Deadlock Detection Mechanism. (Algorithm)

Ans:

Deadlock Detection

- Algorithm for detecting deadlock:
- 1. For each node, N in the graph, perform the following five steps with N as the starting node.
- 2. Initialize L to the empty list, designate all arcs as unmarked.
- 3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.
- 4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6. 5.
- 5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
- 6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.

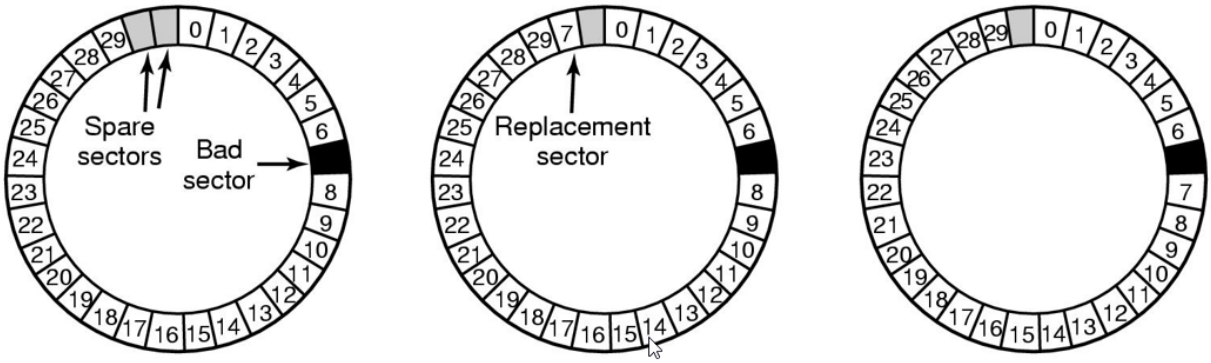
Mass Storage Management

1. How can an OS handle a bad sector in a disk drive?

Ans:

Handling Errors

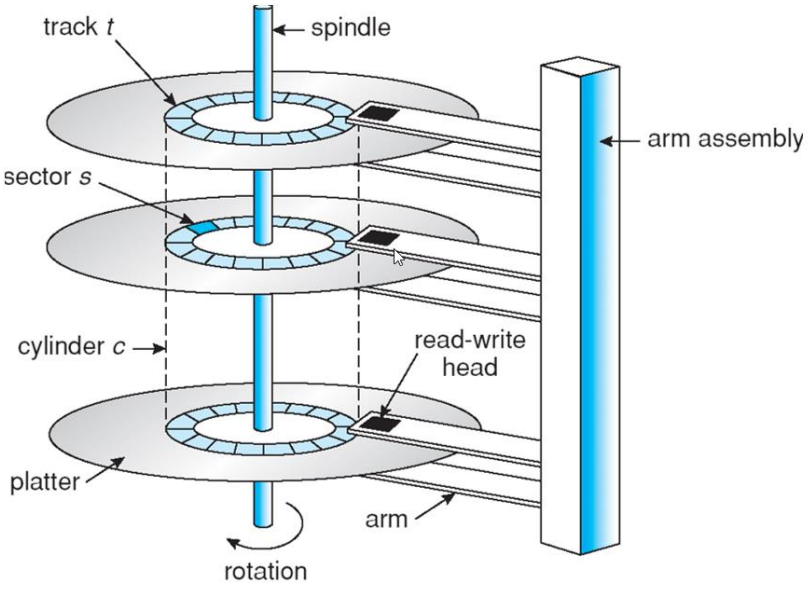
- A disk track with a bad sector
- Solutions:
  - Substitute a spare for the bad sector (sector sparing)
  - Shift all sectors to bypass bad one (sector forwarding)



2. What do you understand by cylinder skew in disk management system?

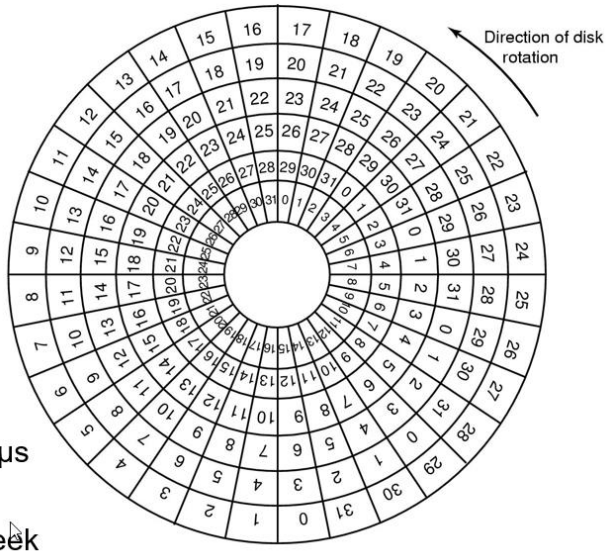
Ans:

What Does The Disk Look Like?



Cylinder Skew

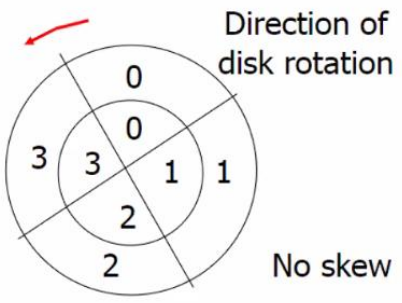
- Why cylinder skew?
- How much skew?
- Example, if
  - 10000 rpm
    - Drive rotates in 6 ms
  - Track has 300 sectors
    - New sector every 20 µs
  - If track seek time 800 µs
    - ⇒ 40 sectors pass on seek

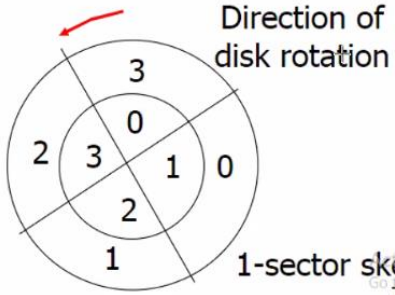


Cylinder skew: 40 sectors

Cylinder Skew

- The position of sector 0 on each track is offset from the previous track. This offset is called *cylinder skew*.
- Allow the disk to read multiple tracks in one continuous operation without losing data





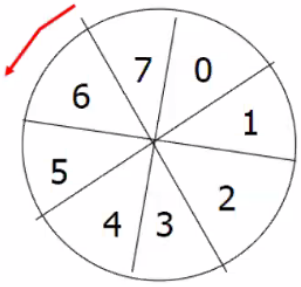
7

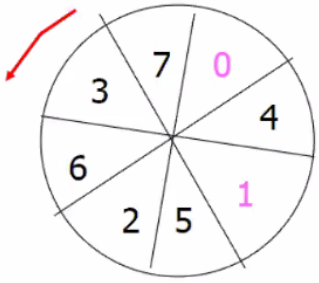
3. What do you understand by sector interleaving?

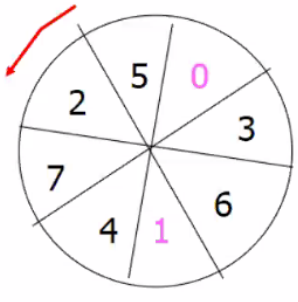
Ans:

Sector Interleaving

- Consider a controller with one sector buffer. A request of reading two consecutive sectors. When the controller is busy with transferring one sector of data to memory, the next sector will fly by the head.
- Solution: sector interleaving







Process Scheduling

1. What is the function of the dispatcher module of an Operating System?

Ans:

Dispatcher

- The **dispatcher** is the module that gives control of the CPU to the process selected by the scheduler. This function involves:
  - Switching context.
  - Switching to user mode.
  - Jumping to the proper location in the newly loaded program.
- The dispatcher needs to be as fast as possible, as it is run on every context switch. The time consumed by the dispatcher is known as **dispatch latency.**

2. Explain the following terms with necessary figure(s) with respect to a multiprocessor-based scheduling system.

i) Asymmetric multiprocessor scheduling

ii) Symmetric multiprocessor scheduling

iii) Pull and Push migration during load balancing

Ans:

(b) (i) Asymetric Multiprocessor Scheduling :

In this system, there is one dedicated CPU for scheduling other processes, it is called the master server. But there is a risk, if the master server fails somehow, all other processors become idle.

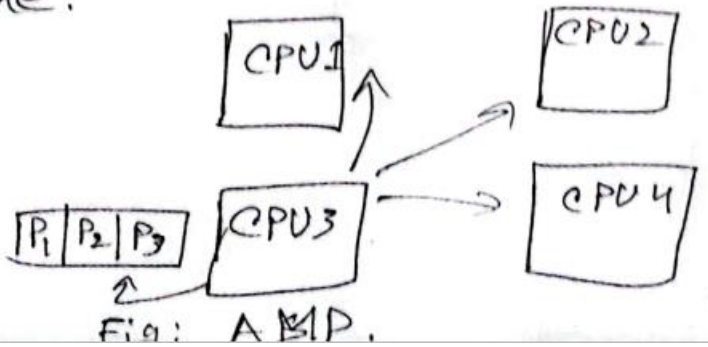


Fig: AMP.

(i.i) Symetric Multiprocessor Scheduling:

In this system, there is no dedicated CPU for scheduling, but each processor has their own scheduler and a common process queue.

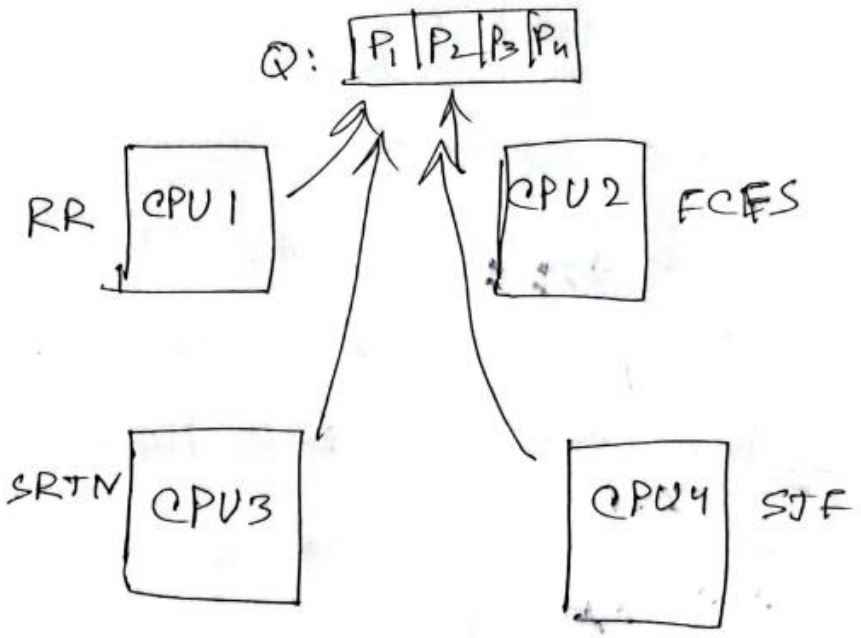
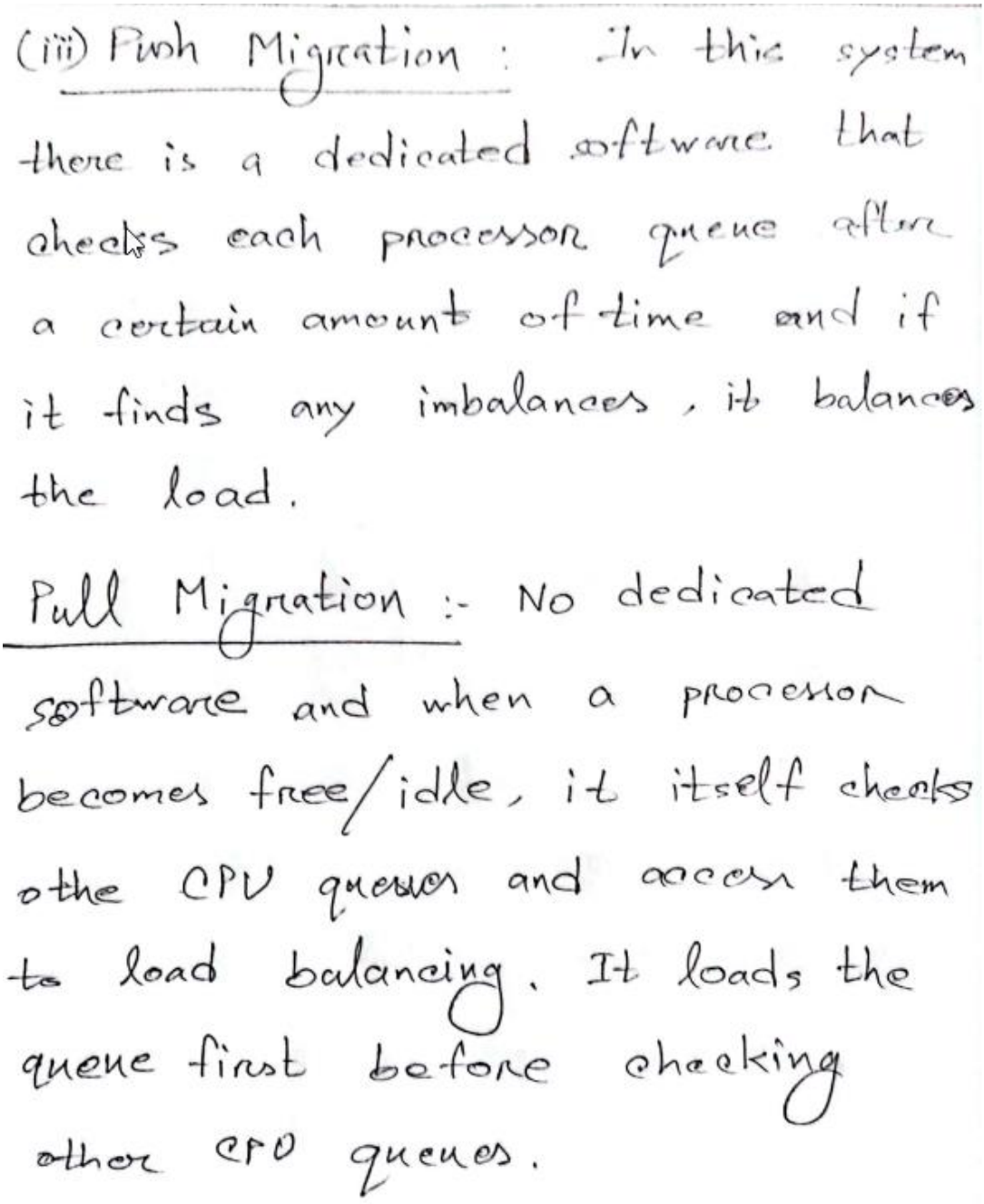


Fig: SMP



Balancing can be achieved through either ***push migration*** or ***pull migration:***

- ***Push migration*** involves a separate process that runs periodically, (e.g., every 200 milliseconds), and moves processes from heavily loaded processors onto less loaded ones.
- ***Pull migration*** involves idle processors taking processes from the ready queues of other processors.

SMP systems attempt to keep processes on the same processor, via ***processor affinity.***

Soft affinity occurs when the system attempts to keep processes on the same processor but makes no guarantees.

Linux and some other OSes support ***hard affinity,*** in which a process specifies that it is not to be moved between processors.

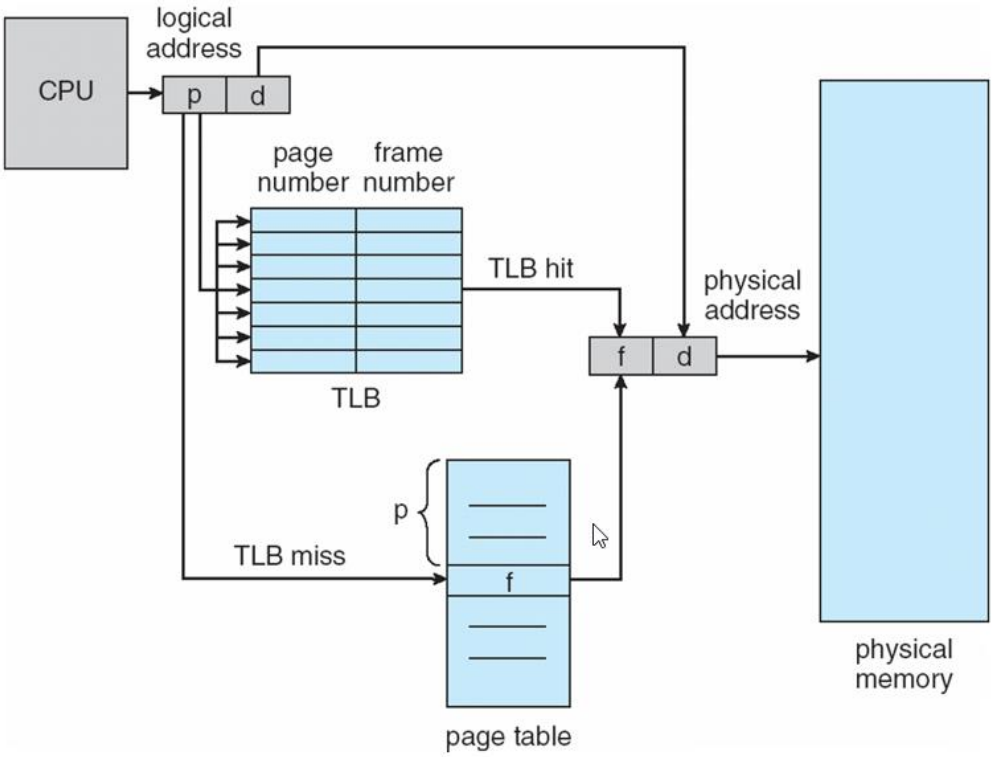
Memory Management

1. Draw the diagram for an address translation process in any virtual memory system with a Translation Look-aside Buffer.
2. Discuss the fundamental concept of using Translation lookaside buffer.

Ans:

Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.

Paging Hardware With TLB

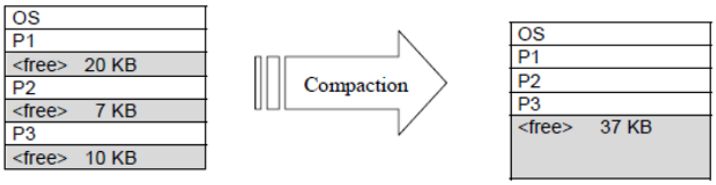


3. Memory compaction

Ans:

Variable Partitioning

- External Fragmentation in variable partitioning
  - Total memory space exists to satisfy a request, but it is not contiguous.
- 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



4. What are the differences between global and local page replacement techniques?

Ans:

Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput so more common
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

Explain the Type-1 and Type-2 hypervisors in a virtual machine Operating System.

TYPE 1 HYPERVISOR
VERSUS
TYPE 2 HYPERVISOR

TYPE 1 HYPERVISOR	TYPE 2 HYPERVISOR
A hypervisor that runs directly on the host's hardware to control the hardware and to manage guest operating systems	A hypervisor that runs on a conventional operating system just as other computer programs do
Called a native or Bare Metal Hypervisor	Called a Host OS Hypervisor
Runs directly on the host's hardware	Runs on an operating system similar to other computer programs
Examples: AntsleOs, Xen, XCP-ng, Microsoft Hyper V, VMware ESX/ESXi, Oracle VM Server for x86	Examples: VMware Workstation, VMware Player, VirtualBox, Parallel Desktop for Mac

Visit www.PEDIAA.com