Final: 2020-21

(1) (a). What are the three main purpose of an Operating System?

⇒ (1) Resource Management:
- OS manages all hardware resources like cpu, memory, I/O...
- It decides which program gets what resource and for how long.

(2) Process Management:
- OS controls and schedules programs/processes.
- It handles running, pausing, resuming and ending process.

(3) File & Derice Management:
- OS manages files, folders and storage.
- It also manages I/O devices like keyboard, mouse, printer etc.

(b) compare Monolithic, Layred, Microkernel and Modular operating systems Structure. How does each structure organize components, and what are the main advantages and disadvantages?

Structure	Organization	Advantage	Disadvantage	Example
Monolithic	All services in one big Kernel	Fast, Simple	Hard to maintain, low security	Unix, Linux
Layered	Os divided into layers	Easy to debug, clean design	slower, hard to design layers	THE OS, windows NT
Microkernel	Only core in kernel, rest in user space	Highly secure, stable	slower due to message passing	Minix, QNX, Mach.
Modular	Core kernel + loadable modules	flexible, good performance.	Modules can crash kernel	Linux, Solaris, Free BSD.

(c) Explain the key difference between Batch operating system, multiprogramming and multitasking. Specifically, describe how each system manages processes, cpu utilization, and user interaction. provide example of scenarios or use case where each type of operating system would be most effective?

⇒ • **Batch operating system:**

Feature	Batch OS	Multiprogramming OS	Multitasking OS
• Process management	Jobs in batches, no switching	Multiple jobs in memory, CPU switches on I/O wait	Time slicing between tasks, rapid switching
• CPU utilization	Medium	High	High + responsive
• User interaction	None	Limitted	full, interactive
• Best use case	payroll, bank processing	servers, scientific jobs	pcs, smartphones.
• Example	IBM bank System	Unix mainframe	Window, Linux, macOS.

(2) (a) What is a process in an operating system? And how does it differ from a program? Explain the different states a process can be in during its lifecycle.

⇒ A process is a program that is currently under execution. An active program can be called process.

Difference between process and program:

Program	process
- passive: A file stored on disk	- Active: program that is currently executing.
- Takes no resources	- Needs cpu, memory, registers
- lifeless	- has a lifecycle.
- One program can create many process.	- Each process is running instance of a program.

process states in its lifecycle:

- **New:**
  - process is just created
  - Os loads process information and allocates memory.
- **Ready:**
  - process is prepared to run.
  - Waiting for cpu to assigned.
  - Stored in the ready queue.
- **Running:**
  - cpu is currently executing the process.
  - All instructions are processed here
- **Waiting/ Blocked:**
  - Process can not continue until an I/o event happens
- **Terminated / Exit:**
  - Process finishes its execution OR is Killed.

(b). Explain the concept of process synchronization. Why is it necessary, and what problems can arise if processes are not properly synchronized?

⇒ Process synchronization means co-ordinating the execution of multiple processes so that they do not interfere with each other when accessing shared resources.

Why process synchronization Necessary:

- prevent data inconsistency: When multiple processes modify the same data at the same time, the result becomes wrong.
- Avoid race conditions: When outcome depends on which process runs first → unsafe and unpredicatable.
- Ensure correct execution order:
  Some processes must run in a specific sequence.
- Maintain data integrity:
  Shared variables, files, databuses must remain correct.

What problems Occur Without proper Synchronization:

① Race condition:
- multiple processes access and update shared data simultaneously.
- Final result becomes unpredictable.

② Data Inconsistency:
- ~~Incorrupted or~~
- Incorrect or corrupted data due to over lapping operations.

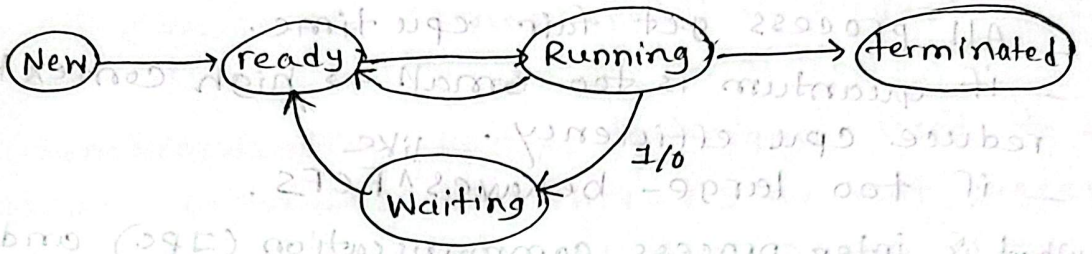
③ Deadlock:
- two processes wait forever for each other's resources.

④ starvation:
- A process never gets cpu or a resource because other keep taking it first.

⑤ Race in I/O Devices:
- Incorrect output or device conflict if two processes try to use the same device at once.

(c) Outline the lifecycle of process, describing each state and the events that trigger transitions between these states.

⇒



(3) (a) Describe process scheduling and the role of the scheduler in managing processes. How do different scheduling algorithm (FCFS, Round Robin) affect process performance and cpu utilization?

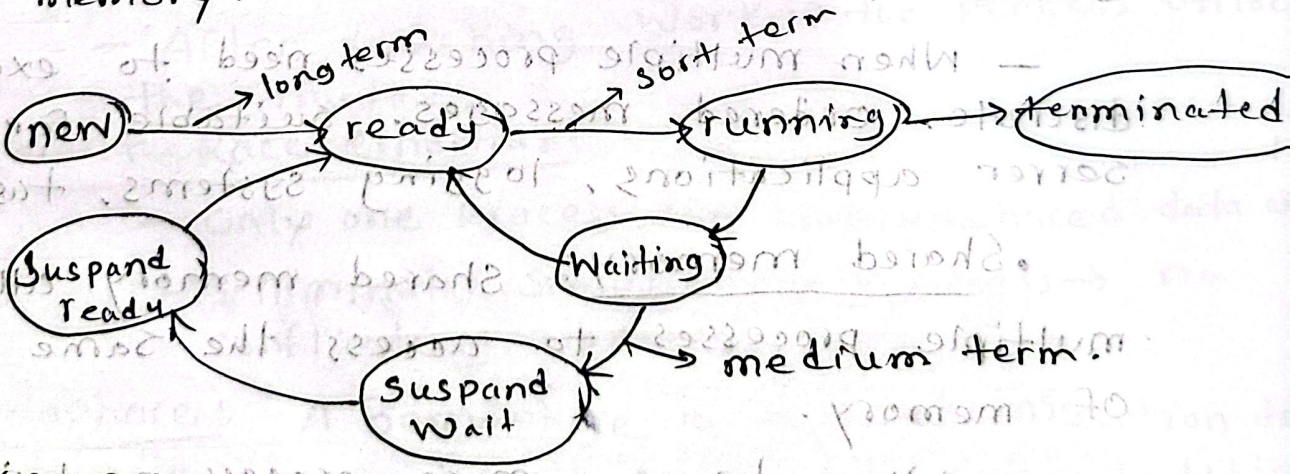
⇒ Process scheduling is the method used by OS to select which process will run next on cpu.

Role of scheduler:

→ Long term scheduler: Describes which process to admit into the ready queue.

→ Short term scheduler: selects which ready proces gets the cpu next.

→ Medium term scheduler: perform swapping to manage memory.



FCFS peformance:

- simple, but may cause long waiting time.
- Suffers from convoy effect
- CPU utilization may decrease if long jobs block the Queue.

Round Robin performance;

- Better response time, suitable for time sharing systems.
- All process get fair cpu time.
- if quantum is too small → high context switching reduce cpu efficiency.
- if too large - behaves like FCFS.

(b). What is inter process communication (IPC) and what are some common IPC mechanisms (pipes, message queues, shared memory)? In which scenarios would each mechanism be most appropriate?

⇒ Inter process communication (IPC) is a mechanism that allows processors to ~~allow~~ exchange data and co-ordinate their activities.

- pipes: A pipe is a Undirectional communication channel that allows data to follow from one process to another.
  - When two related processes need to pass a continious stream of data.
- Message Queue: A message queue allows processes to send and receive structured message stored in a Queue.
  - When multiple processes need to exchange discrete, ordered messages, suitable for client-server applications, logging systems, task scheduling.
- Shared memory: Shared memory allows multiple processes to access the same region of memory.
  - When two or more process need to exchange large amounts of data very quickly.

(c) What is process synchronization? and why it is important in an OS? Describe common synchronizat mechanisms (mutexes, semaphores) and explain how they prevent issues like race conditions.

⇒ process synchronization is the technique used to ensure that multiple processes or threads do not access shared or critical sections at the same time, which could cause incorrect results.

Why synchronization important?

- Race conditions,
- Data inconsistency
- Deadlocks and starvation
- Incorrect program behaviour.

① Mutex: A mutex is a locking mechanism that allows only one process or thread to enter the critical section at a time.

Mechanism:

- When a process wants to enter the critical section → it locks the mutex.
- if another process tries to enter → it mus wait until the mutex is unlocked.
- After finishing work → the process unlock the mutex.

Prevent Race condition:

- only one process can modify shared data at tim
- Eliminates simultaneous access → no conflicting updates.

② Semaphore: A semaphore is a synchronization too represented by an integer that is accessed using two atomic operations:

Wait()

Signal()

Binary semaphore: Works like mutex (0,1)

Counting semaphore: Allows multiple process to access limited shared resources.

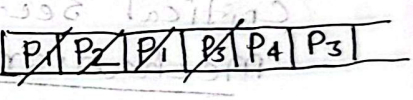
Prevent race condition:

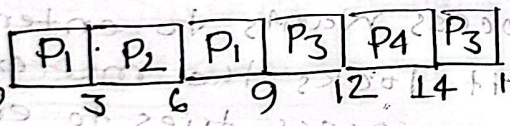
- Wait() decreases the semaphore → blocks if 0.
- Signal() increases the semaphore → allows waiting process to continue.
- ensures only the allowed number of processes enter the critical section.

(4) (a).

process	Arrival	Burst	Completion	turn Around	Waiting
P_1	0	3	9	9	3
P_2	2	4	6	4	1
P_3	4	7	15	11	7
P_4	6	2	14	8	6

time Quantum (3) — Round Robin:





* time Quantum affects

- Small Quantum (very small): behaves like time-Sharin good response time but high context-switch overhea cpu utilization decreases slightly. waiting / turn arou may increase for cpu bound jobs.
- Large Quantum (very large): behaves like FCFS - lower contex switching bat poor response for interactive / short jobs; long jobs hold cpu longer → increase waiting time for others.
- Compared to FCFS: RR gives better response time and fairness but may increase average turn-around vs an optimal schedule, for short jobs.

• Compared of SJF: SJF minimize average turnaround/Waiting time but in non-preemptive OR needs perfect Knowledge; RR gives fairness and responsivenss at cost of larger average turn around than SJF.

⑥ time Quantum = 3. vs time Quantum = 2

$q = 3$ average Waiting = 4.25 (better)

$q = 2$ " " = 5.25

$q = 3$ gives 6 cpu segments (better)

$q = 2$ " 8 " "

So, choose quantum large enough to reduce contex switching but small enough to keep good response. For this job mix $q = 3$ is better.

(b) Difference between user level ~~Kernel~~ thread vs Kernel-level thread

User level thread:

- managed by a user-level library.
- Kernel is unaware of these threads.

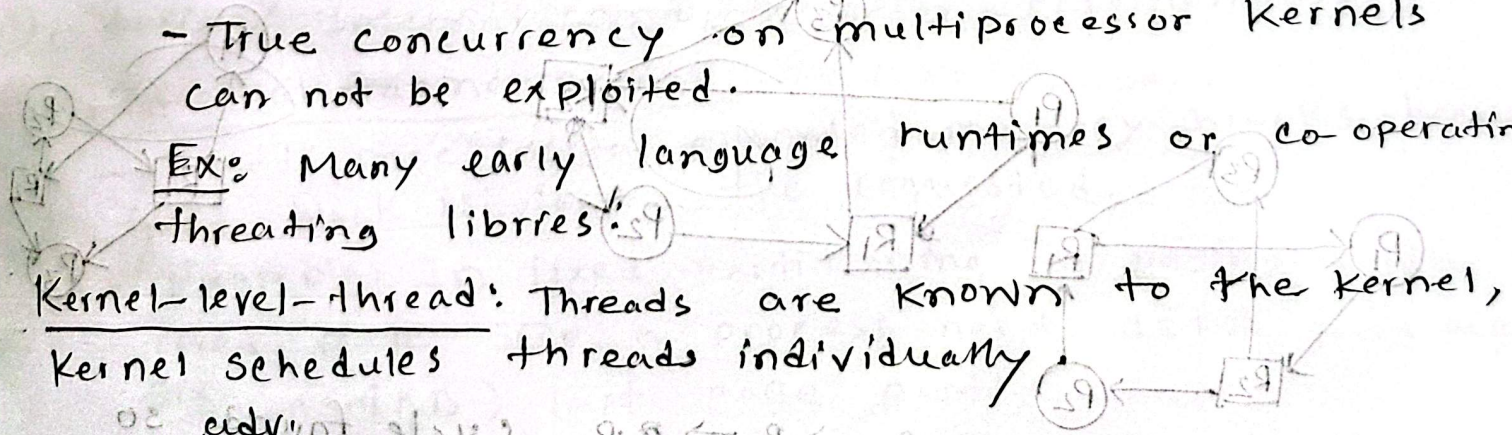
adv:

- very fast thread creation/switching.
- flexible scheduling policies implemented in user library

Disadv:

- If one thread performs blocking system call, all threads in the process block.
- True concurrency on multiprocessor kernels can not be exploited.

Ex: Many early language runtimes or co-operating threading libraries.



Kernel-level-thread: Threads are known to the kernel, Kernel schedules threads individually.

adv:

- if one thread blocks, Kernel can schedule another thread of the same process - no whole process blocking
- true parallelism on multiprocessor machines.

Disadv:

- threads operation are slower.
- More Kernel overhead.

Ex:

POSIX threads implemented as Kernel threads.

(c) Define Short term scheduler and cpu dispatcher.

- Short term scheduler: the os component that selects, from the ready queue; the next process to run on cpu. it turns frequently and decides which process gets the cpu next.
- cpu dispatcher: the mechanism that performs the low level context switch: loads process registers, switches to user mode, switches memory map and starts execution of the selected process. Dispatches gives control of the cpu to the selected process.

5

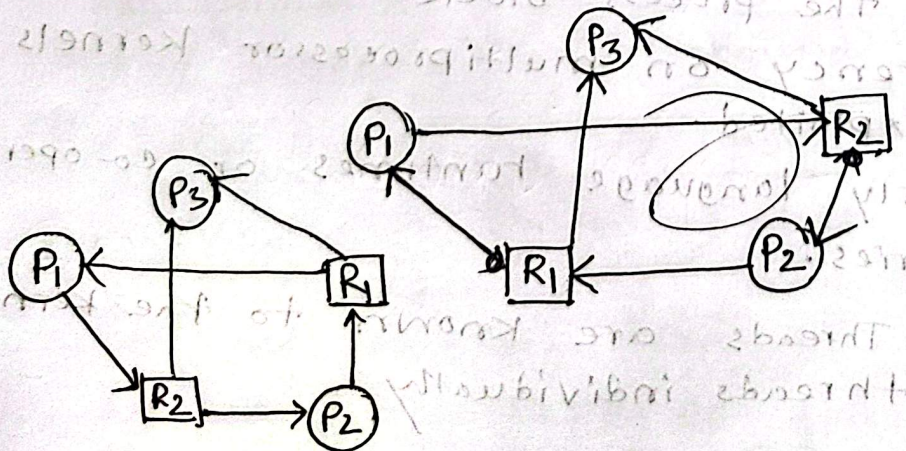
(a)

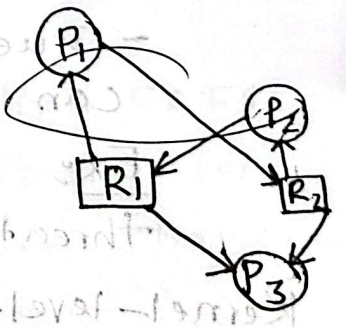
Allocation:

- P_1: $R_1 = 1$, $R_2 = 0$
- P_2: $R_1 = 0$, $R_2 = 1$
- P_3: $R_1 = 1$, $R_2 = 1$

Request:

P_1: $R_1 = 0$, $R_2 = 1$
P_2: $R_1 = 1$, $R_2 = 0$
P_3: $R_1 = 0$, $R_2 = 0$





cycle: $P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$ cycle found so deadlock

(b). difference between deadlock prevention, deadlock avoidance, and deadlock detection. provide examples of techniques for each approach.

- **Deadlock prevention:** Ensure at least one of the necessary conditions for deadlock can't hold.

  technique:

  → Eliminate hold and wait;
  → preemption allowed
  → Impose resouree ordering.

- **Deadlock avoidance:** OS makes dynamic decisions about whether granting a resource will leave system in a safe state.

  Techniques:

  → Banker's Algorithm

- **Deadlock detection and recovery:** allow deadlock to occur, detect them, then recover.

  Technique:

  → periodically run deadlock detection algorithm to find cycles or wait for graph cycles.
  → Recovery by aborting one or more processes or preempting resources.

difference

(c). define Internal and external fragmentation?

- **Internal fragmentation:**

– Wasted space inside allocated memory blocks because fixed block is larger the requested.
– Example: In fixed partitioning or paging with fixed page size, a process needs 18KB and page size is 4KB → last page partly unused.
– Happens within allocated regions.

- External fragmentation:
  - free memory exists but is in small non-contiguous block such that a large request cannot be satisfied despite total free memory being sufficient
  - Example: After many allocations / frees, free spaces are split; a process requiring contiguous block can not be allocated.