Part D

1. What is an operating system, and what are its primary functions?

An operating system (OS) is system software that manages computer hardware and software resources, and provides common services for computer programs. Its primary functions include process management, memory management, file system management, device management, and providing a user interface.

2. Explain the difference between process and thread.

A process is an independent program in execution with its own memory space, while a thread is the smallest unit of a process that can be scheduled and executed. Threads within the same process share the same memory space but execute independently.

3. What is virtual memory, and how does it work?

Virtual memory is a memory management technique that gives an application the impression it has contiguous working memory, while in reality, it may be fragmented and spread across physical memory (RAM) and disk storage. The OS handles this by mapping virtual addresses to physical addresses.

4. Describe the difference between multiprogramming, multitasking, and multiprocessing.

Multiprogramming: Running multiple programs on a single CPU by loading them into memory simultaneously.

Multitasking: Allowing multiple tasks/processes to execute simultaneously by rapidly switching between them.

Multiprocessing: Using multiple CPUs or cores to execute multiple processes simultaneously.

5. What is a file system, and what are its components?

A file system manages how data is stored and retrieved on a storage device. Its components include directories, files, metadata (such as file permissions and timestamps), and the file allocation table or inode table that maps files to the physical blocks on the disk.

6. What is a deadlock, and how can it be prevented?

A deadlock is a situation where two or more processes are unable to proceed because each is waiting for the other to release a resource. Deadlocks can be prevented by avoiding circular wait conditions, using resource allocation strategies like resource ordering, or applying the Banker's algorithm.

7. Explain the difference between a kernel and a shell.

Kernel: The core part of an OS that manages system resources, hardware, and system calls.

Shell: A user interface that allows users to interact with the kernel, either through a command-line interface (CLI) or graphical user interface (GUI).

8. What is CPU scheduling, and why is it important?

CPU scheduling is the process of determining which processes in the ready queue will be allocated CPU time. It is important because it ensures efficient utilization of the CPU, enhances system performance, and provides fairness in resource allocation.

9. How does a system call work?

A system call is a mechanism that allows user-space programs to request services from the kernel. When a system call is made, the CPU switches from user mode to kernel mode, executes the requested service, and then switches back to user mode.

10. What is the purpose of device drivers in an operating system?

Device drivers are specialized programs that allow the OS to communicate with hardware devices. They act as a translator between the hardware and software, converting high-level OS commands into device-specific operations.

11. Explain the role of the page table in virtual memory management.

A page table is a data structure used by the OS to map virtual addresses to physical addresses in memory. It helps the OS efficiently manage memory and enables processes to access their virtual address space without knowing the physical memory layout.

12. What is thrashing, and how can it be avoided?

Thrashing occurs when a system spends more time swapping pages in and out of memory than executing processes, usually due to insufficient physical memory. It can be avoided by increasing the physical memory, optimizing memory usage, or reducing the degree of multiprogramming.

13. Describe the concept of a semaphore and its use in synchronization.

A semaphore is a synchronization tool used to control access to a shared resource by multiple processes in a concurrent system. Semaphores can be binary (0 or 1) or counting, and they help prevent race conditions and ensure mutual exclusion.

14. How does an operating system handle process synchronization?

The OS uses various synchronization mechanisms like semaphores, mutexes, condition variables, and monitors to coordinate the execution of processes and threads, ensuring they do not interfere with each other when accessing shared resources.

15. What is the purpose of an interrupt in operating systems?

An interrupt is a signal sent to the CPU by hardware or software indicating an event that needs immediate attention. The OS responds by pausing the current process, saving its state, and executing an interrupt handler to deal with the event.

16. Explain the concept of a file descriptor.

A file descriptor is an integer that uniquely identifies an open file in a process. It is used by the OS to keep track of all open files and allows processes to read from, write to, or close files.

17. How does a system recover from a system crash?

System recovery involves restoring the system to a stable state after a crash. This can include rolling back incomplete transactions, restoring data from backups, checking and repairing file system integrity, and restarting failed processes.

18. Describe the difference between a monolithic kernel and a microkernel.

Monolithic Kernel: A large kernel that includes many OS services like device drivers, file system management, and network stack.

Microkernel: A smaller kernel that provides minimal services, such as IPC and basic scheduling, with other services running in user space.

19. What is the difference between internal and external fragmentation?

Internal Fragmentation: Wasted space within allocated memory blocks due to fixed block sizes being larger than the requested memory.

External Fragmentation: Wasted space outside of allocated memory blocks due to scattered free memory fragments, which may be too small to use.

20. How does an operating system manage I/O operations?

The OS manages I/O operations using device drivers, buffering, spooling, and interrupt handling. It abstracts hardware details, allowing processes to perform I/O through standard system calls, and manages data transfer between the CPU and I/O devices.

21. Explain the difference between preemptive and non-preemptive scheduling.

Preemptive Scheduling: The OS can interrupt and switch out a process before it finishes executing, allowing another process to use the CPU.

Non-preemptive Scheduling: Once a process starts executing, it runs to completion or until it voluntarily releases the CPU.

22. What is round-robin scheduling, and how does it work?

Round-robin scheduling is a CPU scheduling algorithm where each process is assigned a fixed time quantum. The CPU cycles through the ready queue, giving each process an equal share of CPU time, and if a process doesn't finish within its quantum, it's moved to the back of the queue.

23. Describe the priority scheduling algorithm. How is priority assigned to processes?

Priority scheduling assigns CPU time based on process priorities. Processes with higher priority (lower priority number) are executed first. Priorities can be static (assigned at creation) or dynamic (changing during execution).

24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?

SJN (or SJF, Shortest Job First) schedules the process with the shortest burst time next. It minimizes the average waiting time but can lead to starvation of longer processes. It's often used in batch systems where burst times are known in advance.

25. Explain the concept of multilevel queue scheduling.

Multilevel queue scheduling divides processes into multiple queues based on their priority or type (e.g., system processes, interactive processes). Each queue can have its own scheduling algorithm, and processes are assigned to queues permanently or dynamically.

26. What is a process control block (PCB), and what information does it contain?

A PCB is a data structure used by the OS to store information about a process. It contains process ID, process state, CPU registers, memory management information, I/O status, and scheduling information.

27. Describe the process state diagram and the transitions between different process states.

The process state diagram includes states like New, Ready, Running, Waiting, and Terminated. Transitions occur due to events like process creation, CPU allocation, I/O requests, process termination, and context switching.

28. How does a process communicate with another process in an operating system?

Processes communicate using Inter-Process Communication (IPC) mechanisms like message passing, shared memory, pipes, sockets, and signals. These allow processes to exchange data and synchronize actions.

29. What is process synchronization, and why is it important?

Process synchronization ensures that multiple processes or threads do not interfere with each other while accessing shared resources. It is important to prevent race conditions, ensure data consistency, and maintain system stability.

30. Explain the concept of a zombie process and how it is created.

A zombie process is a process that has completed execution but still has an entry in the process table because its parent has not yet read its exit status. It is created when a process terminates but the parent process does not call wait().

31. Describe the difference between internal fragmentation and external fragmentation.

Internal Fragmentation: Wasted memory within allocated blocks due to fixed-size allocation.

External Fragmentation: Wasted memory outside of allocated blocks due to scattered free memory fragments.

32. What is demand paging, and how does it improve memory management efficiency?

Demand paging loads pages into memory only when they are needed, rather than loading the entire process into memory. This reduces memory usage and allows for better use of available physical memory.

33. Explain the role of the page table in virtual memory management.

The page table maps virtual addresses to physical memory addresses. It allows the OS to translate virtual memory references made by processes into physical memory locations, enabling efficient memory management.

34. How does a memory management unit (MMU) work?

The MMU is a hardware component that handles virtual-to-physical address translation. It uses the page table to translate virtual addresses generated by the CPU into physical addresses used to access memory.

35. What is thrashing, and how can it be avoided in virtual memory systems?

Thrashing occurs when a system spends most of its time swapping pages in and out of memory rather than executing processes. It can be avoided by adjusting the degree of multiprogramming, using efficient page replacement algorithms, or increasing physical memory.

36. What is a system call, and how does it facilitate communication between user programs and the operating system?

A system call is an interface through which a user program requests services from the OS kernel. It allows user programs to perform privileged operations like file manipulation, process control, and communication with hardware.

37. Describe the difference between a monolithic kernel and a microkernel.

Monolithic Kernel: Combines all OS services in one large block of code running in a single address space, offering high performance.

Microkernel: Separates core services (like IPC, memory management) into a small kernel, with other services running in user space, offering modularity and reliability.

38. How does an operating system handle I/O operations?

The OS handles I/O operations by abstracting device-specific details, managing I/O devices through device drivers, buffering data, and using interrupts to manage asynchronous I/O events.

39. Explain the concept of a race condition and how it can be prevented.

A race condition occurs when the outcome of a process depends on the timing or sequence of uncontrollable events. It can be prevented by using synchronization mechanisms like locks, semaphores, and condition variables to enforce mutual exclusion.

40. Describe the role of device drivers in an operating system.

Device drivers are specialized software that enables the OS to communicate with hardware devices. They provide an interface between the OS and hardware, translating OS commands into device-specific operations.

41. What is a zombie process, and how does it occur? How can a zombie process be prevented?

A zombie process is a process that has finished execution but still has an entry in the process table because its parent process has not yet called wait() to read its exit status. It can be prevented by ensuring that the parent process calls wait() or using signal handlers to reap zombie processes automatically.

42. Explain the concept of an orphan process. How does an operating system handle orphan processes?

An orphan process is a child process whose parent has terminated. The OS handles orphan processes by reassigning them to the init process (in Unix-like systems), which then takes responsibility for cleaning them up.

43. What is the relationship between a parent process and a child process in the context of process management?

A parent process creates a child process using the fork() system call. The child process is a duplicate of the parent and inherits most of its attributes. The parent can control or interact with the child, and it is responsible for reading the child's exit status.

44. How does the fork() system call work in creating a new process in Unix-like operating systems?

The fork() system call creates a new process by duplicating the parent process. The child process receives a unique process ID, but otherwise, it is a copy of the parent, including the same code and data. The fork returns 0 to the child and the child's PID to the parent.

45. Describe how a parent process can wait for a child process to finish execution.

A parent process can wait for a child process to finish execution by using the wait() or waitpid() system calls. These calls block the parent until the child terminates and return the child's exit status to the parent.

46. What is the significance of the exit status of a child process in the wait() system call?

The exit status of a child process indicates how the child terminated (normally or via a signal) and provides a status code that the parent process can use to determine if the child executed successfully or encountered errors.

47. How can a parent process terminate a child process in Unix-like operating systems?

A parent process can terminate a child process using the kill() system call, specifying the child's PID and the signal to send (e.g., SIGTERM to terminate the process gracefully).

48. Explain the difference between a process group and a session in Unix-like operating systems.

Process Group: A collection of processes that can be managed as a single unit, often for job control purposes.

Session: A group of process groups, usually associated with a login session, where one process group is the session leader, and the terminal controls are tied to the session.

49. Describe how the exec() family of functions is used to replace the current process image with a new one.

The exec() family of functions replaces the current process image with a new program, effectively transforming the process into a different program. The process retains its PID but gets a new memory image, including the code, data, and stack.

50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?

The waitpid() system call allows a parent process to wait for a specific child process to terminate, identified by its PID. Unlike wait(), which waits for any child process, waitpid() can also be used with options to specify non-blocking behavior or to wait for any child process in the process group.

51. How does process termination occur in Unix-like operating systems?

Process termination occurs when a process completes its execution or is terminated by a signal. The process sends an exit status to the parent, closes all open resources, and changes its state to "terminated" before being reaped by the parent.

52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?

The long-term scheduler, also known as the job scheduler, selects which jobs (processes) from the job pool should be loaded into memory for execution. It controls the degree of multiprogramming by deciding how many processes should be in the ready queue, thus balancing CPU and memory utilization.

53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?

The short-term scheduler, or CPU scheduler, decides which process in the ready queue should be executed next by the CPU. It operates more frequently than the long-term scheduler, making decisions every few milliseconds. The medium-term scheduler manages process swapping between main memory and disk, making decisions based on memory management rather than CPU allocation.

54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.

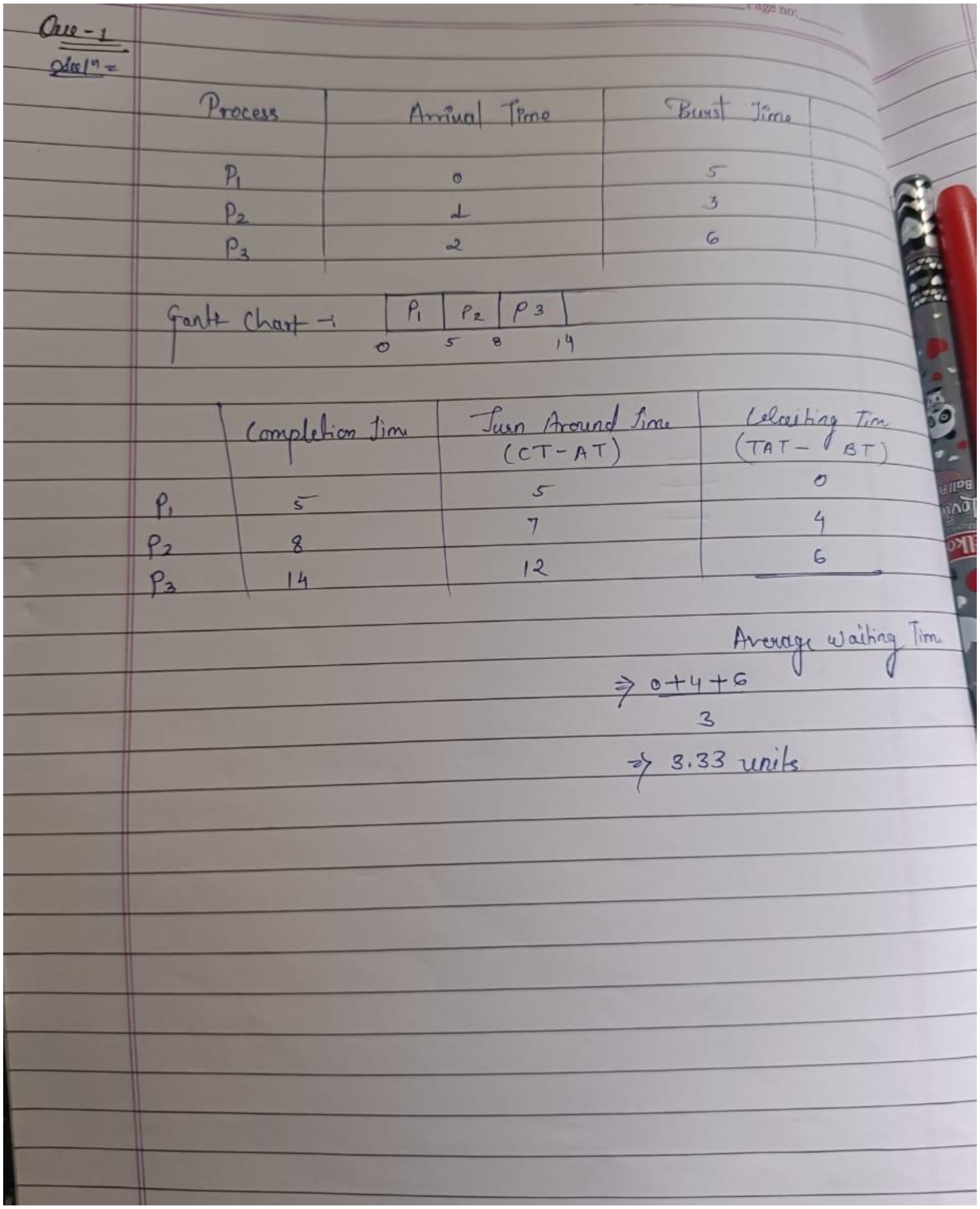
The medium-term scheduler is invoked when the system is low on memory, and it needs to swap out processes to free up memory. For example, if a process is idle or waiting for I/O, the medium-term scheduler may swap it out to disk, reducing memory pressure and allowing other processes to use the freed memory, thus improving system efficiency.

Part E

1. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	6

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.



Que - 1

Sol^n =

Process	Arrival Time	Burst Time
P_1	0	5
P_2	1	3
P_3	2	6

Gantt Chart :-

P_1	P_2	P_3

0 5 8 14

	Completion Time	Turn Around Time (CT - AT)	Waiting Time (TAT - BT)
P_1	5	5	0
P_2	8	7	4
P_3	14	12	6

Average waiting Time

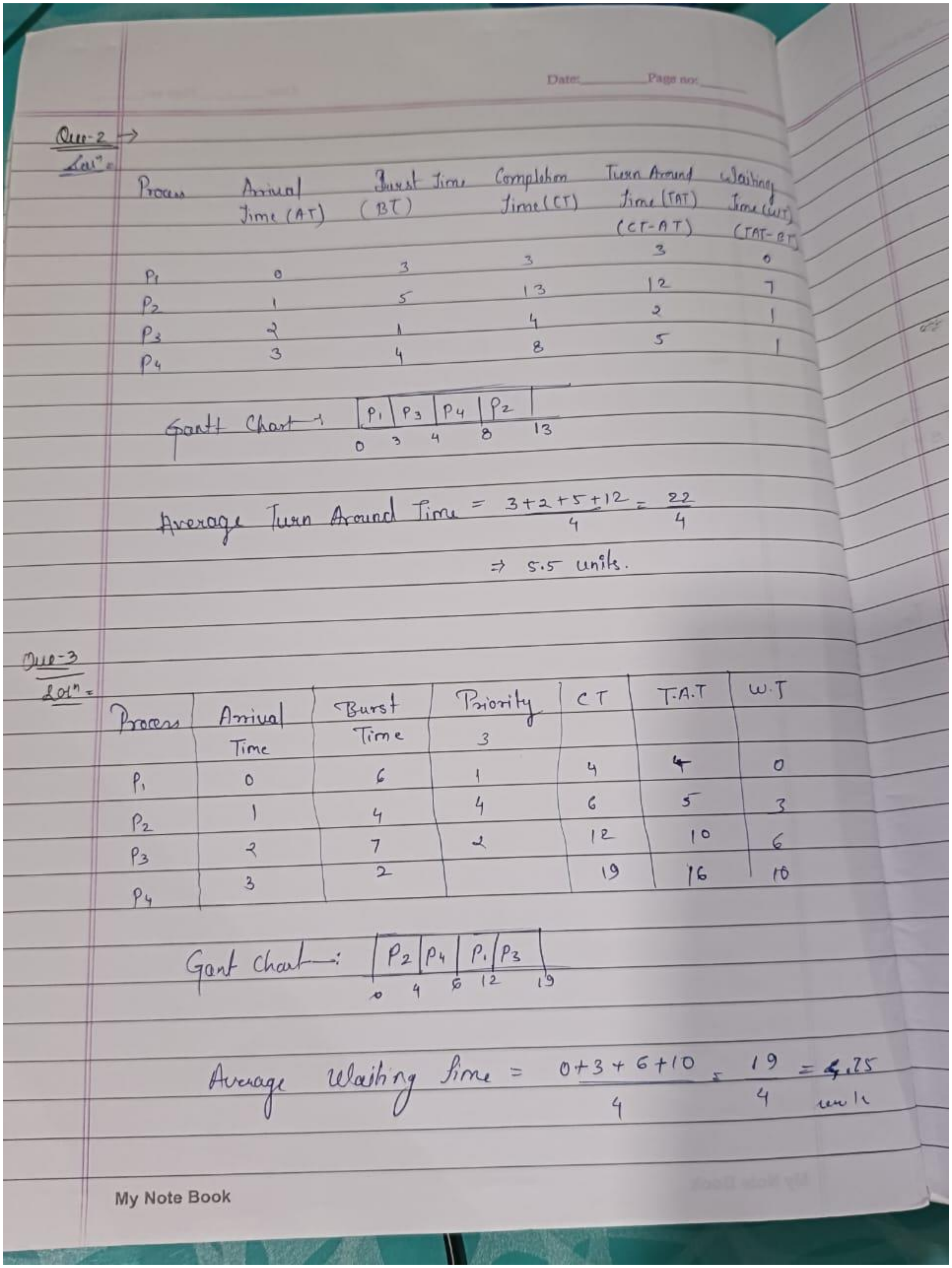
$\Rightarrow \frac{0+4+6}{3}$

$\Rightarrow$ 3.33 units

2. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

Calculate the average turnaround time using Shortest Job First (SJF) scheduling.



Date:________ Page no:________

Que-2 →

Solⁿ =

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) (CT-AT)	Waiting Time (WT) (TAT-BT)
P1	0	3	3	3	0
P2	1	5	13	12	7
P3	2	1	4	2	1
P4	3	4	8	5	1

Gantt Chart →

P1	P3	P4	P2

0 3 4 8 13

Average Turn Around Time $= \frac{3+2+5+12}{4} = \frac{22}{4}$

$\Rightarrow$ 5.5 units.

Que-3

Solⁿ =

Process	Arrival Time	Burst Time	Priority	CT	T.A.T	W.T
			3			
P1	0	6	1	4	4	0
P2	1	4	4	6	5	3
P3	2	7	2	12	10	6
P4	3	2		19	16	10

Gant Chart :

P2	P4	P1	P3

0 4 6 12 19

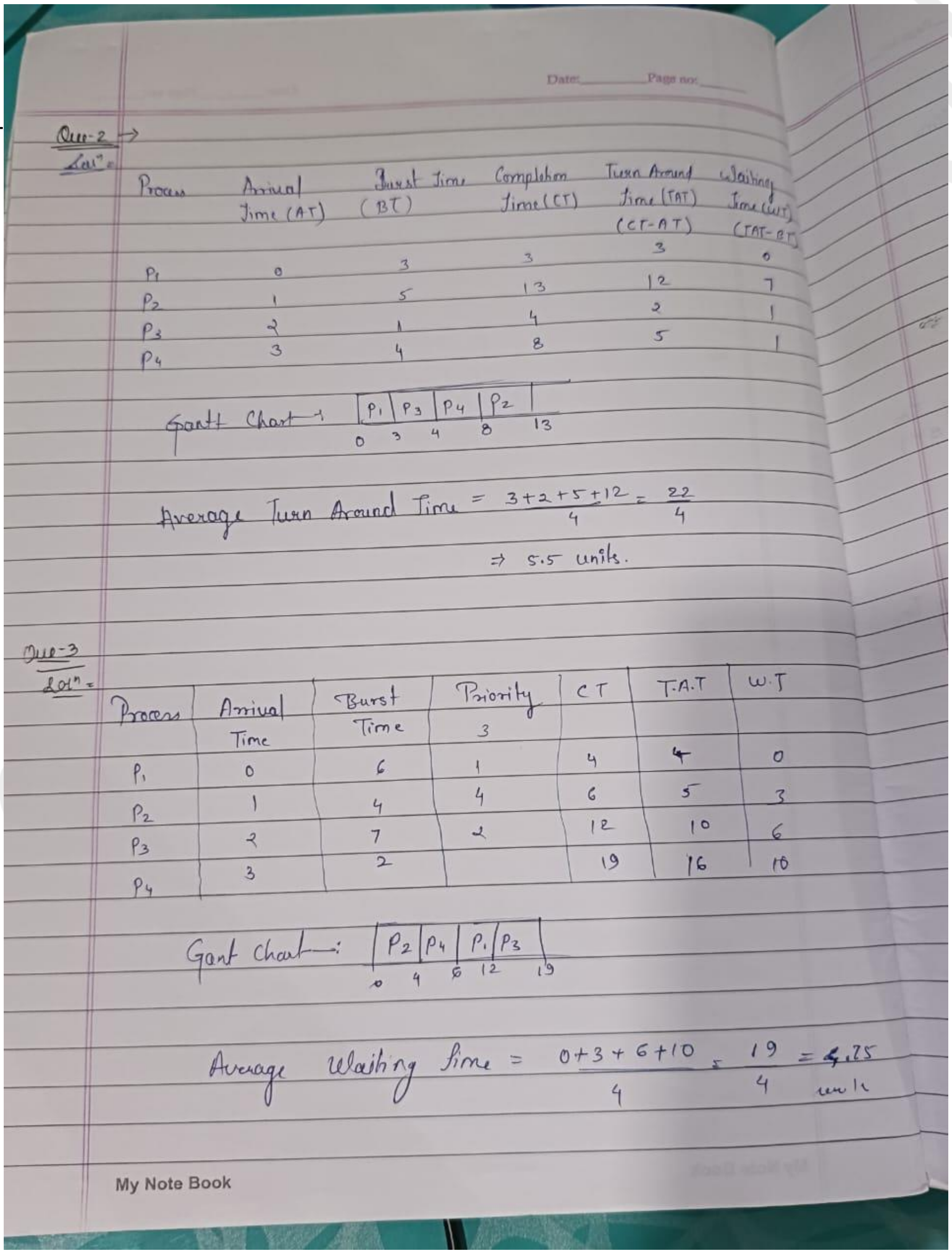
Average Waiting Time $= \frac{0+3+6+10}{4} = \frac{19}{4} = 4.75$ units

My Note Book

3. Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority):

Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Calculate the average waiting time using Priority Scheduling.



Date:________ Page no:________

Que-2 →

Solⁿ =

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT) (CT-AT)	Waiting Time (WT) (TAT-BT)
P1	0	3	3	3	0
P2	1	5	13	12	7
P3	2	1	4	2	1
P4	3	4	8	5	1

Gantt Chart :

```
| P1 | P3 | P4 | P2 |
0    3    4    8    13
```

Average Turn Around Time $= \frac{3+2+5+12}{4} = \frac{22}{4}$

$\Rightarrow$ 5.5 units.

Que-3

Solⁿ =

Process	Arrival Time	Burst Time	Priority	CT	T.A.T	W.T
			3			
P1	0	6	1	4	4	0
P2	1	4	4	6	5	3
P3	2	7	2	12	10	6
P4	3	2		19	16	10

Gant Chart :

```
| P2 | P4 | P1 | P3 |
0    4    6    12   19
```

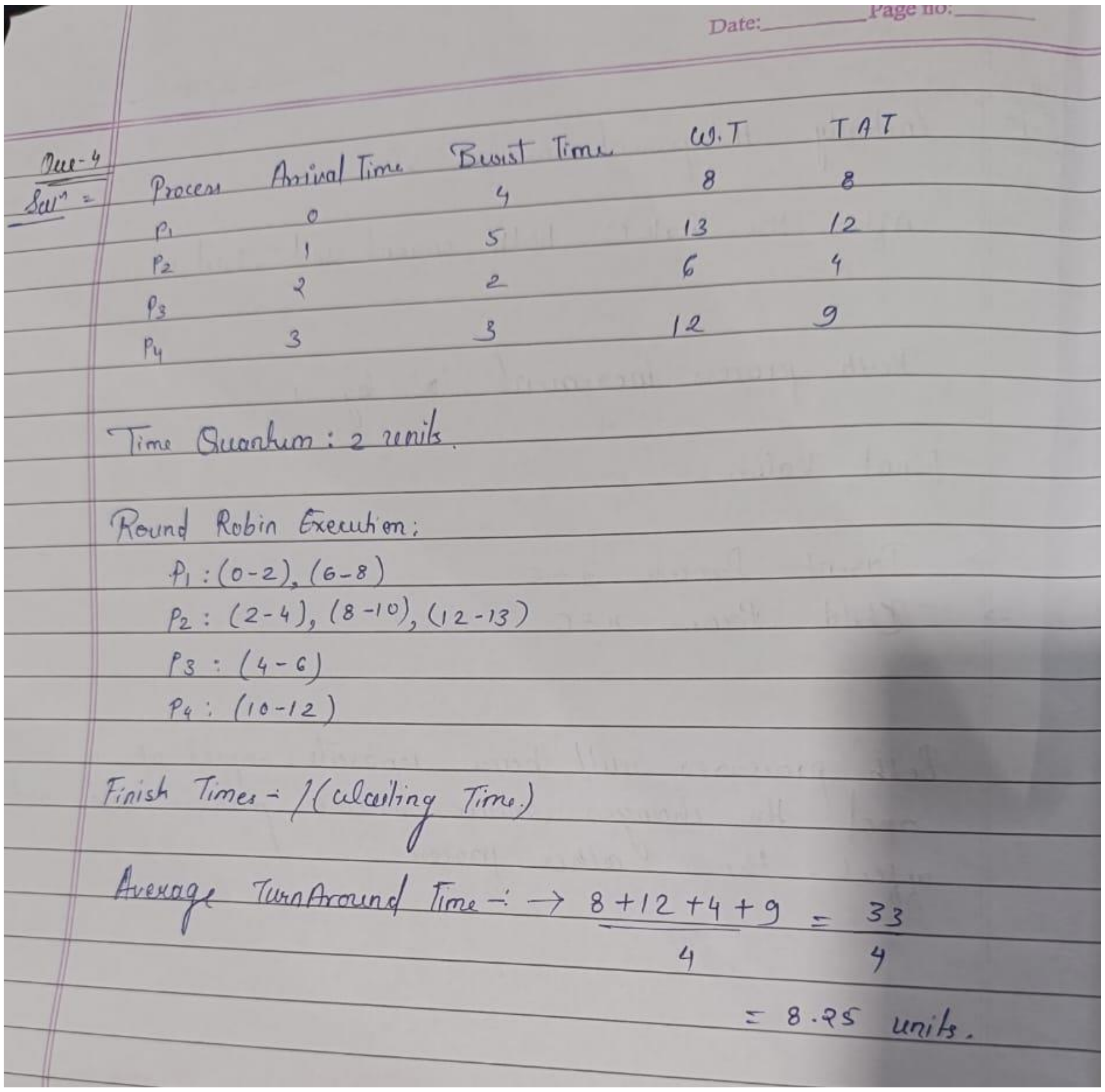
Average Waiting time $= \frac{0+3+6+10}{4} = \frac{19}{4} = 4.25$ unit

My Note Book

4. Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units:

Process	Arrival Time	Burst Time
P1	0	4
P2	1	5
P3	2	2
P4	3	3

Calculate the average turnaround time using Round Robin scheduling.


Date:______ Page no:______

Ques-4

Sol^n =

Process	Arrival Time	Burst Time	W.T	TAT
P_1	0	4	8	8
P_2	1	5	13	12
P_3	2	2	6	4
P_4	3	3	12	9

Time Quantum : 2 units.

Round Robin Execution :

P_1 : (0-2), (6-8)

P_2 : (2-4), (8-10), (12-13)

P_3 : (4-6)

P_4 : (10-12)

Finish Times :- /(Calculating Time.)

Average TurnAround Time :- $\rightarrow \frac{8+12+4+9}{4} = \frac{33}{4}$

$= 8.25$ units.

5. Consider a program that uses the **fork()** system call to create a child process. Initially, the parent process has a variable **x** with a value of 5. After forking, both the parent and child processes increment the value of **x** by 1.
   What will be the final values of **x** in the parent and child processes after the **fork()** call?

- Initially, the parent process has x = 5.
- After the fork(), both parent and child processes have x = 5.
- Both processes increment x by 1.

Final Values:

- **Parent Process:** x = 6
- **Child Process:** x = 6

Both processes will have separate copies of x, and the changes made in one process do not affect the other process.