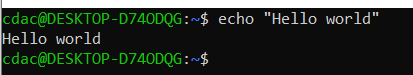
Assignment No 2

1. What will the following commands do?
   1. echo "Hello, World!"

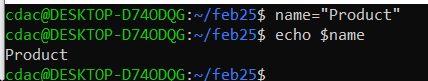
This command is used to Print Hello World at Terminal



* 1. name="Productive"

This Command assign String to a Variable

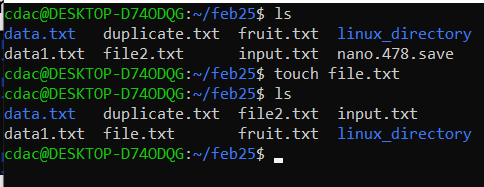
In our case Product String is Assign to Name Variable



* 1. touch file.txt

This Command is Use to Create a File Named as file.txt

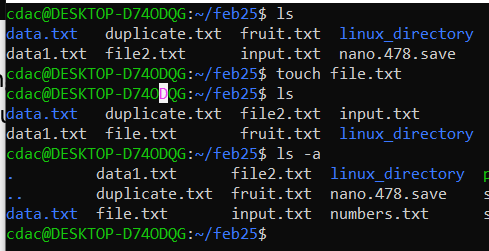
Touch command is use to create a new file



* 1. ls -a

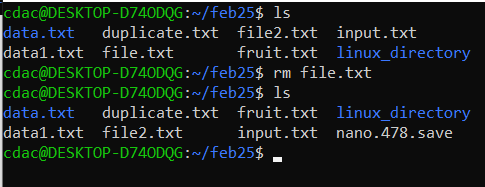
ls command is use to display file containing in working directory

ls -a command is use to display all file including hidden files present in working directory



* 1. rm file.txt

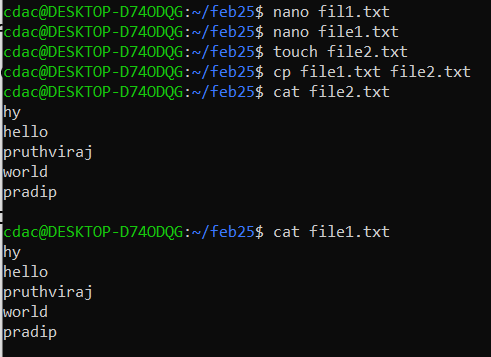
rm Command is use to delete a file permanently



* 1. cp file1.txt file2.txt

cp Command is use to copy the content of one file to another

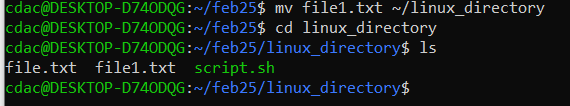
By above command we are copying file1 content to file2



* 1. mv file.txt ~/path/to/directory/

mv command is use to move file or directory another path

we are moving file.txt. to another path



* 1. chmod 755 script.sh

chmod command is use to change the access permission of the file

There are 3 actions for a file

Read-4

Write-2

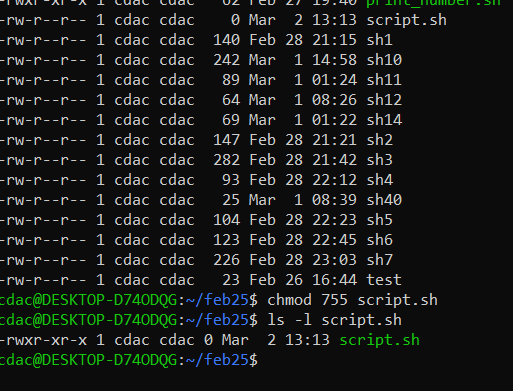
Execute-1

By the above command 7 represents read, write and execute whereas 5 represents read and write .

7 for admin

5 for group

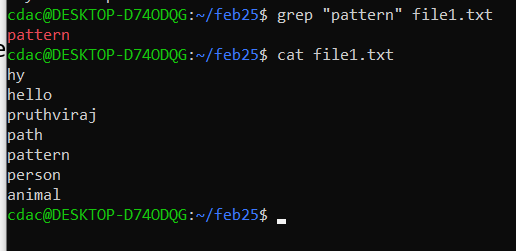
5 others



* 1. grep "pattern" file.txt

grep command is called as regular Expression

by above command it will extract the word pattern from the file named file.txt



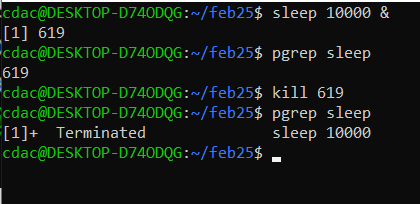
* 1. kill PID

first I have created a process using command **sleep 1000 &** this will create a process at background and process ID will be printed at terminal in our case its process ID=619

To kill the process use **kill 619**

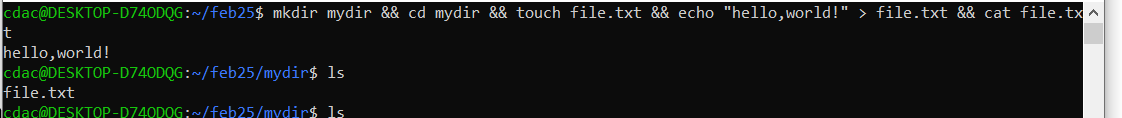
This will terminate the process id 619

**pgrep Sleep** command is used to show the running Sleep processes



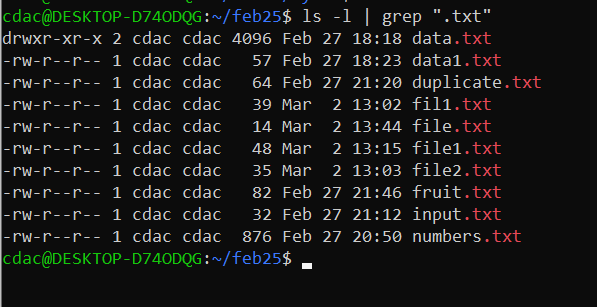
* 1. mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt

The above command will make directory named mydir then open that directory and create a file name as file.txt and then print hello world on terminal



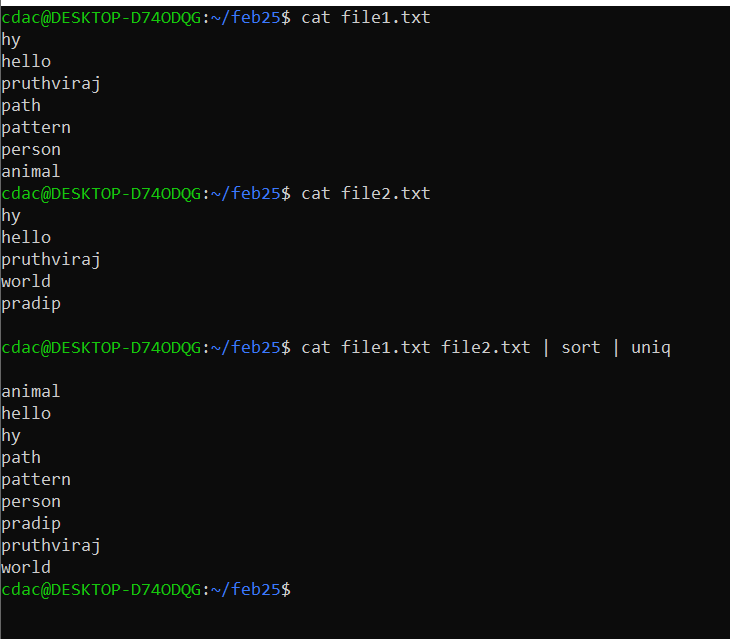
* 1. ls -l | grep ".txt"

The above command will list out all the file with suffix named as .txt



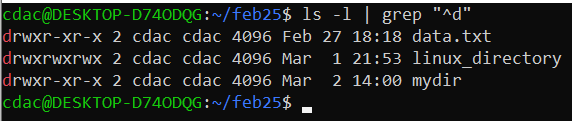
* 1. cat file1.txt file2.txt | sort | uniq

The above command will print all the data from file1.txt and file2.txt in sorted by alphabetical way and removing duplicates



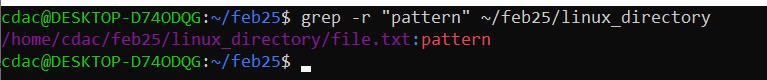
* 1. ls -l | grep "^d"

the above command will list out directory by using d as prefix

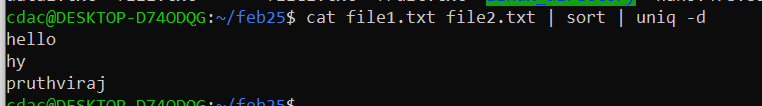


* 1. grep -r "pattern" /path/to/directory/

The above command search for pattern word in a file navigating recursively



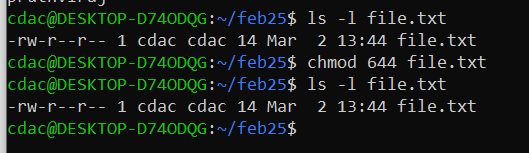
* 1. cat file1.txt file2.txt | sort | uniq –d



It will sort out unique word from file1.txt and file2.txt by sorting out

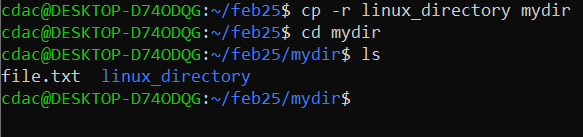
* 1. chmod 644 file.txt

It give read write permission for admin , read permission for group and read permission for others



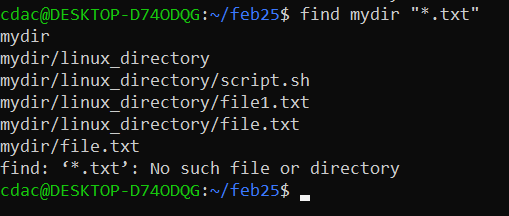
* 1. cp -r source\_directory destination\_directory

This command is used to copy any particular directory to another directory



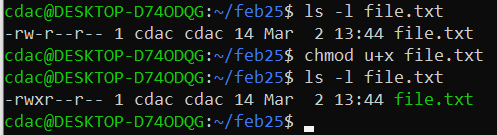
* 1. find /path/to/search -name "\*.txt"

The above command is used to find the file named suffix with .txt



* 1. chmod u+x file.txt

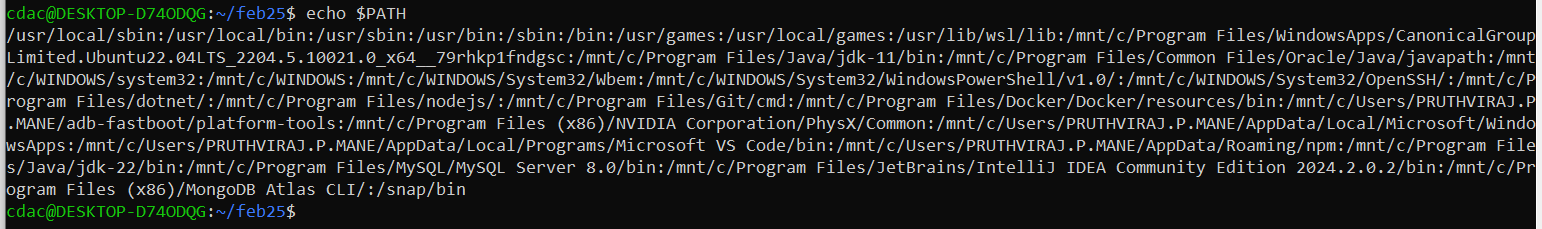
The above command is used to make admin read write and execute over file named as file.txt



* 1. echo $PATH

$PATH is an environment variable it shows the the executable file which system looks for

Echo is used to print the directories and executable file name



**Part B**

**Identify True or False:**

1. ls is used to list files and directories in a directory.

True

1. mv is used to move files and directories.

**True**

1. cd is used to copy files and directories.

**false**

1. pwd stands for "print working directory" and displays the current directory.

**False**

1. grep is used to search for patterns in files.

**True**

1. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute

permissions to group and others.

**True**

7. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1

if directory1 does not exist.

True

1. rm -rf file.txt deletes a file forcefully without confirmation.

True

**Identify the Incorrect Commands:**

1. chmodx is used to change file permissions.

Chmodx is incorrect command instead it should be chmod

1. cpy is used to copy files and directories.

Cp command is used to copy file

1. mkfile is used to create a new file.

Touch is used to create a new file

1. catx is used to concatenate files.

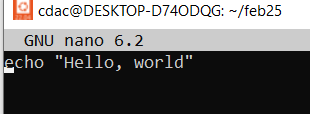
Cat command is used to print content of files

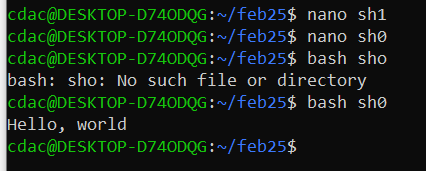
1. rn is used to rename files.

rn is an incorrect command

**Part C**

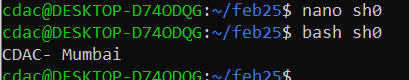
Question 1: Write a shell script that prints "Hello, World!" to the terminal.

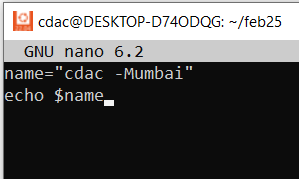




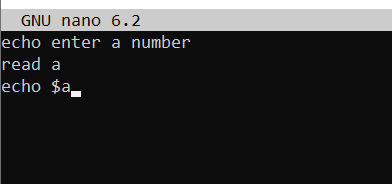
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the

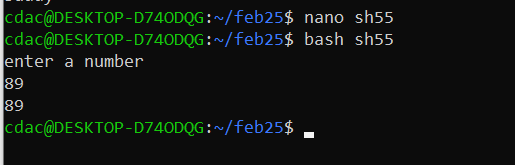
value of the variable.





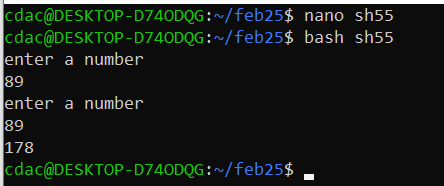
Question 3: Write a shell script that takes a number as input from the user and prints it.

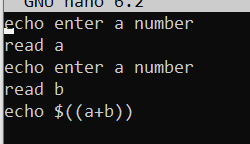




Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the

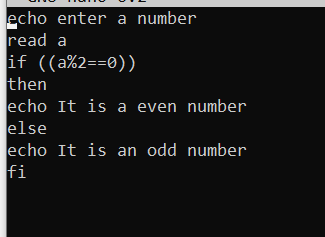
result.

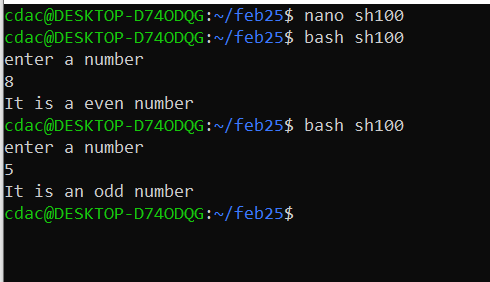




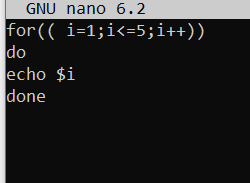
Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise

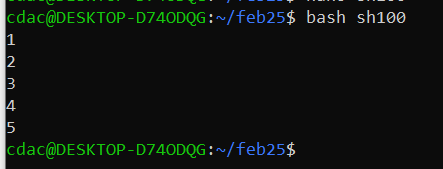
prints "Odd".



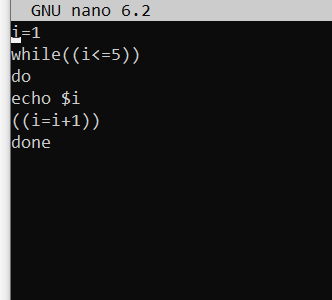


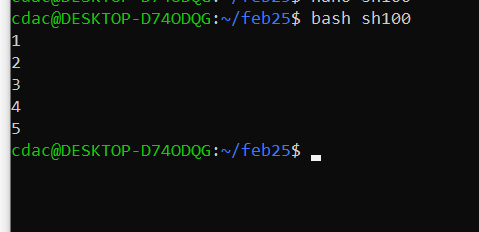
Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.





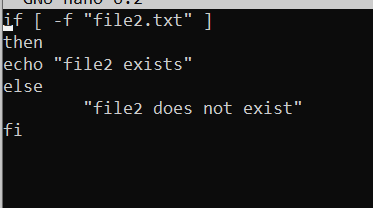
Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.

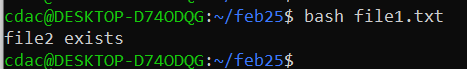




Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it

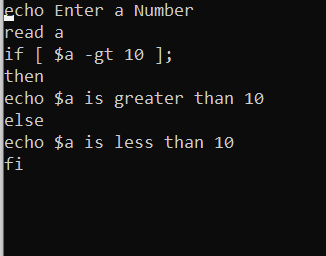
does, print "File exists", otherwise, print "File does not exist".

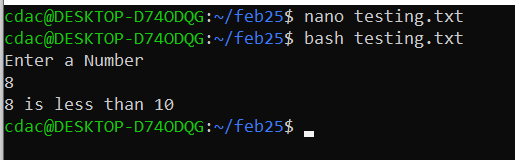




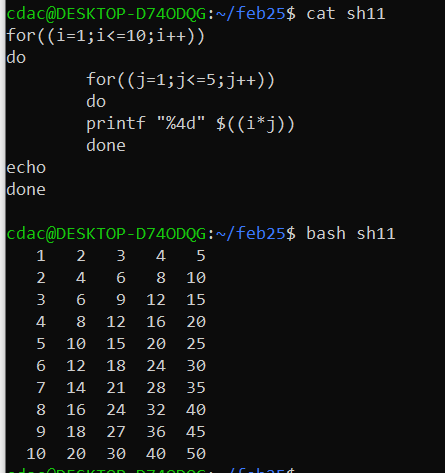
Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and

prints a message accordingly.

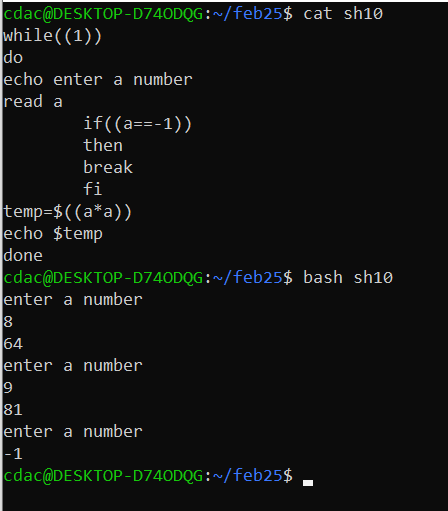




Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.



Question 11: Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.



**Part D**

**Common Interview Questions (Must know)**

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

An operating system (OS) is the software that acts as a bridge between hardware and software. It manages resources like CPU, memory, and storage while also handling user interactions. Its main functions include process management, memory management, file system handling, device management, and security.

Explain the difference between process and thread.

A process is an independent program in execution with its own memory and resources, while a thread is a smaller unit within a process that shares resources like memory and file handles. Threads are lightweight and faster, whereas processes require more overhead to create and manage.

What is virtual memory, and how does it work?

Virtual memory allows a system to use disk space as if it were RAM, enabling programs to run even when physical memory is full. It works by swapping inactive memory pages between RAM and the disk using a technique called paging.

Describe the difference between multiprogramming, multitasking, and multiprocessing.

 Multiprogramming: Multiple programs are loaded into memory, but only one executes at a time.

 Multitasking: Multiple processes appear to run simultaneously by switching between them rapidly.

 Multiprocessing: Multiple CPUs execute multiple processes at the same time.

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

A file system organizes and stores data on a storage device. Its components include:

Files (actual data storage units)

Directories (folders that organize files)

Metadata (file attributes like size, type, and permissions)

File allocation table (FAT) / inode table (keeps track of file locations)

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

A deadlock happens when processes are stuck waiting for resources held by each other. It can be prevented using:

Deadlock avoidance (checking resource availability before allocation)

Resource ordering (allocating resources in a fixed order)

Timeouts (forcing processes to release resources if they wait too long)

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

Kernel: The core of the OS that directly interacts with hardware.

Shell: The interface that allows users to interact with the OS, like command-line interpreters (bash, cmd) or GUI shells (Windows Explorer).

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

CPU scheduling is how the OS decides which process gets the CPU next. It ensures fair resource allocation, maximizes CPU utilization, and improves system performance.

1. How does a system call work?

A system call is a request made by a user program to the OS for performing operations like file handling or process creation. It works by switching from user mode to kernel mode, executing the requested operation, and returning the result.

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

Device drivers act as a bridge between the OS and hardware components, translating OS commands into instructions that hardware can understand.

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

The page table keeps track of the mapping between virtual memory addresses and physical memory addresses, enabling efficient memory access and management.

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

Thrashing happens when excessive paging causes the system to spend more time swapping memory than executing processes. It can be avoided by increasing RAM, tuning the page replacement algorithm, or limiting the number of active processes.

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

A semaphore is a synchronization tool that controls access to shared resources in multi-threaded environments. It can be binary (mutex) or counting to manage multiple resource instances.

14. How does an operating system handle process synchronization?

The OS uses mechanisms like semaphores, monitors, and locks to prevent race conditions and ensure proper execution of processes accessing shared resources.

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

An interrupt allows the OS to respond to external events (like I/O operations) or system conditions (like division by zero) immediately, pausing current execution and handling the event.

1. Explain the concept of a file descriptor.

A file descriptor is a unique number assigned to an open file or I/O stream in Unix-like systems, allowing processes to read, write, or close the file.

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

A system recovers using techniques like system logs, journaling file systems, and checkpointing to restore the last stable state.

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

Monolithic Kernel: Everything (device drivers, file systems) runs in kernel mode.

Microkernel: Only core functionalities run in kernel mode; other services run in user mode, making the system more stable but slower.

19 What is the difference between internal and external fragmentation?

Internal Fragmentation: Unused memory inside allocated space.

External Fragmentation: Free memory is available but scattered in small chunks, making allocation difficult.

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

The OS uses device drivers and buffers to manage I/O requests efficiently through interrupt-driven or DMA-based (Direct Memory Access) methods.

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

Preemptive: The OS can switch processes anytime (e.g., Round Robin).

Non-Preemptive: A process runs until it voluntarily releases the CPU (e.g., FCFS).

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

Round-robin scheduling assigns a fixed time slice (quantum) to each process in a cyclic order, ensuring fairness in CPU time distribution.

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

Processes are assigned priority values, and the CPU executes the highest-priority process first. Priorities can be static (fixed) or dynamic (changing based on execution time or resource needs).

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

SJN selects the process with the shortest execution time first, reducing average wait time but suffering from the starvation issue if short jobs keep arriving.

25. Explain the concept of multilevel queue scheduling.

Multilevel queue scheduling divides processes into multiple priority queues (e.g., foreground and background processes) and applies different scheduling algorithms for each queue.

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

A PCB is like a record that stores all the details about a process in an OS. It contains:

Process ID (PID)

Process state (Ready, Running, Waiting, etc.)

Program counter (next instruction to execute)

CPU registers

Memory info (base and limit registers)

I/O status and open files

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

A process goes through these states:

New → Process is created.

Ready → Waiting for CPU.

Running → Executing on CPU.

Waiting → Waiting for I/O or some event.

Terminated → Process is finished.

Blocked → Waiting for a resource.

Transitions happen based on scheduling, I/O completion, or termination.

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

Processes communicate using Inter-Process Communication (IPC) methods:

Shared Memory → Processes share a common memory space.

Message Passing → Data is sent between processes using pipes, message queues, or sockets.

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

It ensures that multiple processes execute correctly when they share resources. It's needed to prevent race conditions, maintain data consistency, and avoid deadlocks. Methods like semaphores, mutexes, and monitors help in synchronization.

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

A zombie process is one that has completed execution but still has an entry in the process table because the parent hasn't read its exit status. It happens when the parent doesn’t call wait() to clean up the child’s resources.

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

Internal fragmentation → Memory is allocated in fixed blocks, and some space inside a block remains unused.

External fragmentation → Free memory is scattered in small chunks, making it hard to allocate large blocks.

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

Demand paging loads only the required pages into memory instead of loading the whole process, saving RAM. If a needed page isn’t in memory, a page fault occurs, and the OS fetches it from disk.

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

The page table maps logical addresses to physical addresses. It keeps track of which page of a process is in which frame of physical memory, helping in efficient memory access.

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

The MMU is a hardware component that translates logical (virtual) addresses into physical addresses. It works with the page table to ensure proper memory access.

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

Thrashing happens when a system spends most of its time swapping pages in and out of memory instead of executing processes. It can be avoided by:

Working set model → Keeping frequently used pages in memory.

Increasing RAM → Reduces swapping.

Better page replacement algorithms like LRU.

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

operating system?

A system call is a request from a user program to the OS for performing operations like file handling, process control, and memory management. It provides controlled access to system resources.

Describe the difference between a monolithic kernel and a microkernel.

Monolithic Kernel → The entire OS runs as a single large program with all services inside the kernel. (Example: Linux)

Microkernel → The kernel only handles essential tasks like process management and IPC, while other services run in user space. (Example: Minix)

How does an operating system handle I/O operations?

The OS uses I/O scheduling and buffering to manage devices efficiently. It handles input/output requests via device drivers and system calls like read() and write().

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

A race condition occurs when multiple threads/processes try to access shared resources simultaneously, leading to unpredictable results. It can be prevented using locks, semaphores, or atomic operations.

Describe the role of device drivers in an operating system.

Device drivers act as intermediaries between the OS and hardware devices. They translate OS commands into device-specific operations.

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

A zombie process occurs when a child process completes but its exit status isn’t read by the parent. It can be prevented using wait() or waitpid().

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 assigns such processes to init (PID 1) so they don’t remain unmanaged.

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 fork(). The child gets a copy of the parent’s resources but has its own execution space.

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

fork() creates a duplicate process of the caller. The parent gets the child’s PID, and the child gets 0 as the return value.

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

The parent calls wait() or waitpid() to pause until the child finishes, preventing zombie processes.

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

It tells the parent if the child completed successfully or terminated with an error.

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

Using the kill() command or sending a signal like SIGTERM to the child’s PID.

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

 Process Group → A collection of related processes, identified by a group ID.

 Session → A collection of process groups managed by a single shell or terminal.

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

new one.

exec() replaces the running process with a new program, keeping the same PID but loading a different executable.

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

wait()?

waitpid() lets the parent wait for a specific child process, whereas wait() waits for any child.

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

A process terminates in the following ways:

Normal termination → Process finishes execution and calls exit().

Abnormal termination → Due to errors like segmentation faults.

Signal termination → The OS or another process sends a termination signal (e.g., SIGKILL).

Parent termination → If the parent process dies, its child becomes an orphan, which is then adopted by init.

After termination, the process enters the zombie state until the parent collects the exit status using wait().

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 controls the number of processes that enter the ready queue. It decides which jobs should be loaded into memory for execution.

It runs infrequently but plays a major role in multiprogramming.

If too many processes are admitted → The system may slow down due to excessive context switching.

If too few processes are admitted → Underutilization of CPU and memory.

By balancing job admission, the long-term scheduler maintains optimal system performance.

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, also known as the CPU scheduler, runs frequently and is responsible for selecting which process in the ready queue gets to execute on the CPU. Unlike the long-term scheduler, which controls the number of processes entering the system, the short-term scheduler focuses on deciding which process should run next. The medium-term scheduler, on the other hand, is responsible for swapping processes in and out of memory based on system load. The long-term scheduler operates infrequently, making high-level decisions about job admission, while the short-term scheduler runs at a much higher frequency, often several times per second, to manage process execution efficiently. The medium-term scheduler plays a supporting role by temporarily suspending processes to optimize memory usage and prevent system overload.

54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps

manage system resources more efficiently.

Suppose multiple processes are running, but the system is becoming slow due to high memory usage. Some processes are waiting for I/O, while others need CPU time.

Role of Medium-Term Scheduler:

It identifies low-priority or waiting processes and swaps them out to disk (reducing RAM usage).

When resources are available again, it swaps them back into memory for execution.

This prevents thrashing and ensures that active processes get enough CPU and memory.

Example: In virtual memory systems, when RAM is full, inactive processes are swapped to disk using the medium-term scheduler, freeing up space for active processes.

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

| Process | Arrival Time | Burst Time |

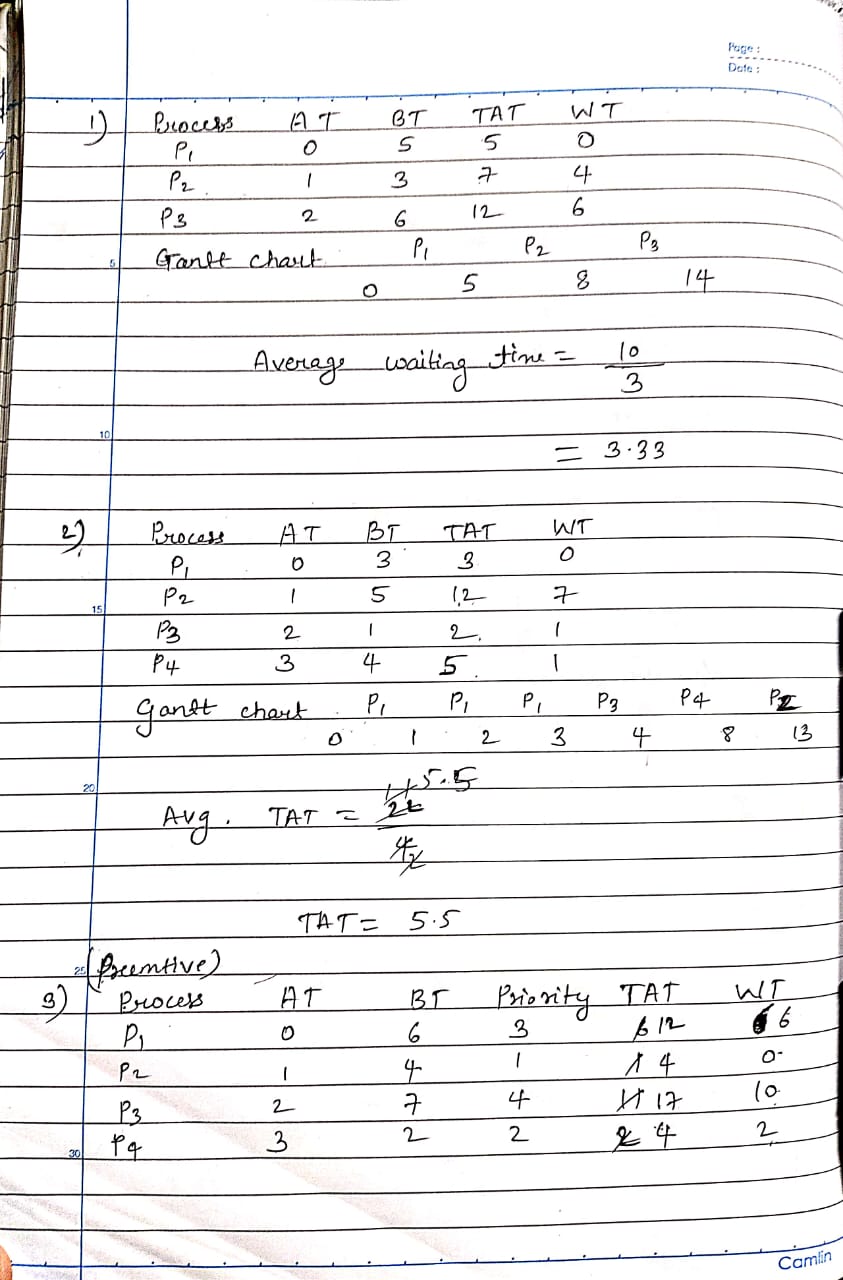
|---------|--------------|------------|

| P1 | 0 | 5 |

| P2 | 1 | 3 |

| P3 | 2 | 6 |

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



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

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

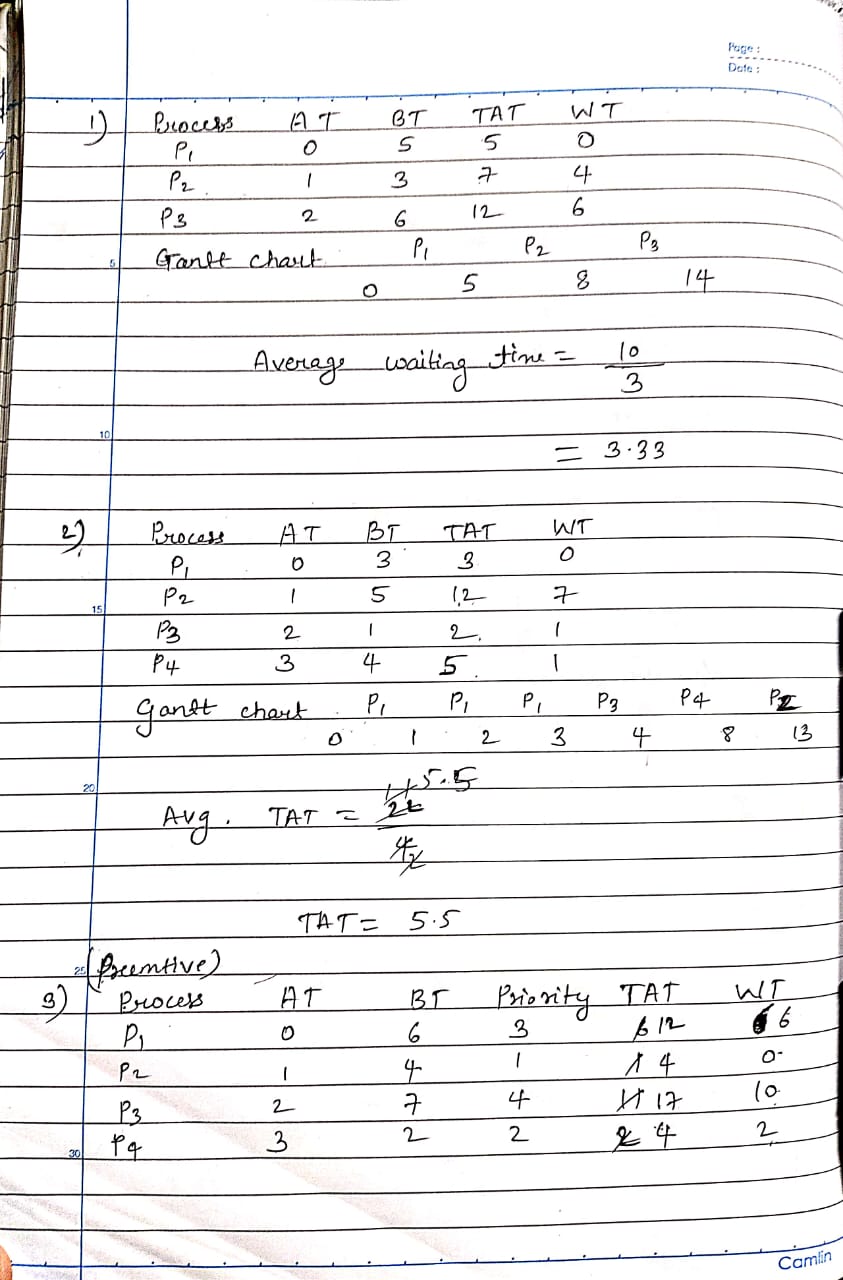
| P1 | 0 | 3 |

| P2 | 1 | 5 |

| P3 | 2 | 1 |

| P4 | 3 | 4 |

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



3. Consider the following processes with arrival times, burst times, and priorities (lower number

indicates higher priority):

| Process | Arrival Time | Burst Time | Priority |

|---------|--------------|------------|----------|

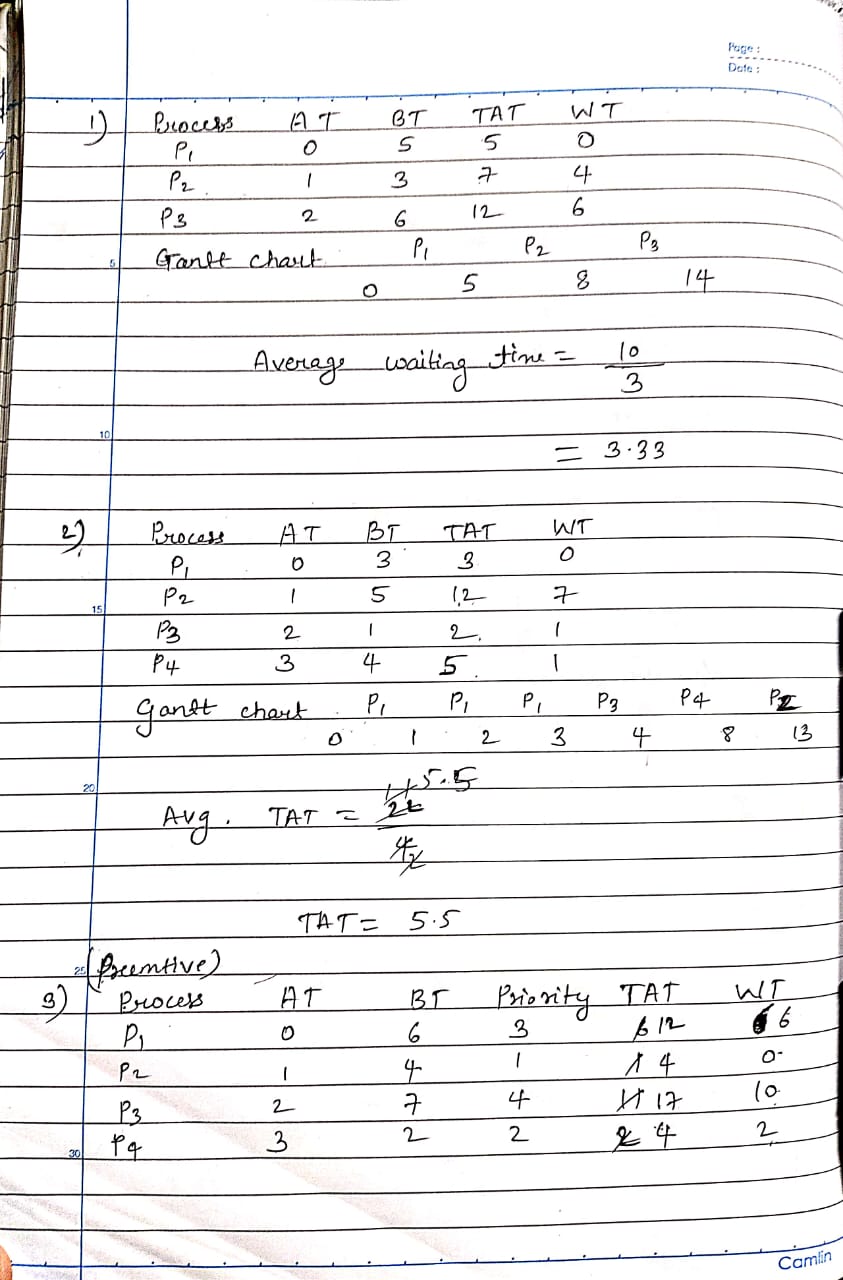
| P1 | 0 | 6 | 3 |

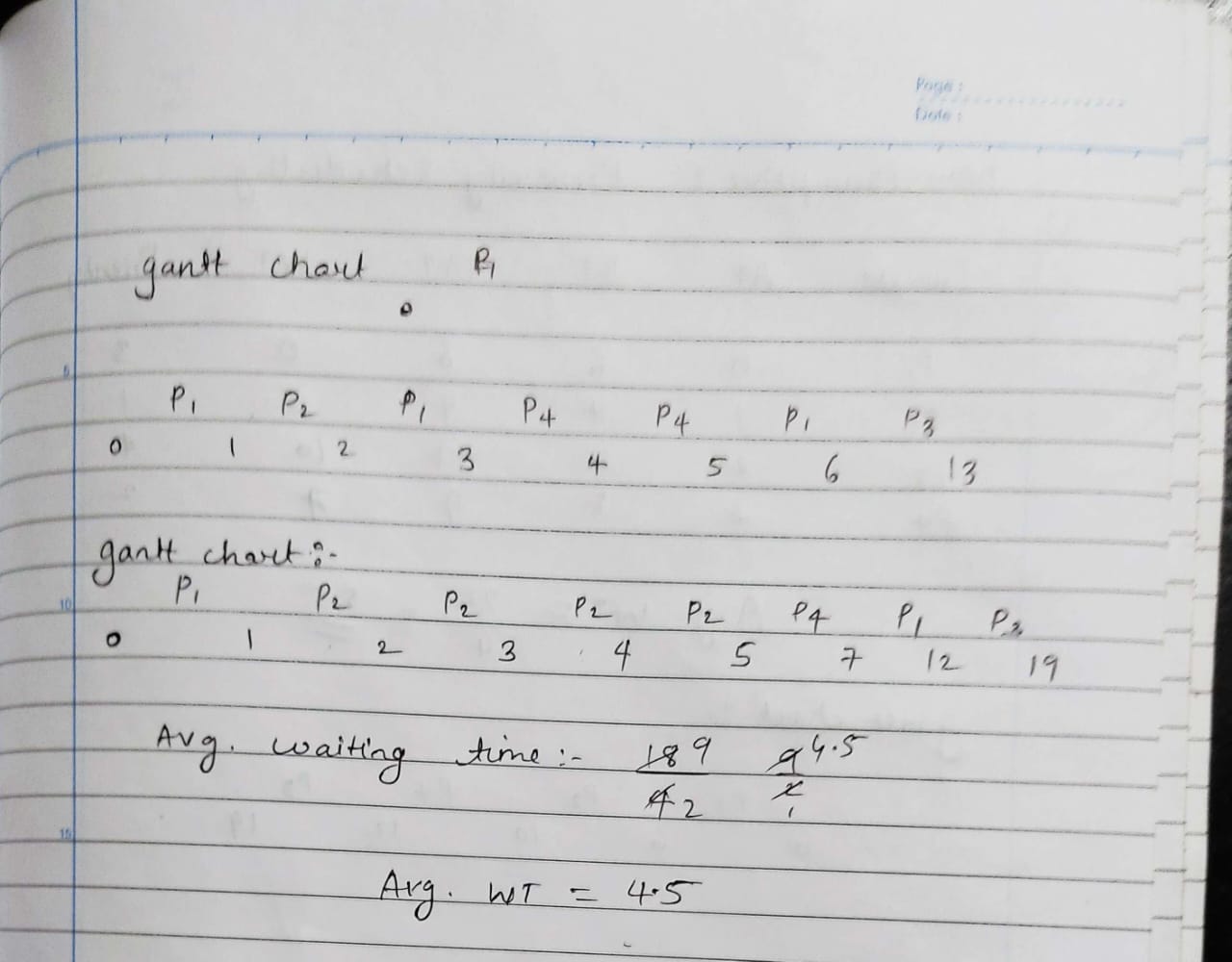
| P2 | 1 | 4 | 1 |

| P3 | 2 | 7 | 4 |

| P4 | 3 | 2 | 2 |

Calculate the average waiting time using Priority Scheduling.





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 |

|---------|--------------|------------|

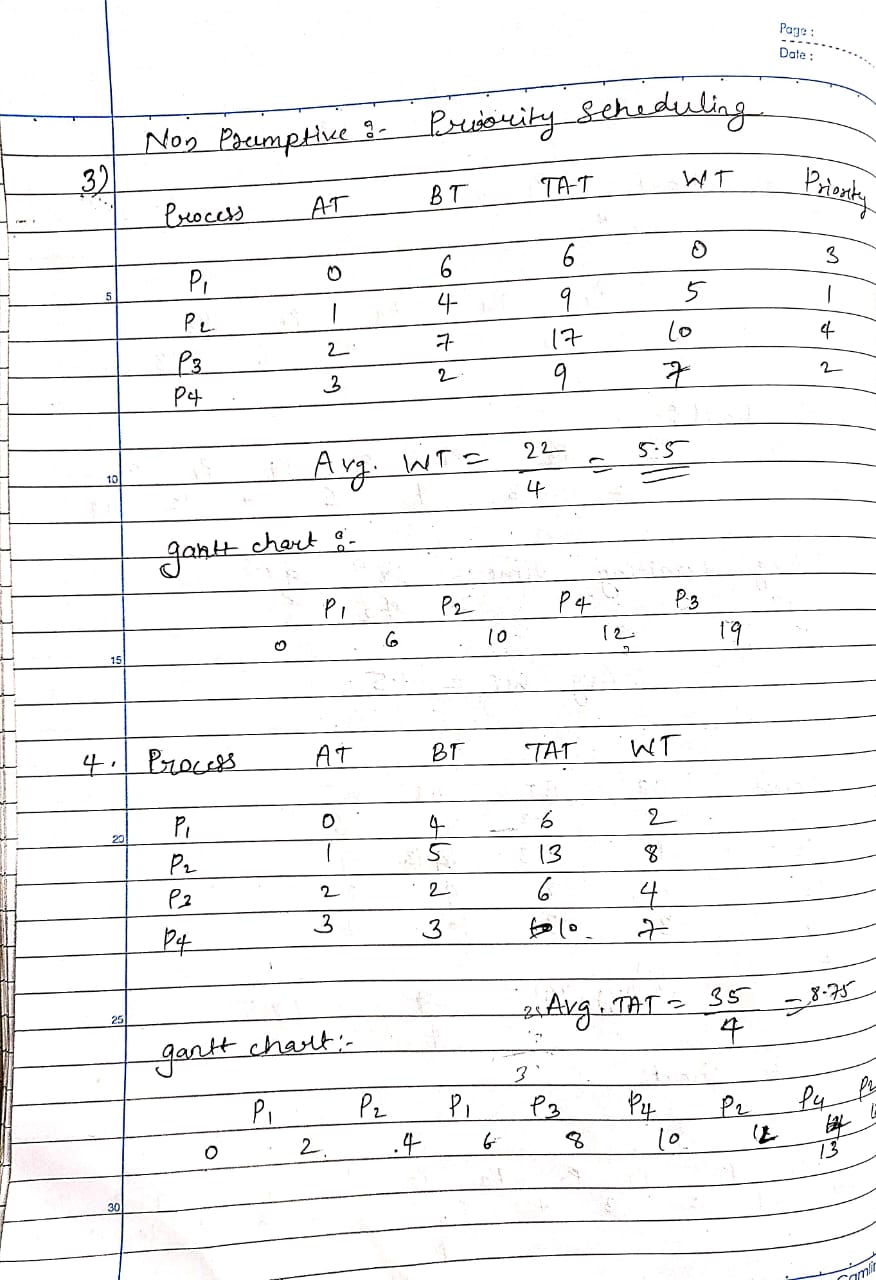
| P1 | 0 | 4 |

| P2 | 1 | 5 |

| P3 | 2 | 2 |

| P4 | 3 | 3 |

Calculate the average turnaround time using Round Robin scheduling.



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?

Answer: 6