* **PART A**

**What will the following commands do?**

**• echo "Hello, World!"**>> Prints Hello,World!

**• name="Productive"**  
>> Stores the value to name.  
   
**• touch file.txt**>> Creates a file named file.txt

**• ls -a**>> Lists all the files and directories with hidden files.

**• rm file.txt**>> removes the file.txt from the directory.

**• cp file1.txt file2.txt**>> Copies the 2 mentioned files.

**• mv file.txt /path/to/directory/**   
>> moves the file into the mentioned directory.

**• chmod 755 script.sh**   
>> It is used to change the permissions of the file. Octal numbers are used  
 7=owner has r,w,x permissions;5=group has r,-,x permissions;5= others has r,-,x permissions .

**• grep "pattern" file.txt**   
>> it is used to fetch specific words from a file. In this case pattern is obtained.

**• kill PID**>> Kills the processes/jobs of specific ID.

**• mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**   
>> Step 1: creates a new directory named mydir.  
 Step 2: changes the directory.  
 Step 3 : creates a file named file.txt  
 Step 4: enter Hello World! in file.txt  
 Step 5: used to see the contents of the file.

**• ls -l | grep ".txt"**  
>> Step 1: lists all files and directories with all info  
 Step 2: finds all the files with .txt extension.

**• cat file1.txt file2.txt | sort | uniq**   
>> Combines the content of the 2 files and sorts the data alphabetically an filters out the duplicates.

**• ls -l | grep "^d"**   
>> lists all the files and directories with info.

**• grep -r "pattern" /path/to/directory/**   
>> recursively searches for “pattern” in the mentioned directory.

**• cat file1.txt file2.txt | sort | uniq –d**>> combines the content of the two files and sorts the

**• chmod 644 file.txt**>> changes the permissions of file.txt  
 6= owner has read and write.  
 4= group has read only.  
 4= others has read only.

**• cp -r source\_directory destination\_directory**>> It copies src directory and dest directory.

**• find /path/to/search -name "\*.txt"**   
>> Finds specific file with .txt extension.

**• chmod u+x file.txt**   
>> gives permission to user for execution.

• **echo $PATH**  
>> prints value saved to PATH.

* PART B
* **Identify True or False:**

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

**2. mv is used to move files and directories.**  
>>TRUE

**3. cd is used to copy files and directories.**>>FALSE

**4. pwd stands for "print working directory" and displays the current directory.**  
>>FALSE

**5. grep is used to search for patterns in files.**>>TRUE

**6. 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

**8. rm -rf file.txt deletes a file forcefully without confirmation.**   
>>TRUE

* **Identify the Incorrect Commands:**

**1. chmodx is used to change file permissions.**  
>> chmod

**2. cpy is used to copy files and directories.**>> cp

**3. mkfile is used to create a new file.**  
>> touch/nano

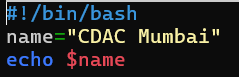
**4. catx is used to concatenate files.**   
>> cat

**5. rn is used to rename files.**>> mv

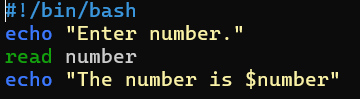
**Question 1: Write a shell script that prints "Hello, World!" to the terminal.  
>>**

**Script:**

**Output: **

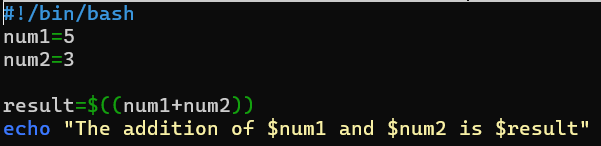
**Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.   
>>  
  
Script:**

****

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

**Script:**

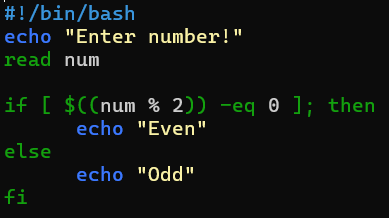
**Output :**

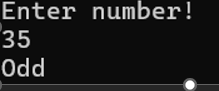
**Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.   
>>**

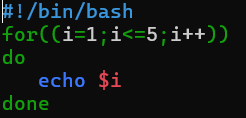
**Script :**

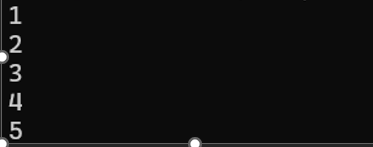
**Output:**

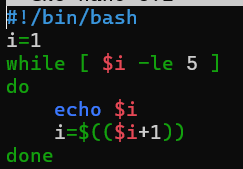
****

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

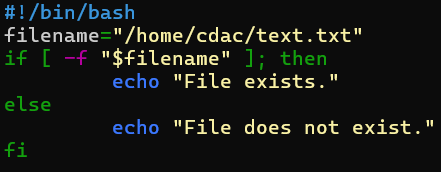
**Output:  
**

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

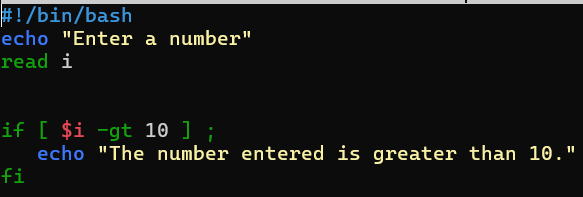
**Output:  
 **

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

**Output:**

**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".   
>>  
 Script:  
**

**Output:**

**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.   
>>  
Script:  
 **

**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)***

**1. What is an Operating System, and What are Its Primary Functions?**

* **Answer: 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.**
* **Book Reference: *Operating System Concepts* (Chapter 1 - Introduction)**

**2. Explain the Difference Between Process and Thread.**

* **Answer: A process is an instance of a program in execution, whereas a thread is a smaller unit of a process that can be scheduled and executed independently. Threads within the same process share resources like memory, while processes have separate memory spaces.**
* **Book Reference: *Operating System Concepts* (Chapter 4 - Threads and Chapter 3 - Processes)**

**3. What is Virtual Memory, and How Does It Work?**

* **Answer: Virtual memory is a memory management technique that gives an application the impression of a large contiguous block of memory while actually using smaller, fragmented physical memory and disk space. It uses techniques like paging and segmentation.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Virtual Memory)**

**4. Describe the Difference Between Multiprogramming, Multitasking, and Multiprocessing.**

* **Answer:**
  + **Multiprogramming: Running multiple programs simultaneously by managing their execution.**
  + **Multitasking: Extends multiprogramming by allowing multiple tasks to run concurrently on a single CPU.**
  + **Multiprocessing: Using multiple CPUs to execute multiple processes simultaneously.**
* **Book Reference: *Operating System Concepts* (Chapter 1 - Introduction)**

**5. What is a File System, and What are Its Components?**

* **Answer: A file system organizes and manages files and directories on storage devices. Its components include files, directories, file descriptors, and metadata. It also defines how data is stored and retrieved.**
* **Book Reference: *Operating System Concepts* (Chapter 11 - File System Implementation)**

**6. What is a Deadlock, and How Can It Be Prevented?**

* **Answer: A deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource that another process holds. Deadlock prevention can be done by ensuring that at least one of the necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, circular wait) is not met.**
* **Book Reference: *Operating System Concepts* (Chapter 7 - Deadlocks)**

**7. Explain the Difference Between a Kernel and a Shell.**

* **Answer: The kernel is the core part of an OS that directly interacts with hardware and manages resources. The shell is a user interface that allows users to interact with the kernel through commands.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 1 - Introduction to UNIX)**

**8. What is CPU Scheduling, and Why is It Important?**

* **Answer: CPU scheduling is the process of determining which processes will run on the CPU and for how long. It is crucial for ensuring efficient CPU utilization and system responsiveness.**
* **Book Reference: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**9. How Does a System Call Work?**

* **Answer: A system call provides an interface between a user program and the operating system. It allows user programs to request services from the OS, such as I/O operations or memory allocation.**
* **Book Reference: *Operating System Concepts* (Chapter 2 - System Structures)**

**10. What is the Purpose of Device Drivers in an Operating System?**

* **Answer: Device drivers act as intermediaries between the OS and hardware devices, translating OS instructions into device-specific commands.**
* **Book Reference: *Operating System Concepts* (Chapter 13 - I/O Systems)**

**11. Explain the Role of the Page Table in Virtual Memory Management.**

* **Answer: The page table is a data structure used in virtual memory systems to map virtual addresses to physical addresses. Each entry in the page table corresponds to a virtual page and contains the address of the corresponding physical page, along with other status information like access rights and whether the page is in memory or on disk.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Virtual Memory)**

**12. What is Thrashing, and How Can It Be Avoided?**

* **Answer: Thrashing occurs when a system spends more time swapping pages in and out of memory than executing actual processes, leading to severe performance degradation. It can be avoided by using techniques like working set model, adjusting the degree of multiprogramming, or by using better page replacement algorithms.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Virtual Memory)**

**13. Describe the Concept of a Semaphore and Its Use in Synchronization.**

* **Answer: A semaphore is a synchronization primitive used to control access to a common resource in concurrent programming. It can be used to solve problems like mutual exclusion and producer-consumer issues. Semaphores are counters that can be incremented or decremented atomically, ensuring that no two processes access the critical section simultaneously.**
* **Book Reference: *Operating System Concepts* (Chapter 6 - Synchronization Tools)**

**14. How Does an Operating System Handle Process Synchronization?**

* **Answer: Process synchronization is handled by using various synchronization primitives like semaphores, mutexes, and monitors. These tools help coordinate the order in which processes execute to prevent race conditions and ensure that shared resources are accessed in a controlled manner.**
* **Book Reference: *Operating System Concepts* (Chapter 6 - Synchronization Tools)**

**15. What is the Purpose of an Interrupt in Operating Systems?**

* **Answer: Interrupts allow the CPU to respond to events such as I/O operations or hardware malfunctions immediately, by pausing the current process, saving its state, and executing an interrupt service routine (ISR). This mechanism ensures efficient and responsive system operation.**
* **Book Reference: *Operating System Concepts* (Chapter 4 - CPU)**

**16. Explain the Concept of a File Descriptor.**

* **Answer: A file descriptor is a low-level handle assigned to a file or other I/O resource, such as a pipe or network socket, used by processes to access the resource. Each process maintains a table of file descriptors, which the OS uses to identify the open files.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 3 - UNIX File System)**

**17. How Does a System Recover from a System Crash?**

* **Answer: Recovery from a system crash involves several steps, such as checking and restoring file system integrity, rolling back incomplete transactions, and rebooting the system. Techniques like journaling in file systems help minimize data loss and ensure consistency.**
* **Book Reference: *Operating System Concepts* (Chapter 11 - File System Implementation)**

**18. Describe the Difference Between a Monolithic Kernel and a Microkernel.**

* **Answer:**
  + **Monolithic Kernel: Includes all the essential OS services like process management, memory management, and device drivers within a single large kernel.**
  + **Microkernel: Only includes the most essential functions in the kernel, like communication between processes and basic I/O operations, while other services run in user space, enhancing modularity and security.**
* **Book Reference: *Operating System Concepts* (Chapter 2 - Operating-System Structures)**

**19. What is the Difference Between Internal and External Fragmentation?**

* **Answer:**
  + **Internal Fragmentation: Occurs when allocated memory may be slightly larger than requested memory, leading to unused memory within allocated regions.**
  + **External Fragmentation: Occurs when free memory is scattered in small blocks across the system, making it difficult to allocate large contiguous blocks.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Memory Management)**

**20. How Does an Operating System Manage I/O Operations?**

* **Answer: The OS manages I/O operations by abstracting hardware specifics through device drivers, scheduling I/O tasks, and handling interrupts. The OS also provides interfaces to user programs for reading and writing data, ensuring efficient and fair access to I/O devices.**
* **Book Reference: *Operating System Concepts* (Chapter 13 - I/O Systems)**

**21. Explain the Difference Between Preemptive and Non-Preemptive Scheduling.**

* **Answer:**
  + **Preemptive Scheduling: Allows the OS to interrupt a running process and switch to another process, ensuring responsive multitasking.**
  + **Non-Preemptive Scheduling: Once a process starts running, it continues until it finishes or voluntarily yields control, reducing context switching overhead.**
* **Book Reference: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**22. What is Round-Robin Scheduling, and How Does It Work?**

* **Answer: Round-Robin scheduling is a preemptive CPU scheduling algorithm where each process is assigned a fixed time slice (quantum). The CPU cycles through all processes, assigning each one the CPU for a time slice in a cyclic order, ensuring fairness.**
* **Book Reference: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**23. Describe the Priority Scheduling Algorithm. How is Priority Assigned to Processes?**

* **Answer: Priority scheduling assigns the CPU to the process with the highest priority. Priorities can be assigned based on factors like process importance, required resources, or user-defined criteria. The algorithm can be preemptive or non-preemptive.**
* **Book Reference: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**24. What is the Shortest Job Next (SJN) Scheduling Algorithm, and When is It Used?**

* **Answer: SJN is a non-preemptive scheduling algorithm that selects the process with the smallest execution time to run next. It minimizes average waiting time but can lead to starvation of longer processes.**
* **Book Reference: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**25. Explain the Concept of Multilevel Queue Scheduling.**

* **Answer: Multilevel queue scheduling involves dividing the ready queue into several smaller queues, each with different priority levels or scheduling algorithms. Processes are permanently assigned to a queue based on some criteria (e.g., process type or priority).**
* **Book Reference: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**26. What is a Process Control Block (PCB), and What Information Does It Contain?**

* **Answer: A PCB is a data structure maintained by the OS that contains all the information about a process, including process state, program counter, CPU registers, memory limits, and I/O status. It is essential for context switching.**
* **Book Reference: *Operating System Concepts* (Chapter 3 - Processes)**

**27. Describe the Process State Diagram and the Transitions Between Different Process States.**

* **Answer: A process typically moves through states such as New, Ready, Running, Waiting, and Terminated. The OS manages transitions between these states based on events like process creation, I/O completion, and CPU scheduling.**
* **Book Reference: *Operating System Concepts* (Chapter 3 - Processes)**

**28. How Does a Process Communicate with Another Process in an Operating System?**

* **Answer: Processes communicate using Inter-Process Communication (IPC) mechanisms like message passing, shared memory, pipes, and signals. These mechanisms allow data exchange and synchronization between processes.**
* **Book Reference: *Operating System Concepts* (Chapter 3 - Processes and Chapter 4 - Threads)**

**29. What is Process Synchronization, and Why is It Important?**

* **Answer: Process synchronization ensures that multiple processes or threads can execute concurrently without causing data inconsistency or race conditions. It is crucial for maintaining data integrity and coordination between processes.**
* **Book Reference: *Operating System Concepts* (Chapter 6 - Synchronization Tools)**

**30. Explain the Concept of a Zombie Process and How It Is Created.**

* **Answer: A zombie process is a process that has completed execution but still has an entry in the process table. It is created when a child process terminates, but its parent has not yet read its exit status using wait(), leaving the process descriptor in a defunct state.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**31. Describe the Difference Between Internal Fragmentation and External Fragmentation.**

* **Answer:**
  + **Internal Fragmentation: Wasted space within allocated memory blocks.**
  + **External Fragmentation: Wasted space between allocated memory blocks that cannot be used for new allocations due to small size.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Memory Management)**

**32. What is Demand Paging, and How Does It Improve Memory Management Efficiency?**

* **Answer: Demand paging loads pages into memory only when they are needed, reducing memory usage and allowing more processes to execute.**

**33. Explain the Role of the Page Table in Virtual Memory Management.**

* **Answer: The page table plays a crucial role in virtual memory management by mapping virtual addresses to physical addresses. Each entry in the page table holds the frame number that corresponds to a page in memory. The page table allows the operating system to translate virtual addresses generated by the CPU into physical addresses used by the hardware. This mapping is essential for implementing paging, which enables efficient memory utilization.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Virtual Memory)**

**34. How Does a Memory Management Unit (MMU) Work?**

* **Answer: The Memory Management Unit (MMU) is a hardware component responsible for handling virtual to physical address translation. It uses the page table to translate virtual addresses to physical addresses. The MMU checks the page table entry for the corresponding page and retrieves the physical frame number to complete the address translation. The MMU also plays a role in enforcing access controls and managing page faults.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Virtual Memory)**

**35. What is Thrashing, and How Can It Be Avoided in Virtual Memory Systems?**

* **Answer: Thrashing occurs when a system spends more time paging (swapping pages in and out of memory) than executing actual processes. It can significantly degrade system performance. Thrashing can be avoided by using working set models to keep track of actively used pages, reducing the degree of multiprogramming, or employing better page replacement algorithms to minimize unnecessary page swaps.**
* **Book Reference: *Operating System Concepts* (Chapter 8 - Virtual Memory)**

**36. What is a System Call, and How Does It Facilitate Communication Between User Programs and the Operating System?**

* **Answer: A system call is a mechanism that allows user-level programs to request services from the operating system. When a program needs to perform an operation that requires OS intervention, such as file access or process control, it makes a system call. The OS then executes the requested service and returns the result to the program. System calls provide a controlled interface between user programs and kernel functions.**
* **Book Reference: *Operating System Concepts* (Chapter 2 - Operating-System Structures)**

**37. Describe the Difference Between a Monolithic Kernel and a Microkernel.**

* **Answer:**
  + **Monolithic Kernel: Includes all operating system services within a single large kernel. This design is fast and efficient but can be less secure and more challenging to maintain.**
  + **Microkernel: Contains only the essential functions like inter-process communication (IPC) and basic I/O. Other services run in user space, which makes the system more modular and secure but can introduce performance overhead.**
* **Book Reference: *Operating System Concepts* (Chapter 2 - Operating-System Structures)**

**38. How Does an Operating System Handle I/O Operations?**

* **Answer: The operating system handles I/O operations by abstracting hardware details through device drivers, managing I/O buffers, scheduling I/O tasks, and handling interrupts. It provides a uniform interface to user applications, allowing them to perform I/O operations without needing to know the underlying hardware specifics. The OS also manages the asynchronous nature of I/O operations, ensuring data integrity and efficient processing.**
* **Book Reference: *Operating System Concepts* (Chapter 13 - I/O Systems)**

**39. Explain the Concept of a Race Condition and How It Can Be Prevented.**

* **Answer: A race condition occurs when the behavior of software or a system depends on the sequence or timing of uncontrollable events, such as concurrent execution of threads or processes. This can lead to unpredictable outcomes. Race conditions can be prevented by using synchronization mechanisms like locks, semaphores, and monitors to control the execution order of critical sections in the code.**
* **Book Reference: *Operating System Concepts* (Chapter 6 - Synchronization Tools)**

**40. Describe the Role of Device Drivers in an Operating System.**

* **Answer: Device drivers are specialized software modules that allow the operating system to communicate with hardware devices. They provide a standard interface for the OS to interact with a wide variety of devices, such as disk drives, printers, and network cards, without requiring the OS to know the specifics of the hardware. The driver translates OS commands into device-specific instructions and manages device-specific operations.**
* **Book Reference: *Operating System Concepts* (Chapter 13 - I/O Systems)**

**41. What is a Zombie Process, and How Does It Occur? How Can a Zombie Process Be Prevented?**

* **Answer: 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 using the wait() system call. This leaves the process in a "defunct" state. Zombie processes can be prevented by ensuring that the parent process calls wait() to clean up the process table entry or by using a signal handler to automatically reap terminated child processes.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**42. Explain the Concept of an Orphan Process. How Does an Operating System Handle Orphan Processes?**

* **Answer: An orphan process is a process whose parent has terminated, leaving it without a parent process. The operating system typically reassigns orphan processes to the init process (process ID 1), which adopts them and ensures they are properly reaped when they terminate.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**43. What is the Relationship Between a Parent Process and a Child Process in the Context of Process Management?**

* **Answer: In process management, a parent process is one that creates a child process using system calls like fork(). The child process is a duplicate of the parent but can execute different code. The parent and child processes can communicate through IPC mechanisms, and the parent can wait for the child process to terminate using wait() or similar system calls.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**44. How Does the fork() System Call Work in Creating a New Process in Unix-like Operating Systems?**

* **Answer: The fork() system call creates a new process by duplicating the calling process. The new process, called the child process, receives a copy of the parent's memory, file descriptors, and other resources. Both the parent and child processes continue execution from the point where fork() was called, but fork() returns a different value in each process to distinguish them.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**45. Describe How a Parent Process Can Wait for a Child Process to Finish Execution.**

* **Answer: A parent process can wait for a child process to finish execution by using the wait() or waitpid() system calls. These calls make the parent process block until the child process exits, at which point the parent can retrieve the child's exit status. This allows the parent to ensure that it doesn't proceed until the child has completed its task.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**46. What is the Significance of the Exit Status of a Child Process in the wait() System Call?**

* **Answer: The exit status of a child process, retrieved through the wait() system call, indicates how the child process terminated. It provides information about whether the process completed successfully, was terminated by a signal, or encountered an error. This status allows the parent process to take appropriate action based on the outcome of the child's execution.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**47. How Can a Parent Process Terminate a Child Process in Unix-like Operating Systems?**

* **Answer: A parent process can terminate a child process by sending it a signal using the kill() system call. The signal can be a termination signal like SIGTERM, which requests the child to terminate gracefully, or SIGKILL, which forces an immediate termination. This allows the parent to control the lifecycle of its child processes.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**48. Explain the Difference Between a Process Group and a Session in Unix-like Operating Systems.**

* **Answer:**
  + **Process Group: A collection of one or more processes that can be managed together. Signals can be sent to the entire group.**
  + **Session: A collection of process groups that are managed as a unit, typically associated with a single login session. The session leader is the process that created the session.**
* **Book Reference: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**49. Describe How the exec() Family of Functions is Used to Replace the Current Process Image with a New One.**

* **Answer: The exec() family of functions replaces the current process image with a new program. When a process calls exec(), the operating system loads the new program into the process's memory, discarding the old one, and starts execution from the entry point of the new program.**

**50. What is the Purpose of the waitpid() System Call in Process Management? How Does It Differ from wait()?**

* **Answer: The waitpid() system call allows a parent process to wait for a specific child process to terminate, rather than waiting for any child process as with the wait() system call. waitpid() provides more control as it can be used with different options to wait for a child process that matches a specific process ID, any child in the process group, or to avoid blocking the parent process if no child has terminated yet. This makes waitpid() more versatile for managing multiple child processes.**
* **Book reference to: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**51. How Does Process Termination Occur in Unix-like Operating Systems?**

* **Answer: In Unix-like operating systems, a process can terminate in several ways: it can exit normally by calling the exit() function, be terminated by a signal, or be killed by another process. When a process terminates, it releases all its resources, such as memory and open files, and sends an exit status to its parent process. If the parent does not immediately collect this status using wait() or waitpid(), the terminated process becomes a zombie.**
* **Book reference to: *Unix Concepts and Applications* (Chapter 5 - Process Control)**

**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?**

* **Answer: The long-term scheduler, also known as the job scheduler, determines which processes are admitted into the system for processing. It controls the degree of multiprogramming, which is the number of processes in memory at a given time. By managing the admission of processes, the long-term scheduler ensures that the system remains balanced and that resources are optimally used without overloading the CPU or memory.**
* **Book reference to: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

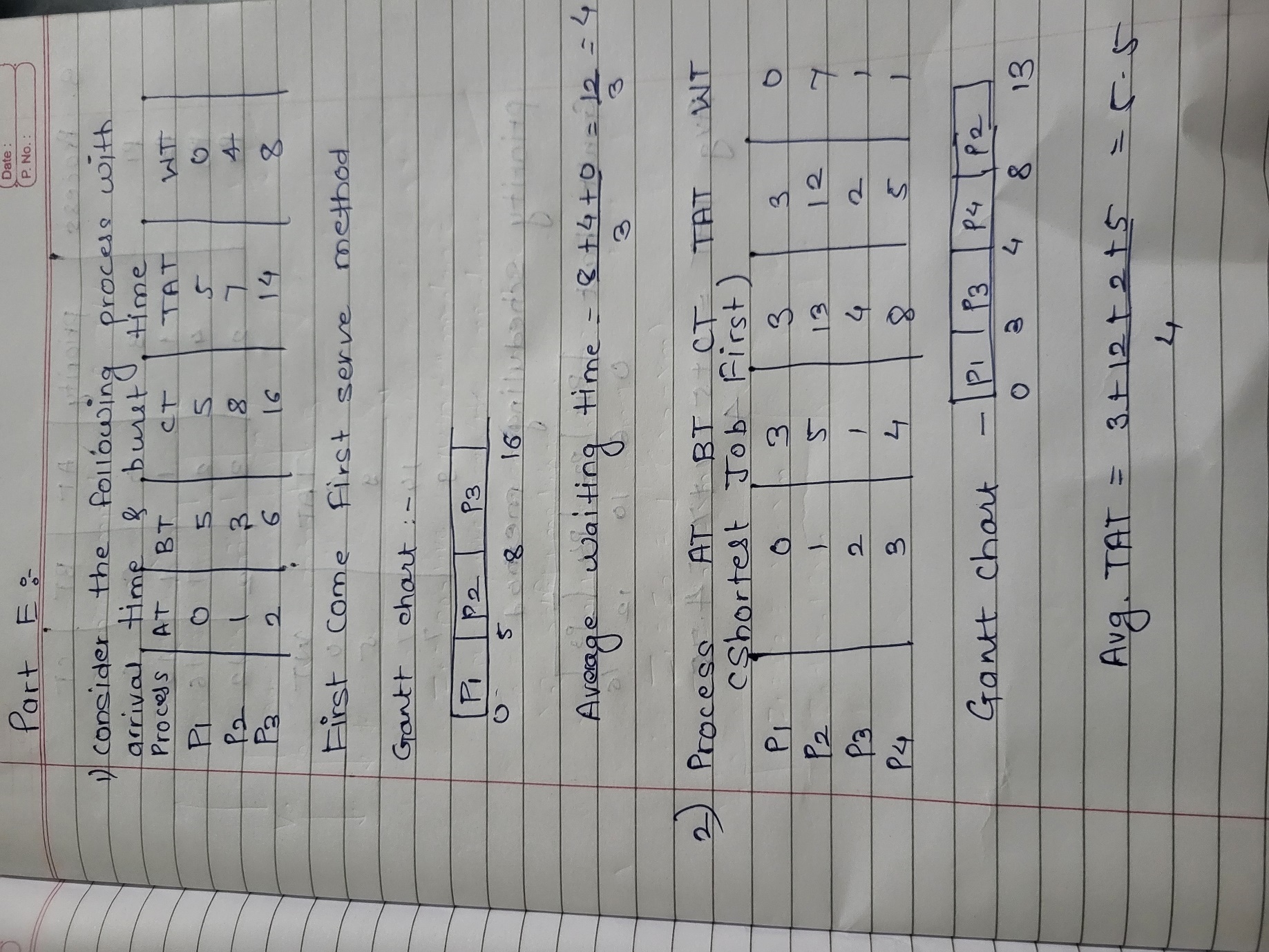
**53. How Does the Short-term Scheduler Differ from the Long-term and Medium-term Schedulers in Terms of Frequency of Execution and the Scope of Its Decisions?**

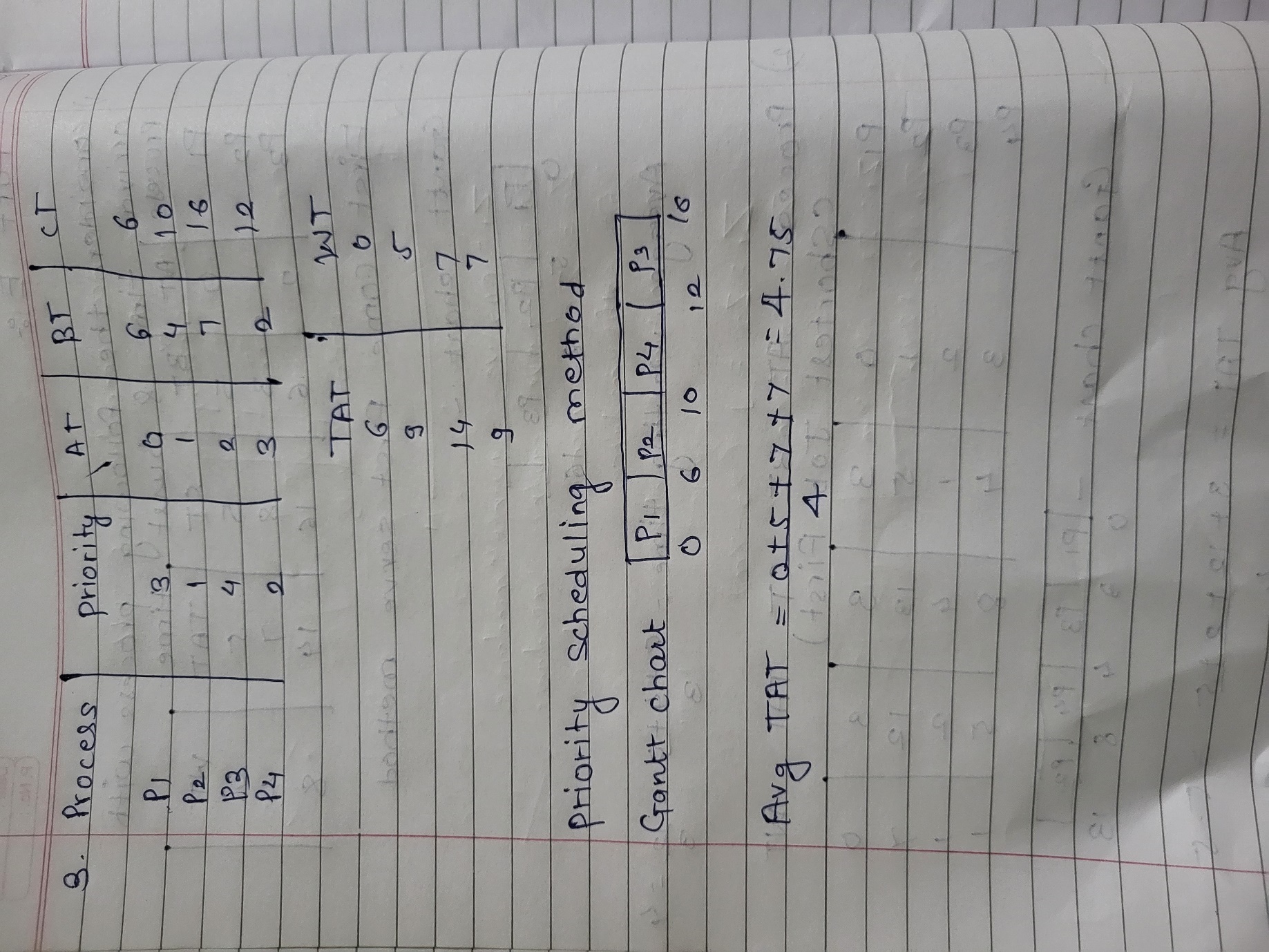
* **Answer: The short-term scheduler, also known as the CPU scheduler, is responsible for selecting which process will run next on the CPU. It operates frequently, making decisions every few milliseconds, and focuses on optimizing CPU utilization. In contrast, the long-term scheduler operates less frequently and decides which processes should be brought into memory, while the medium-term scheduler temporarily removes processes from memory to manage the level of multiprogramming and reduce system load.**
* **Book reference to: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

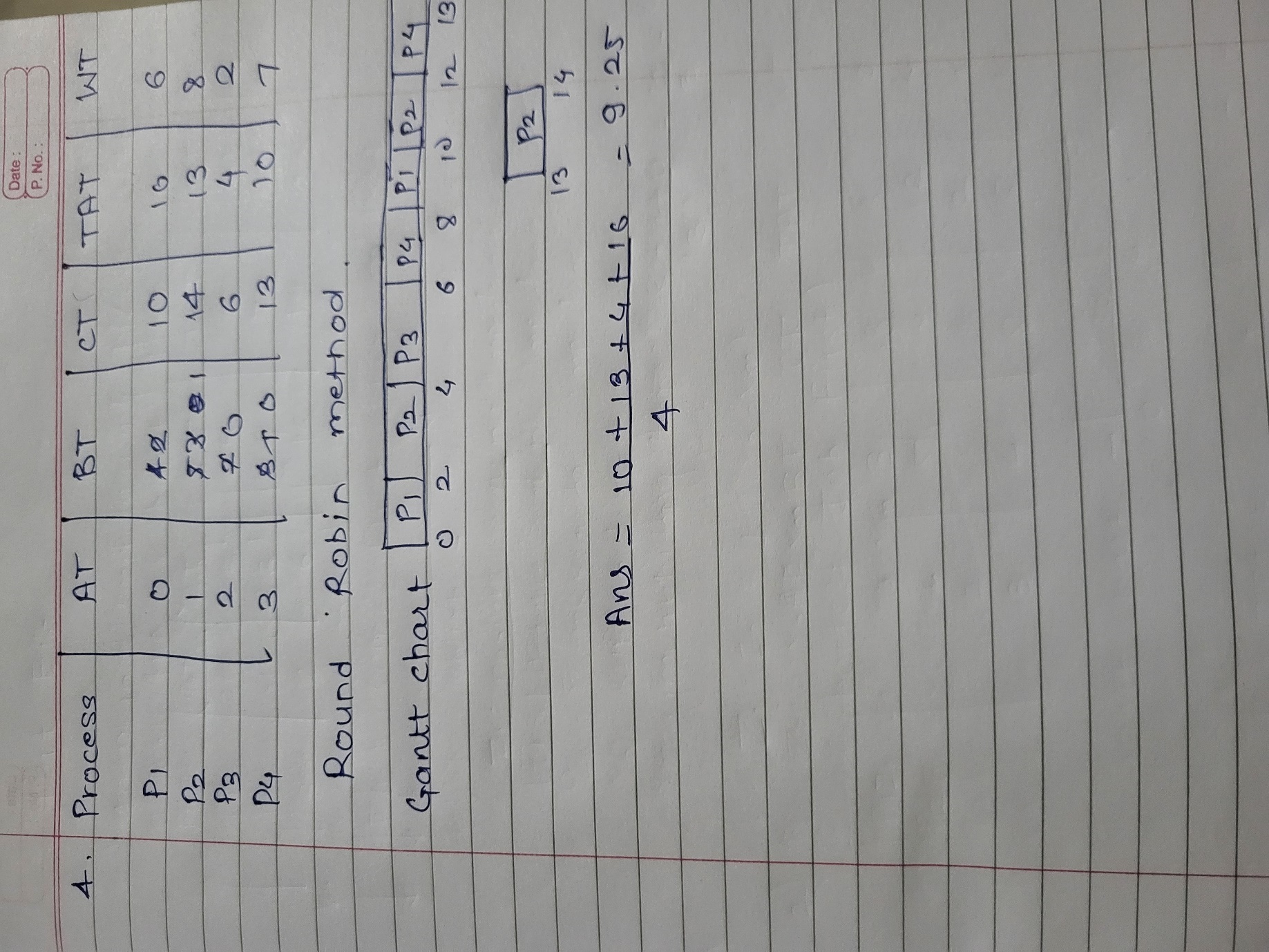
**54. Describe a Scenario Where the Medium-term Scheduler Would Be Invoked and Explain How It Helps Manage System Resources More Efficiently.**

* **Answer: The medium-term scheduler is invoked when the system is experiencing high load or when memory is overcommitted. It swaps out processes from memory to disk (a process known as "swapping") to reduce the degree of multiprogramming temporarily. This allows the system to free up memory and allocate resources more effectively to processes that are currently active, thus preventing thrashing and improving overall system performance.**
* **Book reference to: *Operating System Concepts* (Chapter 5 - CPU Scheduling)**

**Part E**

****

****

****