CDAC MUMBAI

Concepts of Operating System Assignment 2

# Part A

1. **echo "Hello, World!"**  
   Prints "Hello, World!" to the terminal.
2. **name="Productive"**  
   Assigns the string "Productive" to the variable name.
3. **touch file.txt**  
   Creates an empty file named file.txt if it doesn't already exist.
4. **ls -a**  
   Lists all files and directories in the current directory, including hidden ones (those starting with a dot).
5. **rm file.txt**  
   Deletes the file named file.txt.
6. **cp file1.txt file2.txt**  
   Copies the content of file1.txt to file2.txt. If file2.txt does not exist, it will be created.
7. **mv file.txt /path/to/directory/**  
   Moves file.txt to the specified directory.
8. **chmod 755 script.sh**  
   Sets the permissions of script.sh to allow the owner to read, write, and execute; others can read and execute.
9. **grep "pattern" file.txt**  
   Searches for the specified pattern in file.txt and displays the matching lines.
10. **kill PID**  
    Terminates the process with the specified Process ID (PID).
11. **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**  
    Creates a directory named mydir, changes into it, creates file.txt, writes "Hello, World!" to it, and displays the content.
12. **ls -l | grep ".txt"**  
    Lists all files with detailed information and filters the output to show only .txt files.
13. **cat file1.txt file2.txt | sort | uniq**  
    Concatenates file1.txt and file2.txt, sorts the lines, and removes duplicates.
14. **ls -l | grep "^d"**  
    Lists detailed information of files and directories, filtering to show only directories.
15. **grep -r "pattern" /path/to/directory/**  
    Recursively searches for the specified pattern in all files within the specified directory.
16. **cat file1.txt file2.txt | sort | uniq -d**  
    Concatenates file1.txt and file2.txt, sorts the lines, and displays only duplicate lines.
17. **chmod 644 file.txt**  
    Sets permissions on file.txt to allow the owner to read and write, while others can only read.
18. **cp -r source\_directory destination\_directory**  
    Recursively copies source\_directory and its contents to destination\_directory.
19. **find /path/to/search -name "\*.txt"**  
    Searches for all files ending with .txt in the specified path.
20. **chmod u+x file.txt**  
    Grants execute permission to the owner of file.txt.
21. **echo $PATH**  
    Displays the current system path environment variable, which lists directories where executable files are located.

**PART B**

**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 change directories, not copy files.- **False**
4. pwd stands for "print working directory" and displays the current directory. **-True**
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 if they do not exist. **-True**
8. rm -rf file.txt deletes a file forcefully without confirmation. **-True**

**Incorrect Commands**

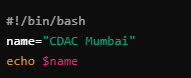
1. Incorrect, the correct command is chmod. **chmodx**
2. Incorrect, the correct command is cp. **cpy**
3. Incorrect, the correct command is touch **mkfile**
4. Incorrect, the correct command is cat. **catx**
5. Incorrect, the correct command is mv for renaming files. **rn**

# Part C

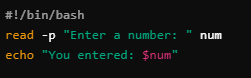
Question 1: Write a shell script that prints "Hello, World!" to the terminal.



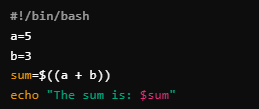
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.



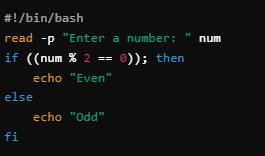
Question 3: Write a shell script that takes a number as input from the user and prints it.



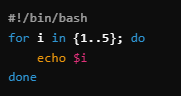
Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.



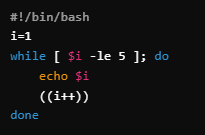
Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".



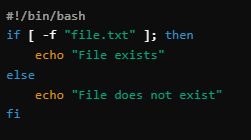
Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.



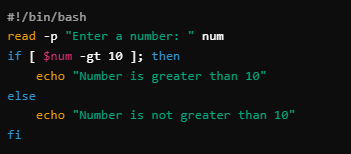
Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.



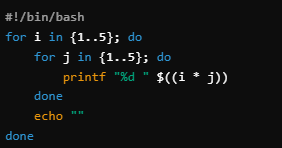
Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".



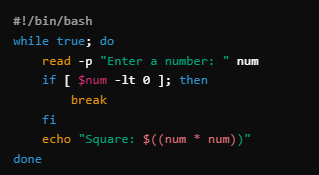
Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.



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)

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

An operating system (OS) is a software program that manages a computer's hardware and software resources. It provides a user interface, handles input/output operations, manages memory, processes, files, and devices, and ensures efficient resource allocation.

**2. Explain the difference between process and thread.**

A **process** is a program in execution, including its memory space, code, data, and resources. A **thread** is a lightweight unit of execution within a process, sharing the process's resources but having its own program counter, stack, and registers.

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

Virtual memory is a technique that allows processes to use more memory than physically available. It maps logical addresses to physical addresses using a page table. When a process references a memory location, the MMU checks the page table to determine if the page is in memory. If not, a page fault occurs, and the operating system loads the required page from disk.

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

* **Multiprogramming:** Multiple processes are loaded into memory simultaneously, but only one process executes at a time.
* **Multitasking:** Multiple processes are executed concurrently, with the OS switching between them to give the illusion of simultaneous execution.
* **Multiprocessing:** Multiple CPUs or cores are used to execute multiple processes simultaneously.

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

A file system is a method for organizing and storing data on a storage device. It consists of:

* **File:** A collection of related data.
* **Directory:** A container for files and other directories.
* **Metadata:** Information about files, such as name, size, creation time, and permissions.

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

A deadlock occurs when two or more processes are waiting for resources held by each other. It can be prevented using:

* **Mutual exclusion:** Only one process can access a resource at a time.
* **Hold and wait:** A process cannot request a resource while holding another.
* **No preemption:** A resource cannot be taken away from a process until it releases it voluntarily.
* **Circular wait:** Processes should not wait for resources in a circular chain.

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

The **kernel** is the core of the operating system, responsible for managing hardware and software resources. The **shell** is a user interface program that provides a command-line interface for interacting with the kernel.

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

CPU scheduling is the process of determining which process should be executed next. It is important for efficient resource utilization and responsiveness.

**9. How does a system call work?**

A system call is a mechanism for a process to request services from the kernel. It involves a software interrupt that transfers control to the kernel, which executes the requested service and returns the result to the process.

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

Device drivers are software programs that provide an interface between the operating system and hardware devices. They handle the specific details of communicating with different devices.

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

The page table maps logical addresses to physical addresses in virtual memory. It contains information about the location of each page in physical memory or on disk.

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

Thrashing occurs when the system spends most of its time swapping pages between memory and disk. It can be avoided by increasing the amount of physical memory or using memory management techniques like demand paging.

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

A semaphore is a synchronization mechanism that allows processes to signal each other. It has a value that can be incremented or decremented, and processes can wait on or signal a semaphore. Semaphores are used to implement mutual exclusion and synchronization between processes.

**14. How does an operating system handle process synchronization?**

An operating system uses synchronization mechanisms like semaphores, mutexes, and monitors to coordinate the activities of multiple processes and prevent race conditions.

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

An interrupt is a signal that causes the CPU to stop its current task and execute an interrupt handler. Interrupts are used to handle hardware events like I/O completion, timer expiration, and exceptions.

**16. Explain the concept of a file descriptor.**

A file descriptor is an integer that represents an open file in an operating system. It is used to refer to files in system calls.

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

A system can recover from a crash by using techniques like journaling, checkpoints, and redundancy. Journaling records changes to files before they are written to disk, allowing them to be recovered in case of a crash. Checkpoints are periodic snapshots of the system state that can be used to restore the system to a consistent state. Redundancy involves duplicating data and resources to ensure that they can be recovered if one fails.

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

A **monolithic kernel** is a single, large executable that contains all the system's components. A **microkernel** is a small kernel that only provides essential services, with most of the system's functionality implemented in user-space servers.

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

* **Internal fragmentation:** Unused space within a memory allocation unit.
* **External fragmentation:** Unused space between allocated units.

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

An operating system uses device drivers to manage I/O operations. It can also use techniques like buffering and caching to improve I/O performance.

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

* **Preemptive scheduling:** The OS can interrupt a running process and switch to another at any time.
* **Non-preemptive scheduling:** A process runs to completion unless it voluntarily releases the CPU.

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

Round-robin scheduling is a preemptive scheduling algorithm that assigns each process a fixed time quantum. Each process runs for its time quantum, and then the CPU is switched to the next process in the queue.

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

Priority scheduling is a non-preemptive algorithm that assigns a priority to each process. The process with the highest priority is executed first. Priority can be assigned based on various factors, such as process type, importance, and resource requirements.

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

SJN is a non-preemptive algorithm that executes the process with the shortest estimated running time next. It is often used in batch systems where the estimated running times are known in advance.

**25. Explain the concept of multilevel queue scheduling.**

Multilevel queue scheduling divides processes into multiple queues based on priority or other criteria. Each queue has its own scheduling algorithm.

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

A PCB is a data structure that contains information about a process, such as its state, program counter, registers, memory allocation, and open files.

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

The process state diagram shows the different states a process can be in, including:

* **New:** Process has been created but not yet ready to run.
* **Ready:** Process is waiting to be executed.
* **Running:** Process is currently executing.
* **Waiting:** Process is waiting for an event (e.g., I/O completion).
* **Blocked:** Process is waiting for a resource.
* **Terminated:** Process has finished execution.

Transitions between states can occur due to various events, such as scheduling decisions, I/O completion, and system calls.

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

Processes can communicate with each other using various mechanisms, including:

* **Pipes:** Unidirectional channels for communication between related processes.
* **Message passing:** Processes exchange messages using system calls.
* **Shared memory:** Processes share a region of memory for communication.
* **Sockets:** Network-based communication between processes on different machines.

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

Process synchronization is the coordination of the activities of multiple processes to ensure that they execute in a predictable and consistent manner. It is important to prevent race conditions and ensure correct program execution.

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

A zombie process is a process that has terminated but its parent process has not yet waited for it. It is created when a child process terminates before the parent process calls wait() or waitpid().

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

* **Internal fragmentation:** Unused space within a memory allocation unit.
* **External fragmentation:** Unused space between allocated units.

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

**Demand paging** is a virtual memory technique that loads pages into physical memory only when they are needed. This improves memory management efficiency by reducing the amount of physical memory required for processes and minimizing the overhead of swapping pages between memory and disk.

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

The **page table** is a data structure that maps virtual addresses to physical addresses. It contains information about the location of each page in physical memory or on disk. When a process references a memory location, the MMU uses the page table to determine the corresponding physical address.

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

The **MMU** is a hardware component that translates virtual addresses to physical addresses. It uses the page table to look up the corresponding physical address for each virtual address. If the page is not in memory, a page fault occurs and the MMU signals the operating system to load the page from disk.

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

**Thrashing** occurs when the system spends most of its time swapping pages between memory and disk. It can be avoided by:

* **Increasing the amount of physical memory:** This reduces the need for swapping.
* **Using a better page replacement algorithm:** A good algorithm can minimize the number of page faults.
* **Adjusting the working set size:** The working set is the set of pages that a process is actively using. By adjusting the working set size, the operating system can reduce the number of page faults.

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

A **system call** is a mechanism for a process to request services from the kernel. It involves a software interrupt that transfers control to the kernel, which executes the requested service and returns the result to the process. System calls provide a well-defined interface for user programs to interact with the operating system.

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

* **Monolithic kernel:** A single, large executable that contains all the system's components.
* **Microkernel:** A small kernel that only provides essential services, with most of the system's functionality implemented in user-space servers.

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

An operating system uses **device drivers** to manage I/O operations. It can also use techniques like **buffering** and **caching** to improve I/O performance.

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

A **race condition** occurs when two or more processes access shared data simultaneously and the outcome depends on the order in which their operations are executed. Race conditions can be prevented using **synchronization mechanisms** like semaphores, mutexes, and monitors.

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

**Device drivers** are software programs that provide an interface between the operating system and hardware devices. They handle the specific details of communicating with different devices.

**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 terminated but its parent process has not yet waited for it. It occurs when a child process terminates before the parent process calls wait() or waitpid(). Zombie processes can be prevented by the parent process calling wait() or waitpid() promptly after creating a child process.

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

An **orphan process** is a process whose parent process has terminated. Orphan processes are typically adopted by the init process, which becomes their new parent. The init process is responsible for cleaning up orphan processes.

**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 copy of the parent process, but it has its own PID and memory space. The parent process can wait for the child process to finish execution or terminate it.

**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 that is an exact copy of the parent process. The child process has its own PID, but it shares the same memory space as the parent process. The return value of fork() is 0 in the child process and the PID of the child process in the parent process.

**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 using the wait() or waitpid() system calls. These functions return the exit status of the child process.

**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 is a value that indicates whether the child process terminated successfully or encountered an error. The parent process can use the exit status to determine the outcome of the child process.

**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. The kill() system call takes the PID of the child process as an argument and sends a signal to it. The signal can be a termination signal, such as SIGTERM or SIGKILL.

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

* **Process group:** A collection of related processes that share a common process group ID (PGID).
* **Session:** A collection of processes that share a common session ID (SID). A session is typically created when a user logs in to a system.

**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 is used to replace the current process image with a new one. This means that the running program is replaced with a new program. The exec() functions take the name of the new program and its arguments as input.

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

The waitpid() system call is similar to wait(), but it allows the parent process to specify which child process to wait for. This is useful when a parent process has multiple child processes.

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

A process can terminate in several ways:

* **Normally:** The process completes its execution and calls the exit() system call.
* **Abnormally:** The process is terminated by a signal (e.g., SIGKILL, SIGTERM).
* **By the operating system:** The operating system can terminate a process if it violates system policies or becomes unresponsive.

**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** is responsible for selecting processes from the job queue and loading them into memory for execution. It influences the degree of multiprogramming by determining how many processes are ready to run at any given time.

**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?**

* **Short-term scheduler:** Executes frequently and makes decisions about which process to run next.
* **Medium-term scheduler:** Executes less frequently and makes decisions about swapping processes between memory and disk.
* **Long-term scheduler:** Executes infrequently and makes decisions about which processes to load into memory.

**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 would be invoked when the system is experiencing high memory utilization or when there are many processes waiting to be executed. It can help manage system resources more efficiently by swapping out processes that are not actively using the CPU or that are using a large amount of memory. This can free up resources for other processes and improve overall system performance.

# Part E

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

| Process | Arrival Time | Burst Time |

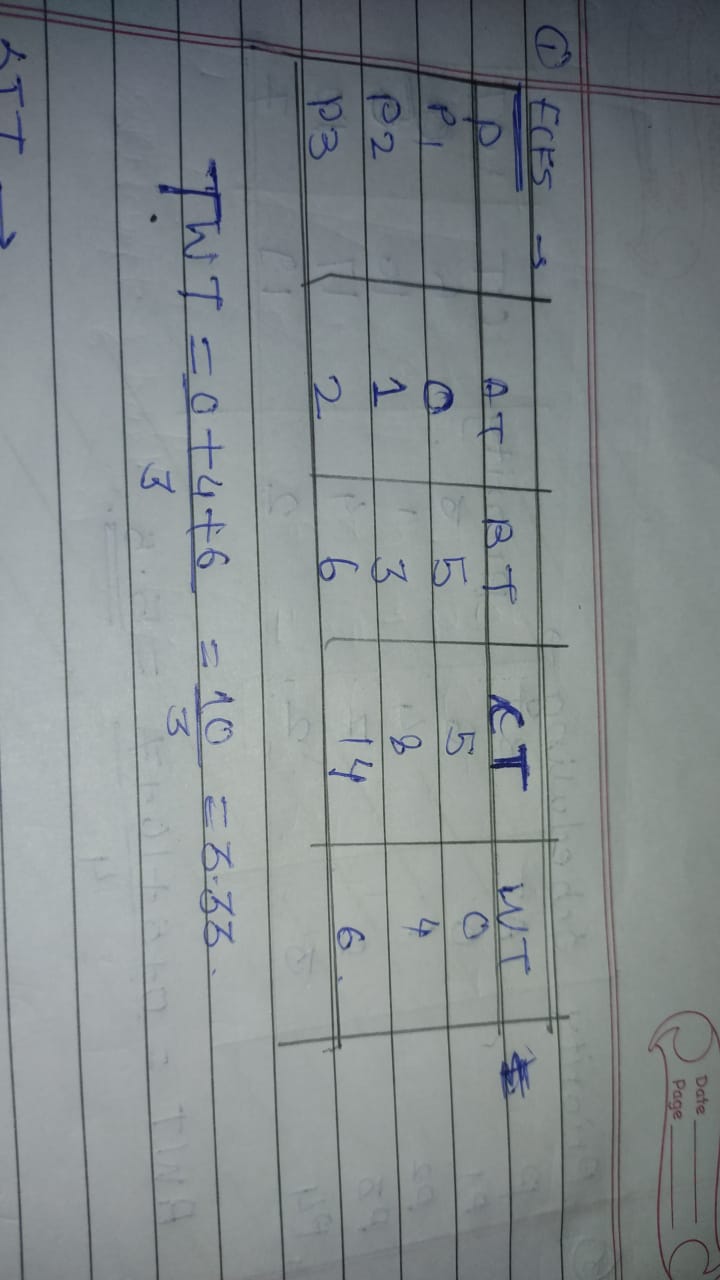
|---------|--------------|------------|

| P1 | 0 | 5 |

| P2 | 1 | 3 |

| P3 | 2 | 6 |

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



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

| Process | Arrival Time | Burst Time |

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

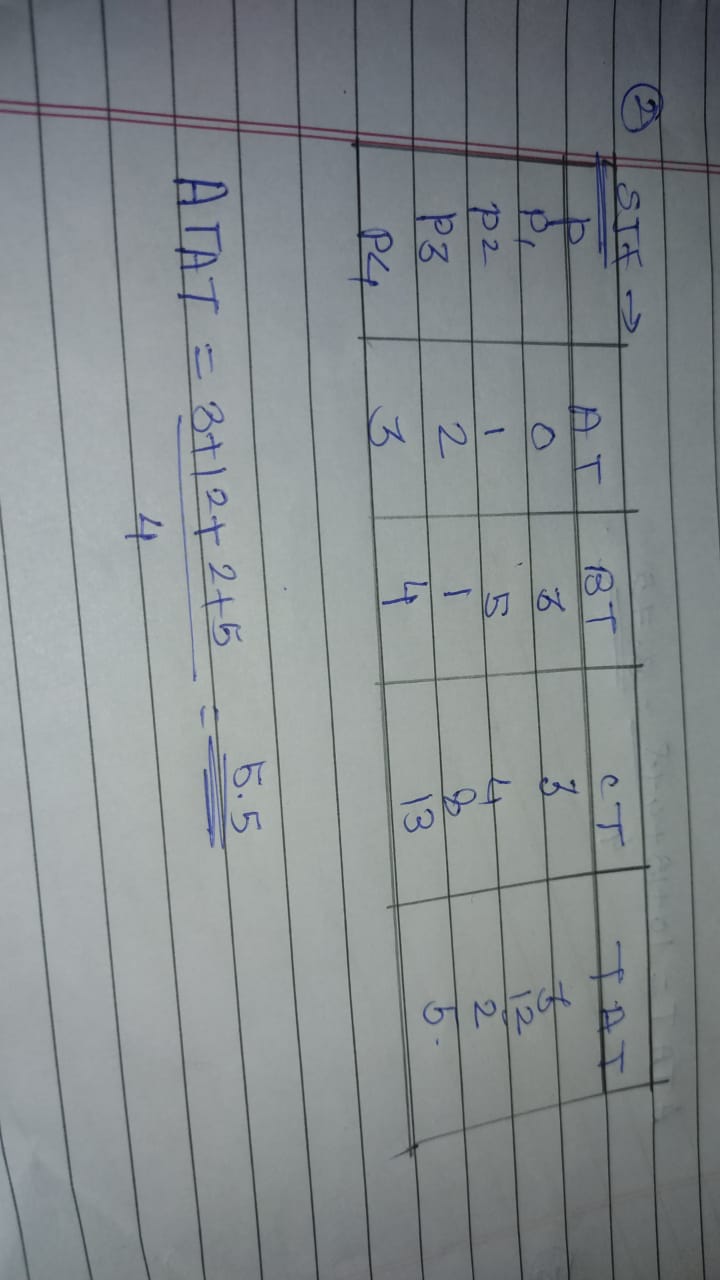
| P1 | 0 | 3 |

| P2 | 1 | 5 |

| P3 | 2 | 1 |

| P4 | 3 | 4 |

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



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

| Process | Arrival Time | Burst Time | Priority |

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

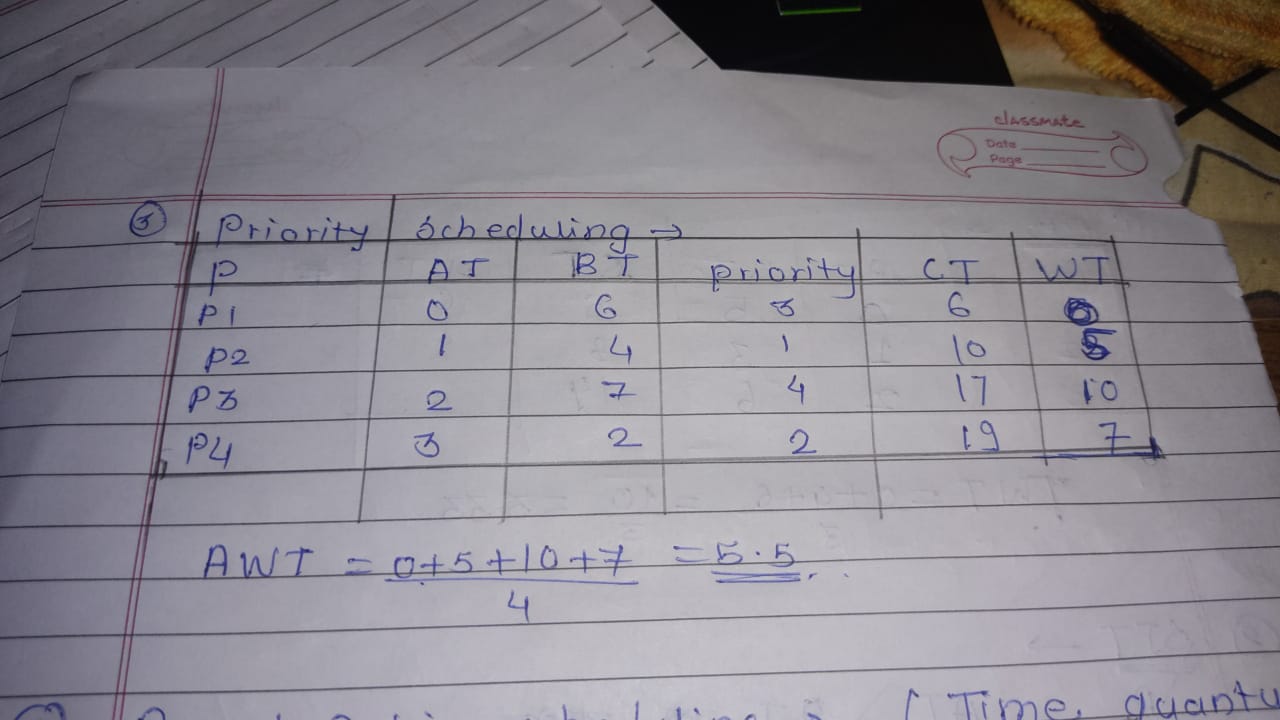
| P1 | 0 | 6 | 3 |

| P2 | 1 | 4 | 1 |

| P3 | 2 | 7 | 4 |

| P4 | 3 | 2 | 2 |

Calculate the average waiting time using Priority Scheduling.



1. 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 |

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

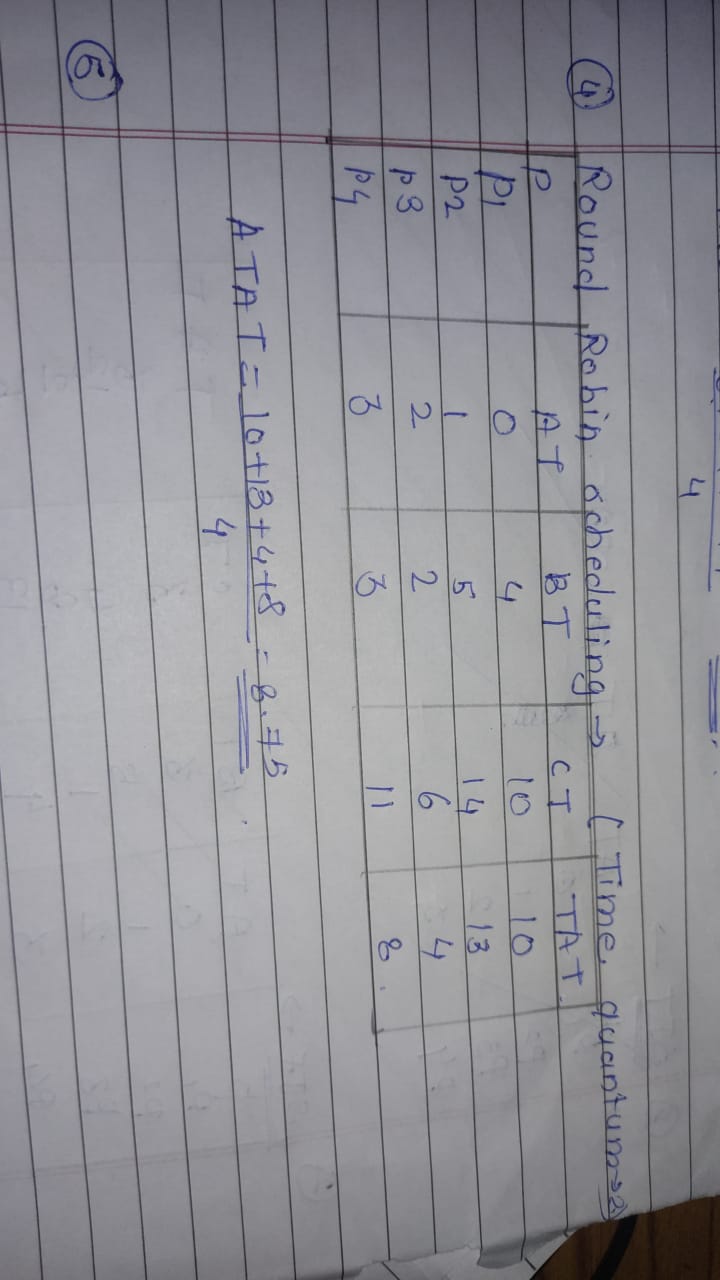
| P1 | 0 | 4 |

| P2 | 1 | 5 |

| P3 | 2 | 2 |

| P4 | 3 | 3 |

Calculate the average turnaround time using Round Robin scheduling.



1. 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?

#include <stdio.h>

void main() {

int x = 5;

fork();

x = x+1;

printf("x = %d\n",x);

}