Name:-Nilesh Fate_JH

Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

- echo "Hello, World!" :-Print On Terminnal

```
cdac@LAPTOP-MLEETRT5:~$ echo "Hello, World!"
Hello, World!
cdac@LAPTOP-MLEETRT5:~$
```

- name="Productive" :- Assigns the string to a variable named name.
- touch file.txt :-create the file
- ls -a :- Lists all files and directories

```
cdac@LAPTOP-MLEETRT5:~$ ls -a
.   .bash_history  .bashrc  .lesshst  .motd_shown  .sudo_as_admin_successful  docs.txt   que1.sh  que2.shy
..  .bash_logout   .cache   .local    .profile     LinuxAssignment.txt        fruit.txt  que2.sh
cdac@LAPTOP-MLEETRT5:~$
```

- rm file.txt :-remove the directory
- cp file1.txt file2.txt :-copy file1.txt into file2.txt
- mv file.txt /path/to/directory/ :-move a file1.txt into directory
- chmod 755 script.sh :-change access of the file it will u a g

```
cdac@LAPTOP-MLEETRT5:~$ chmod 755 docs.txt
cdac@LAPTOP-MLEETRT5:~$ ls -l
total 24
drwxr-xr-x 4 cdac cdac 4096 Aug 29 08:24 LinuxAssignment.txt
-rwxr-xr-x 1 cdac cdac  108 Aug 29 18:29 docs.txt
-rw-r--r-- 1 cdac cdac   64 Aug 29 18:24 fruit.txt
-rw-r--r-- 1 cdac cdac   19 Aug 30 14:59 que1.sh
-rw-r--r-- 1 cdac cdac  211 Aug 30 21:01 que2.sh
-rw-r--r-- 1 cdac cdac  126 Aug 30 16:19 que2.shy
```

- grep "pattern" file.txt :-search the specific word in whole file

```
cdac@LAPTOP-MLEETRT5:~$ grep "h" docs.txt
hello world
hello world
hello world
welcome to the club
cdac@LAPTOP-MLEETRT5:~$
```

• kill PID :- Terminates the process with the specified Process ID

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

:- Creates a directory named mydir, navigates into it, creates a file named file.txt, writes "Hello, World!" into the file, and then displays the contents of file.txt.

```
cdac@LAPTOP-MLEETRT5:~$ mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
Hello, World!
cdac@LAPTOP-MLEETRT5:~/mydir$
```

• ls -l | grep ".txt" :-show the directory list and find the extension “.txt”

```
cdac@LAPTOP-MLEETRT5:~/mydir$ ls -l | grep  ".txt"
-rw-r--r-- 1 cdac cdac 14 Aug 30 21:13 file.txt
cdac@LAPTOP-MLEETRT5:~/mydir$
```

• cat file1.txt file2.txt | sort | uniq :- Concatenates the contents of file1.txt and file2.txt, sorts the combined output, and then removes any duplicate lines.

```
cdac@LAPTOP-MLEETRT5:~$ cat docs.txt fruit.txt | sort | uniq
apple
banana
good evening
good morning
grape
hello world
```

• ls -l | grep "^d":- Lists all files with detailed information and filters the output to show only directories.

```
cdac@LAPTOP-MLEETRT5:~$ ls -l |grep "^d"
drwxr-xr-x 4 cdac cdac 4096 Aug 29 08:24 LinuxAssignment.txt
drwxr-xr-x 2 cdac cdac 4096 Aug 30 21:13 mydir
cdac@LAPTOP-MLEETRT5:~$
```

• grep -r "pattern" /path/to/directory/ :- Recursively searches for the string "pattern" in all files

```
cdac@LAPTOP-MLEETRT5:~$ grep -r "apple" /path/to/directory/
grep: /path/to/directory/: No such file or directory
cdac@LAPTOP-MLEETRT5:~$
```

• cat file1.txt file2.txt | sort | uniq –d :- Concatenates the contents of file1.txt and file2.txt, sorts the combined output, and then displays only the lines that are duplicated.

```
cdac@LAPTOP-MLEETRT5:~$ cat que1.sh que1.sh que2.sh | sort | uniq -d


echo "hello world"
cdac@LAPTOP-MLEETRT5:~$
```

• chmod 644 file.txt:- Changes the permissions of file.txt to allow the owner to read and write, while others can only read.

• cp -r source_directory destination_directory :-it copy command recursively

- find /path/to/search -name "*.txt" :- Searches for all files with a .txt extension

```
cdac@LAPTOP-MLEETRT5:~$ find/path/to/ -name search "*.txt"
-bash: find/path/to/: No such file or directory
cdac@LAPTOP-MLEETRT5:~$
```

- chmod u+x file.txt :-provide the executable access to user

- echo $PATH :-create the PATH variable

```
cdac@LAPTOP-MLEETRT5:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/u
m Files/Common Files/Oracle/Java/javapath:/mnt/c/Program Files (x86)/VMware/VMware Player/b
tem32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/System32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell
t/c/Program Files (x86)/Microsoft VS Code/bin:/mnt/c/Program Files/PuTTY/:/mnt/d/bigdata/sp
rver/150/Tools/Binn/:/mnt/c/Program Files/dotnet/:/mnt/c/Program Files/Git/cmd:/mnt/c/Users
sers/Lenovo/AppData/Local/Programs/Python/Python311/:/mnt/c/Program Files/MySQL/MySQL Shell
odb/mongodb-database-tools-windows-x86_64-100.5.1/bin:/mnt/c/Users/Lenovo/AppData/Local/Pro
/Programs/Python/Python310/:/mnt/c/Users/Lenovo/AppData/Local/Microsoft/WindowsApps:/mnt/c/
cdac@LAPTOP-MLEETRT5:~$
```

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 cd for change the directory

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-owner 5-group 5-others

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. = Incorrect /:-chmod

2. cpy is used to copy files and directories. =Incorrect/:- cp

3. mkfile is used to create a new file. = Incorrect /:-touch

4. catx is used to concatenate files.= Incorrect /:-cat

5. rn is used to rename files. =Incorrect /:-mv

Part C

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

```
cdac@LAPTOP-MLEETRT5:~$ nano que1.sh
cdac@LAPTOP-MLEETRT5:~$ bash que1.sh
hello world
cdac@LAPTOP-MLEETRT5:~$
```

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

```
  GNU nano 6.2
#!/bin/bash
name="CDAC Mumbai"
echo $name
```

```
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
CDAC Mumbai
cdac@LAPTOP-MLEETRT5:~$
```

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

```
#!/bin/bash
echo "Enter the Numbers"
read num
echo $num
```

```
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
Enter the Numbers
1
1
```

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

```
#!/bin/bash
echo "Enter the Numbers1"
read num1
echo "Enter the Numbers2"
read num2
    add=$((num1+num2))
echo addtition of num $add
```

```
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
Enter the Numbers1
1
Enter the Numbers2
2
3
```

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

```
#!/bin/bash
echo "Enter the Numbers:"
read num

 if [ $((num % 2 )) -eq 0 ];
then
          echo "number is  even"
else
          echo "number is odd"
fi
```

```
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
Enter the Numbers:
3
number is odd
cdac@LAPTOP-MLEETRT5:~$
```

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

```
#!bin/bash/


for  i in {1..5};
 do

echo $i

done
```

```
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
1
2
3
4
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".

```
#!bin/bash/


if [ -f "file.txt" ];
then
   echo "file existed"
else
   echo  "file not existed"

fi
```

```
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
file not existed
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$
```

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.

```
#!bin/bash/

echo "enter the number:"
read num
 if [ $num -gt 10 ];
then
        echo "number is greater than 10"
else
      echo "number is less than 10"
fi
```

```
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
enter the number:
12
number is greater than 10
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$
```

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.

```
#!bin/bash/


for i in {1..10}; do
    for j in {1..10}; do
        printf "%4s" $((i * j))
    done
    echo ""
done
```

```
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$
```

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.

```
#!/bin/bash

while true;
 do

    read -p "Enter a number (negative to quit): " number


    if [ "$number" -lt 0 ];
 then
        break
    fi


    echo "Square: $((number * number))"
done
```

```
cdac@LAPTOP-MLEETRT5:~$ bash que2.sh
Enter a number (negative to quit): 2
Square: 4
Enter a number (negative to quit): 2
Square: 4
Enter a number (negative to quit): -2
cdac@LAPTOP-MLEETRT5:~$ nano que2.sh
cdac@LAPTOP-MLEETRT5:~$
```

Part D

Common Interview Questions (Must know)

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

- An **operating system (OS)** is software that manages computer hardware and software resources and provides services for computer programs.
- **Primary Functions:**
    1. **Process management**: Manages processes in the system, including execution, creation, and termination.
    2. **Memory management**: Handles allocation and deallocation of memory space.
    3. **File system management**: Manages files on the disk, including reading, writing, and organizing files.
    4. **Device management**: Manages device communication via drivers.
    5. **Security and access control**: Protects data and resources from unauthorized access.
    6. **User interface**: Provides a user interface (CLI, GUI) to interact with the system.

2. Explain the difference between process and thread.

- A **process** is an independent program in execution, with its own memory space.
- A **thread** is a smaller unit of execution within a process that shares the process's resources (memory, file handles).

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

- **Virtual memory** is a memory management technique that gives an application the impression it has contiguous working memory, while in fact, it may be physically fragmented.
- It works by using **paging** or **segmentation** to map virtual addresses to physical memory, allowing the OS to use disk space as an extension of RAM.

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

- **Multiprogramming**: Running multiple programs on a single CPU by keeping several jobs in memory simultaneously and switching between them.
- **Multitasking**: Extending multiprogramming by rapidly switching between tasks to give the appearance of simultaneous execution.

- **Multiprocessing**: Using multiple CPUs or cores to run processes simultaneously.

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

- A **file system** organizes and manages files on storage devices.
- **Components**:
    - **Files and directories**: Basic units of storage.
    - **Metadata**: Information about files (e.g., size, permissions).
    - **File control blocks (FCBs)**: Data structures that store details about files.
    - **Directories**: Structure that contains references to files.
    - **Access control lists (ACLs)**: Define permissions for files.

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

- A **deadlock** occurs when two or more processes are unable to proceed because each is waiting for the other to release resources.
- **Prevention**:
    - **Avoidance algorithms** like Banker's algorithm.
    - **Resource allocation policies** that ensure no circular wait.
    - **Timeouts**: Force processes to release resources after a certain period.

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

- The **kernel** is the core of the OS, managing hardware resources and system calls.
- The **shell** is a user interface (CLI or GUI) that interprets commands and communicates with the kernel.

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

- **CPU scheduling** determines which process runs at any given time to optimize CPU utilization and system performance.
- It's important for ensuring efficient, fair, and responsive task management in a system.

9. How does a system call work?

- A **system call** is a way for programs to interact with the OS by requesting services like file access or process management.
- It works by switching from user mode to kernel mode, executing the requested service, and then returning to user mode.

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

- **Device drivers** are specialized software that allows the OS to communicate with hardware devices, abstracting hardware details and providing a standard interface.

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

- The **page table** maps virtual addresses to physical memory addresses, enabling the OS to manage memory efficiently through paging.

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

- **Thrashing** occurs when the system spends more time swapping pages in and out of memory than executing processes.
- **Avoidance**: Adjusting the degree of multiprogramming, using better page replacement algorithms, or increasing physical memory.

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

- A **semaphore** is a synchronization tool used to manage concurrent processes by signaling and waiting, preventing race conditions and ensuring mutual exclusion.

14. How does an operating system handle process synchronization?

- The OS uses **locks**, **semaphores**, **monitors**, and **message passing** to ensure that processes execute in a synchronized manner without conflicts.

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

- An **interrupt** signals the CPU to stop its current task and execute a special routine, often in response to hardware events or system calls.

16. Explain the concept of a file descriptor.

- A **file descriptor** is an integer handle used by processes to access files or I/O streams, abstracting file operations in the OS.

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

- **Recovery** involves steps like restoring system state from backups, using journaling file systems, or applying transaction logs to ensure data consistency.

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

- **Monolithic Kernel**: The entire OS, including device drivers, runs in a single address space, leading to efficient but complex management.
- **Microkernel**: The kernel contains only the essential functions, with other services running in user space, leading to a more modular and stable system.

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

- **Internal Fragmentation**: Wasted space within allocated memory blocks due to fixed-size allocation.
- **External Fragmentation**: Wasted space between allocated memory blocks due to variable-size allocation.

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

- The OS manages I/O operations using device drivers, buffering, caching, and scheduling to ensure efficient communication between the CPU and peripheral devices.

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

- **Preemptive Scheduling**: The OS can interrupt a running process to assign the CPU to another process.
- **Non-Preemptive Scheduling**: Once a process starts executing, it runs until it voluntarily releases the CPU.

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

Round-robin scheduling assigns each process a fixed time slice (quantum) to execute, then cycles through processes in a queue, ensuring fair CPU distribution.

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

- **Priority Scheduling** assigns a priority to each process, with the CPU allocated to the process with the highest priority.
- Priorities can be static (assigned at creation) or dynamic (changing based on factors like aging).

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

- **Shortest Job Next (SJN)** selects the process with the shortest expected execution time next, optimizing turnaround time. It's used in batch processing where job lengths are known.

25. Explain the concept of multilevel queue scheduling.

- **Multilevel Queue Scheduling** divides processes into different queues based on priority or type, each with its own scheduling algorithm, allowing differentiated handling of processes.

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

- A **Process Control Block (PCB)** is a data structure containing process information such as process ID, state, CPU registers, memory management information, and I/O status.

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

- **Process State Diagram**: Includes states like New, Ready, Running, Waiting, and Terminated.
    - **Transitions**:
        - New → Ready (Process admitted).
        - Ready → Running (CPU assigned).
        - Running → Waiting (I/O request).
        - Waiting → Ready (I/O completion).
        - Running → Terminated (Process finished).

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

- **Inter-Process Communication (IPC)** mechanisms like pipes, message queues, shared memory, and sockets allow processes to exchange data and synchronize.

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

- **Process Synchronization** ensures that processes execute in the correct order without conflicts, essential for maintaining data consistency in concurrent execution.

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

- A **zombie process** is a process that has completed execution but still has an entry in the process table because its parent hasn't yet read its exit status.

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

- **Internal Fragmentation**: Occurs within allocated memory blocks.
- **External Fragmentation**: Occurs between allocated blocks due to variable allocation sizes.

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

- **Demand paging** is a memory management technique where pages of a program are loaded into memory only when they are needed, rather than preloading all pages.
- **Improvement in Efficiency**:

- Reduces memory usage by only loading necessary pages, freeing up memory for other processes.
- Decreases load times by avoiding loading the entire program into memory upfront.
- Allows larger programs to run on systems with limited memory by swapping pages in and out as needed.

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

- The **page table** is a data structure used in virtual memory systems to map virtual addresses to physical addresses.
- **Role**:
  - Translates virtual addresses generated by a process into physical addresses in RAM.
  - Maintains information about each page, such as its location in physical memory or on disk (if swapped out).
  - Ensures isolation between processes by maintaining separate page tables for each process, preventing one process from accessing another's memory.

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

- The **Memory Management Unit (MMU)** is a hardware component that handles the translation of virtual addresses to physical addresses.
- **How it works**:
  - Uses the page table to look up the physical address corresponding to a virtual address.
  - Checks access permissions and handles page faults (when a page is not in memory).
  - Supports virtual memory by enabling the OS to provide processes with a virtual address space larger than the actual physical memory.

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

- **Thrashing** occurs when a system spends more time swapping pages in and out of memory than executing processes, leading to severe performance degradation.
- **Avoidance**:

- **Working Set Model**: Track the working set of each process (the set of pages it currently needs) and ensure it fits in memory.
- **Page Replacement Algorithms**: Use algorithms like LRU (Least Recently Used) to minimize unnecessary page swaps.
- **Increase Physical Memory**: Adding more RAM can reduce the need for frequent paging.
- **Reduce Multiprogramming Level**: Limit the number of active processes to reduce competition for memory.

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

- A **system call** is a programmed request from a user-level process to the operating system for a service, such as file access, process control, or communication.
- **Facilitation of Communication**:
  - Allows user programs to interact with the hardware indirectly through the OS, maintaining system security and stability.
  - Provides a controlled interface for executing privileged operations, ensuring that user programs do not have direct access to critical system resources.
  - Triggers a switch from user mode to kernel mode, where the OS performs the requested operation before returning to the user process.

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

- **Monolithic Kernel**:
  - All OS services (e.g., device drivers, file systems, networking) run in a single address space in kernel mode.
  - Advantages: Faster due to fewer context switches, simpler communication between services.
  - Disadvantages: Less modular, harder to maintain and debug, greater risk of system crashes.
- **Microkernel**:
  - Only essential services (e.g., process management, memory management) run in kernel mode, while other services run in user space.
  - Advantages: More modular and secure, easier to update and maintain, better fault isolation.

    - Disadvantages: Potentially slower due to more context switches and inter-process communication overhead.

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

- The OS handles I/O operations by:
    - **Device Drivers**: Using device drivers as intermediaries between the hardware devices and the OS, translating OS requests into device-specific commands.
    - **Buffering**: Temporarily storing data in memory buffers to match the speed differences between the CPU and I/O devices.
    - **Caching**: Storing frequently accessed data in faster memory (cache) to improve I/O performance.
    - **Spooling**: Queueing up I/O operations (like print jobs) to be handled sequentially by a device.
    - **Interrupts**: Using hardware interrupts to signal the CPU when an I/O operation is complete, allowing the CPU to process other tasks in the meantime.

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

- A **race condition** occurs when multiple processes or threads access shared resources concurrently, and the final outcome depends on the order of execution, which can lead to unpredictable and erroneous behavior.
- **Prevention**:
    - **Locks/Mutexes**: Ensuring that only one process or thread can access a shared resource at a time.
    - **Semaphores**: Controlling access to resources by using signaling mechanisms to ensure proper execution order.
    - **Atomic Operations**: Using atomic instructions that complete in a single step, preventing interruption during critical operations.
    - **Condition Variables**: Allowing threads to wait for certain conditions to be met before proceeding, ensuring synchronization.
    - **Avoiding Shared State**: Redesigning the program to minimize or eliminate shared data or resources.

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

- **Device drivers** are specialized software modules that act as intermediaries between the OS and hardware devices.

- **Role**:
    - **Translation**: Convert high-level OS commands into hardware-specific operations.
    - **Control**: Manage device-specific functions and provide an interface for the OS to interact with hardware.
    - **Abstraction**: Hide hardware details from the OS and user applications, providing a uniform interface for I/O operations.
    - **Error Handling**: Manage errors and exceptions generated by hardware 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 completed execution but still has an entry in the process table because its parent has not yet read its exit status.
- **Occurrence**:
    - When a process terminates, it sends an exit status to its parent, but if the parent does not use wait() to collect this status, the process remains in the zombie state.
- **Prevention**:
    - **Parent Process Handling**: Ensure that the parent process calls wait() or waitpid() to collect the exit status.
    - **Signal Handling**: Use signal handlers to catch the SIGCHLD signal and handle child process termination.

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

- An **orphan process** is a process whose parent has terminated while the process is still running.
- **Handling**:
    - The **init process** (or a similar designated process) adopts orphan processes. The init process becomes the new parent and ensures that these processes are properly cleaned up when they terminate.

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

- The **parent process** creates a **child process** using the fork() system call.
- **Relationship**:

- The child process inherits some attributes from the parent, such as environment variables and open file descriptors.
- The parent can control or monitor the child process, and the child process can execute new programs using exec() functions.
- The parent is responsible for collecting the child's exit status and handling termination.

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

- The fork() system call creates a new process by duplicating the calling (parent) process.
- **How it works**:
  - The new process (child) receives a unique process ID (PID) and a copy of the parent's address space, file descriptors, and other attributes.
  - Both processes (parent and child) continue execution from the point where fork() was called, but fork() returns different values to each: 0 to the child and the child's PID to the parent.

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

- A parent process can use the wait() or waitpid() system calls to wait for a child process to terminate.
- **wait()**: Suspends the parent process until one of its child processes exits or a signal is received.
- **waitpid()**: Allows the parent process to wait for a specific child process or set of child processes to change state, providing more control and flexibility.

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

- The **exit status** provides information about how the child process terminated:
  - **Exit Code**: Indicates whether the child exited normally (with a specific exit code) or abnormally (due to a signal).
  - **Status Information**: The parent process can use macros like WIFEXITED() and WEXITSTATUS() to interpret the exit status and determine if the process terminated successfully or with an error.

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.

- **How it works**:
    - The parent process sends a signal (e.g., SIGTERM for termination or SIGKILL for forceful termination) to the child process's PID.
    - The child process can handle or ignore signals, but if it does not respond, the OS will terminate it according to the signal sent.

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

- **Process Group**:
    - A collection of processes that can be managed together for job control purposes.
    - All processes in a group receive signals sent to the group, and job control commands (e.g., fg, bg) operate on process groups.
- **Session**:
    - A collection of process groups, where a session is initiated by a session leader process.
    - Sessions are used to manage related process groups and handle terminal control and job management.

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

- The exec() family of functions replaces the current process image with a new process image, effectively loading a new program into the existing process.
- **How it works**:
    - The current process's memory space, including code, data, and stack, is replaced with the new program's content.
    - The exec() call does not return if successful, as the old process image is gone. The new program starts execution from its entry point.

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

- The **waitpid()** system call allows a parent process to wait for a specific child process to change state or terminate.
- **Differences from wait()**:

- waitpid() provides more control by allowing the parent to specify which child process to wait for and whether to wait for all children or just one.
- wait() waits for any child process to terminate, without specifying which child.

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

- **Process Termination**:
  - A process can terminate voluntarily by calling exit(), which performs cleanup and releases resources.
  - It can also be terminated by signals (e.g., SIGTERM, SIGKILL) sent by other processes or the OS.
  - Upon termination, the process's exit status is sent to its parent, and the process's resources are cleaned up, except for its process table entry, which remains until the parent collects the exit status.

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** (or admission scheduler) selects processes from the job pool (disk) and loads them into memory (the ready queue).
- **Role**:
  - Controls the degree of multiprogramming by determining how many processes are loaded into memory and ready for execution.
  - Influences system performance by balancing the number of processes in memory and ensuring efficient use of resources.

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**:
  - Runs frequently (milliseconds) and decides which process in the ready queue gets to use the CPU next.
  - Makes decisions based on process priority, CPU bursts, and other factors to optimize CPU utilization.
- **Long-Term Scheduler**:
  - Runs less frequently (seconds to minutes) and decides which processes to admit to memory from the job pool.

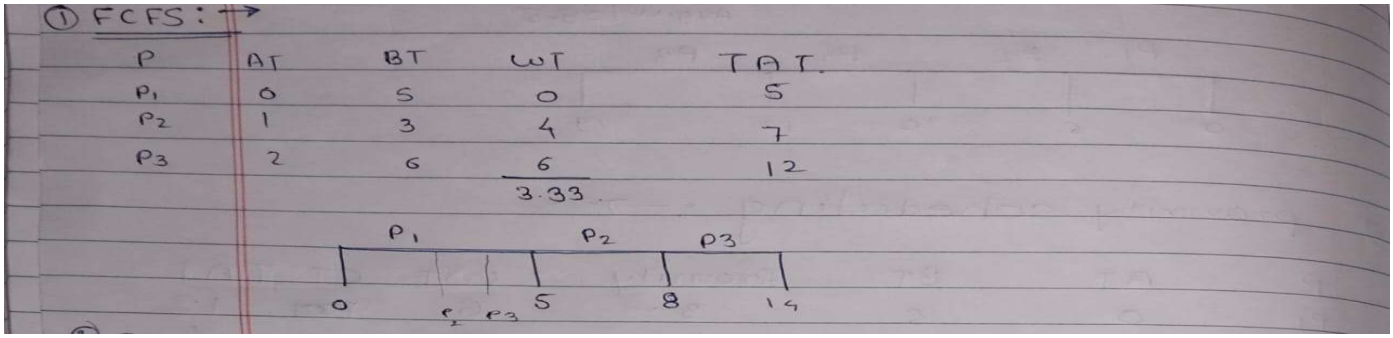
- **Medium-Term Scheduler**:
    - Operates between short-term and long-term schedulers, handling process swapping between memory and disk to manage the active set of processes.

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

- **Scenario**:
    - When the system is experiencing high memory pressure and needs to free up memory for new or more important processes.
- **How it Helps**:
    - The medium-term scheduler may swap out less frequently used processes to disk (paging out) to free up physical memory.
    - It then loads other processes into memory (paging in) based on current demands, thus balancing memory usage and ensuring that active processes have the resources they need while keeping the system responsive and efficient.

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.

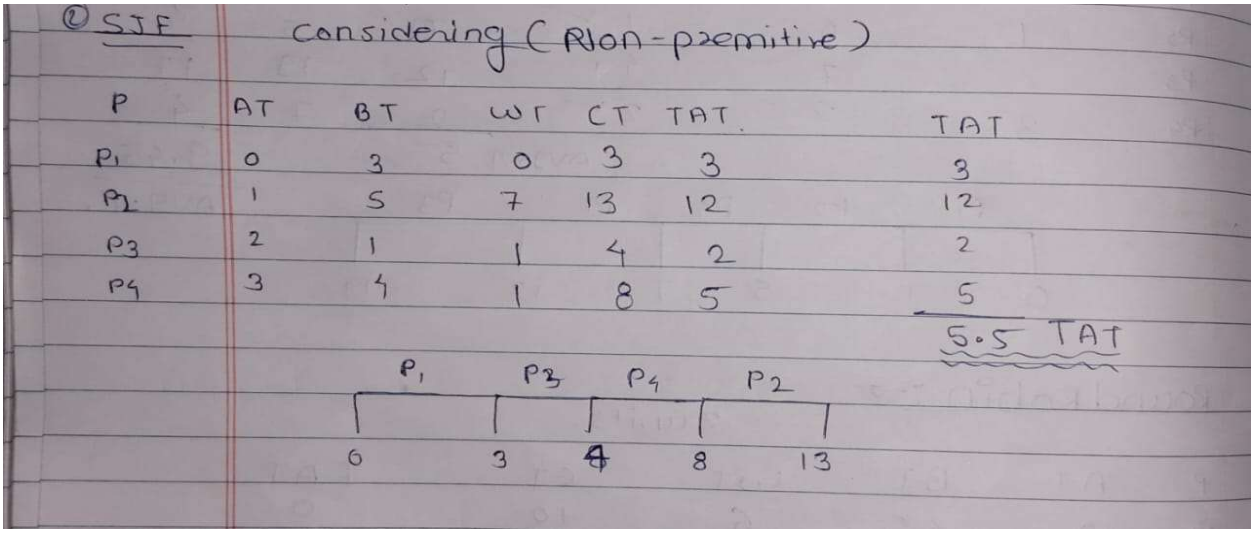


   average waiting time=3.3

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.



average turnaround time=5.5

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

| Process | Arrival Time | Burst Time | Priority |

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

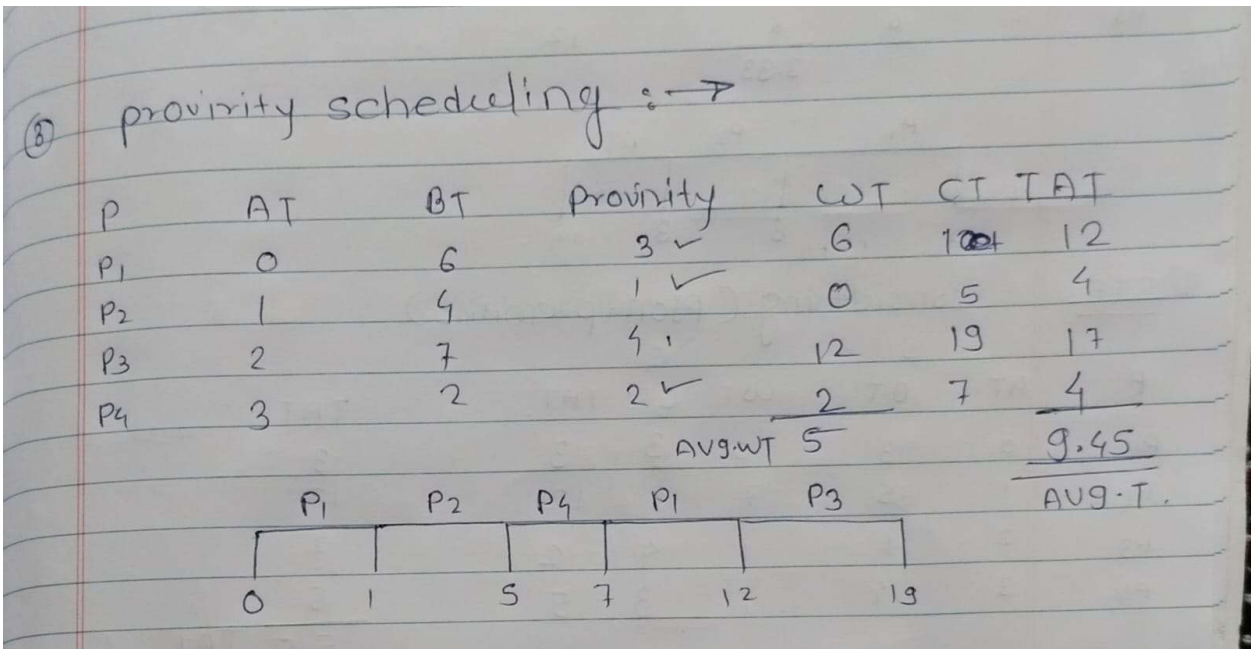
P1 | 0 | 6 | 3 | |

P2 | 1 | 4 | 1 | |

P3 | 2 | 7 | 4 | |

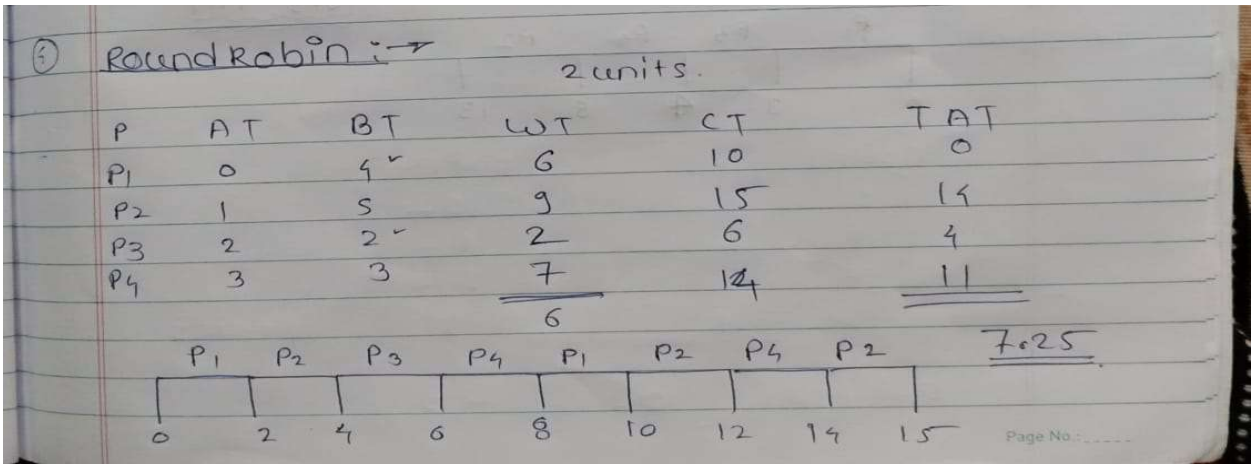
P4 | 3 | 2 | 2 | |

Calculate the average waiting time using Priority Scheduling.



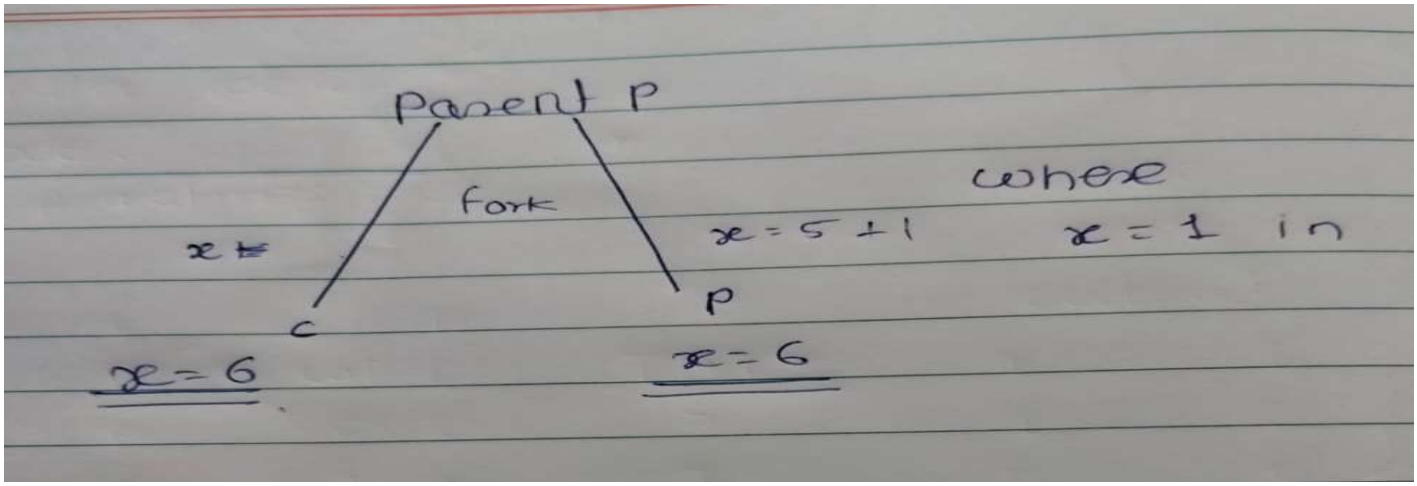
average waiting time=5

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.



average waiting time=6

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?