Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

1. **echo "Hello, World!"**
   Answer: This command prints the string "Hello, World!" to the terminal. It is often used for testing and displaying messages.
2. **name="Productive"**
   Answer: This command assigns the string "Productive" to the variable name. It does not produce any output but sets the variable for use in the current shell session.
3. **touch file.txt**
   Answer: This command creates an empty file named file.txt if it does not already exist. If the file does exist, it updates the file's timestamp to the current time.
4. **ls -a**
   Answer: This command lists all files and directories in the current directory, including hidden files which starting with a dot.
5. **rm file.txt**
   Answer: This command removes the file named file.txt from the current directory. Be cautious, as this action is irreversible.
6. **cp file1.txt file2.txt**
   Answer: This command copies the contents of file1.txt to a new file named file2.txt. If file2.txt already exists, it will be overwritten.
7. **mv file.txt /path/to/directory/**
   Answer: This command moves file.txt to the specified directory (/path/to/directory/).
8. **chmod 755 script.sh**
   Answer: This command changes the permissions of script.sh to 755, which means the owner can read, write, and execute the file, while the group and others can read and execute it.

9. **grep "pattern" file.txt**
   Answer: This command searches for the specified "pattern" in file.txt and prints the lines that contain it. It is useful for finding specific text within files.

10. **kill PID**
    Answer: This command sends a termination signal to the process with the specified Process ID (PID). It is used to stop a running process.

11. **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**
    Answer: This command does the following:

    - Creates a new directory named mydir.
    - Go into that directory.
    - Creates an empty file named file.txt.
    - Writes "Hello, World!" into file.txt.
    - Displays the contents of file.txt, which will show "Hello, World!".

12. **ls -l | grep ".txt"**
    Answer: This command lists all files in the current directory and filters the output to show only files that have ".txt" in their names.

13. **cat file1.txt file2.txt | sort | uniq**
    Answer: This command concatenates the contents of file1.txt and file2.txt, sorts the combined output, and then filters out duplicate lines, displaying only unique lines.

14. **ls -l | grep "^d"**
    Answer: This command lists all files and directories and filters the output to show only directories starting with "d".

15. **grep -r "pattern" /path/to/directory/**
    Answer: This command recursively searches for the specified "pattern" in all files within the specified directory and its subdirectories, printing matching lines.

16. **cat file1.txt file2.txt | sort | uniq -d**
    Answer: This command concatenates file1.txt and file2.txt, sorts the output, and then displays only the duplicate lines that appear in both files.

17. **chmod 644 file.txt**
    Answer: This command changes the permissions of file.txt to 644, allowing the owner to read and write the file, while the group and others can only read it.

18. **cp -r source_directory destination_directory**
    Answer: This command copies the entire source_directory and its contents to destination_directory. The **-r** indicates that the copy should be recursive.

19. **find /path/to/search -name "*.txt"**
    Answer: This command searches for all files with a .txt extension in the specified path and its subdirectories, listing their names.

20. **chmod u+x file.txt**
    Answer: This command adds execute permissions for the user of file.txt.

21. **echo $PATH**
    Answer: This command prints the current value of the PATH environment variable, which contains a list of directories that the shell searches for executable files when a command is entered.

Part B

Identify True or False:

1. **ls is used to list files and directories in a directory.**
   Answer: True

2. **mv is used to move files and directories.**
   Answer: True

3. **cd is used to copy files and directories.**
   Answer: False

4. **pwd stands for "print working directory" and displays the current directory.**
   Answer: True

5. **grep is used to search for patterns in files.**
   Answer: True

6. **chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to group and others.**
   Answer: True
7. **mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1 if directory1 does not exist.**
   Answer: True
8. **rm -rf file.txt deletes a file forcefully without confirmation.**
   Answer: True

Identify the Incorrect Commands:

1. **chmodx is used to change file permissions.**
   Answer: Incorrect

2. **cpy is used to copy files and directories.**
   Answer: Incorrect

3. **mkfile is used to create a new file.**
   Answer: Incorrect

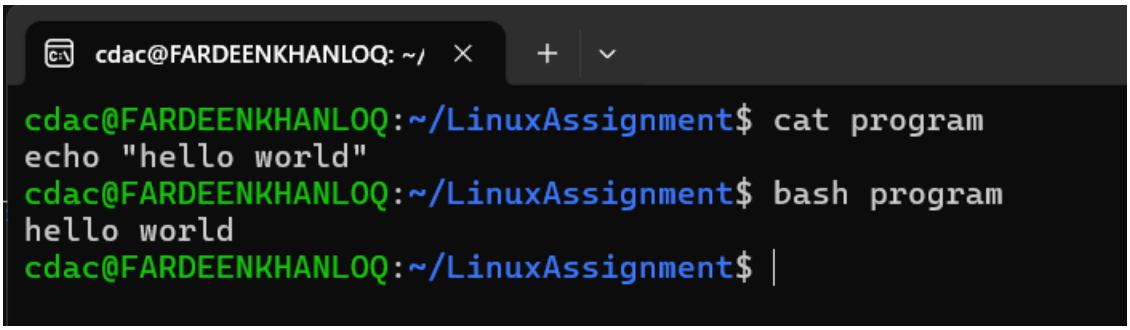
4. **catx is used to concatenate files.**
   Answer: Incorrect

5. **rn is used to rename files.**
   Answer: Incorrect

Part C

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

Answer:



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

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
name="CDAC Mumbai"
echo $name
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
CDAC Mumbai
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
echo enter a number
read number
echo "You entered: $number"

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
enter a number
7
You entered: 7
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
echo enter number1
read num1
echo enter number2
read num2
sum=$((num1 + num2))
echo "The sum of $num1 and $num2 is: $sum"


cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
enter number1
5
enter number2
3
The sum of 5 and 3 is: 8
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
echo "Enter a number:"
read number
if (( number % 2 == 0 )); then
    echo "Number is Even"
else
    echo "Number is Odd"
fi

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
Enter a number:
7
Number is Odd
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
Enter a number:
8
Number is Even
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
for i in 1 2 3 4 5
do
    echo $i
done


cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
1
2
3
4
5
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
a=1
while [ $a -le 5 ]
do
    echo $a
    a=`expr $a + 1`
done

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
1
2
3
4
5
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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".

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ ls
data.txt  docs  duplicate.txt  extracted_docs  input.txt  numbers.txt  output.txt  program
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
if [ -e file.txt ];
then
    echo "File exists"
else
    echo "File does not exist"
fi

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
File does not exist
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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.

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
echo enter a number
read number
if [ $number -gt 10 ]
then
    echo "$number is greater than 10"
else
    echo "$number is not greater than 10"
fi

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
enter a number
7
7 is not greater than 10
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
enter a number
12
12 is greater than 10
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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.

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
for i in 1 2 3 4 5
 do
    for j in 1 2 3 4 5
do
        result=$((i * j))
        echo -n "$result "
    done
    echo ""
done

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

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.

Answer:

```
cdac@FARDEENKHANLOQ:~/LinuxAssignment$ cat program
while true
do
echo "Enter a number:"
    read number
    if [ $number -lt 0 ]
then
        echo "Negative number entered"
        break
    fi

    square=$((number * number))
    echo "The square of $number is: $square"
done

cdac@FARDEENKHANLOQ:~/LinuxAssignment$ bash program
Enter a number:
7
The square of 7 is: 49
Enter a number:
-9
Negative number entered
cdac@FARDEENKHANLOQ:~/LinuxAssignment$
```

Part D

Common Interview Questions

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

An **operating system (OS)** is system software that manages computer hardware and software resources. Its primary functions include:

- **Process Management:** Scheduling and managing running programs.
- **Memory Management:** Allocating and managing the system's memory.
- **File System Management:** Organizing, storing, and retrieving files.
- **Device Management:** Interfacing with peripheral devices via device drivers.
- **Security and Access Control:** Protecting data and system resources.
- **I/O Management:** Handling input/output operations efficiently.

---

2. Explain the difference between process and thread.

- **Process:** An independent program in execution with its own memory space, resources, and system context.
- **Thread:** A lightweight unit of execution within a process that shares the process's memory and resources. Threads enable parallelism within the same application.

---

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

Virtual memory is a memory management technique that gives applications the illusion of a large, contiguous memory space. It works by:

- **Paging/Swapping:** Dividing memory into blocks (pages) and moving them between physical RAM and disk storage as needed.
- **Address Translation:** Using a page table (managed by the Memory Management Unit, MMU) to map virtual addresses to physical addresses. This allows efficient use of physical memory and isolation between processes.

---

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

- **Multiprogramming:** Multiple programs are loaded into memory and executed by the CPU; the OS switches between them to maximize CPU usage.
- **Multitasking:** A form of multiprogramming where the OS rapidly switches between tasks (processes or threads), giving the illusion of simultaneous execution.
- **Multiprocessing:** Utilizes two or more CPUs/cores to execute processes simultaneously, improving overall system performance.

---

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

A **file system** organizes and stores data on storage devices. Its components include:

- **Directories:** Hierarchical structures that organize files.
- **Files:** Collections of data with metadata (permissions, timestamps, etc.).
- **Inodes/File Allocation Tables:** Data structures that store information about files and how they are stored on disk.
- **Superblock:** Contains metadata about the file system itself.

---

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

A **deadlock** occurs when a set of processes are blocked because each process is holding a resource and waiting for another resource held by another process. Prevention methods include:

- **Resource Allocation Strategies:** Avoid circular wait conditions by ordering resource requests.
- **Deadlock Avoidance Algorithms:** Such as the Banker's algorithm.
- **Deadlock Detection and Recovery:** Periodically checking for deadlocks and terminating processes if necessary.

---

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

- **Kernel:** The core part of the OS responsible for managing hardware, memory, and system resources. It operates in privileged (kernel) mode.
- **Shell:** A user interface (command-line or graphical) that allows users to interact with the kernel and run applications.

---

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

CPU scheduling is the process of determining which process in the ready queue gets to use the CPU next. It is important because it affects:

- **System Responsiveness:** Ensuring that interactive tasks receive timely attention.

- **Throughput:** Maximizing the number of processes completed in a given time.
- **Fairness:** Equitably allocating CPU time among processes.

---

9. How does a system call work?

A **system call** is a mechanism that allows a user program to request a service from the operating system's kernel. It involves:

- **Mode Switching:** Transitioning from user mode to kernel mode.
- **Execution of Kernel Code:** The OS performs the requested operation (e.g., I/O, process management).
- **Returning Results:** Control is transferred back to the user program with the results.

---

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

Device drivers are specialized software components that enable the OS to communicate with hardware devices. They abstract the hardware specifics and provide a standardized interface for the OS to perform operations like reading, writing, or controlling the device.

---

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

The **page table** maps virtual addresses to physical memory addresses. It stores information such as:

- **Frame Number:** Which physical frame a virtual page is stored in.
- **Status Bits:** Indicating if the page is in memory, its access rights, and if it has been modified.
  This mapping enables efficient use of memory and ensures protection between processes.

---

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

Thrashing is a state where excessive paging operations (swapping pages in and out of memory) cause the system to slow down dramatically. It can be avoided by:

- **Reducing Multiprogramming:** Lowering the number of active processes.
- **Increasing Physical Memory:** More RAM reduces the need for paging.
- **Improving Locality:** Ensuring processes work on a small set of pages before moving on.

---

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

A **semaphore** is a synchronization tool used to control access to shared resources. There are two main types:

- **Binary Semaphore (Mutex):** Can have only two values (0 or 1) and is used to implement mutual exclusion.
- **Counting Semaphore:** Allows a set number of processes to access a resource concurrently.

---

14. How does an operating system handle process synchronization?

The OS employs various synchronization mechanisms to prevent race conditions, including:

- **Semaphores and Mutexes:** To control access to shared resources.
- **Monitors:** High-level synchronization constructs that combine mutual exclusion with condition variables.
- **Condition Variables:** To block a process until a particular condition is met.

---

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

An **interrupt** is a signal to the CPU that immediately stops the current process and executes an interrupt handler. This mechanism allows the OS to respond quickly to external events (like I/O operations) and improve system responsiveness.

---

16. Explain the concept of a file descriptor.

A **file descriptor** is an integer that uniquely identifies an open file or I/O resource (such as sockets or pipes) within a process. It acts as a handle through which the process can perform operations (read, write, close) on that resource.

---

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

System recovery may involve:

- **Crash Dumps:** Saving the state of the system for debugging.
- **Logging and Journaling:** File systems (like ext3/ext4) use journaling to recover to a consistent state.
- **Restart Mechanisms:** Automatic reboots and service restarts.
- **Backup and Restore Procedures:** To restore data from backups.

---

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

- **Monolithic Kernel:** All OS services (like device drivers, file system management, and system calls) run in kernel space. This can lead to higher performance but less modularity.
- **Microkernel:** Only essential services run in kernel space (e.g., scheduling, IPC). Other services run in user space, which improves modularity and fault isolation, though it might introduce performance overhead.

---

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

- **Internal Fragmentation:** Wasted space within an allocated memory block due to fixed partition sizes.
- **External Fragmentation:** Unused memory spaces between allocated blocks that occur when free memory is split into small chunks.

---

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

The OS manages I/O by:

- **Buffering and Caching:** Temporarily storing I/O data to smooth out differences in speed between devices and the CPU.

- **Spooling:** Queuing data for devices that cannot handle concurrent I/O.
- **Device Drivers:** Providing an interface to communicate with hardware devices.
- **Interrupt Handling:** Managing asynchronous I/O events.

---

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

- **Preemptive Scheduling:** The OS can interrupt a running process to assign CPU time to another process. This allows better responsiveness and fairness.
- **Non-preemptive Scheduling:** A process runs until it voluntarily yields the CPU (or finishes), which can lead to inefficiencies if a process holds the CPU for too long.

---

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

Round-robin scheduling assigns each process a fixed time slice (quantum) in a cyclic order. After the time slice expires, the process is moved to the back of the queue, ensuring that all processes get an equal share of CPU time. This method is fair and simple, especially for time-sharing systems.

---

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

In **priority scheduling**, each process is assigned a priority. The CPU is allocated to the process with the highest priority (lower numerical value or higher ranking, depending on the system). Priorities can be:

- **Static:** Determined at process creation and not changed.

- **Dynamic:** Adjusted during execution based on factors like aging to prevent starvation.

---

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

The **Shortest Job Next (SJN)** algorithm selects the process with the smallest estimated run time for execution. It minimizes the average waiting time but may lead to the starvation of longer processes. It is most effective in environments where job lengths are known in advance.

---

25. Explain the concept of multilevel queue scheduling.

Multilevel queue scheduling separates processes into different queues based on characteristics (e.g., system processes, interactive processes, batch processes). Each queue can have its own scheduling algorithm, and processes do not move between queues. This separation allows the system to treat different types of processes according to their specific needs.

---

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

A **process control block (PCB)** is a data structure in the OS that contains critical information about a process, including:

- Process state (new, ready, running, waiting, terminated)
- Program counter and CPU registers
- Memory management information (e.g., page tables)
- Process identification numbers (PID, parent PID)
- I/O status and resource usage

---

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

A typical **process state diagram** includes:

- **New:** The process is being created.
- **Ready:** The process is waiting to be assigned to the CPU.
- **Running:** The process is executing on the CPU.
- **Waiting (Blocked):** The process is waiting for an event (like I/O completion).
- **Terminated:** The process has finished execution.
  Transitions occur via process creation, scheduling, blocking on I/O, and termination.

---

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

Processes communicate using **Inter-Process Communication (IPC)** mechanisms such as:

- **Pipes:** For unidirectional communication.
- **Message Queues:** To send and receive messages asynchronously.
- **Shared Memory:** Allowing processes to access the same memory space.
- **Sockets:** For communication over a network or between processes on the same machine.

---

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

Process synchronization involves coordinating the execution of processes to ensure that they do not conflict when accessing shared resources. It is important to prevent race conditions, data inconsistency, 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 completed execution but still has an entry in the process table because its parent has not yet read its exit status (via a wait system call). This occurs when the parent neglects to reap the child process, leaving behind a “zombie.”

---

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

- **Internal fragmentation** occurs when fixed-size memory blocks are allocated, but the process does not fully utilize them, leading to wasted space within the allocated block. This happens in systems using fixed partitioning or paging. A common solution is using smaller allocation units or dynamic memory allocation.
- **External fragmentation** occurs when free memory is divided into small, non-contiguous blocks due to dynamic allocation and deallocation. Even if enough total free memory exists, it may not be usable due to fragmentation. This commonly occurs in systems using segmentation or dynamic partitioning. Solutions include memory compaction or using segmentation to merge free spaces.

---

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

Demand paging is a memory management scheme where pages are loaded into memory only when they are needed (on-demand) rather than loading an entire program at startup. This reduces memory usage and speeds up program startup by only using memory for actively used pages.

---

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

The page table maps virtual addresses to physical addresses, enabling efficient memory management in an OS. It supports address translation, memory protection, and paging, allowing non-contiguous memory allocation and process isolation. Each entry stores the physical frame number, access permissions, and status bits (valid, dirty, reference). Types include single-level, multi-level, and inverted page tables, optimizing memory usage and translation speed.

---

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

The **Memory Management Unit (MMU)** is hardware that handles virtual-to-physical address translation using page tables. It also enforces memory protection, caching, and manages other aspects of memory access.

---

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

Thrashing occurs when excessive paging operations degrade performance. It can be mitigated by:

- Reducing the degree of multiprogramming
- Increasing physical memory
- Enhancing locality of reference in programs

---

36. What is a system call, and how does it facilitate communication between user programs and the operating system?
A **system call** is the mechanism by which a user program requests a service from the kernel. It provides a controlled interface for performing operations like file handling, process creation, and I/O operations.

---

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

- **Monolithic Kernel:** Integrates most services in kernel space for performance.
- **Microkernel:** Runs minimal services in kernel space, moving other services to user space for modularity and reliability.

---

38. How does an operating system handle I/O operations?
The OS handles I/O by coordinating device drivers, buffering data, scheduling I/O requests, and using interrupts to manage asynchronous I/O events efficiently.

---

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 data concurrently, and the final result depends on the sequence of access. It can be prevented by using synchronization mechanisms such as locks, semaphores, and critical sections to ensure that only one process accesses the shared resource at a time.

---

40. Describe the role of device drivers in an operating system.
Device drivers serve as the interface between the OS and hardware devices, translating high-level OS requests into device-specific commands and managing hardware resources.

---

41. What is a zombie process, and how does it occur? How can a zombie process be prevented?
A **zombie process** is one that has terminated but still occupies an entry in the process table because the parent has not yet called wait() to read its exit status. It can be prevented by ensuring that parent processes properly reap their child processes using wait or waitpid.

---

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

An **orphan process** is a process whose parent has terminated before it. The OS typically reassigns orphan processes to the **init process** (or a similar system process), which then becomes responsible for cleaning up after them once they complete execution.

---

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** (usually via fork()). The child inherits many attributes from the parent (such as environment variables and open file descriptors), and the parent is responsible for managing the child's termination (by calling wait or waitpid).

---

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

The **fork()** system call duplicates the calling process, creating a child process that is nearly identical to the parent. Both processes continue execution independently, with the child receiving a unique process ID. The fork() return value is used to differentiate between the parent (receives child's PID) and the child (receives 0).

---

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 suspend its execution until one or more child processes have terminated. This helps prevent zombie processes by ensuring the exit status of the child is collected.

---

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

The **exit status** indicates how a child process terminated (e.g., successful completion, error, or termination by signal). The parent process can inspect this status after wait() returns, enabling error handling and proper process management.

---

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

A parent can terminate a child by sending it a signal using the **kill()** system call (e.g., SIGTERM for a graceful shutdown or SIGKILL for immediate termination).

---

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

- **Process Group:** A collection of one or more processes that can be managed together, typically used for job control in a shell.
- **Session:** A collection of process groups, often created at login, representing a user session. A session helps manage foreground and background processes.

---

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

The **exec()** functions load a new program into the current process, replacing its code, data, and stack with that of the new program. This is often used after fork() in the child process to run a different executable while retaining the same process ID.

---

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

waitpid() allows a parent process to wait for a specific child process (or set of child processes) to change state, whereas **wait()** waits for any child process to finish. waitpid() provides more precise control over process synchronization.

---

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

Process termination occurs when a process calls **exit()** or is terminated by a signal. The OS then cleans up the process's resources, and the process's PCB remains in the process table until the parent retrieves its 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 job scheduler) controls the admission of processes into the system, thus regulating the degree of multiprogramming. By deciding which processes to load into memory, it influences system load and overall performance.

---

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

- **Short-Term Scheduler:** Runs very frequently (milliseconds) to select which process in the ready queue is allocated the CPU next.
- **Long-Term Scheduler:** Runs infrequently (seconds to minutes) to control the number of processes in memory.
- **Medium-Term Scheduler:** Operates between the two, often handling the suspension and resumption of processes to manage memory and improve responsiveness.

---

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

A **medium-term scheduler** might be invoked when the system is under heavy memory pressure. For example, if many processes are waiting for I/O and some are not actively executing, the OS might swap out (suspend) those inactive processes to disk. This frees up RAM for processes that need to run immediately, thus balancing system load and improving overall performance.

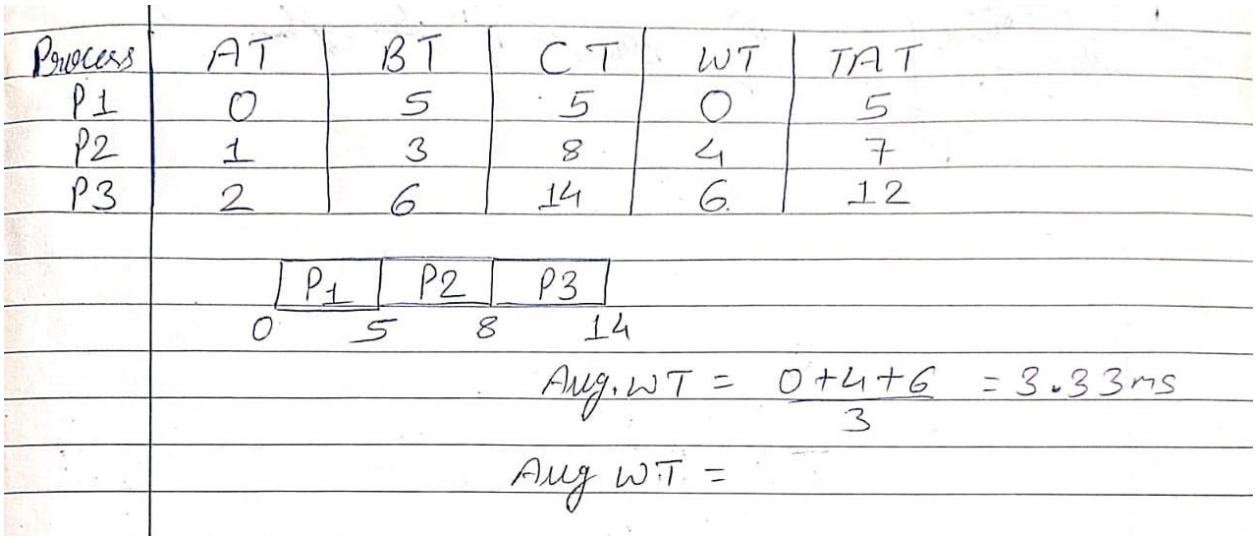
Part E

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.

Answer:



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

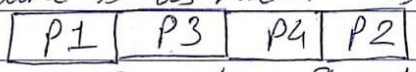
Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

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

Answer:

Process	AT	BT	CT	WT	TAT
P1	0	3	3	0	3
P2	1	5	13	7	12
P3	2	1	4	1	2
P4	3	4	8	1	5

Take it as Non - Preemtive :-



$$\text{Avg. TAT} = \frac{3+12+2+5}{4}$$

$$= \frac{22}{4}$$

$$= 5.5 \text{ ms}$$

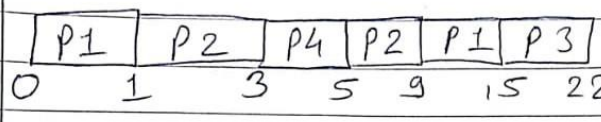
3.Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority):

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.

Answer:

Process	AT	BT	Priority	CT	TAT	WT	lower no. higher priority.
P1	0	6	3	15	15	9	
P2	1	4	1	9	8	4	
P3	2	7	4	22	20	13	
P4	3	2	2	5	2	0	(pre-ordine)



$$\text{Avg. WT} = \frac{9+4+13+0}{4}$$

$$= \frac{26}{4} = 6.5\text{ms}$$

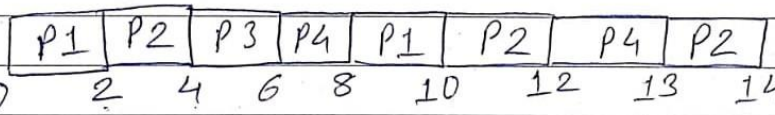
4.Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units:

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.

Answer:

Process	AT	BT	CT	TAT	WT	
P1	0	4	10	10	6	Round Robin
P2	1	5	14	13	8	Time = 2ms
P3	2	2	6	4	2	
P4	3	3	13	10	7	



$$\text{Avg. TAT} = \frac{10+13+4+10}{4} = 9.25\text{ms}$$

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

Answer:

1. The parent process has **x = 5**.
2. The **fork()** system call is executed, creating a child process. At this point, both the parent and child processes have their own copy of **x**, which is still 5.
3. The parent process increments its copy of **x** by 1, so now the parent's **x** becomes 6.
4. The child process also increments its copy of **x** by 1, so the child's **x** becomes 6 as well.

Thus, after both processes have incremented their copies of **x**, the final values will be:

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

The parent process has x = 5.

The fork() system call is executed, creating a child process. At this point, both the parent and child processes have their own copy of x, which is still 5.

The parent process increments its copy of x by 1, so now the parent's x becomes 6.

The child process also increments its copy of x by 1, so the child's x becomes 6 as well.

Thus, after both processes have incremented their copies of x, the final values will be:

Parent process: x = 6

Child process: x = 6