**CDAC MUMBAI**

Concepts of Operating System

**Assignment 2**

**Name: Swati Ashok Jadhav  
Form Number: 241204235**

**Part A**

What will the following commands do?

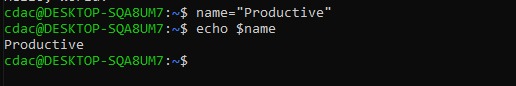
**1] echo "Hello, World!"**

* Prints "Hello, World!" to the terminal.



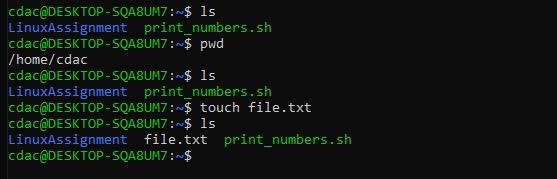
**2] name="Productive"**

* Assigns the string "Productive" to the variable name.



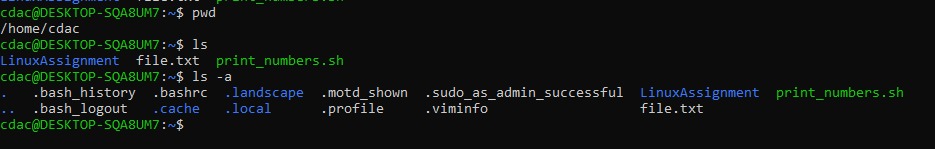
**3] touch file.txt**

* Creates an empty file named file.txt if it doesn't exist or updates its last modified timestamp if it does.



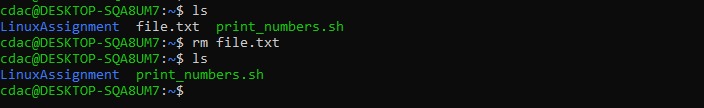
**4] ls -a**

* Lists all files and directories, including hidden ones (starting with .).



**5] rm file.txt**

* Delete file.txt.



**6] cp file1.txt file2.txt**

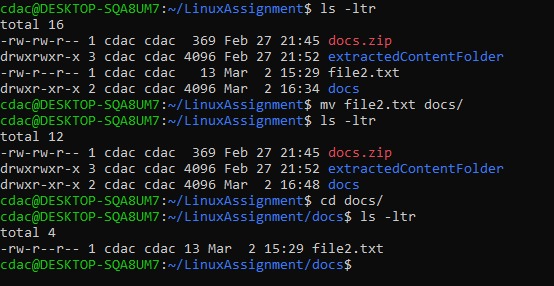
* Copies file1.txt to file2.txt.

A computer screen with text and numbers

AI-generated content may be incorrect.

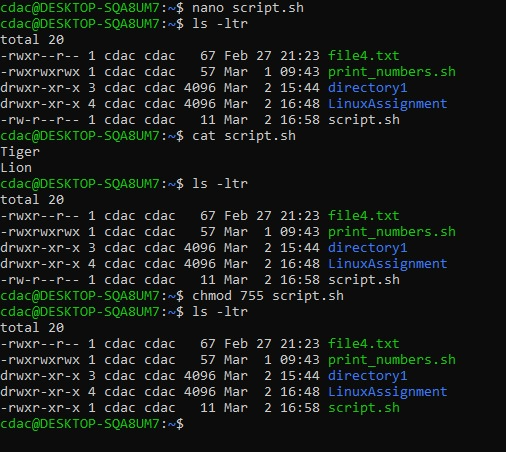
**7] mv file.txt /path/to/directory/**

* Move file.txt to the specified directory.



**8] chmod 755 script.sh**

Grants the owner full permissions (rwx), and read & execute (r-x) permissions to group and others on script.sh.



**9] grep "pattern" file.txt**

Searches for the word "pattern" inside file.txt.

A screen shot of a computer

AI-generated content may be incorrect.

**10] kill PID**

Terminates the process with the given PID.

**A computer screen with white text

AI-generated content may be incorrect.**

**A computer screen with white text

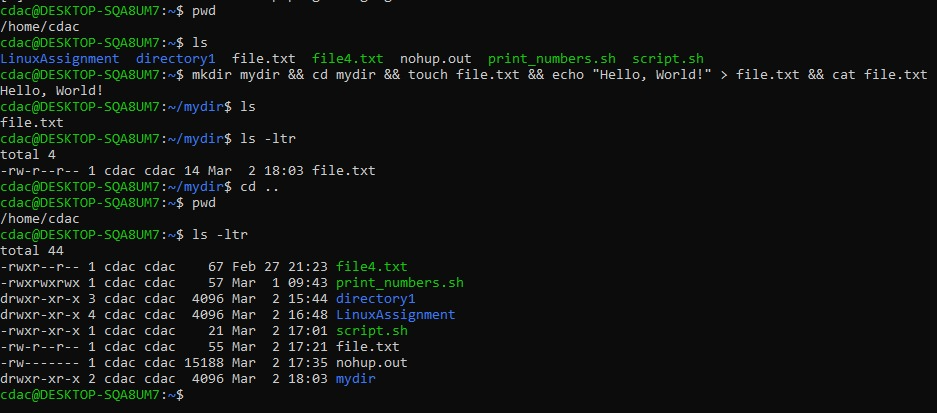
AI-generated content may be incorrect.**

**A computer screen with white text

AI-generated content may be incorrect.**

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

Creates a directory mydir, enters it, creates file.txt, writes "Hello, World!" to file.txt, and displays its content.

****

**12] ls -l | grep ".txt"**

Lists files in long format and filters only those containing .txt.

**A screen shot of a computer

AI-generated content may be incorrect.**

**13] cat file1.txt file2.txt | sort | uniq**

Combines the contents of file1.txt and file2.txt, sorts them, and removes duplicate lines.

A screenshot of a computer program

AI-generated content may be incorrect.

**14] ls -l | grep "^d"**

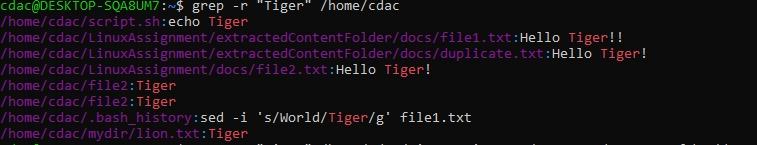
Lists only directories (lines starting with d in ls -l output).

A computer screen with white text and green text

AI-generated content may be incorrect.

**15] grep -r "pattern" /path/to/directory/**

Recursively searches for "pattern" in all files under the given directory.



**16] cat file1.txt file2.txt | sort | uniq -d**

Displays only duplicate lines between file1.txt and file2.txt.

A computer screen shot of a computer program

AI-generated content may be incorrect.

A computer screen with many white and green text

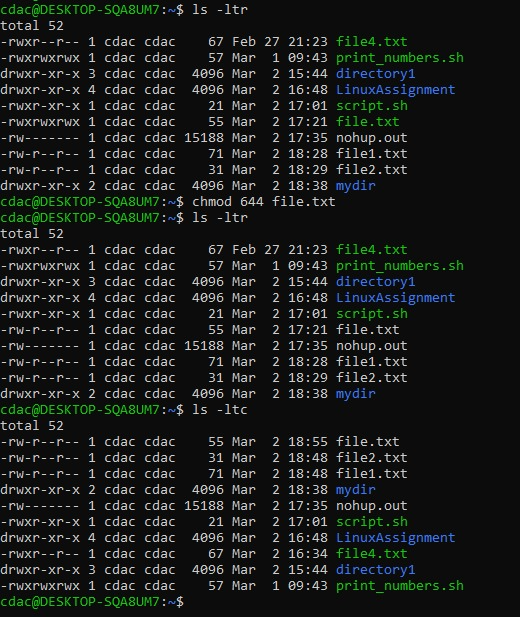
AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

**17] chmod 644 file.txt**

Sets file.txt permissions to rw-r--r-- (read & write for owner, read-only for others).



**18] cp -r source\_directory destination\_directory**

Recursively copies the contents of source\_directory to destination\_directory.

A computer screen shot of a program

AI-generated content may be incorrect.

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

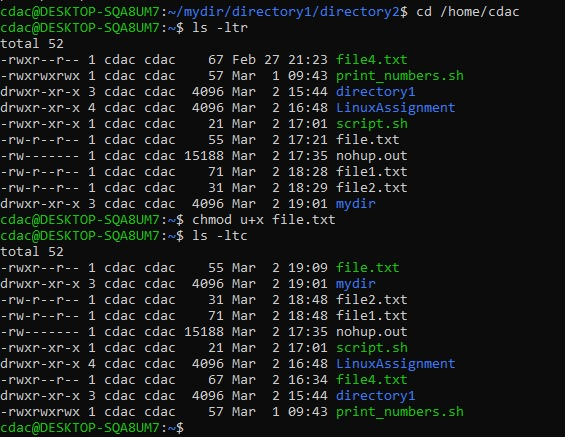
Searches for all .txt files in the specified path.

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**20] chmod u+x file.txt**

Grants the user (u) execute (x) permission on file.txt.

****

**21] echo $PATH**

Displays the system's PATH environment variable.



**Part B**

**Identify True or False:**

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

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

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

4. pwd stands for "print working directory" and displays the current directory. **-- 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 executepermissions to group and others. **-- True**

7. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1 if directory1 does not exist. **-- True**

8. rm -rf file.txt deletes a file forcefully without confirmation. **– True**

**Identify the Incorrect Commands:**

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

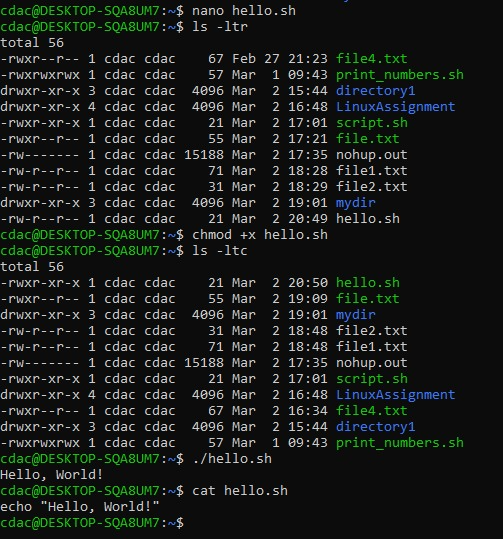
**-- mv**

A computer screen with many green and blue text

AI-generated content may be incorrect.

**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.**

A computer screen with white text

AI-generated content may be incorrect.

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

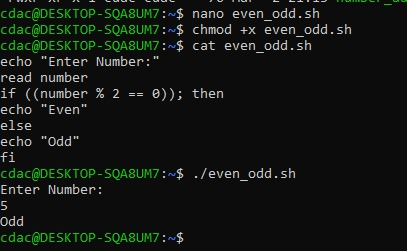
****

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

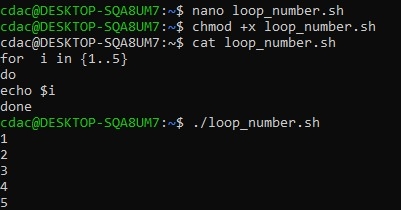
**A computer screen with white text

AI-generated content may be incorrect.**

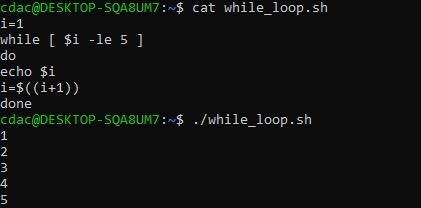
**Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".**

****

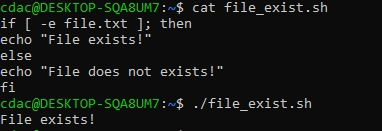
**Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.**

****

**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.**

**A computer screen with white text

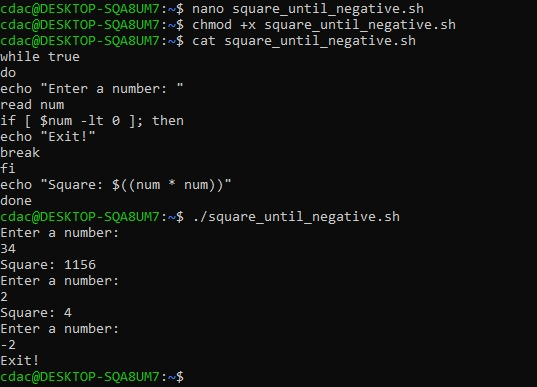
AI-generated content may be incorrect.**

**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.**

**A computer screen with white text

AI-generated content may be incorrect.**

**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**

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

An operating system (OS) helps manage the computer's hardware and software. It controls processes, memory, input/output, and file systems so that everything works together smoothly. Without an OS, the user would have to interact directly with hardware, which is impractical for everyday use.

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

A process is a running program, while a thread is a smaller part of that process that can run at the same time as other threads. Multiple threads in a process can share resources like memory, making tasks run faster and more efficiently. Threads are the basic units of execution within a process.

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

Virtual memory makes it seem like the computer has more RAM than it actually does by using space on the hard drive when RAM is full. This is done by swapping data back and forth between physical memory and storage to ensure the system runs smoothly. It allows programs to use more memory than physically available.

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

Multiprogramming runs multiple programs at once, multitasking allows a single program to do many tasks, and multiprocessing uses multiple CPUs to run many tasks at the same time. These techniques aim to maximize CPU utilization and system efficiency. Multiprogramming increases the efficiency of the system by keeping the CPU busy.

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

A file system organizes files on a computer. It includes directories, files, and the info needed to keep track of them. File systems also define how files are named, stored, and accessed, allowing the OS to keep data organized and retrieve it quickly.

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

A deadlock happens when two or more processes get stuck waiting for each other. To prevent it, we can avoid circular waiting, ensure resources are allocated in a way that prevents deadlock, or use timeouts to avoid indefinite waiting. Deadlocks can freeze a system if not handled properly.

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

The kernel is the main part of the OS that controls the computer, while the shell is the user interface where you type commands. The kernel interacts directly with hardware, while the shell acts as an intermediary between the user and the OS, translating user input into commands the kernel can execute.

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

CPU scheduling is how the OS decides which task gets the CPU next. It’s important for managing time and making sure tasks run smoothly. Effective CPU scheduling helps to maximize performance, minimize delays, and ensure fairness among processes.

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

A system call is when a program asks the OS for help to do something, like opening a file or running a command. It’s a way for user programs to request services that only the OS can provide, such as managing hardware resources or creating new processes.

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

Device drivers let the OS communicate with hardware, like printers or keyboards, by sending the right commands. They act as translators between the OS and hardware, converting high-level instructions from the OS into low-level commands understood by the device.

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

The page table helps translate virtual memory addresses into physical addresses so programs can access data correctly. It is crucial in managing the virtual memory system, allowing the OS to swap data between the hard drive and RAM without confusing the program.

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

Thrashing happens when the system spends too much time swapping data between memory and the disk. Adding more memory can help avoid it. Thrashing makes the system slow because the CPU and memory are too busy managing data instead of running processes.

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

A semaphore is a way to manage access to shared resources between processes to prevent conflicts or errors. Semaphores are critical for ensuring that only one process can access a resource at a time, preventing race conditions.

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

The OS uses locks and semaphores to make sure processes don’t interfere with each other when they need the same resource. Without synchronization, processes might overwrite each other’s data or cause inconsistencies.

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

An interrupt tells the OS to stop what it's doing and take care of something urgent, like user input or hardware requests. Interrupts allow the system to respond to external events promptly without constantly checking for them.

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

A file descriptor is like a number that identifies an open file, so the OS knows which file to read or write. Once a file is opened, the OS assigns it a descriptor that it uses to refer to that file during read/write operations.

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

When a system crashes, the OS uses logs or backups to restore the system to where it was before the crash. The recovery process helps avoid data loss and ensures that the system can start from a consistent state.

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

A monolithic kernel has everything in one big program, while a microkernel keeps only the essential parts, making it more modular. A microkernel can offer better security and stability because it isolates different parts of the OS, reducing the impact of failures.

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

Internal fragmentation is wasted memory inside an allocated space, while external fragmentation is unused space between allocations. Both types of fragmentation can cause inefficient memory usage, but external fragmentation is harder to manage.

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

The OS manages I/O by scheduling tasks and making sure data is transferred properly between the CPU and hardware. I/O management ensures devices are used effectively, without conflict, and that data is transferred in an orderly way.

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

Preemptive scheduling lets the OS stop a process to give time to others, while non-preemptive scheduling lets a process finish before switching. Preemptive scheduling is more efficient in systems where multiple processes need to share the CPU quickly.

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

Round-robin scheduling gives each process a set amount of time to use the CPU, then moves to the next one. It ensures fairness, giving each process equal CPU time and is especially useful in time-sharing systems.

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

Priority scheduling gives higher priority to more important processes, and their priority can be set by the system or the user. This ensures that critical tasks get executed before less important ones.

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

Shortest Job Next (SJN) runs the process that will finish the quickest. It’s good for reducing wait time but needs to know process times in advance, which can be difficult in real-time systems.

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

Multilevel queue scheduling uses different queues for different types of processes, with each queue having its own scheduling rule. It allows efficient management of processes by prioritizing types of work like interactive or batch processes.

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

A Process Control Block (PCB) stores information about a process, like its ID, state, and memory. It helps the OS keep track of each process and ensure the correct execution of tasks.

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

The process state diagram shows different stages a process can be in, like running, waiting, or finished. Transitions occur as processes wait for resources or complete their tasks.

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

Processes communicate using methods like shared memory or message passing, allowing them to exchange data or signals. This helps processes collaborate, share data, or coordinate actions without interfering with each other.

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

Process synchronization makes sure processes don’t mess with each other’s data. It’s important for avoiding errors like race conditions where processes try to access shared resources simultaneously.

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

A zombie process is a finished process that still has an entry in the system because its parent hasn’t collected its exit status yet. It doesn't do anything but take up system resources.

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

Internal fragmentation is wasted memory inside allocated blocks, while external fragmentation is wasted space outside of them. Both affect how efficiently memory is used and can slow down the system.

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

Demand paging loads only the parts of a program into memory when needed, helping manage memory better. It allows large programs to run without taking up too much memory.

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

The page table is used to map virtual memory to physical memory, making sure programs access the correct memory locations. It is key to how virtual memory works and how processes get their data.

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

The Memory Management Unit (MMU) translates virtual memory addresses into physical addresses, helping manage memory access. It allows the OS to protect memory and allocate it effectively to programs.

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

Thrashing happens when the system spends too much time swapping data, slowing it down. More memory or better memory use can prevent this. Thrashing occurs when memory is overloaded and the system can’t keep up.

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

A system call is a way for programs to ask the OS to do something, like create a file or start a process. It allows programs to request system-level services safely.

**Duplicate question 18 and 37.**

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

The OS handles I/O by using drivers and buffers to manage data flow between devices like printers or hard drives. It ensures smooth data transfer by controlling how data is read from and written to devices.

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

A race condition happens when two processes try to access the same resource at the same time. Using locks or semaphores can prevent it by ensuring only one process can use the resource at a time.

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

Device drivers act as translators between the operating system and hardware devices. They convert general OS commands into specific instructions that a device can understand and respond to, enabling smooth communication between the system and peripherals like printers or network adapters.

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

A zombie process is a finished process that still has an entry in the process table because its parent hasn’t collected its exit status. It occurs when the parent process doesn’t call wait() to read the child’s exit status. To prevent zombie processes, the parent can call wait() or use a signal handler to clean up the process after it terminates.

**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, leaving it without a parent. The operating system handles orphan processes by making them children of the init process, which ensures they are properly cleaned up after they finish.

**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, which inherits some of the parent’s resources. The parent manages the child’s execution and can wait for its completion or terminate it. The relationship helps the OS track and manage processes effectively.

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

The fork() system call in Unix-like systems creates a new process by duplicating the parent process. The child process gets an exact copy of the parent’s memory space, and both processes continue executing from the point where fork() was called.

**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 using the wait() system call. This causes the parent to pause execution until the child completes, and it allows the parent to retrieve the child’s exit status.

**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 important because it indicates whether the child completed successfully or encountered an error. The wait() system call uses this exit status to allow the parent process to check the outcome of the child’s execution.

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

A parent process can terminate a child process by sending it a signal, such as SIGKILL or SIGTERM. The signal causes the child process to stop, allowing the parent to manage resources and prevent orphan processes.

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

In Unix-like systems, a process group is a set of processes that share the same process group ID, typically used for managing signals. A session is a collection of one or more process groups and is used to manage terminal control and job control, especially in shell environments.

**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 one. It loads a new program into the process’s memory space, effectively replacing the running program without creating a new process. This is useful for running different programs within the same process.

**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 finish. Unlike wait(), which waits for any child, waitpid() provides more control, allowing the parent to specify which child process to wait for, and can also allow non-blocking operation.

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

Process termination in Unix-like systems occurs when a process completes its task or is terminated by a signal. The OS removes the process from the process table and deallocates its resources. The parent process can retrieve the exit status of the terminated process.

**52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?**

The long-term scheduler controls the degree of multiprogramming by deciding which processes should be admitted into the system for execution. It makes decisions about which processes to bring into memory and when, balancing the load and ensuring efficient resource use.

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

The short-term scheduler, or CPU scheduler, makes decisions frequently (milliseconds) and determines which process runs on the CPU next. Unlike the long-term scheduler, which controls overall system load, the short-term scheduler focuses on the immediate task at hand.

**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 is invoked when processes are swapped in and out of memory to manage the system’s memory more effectively. For example, if the system is overloaded, the medium-term scheduler may suspend some processes and swap them out to free up space for others.

**Part E**

A white background with black text

AI-generated content may be incorrect.

**Completion Time**

Process Arrival Time Burst Time Completion Time

P1 0 5 5

P2 1 3 8

P3 2 6 14

**Finding Turnaround Time (TAT)**

Process Completion Time Arrival Time Turnaround Time (TAT)

P1 5 0 5 - 0 = 5

P2 8 1 8 - 1 = 7

P3 14 2 14 - 2 = 12

**Finding Waiting Time (WT)**

Process Turnaround Time (TAT) Burst Time Waiting Time (WT)

P1 5 5 5 - 5 = 0

P2 7 3 7 - 3 = 4

P3 12 6 12 - 6 = 6

(0+4+6)/3=10/3=3.33

**A white background with black text

AI-generated content may be incorrect.**

**Process Execution Order (Non-Preemptive SJF)**

The execution order is:  
P1 → P3 → P4 → P2

**Finding Completion Time**

Process Arrival Time Burst Time Completion Time

P1 0 3 3

P3 2 1 4

P4 3 4 8

P2 1 5 13

**Finding Turnaround Time (TAT)**

Process Completion Time Arrival Time Turnaround Time (TAT)

P1 3 0 3 - 0 = 3

P3 4 2 4 - 2 = 2

P4 8 3 8 - 3 = 5

P2 13 1 13 - 1 = 12

(3+2+5+12)/4=22/4=5.5

A white background with black text

AI-generated content may be incorrect.

The execution order is:  
P1 → P2 → P4 → P3

**Finding Completion Time**

Process Arrival Time Burst Time Priority Completion Time

P1 0 6 3 6

P2 1 4 1 10

P4 3 2 2 12

P3 2 7 4 19

**Finding Waiting Time (WT)**

Process Completion Time Arrival Time Burst Time Waiting Time (WT)

P1 6 0 6 6 - 0 - 6 = 0

P2 10 1 4 10 - 1 - 4 = 5

P4 12 3 2 12 - 3 - 2 = 7

P3 19 2 7 19 - 2 - 7 = 10

(0+5+7+10)/4=22/4=5.5ms

A white background with black text

AI-generated content may be incorrect.

Execution Order:  
P1 → P2 → P3 → P4 → P1 → P2 → P4 → P2

**Calculating Completion Time**

Process Completion Time

P1 10

P2 14

P3 6

P4 13

**Calculate Turnaround Time**

Process Completion Time Arrival Time Turnaround Time (TAT)

P1 10 0 10 - 0 = 10

P2 14 1 14 - 1 = 13

P3 6 2 6 - 2 = 4

P4 13 3 13 - 3 = 10

(10+13+4+10)/4=37/4=9.25ms

**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?**

Before fork():

-Parent process: x = 5

After fork():

-Parent process has x = 5

-Child process has a separate copy of x = 5

Both increment x by 1:

-Parent updates its x → x = 6

-Child updates its x → x = 6

Final value:

**Parent process: x = 6**

**Child process: x = 6**