# CDAC MUMBAI

Concepts of Operating System Assignment 2

## Part A

What will the following commands do?

- echo "Hello, World!"

➔ Prints “Hello, World!” to the terminal.

- name="Productive"

➔ assign value “productive” to the variable name.

- touch file.txt

➔ create a new file.

- ls -a

➔ list all files and directories.

- rm file.txt

➔ delete the file.

- cp file1.txt file2.txt

➔ copy the content of file1 to file2.

- mv file.txt /path/to/directory/

➔ move the file to above given path.

- chmod 755 script.sh

➔ this command changes the permission of script.sh file to allow owner to read, write, execute and group and others to read and execute.

- grep "pattern" file.txt

➔ searches for the pattern in above file.

- 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  ls -l | grep ".txt"

➔ creates directory and after navigating to it creare new file that prints hello world and then print content of that file.

- cat file1.txt file2.txt | sort | uniq

➔ concatenates the content of file1 to file then sorts the output and remove duplicates.

- ls -l | grep "^d"

➔ list all files and directories and searches for directories.

- grep -r "pattern" /path/to/directory/

➔ searches for pattern in the specified directory.

- cat file1.txt file2.txt | sort | uniq –d

➔ Concatenates the contents of file1.txt and file2.txt, sorts the output, removes duplicate lines, and only prints the duplicate lines.

- chmod 644 file.txt

➔ Changes the permissions of file.txt to allow the owner to read and write, and the group and others to read.

- cp -r source_directory destination_directory

➔ copies the contents of source directory to destination directory.

- find /path/to/search -name "*.txt"

➔ Searches for files with the .txt extension within the specified directory.

- chmod u+x file.txt

➔ adds execute permission to the above file.

- echo $PATH

➔ prints the value of path.

# Part B

Identify True or False:

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

➔ True

2. mv is used to move files and directories.

➔ True

3. cd is used to copy files and directories.

➔ False

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

➔ True

5. grep is used to search for patterns in files.

➔ True

6. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to group and others.

➔ True

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

➔ True

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

➔ False

Identify the Incorrect Commands:

1. chmodx is used to change file permissions.

➔ chmodx - chmod

2. cpy is used to copy files and directories.

➔ cpy – cp

3. mkfile is used to create a new file.

➔ mkfile – touch

4. catx is used to concatenate files.

➔ cats -> cat

5. rn is used to rename files.

➔ rn -> mv

# Part C

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

➔ echo "Hello, World!"

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

➔ name="CDAC Mumbai"; echo $name

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

➔ read -p "Enter a number: " num; echo $num

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

➔ num1=5; num2=3; sum=$((num1 + num2)); echo $sum

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

➔ read -p "Enter a number: " num; if (( num % 2 == 0 )); then echo "Even"; else echo "Odd"; fi

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

➔ for i in {1..5}; do echo $i; done

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

➔ i=1; while (( i <= 5 )); do echo $i; ((i++)); done

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".

➔ if [ -f "file.txt" ]; then echo "File exists"; else echo "File does not exist"; fi

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.

➔ read -p "Enter a number: " num; if (( num > 10 )); then echo "Number is greater than 10"; else echo "Number is less than or equal to 10"; fi

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.

➔
```
for i in {1..5}; do  for j in {1..5}; do  printf "%4d" $((i*j)) done
echo done
```

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.

➔
```
while true; do  read -p "Enter a number: " num  if (( num < 0 )); then
break fi echo "Square of $num is $((num**2))" done
```

# Part D

Common Interview Questions (Must know)

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

➔ An operating system (OS) manages computer hardware resources and provides a platform for running application software. Its primary functions include process management, memory management, file management, I/O management, and security.

2. Explain the difference between process and thread.

➔ A process is an independent unit of execution, with its own memory space and resources. A thread is a lightweight process that shares the same memory space and resources as other threads within the same process.

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

➔ Virtual memory is a memory management technique that enables a computer to use both physical RAM and hard drive storage to provide more memory than is physically available. It works by dividing the program into pages and swapping them between RAM and disk storage as needed.

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

➔ Multiprogramming allows multiple programs to run concurrently by switching between them. Multitasking allows multiple tasks to run concurrently within the same program. Multiprocessing uses multiple processors to execute multiple programs or tasks concurrently.

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

➔ A file system is a way of organizing and storing files on a computer. Its components include files, directories, file names, and metadata.

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

➔ A deadlock occurs when two or more processes are blocked indefinitely, each waiting for the other to release a resource. Deadlocks can be prevented by ensuring that resources are allocated in a way that avoids circular waits.

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

➔ A kernel is the core part of an operating system that manages hardware resources and provides basic services to applications. A shell is a program that provides a user interface to interact with the operating system and run commands.

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

➔ CPU scheduling is the process of allocating the CPU to different processes or threads. It is important because it helps to optimize system performance, ensure fairness, and prevent starvation.

9. How does a system call work?

➔ A system call is a request from an application to the operating system to perform a specific service, such as process creation or file access. The application makes the request through a library function, which then invokes the system call.

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

➔ Device drivers act as an interface between the operating system and hardware devices, allowing the operating system to communicate with and control the devices.

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

➔ he page table is a data structure used by the operating system to keep track of the mapping between virtual addresses and physical addresses in memory.

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

➔ Thrashing occurs when the operating system spends more time swapping pages between RAM and disk storage than executing actual program instructions. Thrashing can be avoided by using techniques such as page replacement algorithms and disk caching.

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

➔ A semaphore is a variable used to control access to a shared resource by multiple processes. It can be used to implement synchronization mechanisms such as mutual exclusion and condition variables.

14. How does an operating system handle process synchronization?

➔ An operating system handles process synchronization using various mechanisms such as semaphores, monitors, and message passing.

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

➔ An interrupt is a signal to the operating system that an event has occurred, such as a hardware device completing an operation. The operating system responds to the interrupt by executing an interrupt handler.

16. Explain the concept of a file descriptor.

➔ A file descriptor is a unique integer assigned to an open file or device, used by the operating system to identify and manage the file.

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

➔ A system can recover from a crash by using techniques such as checkpointing, journaling, and rebooting.

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

➔ A monolithic kernel is a single, self-contained kernel that provides all operating system services. A microkernel is a smaller kernel that provides only basic services, with other services provided by user-level applications.

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

➔ Internal fragmentation occurs when a block of memory is allocated but not fully used. External fragmentation occurs when free memory is broken into small, non-contiguous blocks.

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

➔ An operating system manages I/O operations using various techniques such as buffering, caching, and interrupt-driven I/O.

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

➔ Preemptive scheduling allows the operating system to interrupt a running process and allocate the CPU to another process.

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

➔ Round-robin scheduling is a CPU scheduling algorithm that assigns a fixed time slice (called a time quantum) to each process in a circular order. Each process is given a chance to execute for the time quantum, and then the CPU is allocated to the next process.

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

➔ Priority scheduling is a CPU scheduling algorithm that assigns a priority to each process. The process with the highest priority is executed first. Priority can be assigned based on factors such as process type, memory requirements, and user-defined priorities.

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

➔ The shortest job next (SJN) scheduling algorithm is a CPU scheduling algorithm that executes the process with the shortest burst time (CPU time required) first. SJN is used in systems where the burst time of each process is known in advance.

25. Explain the concept of multilevel queue scheduling.

➔ Multilevel queue scheduling is a CPU scheduling algorithm that uses multiple queues to hold processes with different priorities. Each queue has its own scheduling algorithm, and the queues are served in a priority order.

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

➔ A process control block (PCB) is a data structure that contains information about a process, including its process ID, program counter, registers, memory allocation, and priority.

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

➔ The process state diagram shows the different states a process can be in, including new, ready, running, waiting, and zombie. The transitions between these states are triggered by events such as process creation, CPU allocation, I/O completion, and process termination.

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

➔ Processes can communicate with each other using inter-process communication (IPC) mechanisms, such as pipes, sockets, shared memory, and message queues.

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

➔ Process synchronization is the coordination of multiple processes to ensure that they access shared resources in a safe and efficient manner. Process synchronization is important because it prevents problems such as data inconsistency, deadlock, and starvation.

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

➔ A zombie process is a process that has finished execution but still exists in the system because its parent process has not yet acknowledged its termination. A zombie process is created when a process terminates, but its parent process does not call the wait() system call to retrieve its exit status.

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

➔ Internal fragmentation occurs when a block of memory is allocated but not fully used. External fragmentation occurs when free memory is broken into small, non-contiguous blocks.

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

➔ Demand paging is a memory management technique that loads pages into memory only when they are actually needed. Demand paging improves memory management efficiency by reducing the amount of memory needed to run a program.

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

➔ The page table is a data structure that maps virtual addresses to physical addresses in memory. The page table is used to translate virtual addresses used by a program into physical addresses in memory.

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

➔ A memory management unit (MMU) is a hardware component that translates virtual addresses to physical addresses in memory. The MMU uses a page table to perform the translation.

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

➔ Thrashing occurs when the operating system spends more time swapping pages between RAM and disk storage than executing actual program instructions. Thrashing can be avoided by using techniques such as page replacement algorithms and disk caching.

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

➔ A system call is a request from a user program to the operating system to perform a specific service, such as process creation or file access. System calls facilitate communication between user programs and the operating system by providing a well-defined interface for requesting services.

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

➔ A monolithic kernel is a single, self-contained kernel that provides all operating system services. A microkernel is a smaller kernel that provides only basic services, with other services provided by user-level applications.

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

➔ An operating system handles I/O operations using various techniques such as buffering, caching, and interrupt-driven I/O.

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

➔ A race condition occurs when multiple processes access shared data and the outcome depends on the relative timing of their accesses. To prevent race conditions, synchronization mechanisms such as locks, semaphores, and monitors can be used to coordinate access to shared data.

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

➔ Device drivers act as an interface between the operating system and hardware devices, allowing the operating system to communicate with and control the 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 finished execution but still exists in the system because its parent process has not yet acknowledged its termination. Zombie processes can be prevented by ensuring that parent processes properly wait for their child processes to finish execution.

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

➔ An orphan process is a process that is no longer associated with a parent process. Operating systems typically handle orphan processes by assigning them to a special process, such as the init process, which waits for them to finish 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 using the fork() system call. The child process inherits many of the parent's attributes, such as its memory space and open files.

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 process. The new process, called the child process, is a copy of the parent process.

45. Describe how a parent process can wait for a child process to finish execution.
➔ A parent process can wait for a child process to finish execution using the wait() system call. The wait() call blocks the parent process until the child process terminates.

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 indicates whether the process terminated normally or abnormally. The wait() system call returns the exit status of the child process to the parent process.

47. How can a parent process terminate a child process in Unix-like operating systems?
➔ A parent process can terminate a child process using the kill() system call. The kill() call sends a signal to the child process, which can cause it to terminate.

48. Explain the difference between a process group and a session in Unix-like operating systems.
➔ A process group is a collection of processes that can receive signals from the same source. A session is a collection of process groups that can be controlled together.

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 by loading a new program into memory and starting its execution.

50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?
➔ The waitpid() system call is used to wait for a specific child process to finish execution. It differs from wait() in that it allows the parent process to wait for a specific child process, rather than any child process.

51. How does process termination occur in Unix-like operating systems?
➔ Process termination occurs when a process receives a signal that causes it to exit, or when it completes its execution and calls the exit() system call.

52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?
➔ The long-term scheduler is responsible for selecting which processes to admit to the ready queue. It influences the degree of multiprogramming by controlling the number of processes that are allowed to compete for the CPU.

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 executes frequently and makes decisions about which process to execute next. The long-term scheduler executes less frequently and makes decisions about which processes to admit to the ready queue. The medium-term scheduler executes at a frequency between the short-term and long-term schedulers and makes decisions about which processes to swap out of memory.

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

➔ The medium-term scheduler would be invoked when the system is running low on memory and needs to swap out some processes to make room for others. The medium-term scheduler helps manage system resources more efficiently by selecting which processes to swap out based on factors such as their memory usage and CPU requirements.

# 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.

| ➔ **Process** | ➔ **Arrival Time** | ➔ **Burst Time** | ➔ **Waiting Time** |
|---|---|---|---|
| ➔ P1 | ➔ 0 | ➔ 5 | ➔ 0 |
| ➔ P2 | ➔ 1 | ➔ 3 | ➔ 5 |
| ➔ P3 | ➔ 2 | ➔ 6 | ➔ 8 |
| ➔ Average Waiting Time = (0 + 5 + 8) / 3 = 13 / 3 = 4.33 | ➔ | ➔ | ➔ |

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.

| ➔ **Process** | ➔ **Arrival Time** | ➔ **Burst Time** | ➔ **Turnaround Time** |
|---|---|---|---|
| ➔ P1 | ➔ 0 | ➔ 3 | ➔ 3 |

| ➔ **Process** | ➔ **Arrival Time** | ➔ **Burst Time** | ➔ **Turnaround Time** |
|---|---|---|---|
| ➔ P3 | ➔ 2 | ➔ 1 | ➔ 3 |
| ➔ P2 | ➔ 1 | ➔ 5 | ➔ 8 |
| ➔ P4 | ➔ 3 | ➔ 4 | ➔ 7 |
| ➔ Average Turnaround Time = (3 + 3 + 8 + 7) / 4 = 21 / 4 = 5.25 | ➔ | ➔ | ➔ |

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.

| ➔ **Process** | ➔ **Arrival Time** | ➔ **Burst Time** | ➔ **Priority** | ➔ **Waiting Time** |
|---|---|---|---|---|
| ➔ P2 | ➔ 1 | ➔ 4 | ➔ 1 | ➔ 0 |
| ➔ P4 | ➔ 3 | ➔ 2 | ➔ 2 | ➔ 5 |
| ➔ P1 | ➔ 0 | ➔ 6 | ➔ 3 | ➔ 6 |
| ➔ P3 | ➔ 2 | ➔ 7 | ➔ 4 | ➔ 11 |
| ➔ Average Waiting Time = (0 + 5 + 6 + 11) / 4 = 22 / 4 = 5.5 | ➔ | ➔ | ➔ | ➔ |

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.

| ➔ **Process** | ➔ **Arrival Time** | ➔ **Burst Time** | ➔ **Turnaround Time** |
|---|---|---|---|
| ➔ P1 | ➔ 0 | ➔ 4 | ➔ 6 |
| ➔ P2 | ➔ 1 | ➔ 5 | ➔ 9 |
| ➔ P3 | ➔ 2 | ➔ 2 | ➔ 4 |
| ➔ P4 | ➔ 3 | ➔ 3 | ➔ 6 |

| ➔ **Process** | ➔ **Arrival Time** | ➔ **Burst Time** | ➔ **Turnaround Time** |
|---|---|---|---|
| ➔ Average Turnaround Time = (6 + 9 + 4 + 6) / 4 = 25 / 4 = 6.25 | ➔ | ➔ | ➔ |

5. 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?

➔ Parent Process: x = 6
Child Process: x = 6

Submission Guidelines:

- Document each step of your solution and any challenges faced.
- Upload it on your GitHub repository

Additional Tips:

- Experiment with different options and parameters of each command to explore their functionalities.
- This assignment is tailored to align with interview expectations, CCEE standards, and industry demands.
- If you complete this then your preparation will be skyrocketed.