

CDAC MUMBAI
Concepts of Operating System
Assignment 2

Submitted By: Mallikarjun Gundappa Konchurkar_JH

Part A

Question: What will the following commands do?

1. echo "Hello, World!"

Ans : Prints the text "Hello, World!" to the terminal.

2. name="Productive"

Ans : Assigns the value "Productive" to the variable name.

3. touch file.txt

Ans : Creates a new empty file named file.txt in the current directory if it does not already exist. If it exists, it updates its timestamp.

4. ls -a

Ans : Lists all files and directories in the current directory, including hidden ones (those starting with a dot).

5. rm file.txt

Ans : Deletes the file named file.txt.

6. cp file1.txt file2.txt

Ans : Copies the contents of file1.txt to a new file named file2.txt.

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

Ans : Moves the file.txt to the specified directory (/path/to/directory/).

8. chmod 755 script.sh

Ans : Changes the permissions of the script.sh file to rwxr-xr-x (owner can read/write/execute, others can read/execute).

9. grep "pattern" file.txt

Ans : Searches for the specified pattern (text) within file.txt and prints the matching lines.

10. kill PID

Ans : Terminates the process with the specified Process ID (PID)

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

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

12. ls -l | grep ".txt"

Ans : Lists all files in the current directory with detailed information and filters the output to show only .txt files.

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

Ans : Concatenates the contents of file1.txt and file2.txt, sorts the lines, and then removes duplicate lines (outputs only unique lines).

14. ls -l | grep "^d"

Ans : Lists all files and directories with detailed information, and filters the output to show only directories (lines starting with "d").

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

Ans : Recursively searches for the pattern in all files within the specified directory and its subdirectories.

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

Ans : Concatenates the contents of file1.txt and file2.txt, sorts the lines, and then filters out only the duplicate lines (lines that appear more than once).

17. chmod 644 file.txt

Ans : Changes the permissions of file.txt to rw-r--r-- (owner can read/write, others can read).

18. cp -r source_directory destination_directory

Ans : Recursively copies all contents of source_directory to destination_directory.

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

Ans : Searches for all files with the .txt extension within the specified directory and subdirectories.

20. chmod u+x file.txt

Ans : Adds execute permissions to the file.txt file for the file's owner (user).

21. echo \$PATH

Ans : Displays the current system's PATH variable, which contains a list of directories where executable programs are located.

Part B

Que: Identify True or False:

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

Ans : True

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

Ans : True

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

Ans : False **cd** is used to change the current working directory.

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

Ans : True

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

Ans : True

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

Ans : True

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

Ans : True

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

Ans : True

Question: Identify the Incorrect Commands:

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

Ans : The correct command is **chmod**

2. **cpy** is used to copy files and directories.

Ans : The correct command is **cp**

3. **mkfile** is used to create a new file.

Ans : The correct command is **touch**, not **mkfile**. While **mkfile** can exist on some systems, it's not a standard Linux command for creating files.

4. **catx** is used to concatenate files.

Ans : The correct command is **cat**

5. **rn** is used to rename files.

Ans : The correct command is **mv**

Part C

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

Ans:

```
mallikarjun@MAYUR: ~/assign  X + v
mallikarjun@MAYUR:~$ mkdir assignment2
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano hello.sh
mallikarjun@MAYUR:~/assignment2$ cat hello.sh
echo "hello, world!"
mallikarjun@MAYUR:~/assignment2$ bash hello.sh
hello, world!
mallikarjun@MAYUR:~/assignment2$ |
```

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

Ans:

```
mallikarjun@MAYUR: ~/assign  X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano name.sh
mallikarjun@MAYUR:~/assignment2$ cat name.sh
name="CDAC Mumbai"
echo $name

mallikarjun@MAYUR:~/assignment2$ bash name.sh
CDAC Mumbai
mallikarjun@MAYUR:~/assignment2$
```

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

Ans:

```
mallikarjun@MAYUR: ~/assign  X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que3.sh
mallikarjun@MAYUR:~/assignment2$ cat que3.sh
echo "Enter a Number"
read a
echo Your Number is $a
mallikarjun@MAYUR:~/assignment2$ bash que3.sh
Enter a Number
56
Your Number is 56
mallikarjun@MAYUR:~/assignment2$ |
```

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

Ans:

```
mallikarjun@MAYUR: ~/assign X + v
mallikarjun@MAYUR:~/assignment2$ nano que4.sh
mallikarjun@MAYUR:~/assignment2$ cat que4.sh
echo "Enter a Number"
read a
echo "Enter a Number"
read b
sum=$(expr $a + $b)
echo "Sum of $a and $b is $sum"
mallikarjun@MAYUR:~/assignment2$ bash que4.sh
Enter a Number
5
Enter a Number
3
Sum of 5 and 3 is 8
mallikarjun@MAYUR:~/assignment2$
```

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

Ans:

```
mallikarjun@MAYUR: ~/assign X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que5.sh
mallikarjun@MAYUR:~/assignment2$ cat que5.sh
echo "Enter a number:"
read number
if [ $((number % 2)) -eq 0 ]; then
    echo "Even"
else
    echo "Odd"
fi
mallikarjun@MAYUR:~/assignment2$ bash que5.sh
Enter a number:
9
Odd
mallikarjun@MAYUR:~/assignment2$ bash que5.sh
Enter a number:
2
Even
mallikarjun@MAYUR:~/assignment2$
```

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

Ans:

```
mallikarjun@MAYUR: ~/assigng  X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que6.sh
mallikarjun@MAYUR:~/assignment2$ cat que6.sh
for i in {1..5}
do
    echo $i
done
mallikarjun@MAYUR:~/assignment2$ bash que6.sh
1
2
3
4
5
mallikarjun@MAYUR:~/assignment2$
```

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

Ans:

```
mallikarjun@MAYUR: ~/assigng  X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que7.sh
mallikarjun@MAYUR:~/assignment2$ cat que7.sh
# Initialize the counter variable
i=1

# Using a while loop to print numbers from 1 to 5
while [ $i -le 5 ]
do
    echo $i
    ((i++)) # Increment the counter
done
mallikarjun@MAYUR:~/assignment2$ bash que7.sh
1
2
3
4
5
mallikarjun@MAYUR:~/assignment2$
```

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".

Ans:

```
mallikarjun@MAYUR: ~/assign X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que8.sh
mallikarjun@MAYUR:~/assignment2$ cat que8.sh
# Check if file.txt exists in the current directory
if [ -e "file.txt" ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
mallikarjun@MAYUR:~/assignment2$ bash que8.sh
File does not exist
mallikarjun@MAYUR:~/assignment2$
```

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.

Ans:

```
mallikarjun@MAYUR: ~/assign X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que9.sh
mallikarjun@MAYUR:~/assignment2$ cat que9.sh
# Prompt the user to enter a number
echo "Enter a number:"
read number

# Check if the number is greater than 10
if [ $number -gt 10 ]; then
    echo "The number is greater than 10."
else
    echo "The number is not greater than 10."
fi
mallikarjun@MAYUR:~/assignment2$ bash que9.sh
Enter a number:
6
The number is not greater than 10.
mallikarjun@MAYUR:~/assignment2$ bash que9.sh
Enter a number:
78
The number is greater than 10.
mallikarjun@MAYUR:~/assignment2$ |
```

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.

Ans:

```
mallikarjun@MAYUR: ~/assigni X + v
mallikarjun@MAYUR:~$ cd assignment2
mallikarjun@MAYUR:~/assignment2$ nano que10.sh
mallikarjun@MAYUR:~/assignment2$ cat que10.sh
# Print the multiplication table from 1 to 5
for i in {1..5}
do
    for j in {1..5}
    do
        # Print the product of i and j, with formatting for alignment
        printf "%-4d" $($i * $j)
    done
    # Print a newline after each row
    echo ""
done
mallikarjun@MAYUR:~/assignment2$ bash que10.sh
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
mallikarjun@MAYUR:~/assignment2$
```


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.

Ans:

```
mallikarjun@MAYUR: ~/assigng X + v
mallikarjun@MAYUR:~/assignment2$ cat que11.sh
# Loop to read numbers from the user
while true
do
    # Prompt the user to enter a number
    echo "Enter a number (negative number to exit):"
    read number

    # Check if the number is negative
    if [ $number -lt 0 ]; then
        break # Exit the loop if a negative number is entered
    fi

    # Calculate and print the square of the number if it is positive
    square=$((number * number))
    echo "The square of $number is $square"
done

echo "You entered a negative number, exiting the program."
mallikarjun@MAYUR:~/assignment2$ bash que11.sh
Enter a number (negative number to exit):
10
The square of 10 is 100
Enter a number (negative number to exit):
7
The square of 7 is 49
Enter a number (negative number to exit):
-6
You entered a negative number, exiting the program.
mallikarjun@MAYUR:~/assignment2$ |
```

Part D

Common Interview Questions (Must know)

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

Ans : An operating system (OS) is system software that manages hardware resources and provides services for computer programs. Its primary functions include:

- **Process management:** Scheduling and controlling processes.
- **Memory management:** Allocating and deallocating memory.
- **File system management:** Organizing and managing data storage.
- **Device management:** Controlling hardware devices.
- **Security and access control:** Ensuring proper access to resources.

2. Explain the difference between process and thread.

Ans : Process: A program in execution, having its own memory space.

Thread: A lightweight process that shares the same memory space with other threads in the same process. Multiple threads can exist within a single process.

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

Ans : Virtual memory is an abstraction that allows programs to use more memory than physically available by swapping data between RAM and disk storage. It works by using a page table to map virtual addresses to physical addresses.

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

Ans :

- **Multiprogramming:** Running multiple programs simultaneously by switching between them.
- **Multitasking:** The OS executes multiple tasks (processes or threads) concurrently.
- **Multiprocessing:** Use of multiple processors (CPUs) to run tasks in parallel.

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

Ans : A file system is a method for storing and organizing computer files. Its components include:

- **File control block (FCB):** Information about files.
- **Directories:** Organize files.
- **Data blocks:** Store actual file data.

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

Ans : A deadlock occurs when two or more processes are blocked forever, waiting for each other. It can be prevented by using techniques like resource allocation graphs or the Banker's algorithm.

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

Ans : Kernel: The core part of the OS that manages hardware and system resources.

Shell: A user interface that allows users to interact with the OS.

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

Ans : CPU scheduling is the method the OS uses to decide which process runs next on the CPU. It is important for optimizing CPU usage and system performance.

9. How does a system call work?

Ans : A system call is a request from a program to the OS for services such as file manipulation, process control, or I/O operations.

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

Ans : Device drivers are programs that enable the operating system to communicate with hardware devices (e.g., printers, hard drives).

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

Ans : A page table maps virtual memory addresses to physical memory addresses, enabling efficient memory management.

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

Ans : Thrashing occurs when the OS spends more time swapping data between disk and memory than executing processes. It can be avoided by managing the degree of multiprogramming or using page replacement algorithms.

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

Ans : A semaphore is a synchronization tool used to control access to a shared resource by multiple processes in concurrent programming.

14. How does an operating system handle process synchronization?

Ans : The OS ensures that processes coordinate and avoid conflicts when accessing shared resources, typically using locks, semaphores, or monitors.

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

Ans : An interrupt is a signal to the processor indicating an event that needs immediate attention, allowing the OS to pause the current process and handle the interrupt.

16. Explain the concept of a file descriptor.

Ans : A file descriptor is a unique identifier used by the OS to access an open file or other input/output resources.

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

Ans : After a system crash, the OS typically uses a technique like journaling to recover data or performs a file system check to restore consistency.

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

Ans : Monolithic Kernel: A large, single kernel that directly controls hardware and system resources.

Microkernel: A smaller kernel that only provides essential services, with other services running in user space.

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

Ans : Internal Fragmentation: Wasted space within allocated memory blocks.

External Fragmentation: Wasted space between allocated memory blocks.

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

Ans : The OS manages I/O operations through device drivers, buffering, and scheduling to ensure efficient data transfer.

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

Ans : Preemptive: The OS can interrupt a running process to allocate CPU time to another process.

Non-preemptive: A process runs until it voluntarily relinquishes control.

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

Ans : Round-robin scheduling is a preemptive scheduling algorithm where each process is assigned a fixed time slice (quantum) to run before being swapped out.

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

Ans : Processes are assigned priorities, and the OS executes the process with the highest priority. Priorities can be static or dynamic.

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

Ans : SJN schedules the process with the shortest burst time next. It is ideal for minimizing average waiting time but requires knowing process durations in advance.

25. Explain the concept of multilevel queue scheduling.

Ans : Processes are divided into multiple queues based on priority or other criteria. Each queue may have its own scheduling algorithm.

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

Ans : A PCB is a data structure containing information about a process, such as its state, program counter, CPU registers, memory limits, and I/O status.

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

Ans : A diagram illustrating the different states a process can be in (e.g., running, waiting, ready, terminated) and transitions between them.

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

Ans : Processes communicate through mechanisms such as pipes, message queues, shared memory, or sockets.

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

Ans : Process synchronization ensures that processes do not interfere with each other when accessing shared resources, typically using locks, semaphores, or condition variables.

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

Ans : A zombie process is a process that has completed execution but still has an entry in the process table. It occurs when the parent doesn't read the exit status.

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

Ans : Internal Fragmentation: Wasted space within allocated memory blocks.

External Fragmentation: Wasted space between allocated memory blocks.

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

Ans : Demand paging is a memory management technique where pages are only loaded into memory when needed, improving memory efficiency.

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

Ans : The page table maps virtual memory addresses to physical memory addresses, enabling efficient memory management.

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

Ans : The MMU is a hardware component that maps virtual addresses to physical addresses using the page table, enabling virtual memory.

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

Ans : Thrashing occurs when the system spends too much time swapping between disk and memory. It can be avoided by controlling the degree of multiprogramming or using efficient page replacement algorithms.

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

Ans : A system call allows user programs to request services such as file manipulation, process control, or I/O operations from the OS.

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

Ans : Monolithic Kernel: A large, single kernel that directly controls hardware and system resources.

Microkernel: A smaller kernel that only provides essential services, with other services running in user space.

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

Ans : The OS manages I/O operations through device drivers, buffering, and scheduling to ensure efficient data transfer.

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

Ans : A race condition occurs when the outcome of a program depends on the order of execution of concurrent processes. It can be prevented using synchronization techniques like locks.

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

Ans : Device drivers enable the operating system to communicate with hardware devices (e.g., printers, hard drives).

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

Ans : A zombie process is a process that has finished execution but still has an entry in the process table. It occurs when the parent doesn't read the exit status. It can be prevented by ensuring that the parent process collects the exit status using `wait()`.

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

Ans : An orphan process is a process whose parent has terminated. The OS typically assigns a new parent (`init` process) to handle orphan processes.

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

Ans : The parent process creates child processes. The child inherits certain attributes from the parent and can communicate with it.

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

Ans : The `fork()` system call creates a new child process by duplicating the parent process.

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

Ans : The parent process can use the `wait()` system call to wait for the child process to finish execution.

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

Ans : The exit status indicates whether a child process completed successfully or encountered an error.

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

Ans : A parent can terminate a child process using the kill() system call or by sending a termination signal.

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

Ans : A **process group** is a collection of related processes, while a **session** is a group of process groups and represents the entire login session.

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

Ans : The exec() family replaces the current process image with a new one, allowing the running process to execute a different program.

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

Ans : waitpid() allows the parent to wait for a specific child process, whereas wait() waits for any child process to finish execution.

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

Ans : When a process terminates, its resources are reclaimed, and the process state is set to terminated. The parent is notified through signals or the wait() 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?

Ans : The long-term scheduler decides which processes should be admitted into the ready queue, influencing the degree of multiprogramming.

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?

Ans : Short-term Scheduler: Executes frequently to select the next process to run on the CPU.

Long-term Scheduler: Executes less frequently and decides which processes enter the ready queue.

Medium-term Scheduler: Handles process swapping and manages memory by temporarily removing processes from the ready queue.

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

Ans : The medium-term scheduler is invoked when processes need to be swapped in or out of memory to optimize resource usage and prevent thrashing. It ensures efficient system resource management.

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.

Ans:

First-Come, First-Served (FCFS) Scheduling:

GoodLuck Page No. _____
Date _____

Q1) ~~Q2~~ Algorithm used: FCFS

Ans

| Process | Arrival time | Burst time | Waiting time |
|---------|--------------|------------|--------------|
| P1 | 0 | 5 | 0 |
| P2 | 1 | 3 | 4 |
| P3 | 2 | 6 | 6 |

Gantt chart :

0 5 8 14

P1 P2 P3

Avg. waiting time = $(0 + 4 + 6) / 3$
 $= 10 / 3$

Avg waiting time = 3.333 units

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.

Ans :

Shortest Job First (SJF) Scheduling:

Q2) ~~Q2~~ Algorithm used : SJF (non-preemptive)

Ans

| Process | Arrival time | Burst time | waiting time | Turnaround Time |
|---------|--------------|------------|--------------|-----------------|
| P1 | 0 | 3 | 0 | 3 |
| P2 | 1 | 5 | 7 | 12 |
| P3 | 2 | 1 | 1 | 2 |
| P4 | 3 | 4 | 1 | 5 |

Gantt chart :

| | | | | |
|----|----|----|----|----|
| 0 | 3 | 4 | 8 | 13 |
| P1 | P3 | P4 | P2 | |

Avg. Turnaround Time = $(3+12+2+5)/4$
 $= 22/4$
Avg. Turnaround Time = 5.5 units

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.

Ans:

Q3) Algorithm used: priority scheduling (Non-preemptive)

Ans

| Process | Arrival time | Burst time | Completion time | Turnaround time (TAT) | Waiting time (WT) |
|---------|--------------|------------|-----------------|-----------------------|-------------------|
| P1 | 0 | 6 | 6 | 6 | 0 |
| P2 | 1 | 4 | 10 | 9 | 5 |
| P3 | 2 | 7 | 19 | 17 | 10 |
| P4 | 3 | 2 | 12 | 9 | 7 |

Avg waiting time = $(0 + 5 + 7 + 10) / 4$
 $= 22 / 4$
Avg waiting time (AWT) = 5.5 units

Gantt chart :-

```

0       6       10      12      19
| P1 | | P2 | | P4 | | P3 |
  
```

* for preemptive priority scheduling :-

Gantt chart :-

```

0       1       5       7       12      19
| P1 | | P2 | | P4 | | P1 | | P3 |
  
```

| Process | Arrival time | Burst time | Completion time | Turnaround time (TAT) | Waiting time (WT) |
|---------|--------------|------------|-----------------|-----------------------|-------------------|
| P1 | 0 | 6 | 12 | 12 | 6 |
| P2 | 1 | 4 | 5 | 4 | 0 |
| P3 | 2 | 7 | 19 | 17 | 10 |
| P4 | 3 | 2 | 7 | 4 | 2 |

Avg waiting time = $(6 + 0 + 2 + 10) / 4 = 18 / 4$
Avg waiting time = 4.5 units

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.

Ans :

Round Robin Scheduling (Time quantum = 2 units):

Q4)
Ans

Algorithm used : Round Robin scheduling
Quantum = 2 units

| process | Arrival time | Burst time | Turnaround time | waiting time |
|---------|--------------|------------|-----------------|--------------|
| P1 | 0 | 4 | 10 | 6 |
| P2 | 1 | 5 | 13 | 8 |
| P3 | 2 | 2 | 4 | 2 |
| P4 | 3 | 3 | 10 | 7 |

Gantt chart :

0 2 4 6 8 10 12 13 14

[P1 | P2 | P3 | P4 | P1 | P2 | P4 | P2]

Avg Turnaround time = $(10 + 13 + 4 + 10) / 4$
 $= 27 / 4$

Avg Turnaround time = 9.25 units.

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?

Ans :

GoodLuck Page No. _____
Date _____

Q5) fork() & variable copying concepts

Ans

① Before fork() :-
parent process has variable $x = 5$

② After fork() :-
Two separate process exist (child & parent)
both copy $x = 5$ (i.e. Both process have own copy)

③ Incrementing x :-

in child process :-
 $x = x + 1$
 $= 5 + 1$
 $x = 6$

in parent process :-
 $x = x + 1$
 $= 5 + 1$
 $x = 6$

Final values of x :-

| process | value of x |
|---------|------------|
| parent | 6 |
| child | 6 |