COS ASSIGNMENT 2

1. **echo "Hello, World!"**
   - Prints "Hello, World!" to the terminal.
2. **name="Productive"**
   - Assigns the value "Productive" to the variable name in the current shell session.
3. **touch file.txt**
   - Creates an empty file named file.txt (or updates its timestamp if it already exists).
4. **ls -a**
   - Lists all files and directories in the current directory, including hidden files (those starting with .).
5. **rm file.txt**
   - Deletes the file file.txt.
6. **cp file1.txt file2.txt**
   - Copies file1.txt to file2.txt.
7. **mv file.txt /path/to/directory/**
   - Moves file.txt to /path/to/directory/.
8. **chmod 755 script.sh**
   - Changes the permissions of script.sh to **rwxr-xr-x** (owner can read, write, and execute; group and others can read and execute).
9. **grep "pattern" file.txt**
   - Searches for "pattern" inside file.txt and prints matching lines.
10. **kill PID**
    - Terminates the process with the given **PID** (Process ID).
11. **mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt**
    - Creates a directory named mydir.
    - Changes into mydir.
    - Creates an empty file named file.txt.
    - Writes "Hello, World!" into file.txt.

   - Displays the contents of file.txt.

12. **ls -l | grep ".txt"**
    - Lists files in long format (-l), then filters and displays only files containing ".txt" in their name.
13. **cat file1.txt file2.txt | sort | uniq**
    - Concatenates the contents of file1.txt and file2.txt, sorts them, and removes duplicate lines.
14. **ls -l | grep "^d"**
    - Lists files in long format and filters only directories (lines that start with "d").
15. **grep -r "pattern" /path/to/directory/**
    - Recursively searches for "pattern" in all files under /path/to/directory/.
16. **cat file1.txt file2.txt | sort | uniq –d**
    - Concatenates file1.txt and file2.txt, sorts them, and prints only the duplicate lines.
17. **chmod 644 file.txt**
    - Sets permissions for file.txt to **rw-r--r--** (owner can read and write; group and others can only read).
18. **cp -r source_directory destination_directory**
    - Recursively copies source_directory and all its contents to destination_directory.
19. **find /path/to/search -name "*.txt"**
    - Searches for all .txt files under /path/to/search.
20. **chmod u+x file.txt**
    - Grants execute permission to the owner of file.txt.
21. **echo $PATH**
    - Displays the system's **PATH** variable, which lists directories where executable files are searched for.

Identify True Or False

1) **True** – ls lists files and directories in a directory.

2) **True** – mv is used to move (or rename) files and directories.

3) **False –** cd is used to change directories, not copy files. cp is used for copying.

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

5) True – grep is used to search for patterns in files.

6) **True** – chmod 755 file.txt gives the owner **read (r), write (w), and execute (x)** permissions, and **read (r) and execute (x)** permissions to group and others.

7) **True** – mkdir -p directory1/directory2 creates directory1 if it doesn't exist and then creates directory2 inside it.

8) **True** – rm -rf file.txt forcefully deletes file.txt without asking for confirmation.

Identify the incorrect commands

All the given commands are **incorrect** because they do not exist in standard Linux commands:

1. **chmodx –** Incorrect. The correct command is **chmod** to change file permissions.
2. **cpy –** Incorrect. The correct command is **cp** to copy files and directories.
3. **mkfile –** Incorrect. The correct command to create a new file is **touch filename** or **echo "" > filename**.
4. **catx –** Incorrect. The correct command is **cat** to concatenate and display files.
5. **rn –** Incorrect. The correct command to rename a file is **mv oldname newname**.

Question 1: Print "Hello, World!"

```
echo "Hello, World!"
```

Question 2: Declare a variable and print its value

```
name="CDAC Mumbai"

echo "The value of name is: $name"
```

Question 3: Take a number as input and print it

```
read -p "Enter a number: " num
echo "You entered: $num"
```

Question 4: Add two numbers and print the result

```
num1=5
num2=3
sum=$((num1 + num2))
echo "Sum: $sum"
```

Question 5: Check if a number is even or odd

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

Question 6: Print numbers from 1 to 5 using a for loop

```
for i in {1..5}; do
  echo $i
done
```

Question 7: Print numbers from 1 to 5 using a while loop

```
i=1
while [ $i -le 5 ]; do
  echo $i
  ((i++))
done
```

Question 8: Check if "file.txt" exists

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

Question 9: Check if a number is greater than 10

```
read -p "Enter a number: " num
if (( num > 10 )); then
   echo "The number is greater than 10"
else
   echo "The number is not greater than 10"
fi
```

Question 10: Print a multiplication table (1 to 5)

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

Question 11: Read numbers until a negative number is entered

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

```
done
echo "Negative number entered. Exiting..."
chmod +x script.sh
./script.sh
```

Problem 1: FCFS (First-Come, First-Served) Scheduling

Given Processes

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	6

Step 1: Calculate Completion Time (CT)

FCFS executes processes in the order of arrival.

1. **P1** starts at **0** and runs for **5** units → **CT = 5**
2. **P2** starts at **5** and runs for **3** units → **CT = 8**
3. **P3** starts at **8** and runs for **6** units → **CT = 14**

Process	AT	BT	CT
P1	0	5	5
P2	1	3	8
P3	2	6	14

Step 2: Calculate Turnaround Time (TAT)

TAT=CT−ATTAT = CT - ATTAT=CT−AT

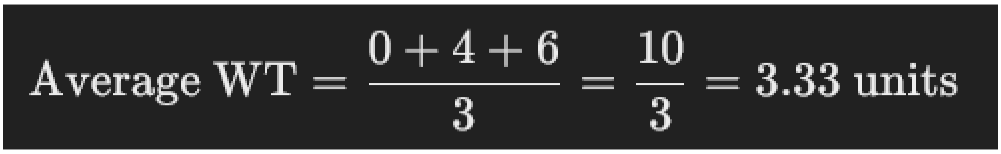
Process	AT	BT	CT	TAT
P1	0	5	5	5
P2	1	3	8	7
P3	2	6	14	12

Step 3: Calculate Waiting Time (WT)

WT=TAT−BTWT = TAT - BTWT=TAT−BT

Process	**AT**	**BT**	**CT**	**TAT**	**WT**
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	6	14	12	6

Step 4: Calculate Average Waiting Time



$$\text{Average WT} = \frac{0+4+6}{3} = \frac{10}{3} = 3.33 \text{ units}$$

Answer: The average waiting time using FCFS is **3.33 units**.

Problem 2: SJF (Shortest Job First) Scheduling

Given Processes

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

Step 1: Arrange Processes Based on SJF (Non-Preemptive)

- At **t = 0**, only **P1** is available → Execute **P1**.
- At **t = 3**, **P2, P3, P4** are available. Choose **P3** (shortest burst time).
- At **t = 4**, **P2, P4** are available. Choose **P4** (shorter burst time).
- At **t = 8**, **P2** is the only one left.

Step 2: Calculate Completion Time (CT)

Order	**Process**	**AT**	**BT**	**Start Time**	**Completion Time**
1	P1	0	3	0	3
2	P3	2	1	3	4

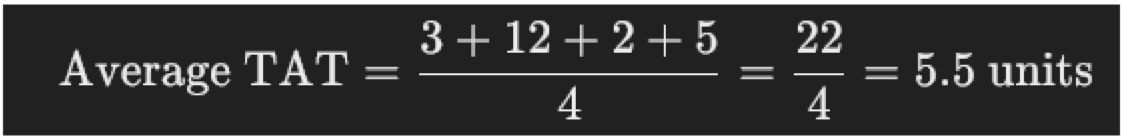
Order	Process	AT	BT	Start Time	Completion Time
3	P4	3	4	4	8
4	P2	1	5	8	13

Step 3: Calculate Turnaround Time (TAT)

TAT=CT−ATTAT = CT - ATTAT=CT−AT

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

Step 4: Calculate Average Turnaround Time



$$\text{Average TAT} = \frac{3+12+2+5}{4} = \frac{22}{4} = 5.5 \text{ units}$$

Answer: ✅ The average turnaround time using SJF is **5.5 units**.

Problem 3: Priority Scheduling (Non-Preemptive)

Given Processes

Process	Arrival Time	Burst Time	Priority (Lower is Higher)
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Step 1: Arrange Processes by Priority

1. The process with the **lowest priority number (highest priority)** is scheduled first.
2. If two processes have the same priority, FCFS is used.

Arrival Time	Process	Priority
1	P2	1 (Highest)
3	P4	2
0	P1	3
2	P3	4 (Lowest)

Step 2: Calculate Completion Time (CT)

Order	Process	AT	BT	Start Time	Completion Time
1	P1	0	6	0	6
2	P2	1	4	6	10
3	P4	3	2	10	12
4	P3	2	7	12	19

Step 3: Calculate Turnaround Time (TAT)

TAT=CT−ATTAT = CT - ATTAT=CT−AT

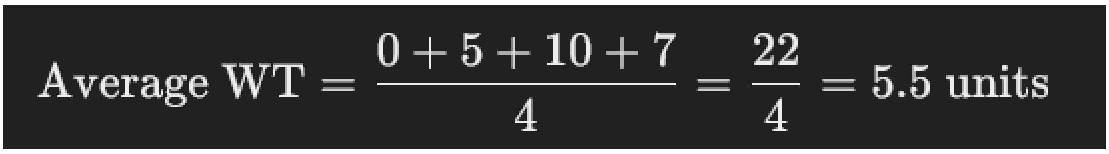
Process	AT	BT	CT	TAT
P1	0	6	6	6
P2	1	4	10	9
P3	2	7	19	17
P4	3	2	12	9

Step 4: Calculate Waiting Time (WT)

WT=TAT−BTWT = TAT - BTWT=TAT−BT

Process	AT	BT	CT	TAT	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

Step 5: Calculate Average Waiting Time



$$\text{Average WT} = \frac{0+5+10+7}{4} = \frac{22}{4} = 5.5 \text{ units}$$

Answer: The average waiting time using Priority Scheduling is **5.5 units**.

Problem 4: Round Robin Scheduling (Time Quantum = 2 Units)

Given Processes

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

Step 1: Execute Using Round Robin (Time Quantum = 2)

1. **P1** executes for **2** (Remaining: 2)
2. **P2** executes for **2** (Remaining: 3)
3. **P3** executes for **2** (Finished)
4. **P4** executes for **2** (Remaining: 1)
5. **P1** executes for **2** (Finished)
6. **P2** executes for **2** (Remaining: 1)
7. **P4** executes for **1** (Finished)
8. **P2** executes for **1** (Finished)

Step 2: Calculate Completion Time (CT)

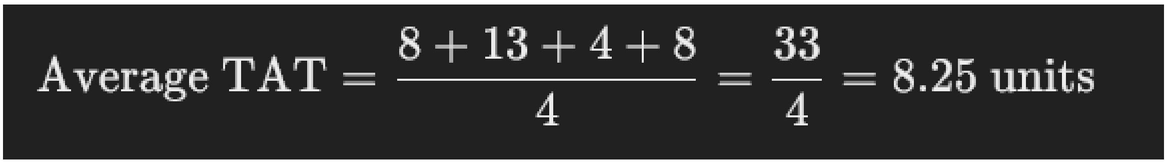
Process	**AT**	**BT**	**CT**
P1	0	4	8
P2	1	5	14
P3	2	2	6
P4	3	3	11

Step 3: Calculate Turnaround Time (TAT)

TAT=CT−ATTAT = CT - ATTAT=CT−AT

Process	AT	BT	CT	TAT
P1	0	4	8	8
P2	1	5	14	13
P3	2	2	6	4
P4	3	3	11	8

Step 4: Calculate Average Turnaround Time



$$\text{Average TAT} = \frac{8+13+4+8}{4} = \frac{33}{4} = 8.25 \text{ units}$$

Answer: The average turnaround time using Round Robin is **8.25 units**.

Problem 5: fork() System Call and Variable Modification

Explanation

- Initially, **x = 5**.
- When fork() is called, a **new child process is created** with a copy of x.
- Both parent and child **increment x independently**.
- Since they are separate processes, changes in one do **not** affect the other.

Final Values:

- In **Parent Process**: x = 6
- In **Child Process**: x = 6

Answer: Both parent and child will have **x = 6** separately.