Q.1. Implement internal commands of Linux (Any one)

Terminal temple / online bash shell

1 Command to display the current working directory. command to navigate to a directory of your choice. Then, again verify your new location.

```
terminal@terminal-temple ~ $ pwd
/home/terminal
terminal@terminal-temple ~ $ cd Documents
terminal@terminal-temple Documents $ pwd
/home/terminal/Documents
```

mkdir pranjal
cd pranjal
pwd

```
/home/pranjal
```

2 Create a new directory named "TestDir" in your current working directory . Create an empty file named "testfile.txt" inside the "TestDir" directory list the contents of "TestDir" and verify that "testfile.txt" is present. Delete the file "testfile.txt" and verify its removal

```
terminal@terminal-temple ~ $ mkdir TestDir
mkdir: TestDir: File exists
terminal@terminal-temple ~ $ touch TestDir/testfile.txt
terminal@terminal-temple ~ $ ls TestDir
testfile.txt
terminal@terminal-temple ~ $ rm TestDir/testfile.txt
terminal@terminal-temple ~ $ ls TestDir
terminal@terminal-temple ~ $
```

mkdir TestDir
ls

cd TestDir
nano testfile.txt
ls

rm testfile.txt
ls

3. Command to display the current system date and time.

echo "Date and type of system : $(date)"
date

```
Date and type of system : Wed Apr 23 10:48:38 AM UTC 2025
Wed Apr 23 10:48:38 AM UTC 2025
```

Q.4 ommand to display how long the system has been running.

echo "system has been running : $(uptime)"

```
system has been running :  10:51:03 up 12 days, 23:54,  0 user,  load average: 7.17, 6.00, 5.01
```

Q.5command to display the current user.

echo "curent user : $(whoami)"
echo "current logname : $(tty)"
echo "Log Name: $LOGNAME"

```
curent user : runner8
current logname : /dev/pts/2
```

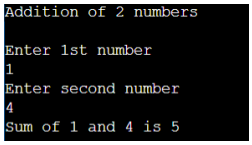
Q.2. Write Shell Scripts incorporating Linux Commands (Any two will be asked)

1.Display OS version, release number, kernel version

```
#!/bin/bash
echo "OS Version and release no :"
lsb_release -a|grep Description
lsb_release -d
echo -e "\nKernel Version:"
uname -r
uname -a
```

2.Write the shell Script to add two numbers and display the output.

```
#!/bin/bash
echo -e "\nAdd Two Numbers"
echo "Enter first number:"
read num1
echo "Enter second number:"
read num2
sum=$((num1 + num2))
echo "The sum of $num1 and $num2 is: $sum"
```



3.Accept a number from the user and write a shell script to check whether it is even or odd.

```
#!/bin/bash
#!/bin/bash

echo -e "\n odd & even"
echo "Enter first number:"
read num

if (( num % 2 == 0 )); then
   echo "$num is even"
else
   echo "$num is odd"
fi

echo -e "\n Odd & Even"
echo "Enter a number:"
read num

if [ $((num % 2)) -eq 0 ]
then
   echo "$num is even"
else
   echo "$num is odd"
fi
```

4.Display processes with highest memory usage.

```
#!/bin/bash
echo  -e "Top Processes by highest Memory Usage"
ps aux --sort=-%mem | head -n 2
```

5.Display top 10 processes in descending order

```
#!/bin/bash
echo -e "\nTop 10 Processes by CPU usage in descending order"
ps aux --sort=-%cpu | head -n 11
echo
```

nano os.sh

chmod +x os.sh

./os.sh

```
OS version and release nuber:
No LSB modules are available.
Description:    Ubuntu 22.04.3 LTS
Kernal Version:
Linux b4-HP-Pro-Tower-400-G9-PCI-Desktop-PC 6.5.0-15-generic #15~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Jan 12 18:54:30 UTC 2 x86_64 x86_64 x8
6_64 GNU/Linux
TOP 10 processes in descending ordee:

TOP 10 processes by CPU usage:
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
b4          1738  1.0  1.7 5911216 276264 ?      Ssl  10:43   1:00 /usr/bin/gnome-shell
avahi        572  0.3  0.0   8032  3968 ?        Ss   10:42   0:17 avahi-daemon: running [b4-HP-Pro-Tower-400-G9-PCI-Desktop-PC.local]
b4          4262  0.2  0.3 555872 53788 ?        Ssl  10:49   0:15 /usr/libexec/gnome-terminal-server
b4          1863  0.1  0.0 315092 11624 ?        Sl   10:43   0:05 /usr/bin/ibus-daemon --panel disable
root           1  0.0  0.0 166776 11084 ?        Ss   10:42   0:00 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    10:42   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   10:42   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   10:42   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   10:42   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   10:42   0:00 [netns]

Processes with highest memory usage:
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
b4          1738  1.0  1.7 5911216 276264 ?      Ssl  10:43   1:00 /usr/bin/gnome-shell

Current user: b4
Logname: b4-HP-Pro-Tower-400-G9-PCI-Desktop-PC

Enter first number:
78
Enter second number:
4556

 The sum of 78 and 4556 is: 4634
```

Q. 3. Create a child process using fork system call. Obtain process ID of parent and child using getppid and getpid system calls

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid = fork();

    if (pid < 0)
    {
        printf("Fork failed!\n");
        return 1;
    }

    if (pid == 0)
    {
        printf("Child Process:\n");
        printf("PID (Child): %d\n", getpid());
        printf("PPID (Parent): %d\n", getppid());
    }
    else
    {
        printf("Parent Process:\n");
        printf("PID (Parent): %d\n", getpid());
        printf("Child PID: %d\n", pid);
    }

    return 0;
}
```

```
/* Child Process:
PID (Child): 12346
PPID (Parent): 12345

Parent Process:
PID (Parent): 12345
Child PID: 12346 */
```

4. Write a program to implement a preemptive process scheduling algorithm (SJF, PRIORITY)

SJF

```c
#include <stdio.h>

int main()
{
    int n, i, time = 0, count = 0, smallest;
    int a[10], b[10], x[10];
    double wt = 0, tat = 0, end;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter arrival times:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("Enter burst times:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &b[i]);
        x[i] = b[i];    /*copy arrival time*/
    }

    b[9] = 9999;

    printf("Gantt Chart:\n");
    for (time = 0; count != n; time++)
    {
        smallest = 9;
        for (i = 0; i < n; i++)
```

```
        {
            if (a[i] <= time && b[i] < b[smallest] && b[i] > 0)
            {
                smallest = i;
            }
        }

        b[smallest]--;
        printf("| P%d ", smallest + 1);

        if (b[smallest] == 0)
        {
            count++;
            end = time + 1;
            wt = wt +  end - a[smallest] - x[smallest];
            tat = tat + end - a[smallest];
        }
    }
    printf("|\n");

    printf("Average Waiting Time = %.2lf", wt / n);
    printf("\nAverage Turnaround Time = %.2lf", tat / n);

    return 0;
}

//6
//000000
//7 5 3 1 2 1

/* Gantt Chart:
| P4 | P6 | P5 | P5 | P3 | P3 | P3 | P2 | P2 | P2 | P2 | P2 | P1 | P1 | P1 |
P1 | P1 | P1 | P1 |
Average Waiting Time = 4.33
Average Turnaround Time = 7.50 */
```

PRIORITY

```
#include <stdio.h>

int main()
{
    int n, i, time = 0, highest, count = 0;
    int a[10], b[10], remaining[10], p[10];
    double wt = 0, tat = 0, end;

    printf("Enter the number of processes: ");
```

```
    scanf("%d", &n);

    printf("Enter arrival times:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("Enter burst times:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &b[i]);
        remaining[i] = b[i];
    }

    printf("Enter priorities (lower number = higher priority):\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &p[i]);
    }

    p[9] = 9999;

    printf("\nGantt Chart:\n");
    for (time = 0; count != n; time++)
    {
        highest = 9;
        for (i = 0; i < n; i++) {
            if (a[i] <= time && p[i] < p[highest] && remaining[i] > 0)
            {
                highest = i;
            }
        }

        remaining[highest]--;
        printf("| P%d ", highest + 1);

        if (remaining[highest] == 0) {
            count++;
            end = time + 1;
            wt += end - a[highest] - b[highest];
            tat += end - a[highest];
        }
    }
    printf("|\n");

    printf("\nAverage Waiting Time = %.2lf\n", wt / n);
    printf("Average Turnaround Time = %.2lf\n", tat / n);
```

```
    return 0;
}

//5
// 0 1 2 3 4
//4 3 1 5 2
//1 2 3 4 4

/*  Gantt Chart:
| P2 | P5 | P5 | P5 | P5 | P5 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 |
P1 | P3 | P3 | P4 |

Average Waiting Time = 8.20
Average Turnaround Time = 12.00 */
```

Q.5.To study and implement disk scheduling algorithms (FCFS)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int max,n,i,queue[20],head,seek=0,diff;
    float avg;
    printf("Max range of disk: ");
    scanf("%d",&max);

    printf("Size of pending request: ");
    scanf("%d",&n);

    printf("Queue of pending request \n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&queue[i]);
    }

    printf("Initial head position: ");
    scanf("%d",&head);

    printf("Disk head movements:\n");
    for (i = 0; i < n; i++)
    {
        int diff = abs(queue[i] - head);
        seek = seek + diff;
        printf("From %d to %d with seek %d\n", head, queue[i], diff);
        head = queue[i];
    }
```

```
    printf("Total seek time is %d",seek);
    avg=(float)seek/n;
    printf("Average seek time is %f\n",avg);
    return 0;
}

//200
//55 58 39 18 90 160 150 38 184
//100
//498
//55.3333332
```