GLA UNIVERSITY MATHURA

Department	Computer Engineering and Application
Program	B. Tech CSE [AIML & IoT]
University Roll No.	2315510150
Section	CC
Class Roll No.	36
Assignment No.	01
Subject Name	Operating System Lab
Subject Code	BCSC 0803

Submitted To:

Mrs. Munmi Gogoi

Submitted By:

Prem Narayan Sharma

INDEX

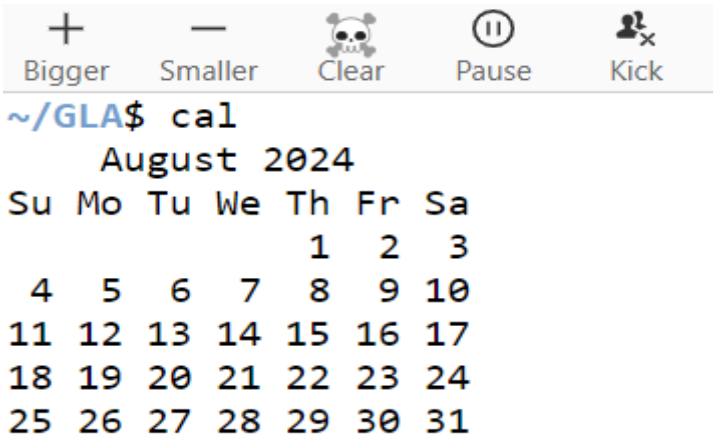
S.No.	Experiment	Date	Remark
1	Implement the Date & Cal commands used in LINUX OS	13/8/2024	
2	Implement the Directory & File commands used in LINUX OS	20/8/2024	
3	Implement the Sort commands used in LINUX OS	27/8/2024	
4	Using SED & Vi Editors used for LINUX OS	03/9/2024	
5	Implement the CHMOD commands used in LINUX OS	10/9/2024	
6	Display the Date & Time in LINUX OS	17/9/2024	
7	Create Nested Directories using Linux Terminal in LINUX OS	24/9/2024	
8	Creating files and editing the content in LINUX OS	15/10/2024	
9	Implement the for and while loops in LINUX OS	22/10/2024	
10	Displaying each element of an array via loops in LINUX OS	29/10/2024	
11	Displaying Factorial of any number using loops	05/11/2024	
12	Comparing two Strings in LINUX OS	12/11/2024	
13	Implement the CPU Scheduling Algorithms	19/11/2024	

CONTENT

LINUX COMMANDS WITH THEIR EXAMPLES

01.cal command: The cal utility displays a simple calendar in traditional format. If arguments are not specified, the current month is displayed.

Execution:



Examples:

I. cal -3 : Displays previous, current and next month's calendar

```
~/GLA$ cal -3
                                2024
        July                   August                 September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6                  1  2  3   1  2  3  4  5  6  7
 7  8  9 10 11 12 13   4  5  6  7  8  9 10   8  9 10 11 12 13 14
14 15 16 17 18 19 20  11 12 13 14 15 16 17  15 16 17 18 19 20 21
21 22 23 24 25 26 27  18 19 20 21 22 23 24  22 23 24 25 26 27 28
28 29 30 31           25 26 27 28 29 30 31  29 30
```

II. cal 2025 : Shows calendar of particular year, 2025 in this case

```
~/GLA$ cal 2025
                                2025
      January               February                March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
            1  2  3  4                     1                     1
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   2  3  4  5  6  7  8
12 13 14 15 16 17 18   9 10 11 12 13 14 15   9 10 11 12 13 14 15
19 20 21 22 23 24 25  16 17 18 19 20 21 22  16 17 18 19 20 21 22
26 27 28 29 30 31     23 24 25 26 27 28     23 24 25 26 27 28 29
                                            30 31

       April                 May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5                  1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30

        July                 August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5                  1  2      1  2  3  4  5  6
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   7  8  9 10 11 12 13
13 14 15 16 17 18 19  10 11 12 13 14 15 16  14 15 16 17 18 19 20
20 21 22 23 24 25 26  17 18 19 20 21 22 23  21 22 23 24 25 26 27
27 28 29 30 31        24 25 26 27 28 29 30  28 29 30
                      31

      October              November              December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
            1  2  3  4                     1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
19 20 21 22 23 24 25  16 17 18 19 20 21 22  21 22 23 24 25 26 27
26 27 28 29 30 31     23 24 25 26 27 28 29  28 29 30 31
                      30
```

III. cal 09 2024 : Shows calendar of given month and year

```
~/GLA$ cal 09 2024
   September 2024
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

02.date command: Display the current time in the given FORMAT, or set the system date.

Execution:

```
~/GLA$ date
Tue Aug 20 09:41:21 UTC 2024
```

Examples:

I. date "+%T" : Shows current time

```
~/GLA$ date "+%T"
09:47:50
```

II. date "+%a" : Shows today's weekday name in abbreviation

```
~/GLA$ date "+%a"
Tue
```

III. date "+%y" : Shows last two digits of current year

```
~/GLA$ date "+%y"
24
```

03.mkdir command: Create the directory(ies), if they do not already exist.

Execution:

```
~$ mkdir NewDirectory
~$ cd NewDirectory
~/NewDirectory$
```

Examples:

I. Multiple Directories

```
~/NewDirectory$ mkdir New1 New2
~/NewDirectory$ ls
New1  New2
```

II. Parent and Child Directories

```
~/NewDirectory$ mkdir -p DirA/DirB/DirC
~/NewDirectory$ tree
.
├── DirA
│   └── DirB
│       └── DirC
├── New1
└── New2
```

III. Multiple parent and child directories

```
~/NewDirectory$ mkdir -p dirW/dirX dirY/dirZ
~/NewDirectory$ tree
.
├── dirW
│   └── dirX
└── dirY
    └── dirZ
```

04.rmdir command: Remove empty directory(ies).

Execution:

```
~/main$ ls
NewDirectory
~/main$ rmdir NewDirectory
~/main$ ls
~/main$
```

Examples:

I. Multiple Directories

```
~/main$ mkdir Dir1 Dir2
~/main$ ls
Dir1  Dir2
~/main$ rmdir Dir1 Dir2
~/main$ ls
~/main$
```

II. Parent and Child Directories

```
~/main$ tree
.
└── DirA
    └── DirB
        └── DirC

3 directories, 0 files
~/main$ rmdir -p DirA/DirB/DirC
~/main$ tree
.
```

III. Multiple parent and child directories

```
~/main$ tree
.
├── dirW
│   └── dirX
└── dirY
    └── dirZ

4 directories, 0 files
~/main$ rmdir -p dirW/dirX dirY/dirZ
~/main$ tree
.
```

05.touch command: A file argument that does not exist is created empty, unless -c or -h is supplied.

Execution:

```
~/main$ touch File1.txt
~/main$ ls
File1.txt
```

Examples:

I. Multiple Files

```
~/main$ touch File2.txt File.txt
~/main$ ls
File.txt_ File2.txt
```

06.rm command: Removes each specified file. By default, it does not remove directories.

Execution:

```
~/main$ ls
File1.txt
~/main$ rm File1.txt
~/main$ ls
```

Examples:

I. Multiple Files

```
~/main$ ls
File01.txt  File02.txt
~/main$ rm File01.txt File02.txt
~/main$ ls
~/main$
```

SHELL SCRIPTING

A shell script is a list of commands in a computer program that is run by the Unix shell which is a command line interpreter.

A shell script usually has comments that describe the steps.

The different operations performed by shell scripts are:

1. *Automating the code compiling process.*
2. *Running a program or creating a program environment.*
3. *Completing batch*
4. *Manipulating files.*
5. *Linking existing programs together.*
6. *Executing routine backups.*
7. *Monitoring a system.*

SHELL SCRIPTING PROGRAMS

1. Write shell script programs to print name which input by user

```
~/shellscripts$ cat 1_display_name.sh
echo "Enter your name: "
read name
echo "Your name is $name"

~/shellscripts$ ./1_display_name.sh
Enter your name:
Alok Bhadauria
Your name is Alok Bhadauria
```

2. Write shell script programs to sum of two numbers

```
~/shellscripts$ cat 2_sum_numbers.sh
echo "Enter num1: "
read num1
echo "Enter num2: "
read num2
sum=$(( $num1+$num2 ))
echo "Sum of numbers: $sum"

~/shellscripts$ ./2_sum_numbers.sh
Enter num1:
5
Enter num2:
10
Sum of numbers: 15
```

3. Write shell script programs to print greater number among two numbers

```
~/shellscripts$ cat 3_greater_number.sh
echo "Enter num1: "
read num1
echo "Enter num2: "
read num2

#if test $num1 -gt $num2
if [ $num1 -gt $num2 ]

then
        echo $num1 is greater than $num2
else
        echo $num2 is greater than $num1
fi

~/shellscripts$ ./3_greater_number.sh
Enter num1:
5
Enter num2:
10
10 is greater than 5
```

4. Write shell script program to check whether a number is positive or negative

```
~/shellscripts$ cat 4_check_positive.sh
echo "Enter a number: "
read num

if [ $num -lt 0 ]
then
        echo "Negative"
elif [ $num -gt 0 ]
then
        echo "Positive"
else
        echo "Neither Positive nor Negative"
fi

~/shellscripts$ ./4_check_positive.sh
Enter a number:
8
Positive
```

5. Write shell program to check entered number is odd or even

```
~/shellscripts$ cat 5_odd_even.sh
echo "Enter the number: "
read num
if [ $(($num % 2)) -eq 0 ]
then
        echo "$num is an Even number"
else
        echo "$num is an Odd number"
fi

~/shellscripts$ ./5_odd_even.sh
Enter the number:
3
3 is an Odd number
```

6. Write shell script program to sum of all digit enter by user

```
~/shellscripts$ cat 6_sum_digits.sh
echo "Enter a number: "
read num
sum=0
while [ $num -gt 0 ]
do
        mod=$((num % 10))
        sum=$((sum + mod))
        num=$((num / 10))
done
echo $sum
~/shellscripts$ ./6_sum_digits.sh
Enter a number:
1024
7
```

7. Write shell script program to print numbers from 1 to 100 using for loop

```
~/shellscripts$ cat 7_for_loop.sh
for((i=1;i<=100;i++))
do
        echo $i
done
~/shellscripts$ ./7_for_loop.sh
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

(...continued till 100)

8. Write shell script program to print numbers from 1 to 10 using while loop

```
~/shellscripts$ cat 8_while_loop.sh
i=1
while [ $i -le 10 ]
do
        echo $i
        i=$(($i+1))
done

~/shellscripts$ ./8_while_loop.sh
1
2
3
4
5
6
7
8
9
10
```

9. Write shell script program to print factorial of a given number using for loop

```
~/shellscripts2$ cat 9_for_factorial.sh
echo "Enter a number: "
read num
fact=1
for((i=2;i<=num;i++))
{
        fact=$((fact * i))
}
echo "Factorial of $num is : $fact"
~/shellscripts2$ ./9_for_factorial.sh
Enter a number:
5
Factorial of 5 is : 120
```

10. Write shell script program to print factorial of a given number using while loop

```
~/shellscripts2$ cat 10_while_factorial.sh
echo "Enter a number: "
read num
a=$num
fact=1
while [ $num -gt 1 ]
do
        fact=$((fact * num))
        num=$((num - 1))
done
echo "Factorial of $a is : $fact"
~/shellscripts2$ ./10_while_factorial.sh
Enter a number:
10
Factorial of 10 is : 3628800
~/shellscripts2$
```

11. Write shell script program to print elements using array

```
~/shellscripts2$ cat 11_print_array_elements.sh
fruits=('apple' 'banana' 'cherry')
for i in "${fruits[@]}"
do
        echo I like $i
done
~/shellscripts2$ ./11_print_array_elements.sh
I like apple
I like banana
I like cherry
~/shellscripts2$
```

12. Write shell script program to print two array elements using nested for loop

```
~/shellscripts2$ cat 12_array_nested_loop.sh
fruits=('guava' 'banana' 'cherry')
colors=('green' 'yellow' 'red')
for i in "${!fruits[@]}"
do
        echo The ${fruits[$i]} is ${colors[$i]}
done
~/shellscripts2$ ./12_array_nested_loop.sh
The guava is green
The banana is yellow
The cherry is red
~/shellscripts2$
```

13. Write shell script program to print sum of array elements

```
~/shellscripts2$ cat 13_array_sum.sh
arr=(10 20 30 40 50)
sum=0
for i in ${arr[@]}
do
        sum=$(($sum + $i))
done
echo "Sum of array : $sum"
~/shellscripts2$ ./13_array_sum.sh
Sum of array : 150
~/shellscripts2$
```

14. Write shell script program to compare two strings are equal or not equal

```
~/shellscripts2$ cat 14_string_compare.sh
read -p "Enter first string: " str1
read -p "Enter second string: " str2
if [ $str1 == $str2 ]
then
        echo "Equal"
else
        echo "Not Equal"
fi
~/shellscripts2$ ./14_string_compare.sh
Enter first string: Alok
Enter second string: Linux
Not Equal
~/shellscripts2$
```

CPU SCHEDULING ALGORITHM PROGRAMMING

1. Implement FCFS CPU Scheduling Algorithm in Shell Script language to determine the average waiting time and average turnaround time given n processes and their burst times.

```
#!/bin/bash

# Function to draw a border
draw_border() {
    printf '%s\n' "----------------------------------------------------------"
}

# Input number of processes
read -p "Enter the number of processes: " n

# Declare arrays
declare -a pid arrival_time burst_time waiting_time tat completion_time

# Input process details
for ((i = 0; i < n; i++)); do
    read -p "Enter Process Id: " pid[i]
```

```
read -p "Enter arrival time: " arrival_time[i] read -p
"Enter burst time: " burst_time[i]
    done

    # Sort based on arrival time using Bubble Sort
    for ((i = 0; i < n - 1; i++)); do
        for ((j = 0; j < n - i - 1; j++)); do
            if ((arrival_time[j] > arrival_time[j + 1])); then
                # Swap arrival time
                temp=${arrival_time[j]}
                arrival_time[j]=${arrival_time[j + 1]}
                arrival_time[j + 1]=$temp

                # Swap burst time
                temp=${burst_time[j]}
                burst_time[j]=${burst_time[j + 1]}
                burst_time[j + 1]=$temp

                # Swap process ID
                temp=${pid[j]}
                pid[j]=${pid[j + 1]}
                pid[j + 1]=$temp
            fi
        done
    done

    # Calculate waiting time, turnaround time, and completion time
    total_wt=0
    total_tat=0
    completion_time[0]=$((arrival_time[0] + burst_time[0]))
```

```
tat[0]=${completion_time[0]}
waiting_time[0]=0
total_tat=${tat[0]}

for ((i = 1; i < n; i++)); do
    # Calculate completion time
    completion_time[i]=$((completion_time[i - 1] + burst_time[i]))

    # Calculate turnaround time
    tat[i]=$((completion_time[i] - arrival_time[i]))
    total_tat=$((total_tat + tat[i]))

    # Calculate waiting time
    waiting_time[i]=$((tat[i] - burst_time[i]))
    total_wt=$((total_wt + waiting_time[i]))
done

# Print table headers
draw_border
printf "| %-12s | %-12s | %-12s | %-12s | %-12s |\n" "Process Id" "Burst time" "Arrival time" "Waiting time" "Turn around time"
draw_border

# Print table rows
for ((i = 0; i < n; i++)); do
    printf "| %-12s | %-12s | %-12s | %-12s | %-12s |\n" "${pid[i]}" "${burst_time[i]}" "${arrival_time[i]}" "${waiting_time[i]}" "${tat[i]}"
done

draw_border
```

```
# Calculate and print averages
avg_wt=$(echo "scale=3; $total_wt / $n" | bc)
avg_tat=$(echo "scale=3; $total_tat / $n" | bc)
echo "Average waiting time = $avg_wt"
echo "Average turn around time = $avg_tat"

# Print Gantt chart
echo "----------------"
echo -n "|"
for ((i = 0; i < n; i++)); do
    printf " P%-2s |" "${pid[i]}"
done
echo
echo "----------------"
printf "%-3s" "0"
for ((i = 0; i < n; i++)); do
    printf "%-4s" "${completion_time[i]}"
done
echo
```

OUTPUT:

```
~$ nano fcfs.sh
~$ chmod +x fcfs.sh
~$ ./fcfs.sh
Enter the number of processes: 5
Enter Process Id: 0
Enter arrival time: 1
Enter burst time: 8
Enter Process Id: 1
Enter arrival time: 2
Enter burst time: 4
Enter Process Id: 2
Enter arrival time: 3
Enter burst time: 5
Enter Process Id: 3
Enter arrival time: 5
Enter burst time: 11
Enter Process Id: 4
Enter arrival time: 6
Enter burst time: 3
-------------------------------------------------------------
| Process Id    | Burst time    | Arrival time | Waiting time | Turn around time |
-------------------------------------------------------------
| 0             | 8             | 1            | 0            | 9                |
| 1             | 4             | 2            | 7            | 11               |
| 2             | 5             | 3            | 10           | 15               |
| 3             | 11            | 5            | 13           | 24               |
| 4             | 3             | 6            | 23           | 26               |
-------------------------------------------------------------
Average waiting time = 10.600
Average turn around time = 17.000
-----------------
| P0  | P1  | P2  | P3  | P4  |
-----------------
0  9   13  18  29  32
~$
```

Q 2) Implement Priority CPU Scheduling Algorithm in Shell Script programming language to determine the average waiting time and average turnaround time given n processes and their burst times.

```
#!/bin/bash


# Function to draw a border
border() {
    for ((i = 0; i < 121; i++)); do
        echo -n "-"
    done
    echo ""
}
```

```
# Function to arrange processes by Arrival Time
arrangeArrival() {
    for ((i = 0; i < n; i++)); do
        for ((j = i + 1; j < n; j++)); do
            if [ ${arrival_time[i]} -gt ${arrival_time[j]} ]; then
                # Swap Arrival Time
                temp=${arrival_time[i]}
                arrival_time[i]=${arrival_time[j]}
                arrival_time[j]=$temp

                # Swap Burst Time
                temp=${burst_time[i]}
                burst_time[i]=${burst_time[j]}
                burst_time[j]=$temp

                # Swap Priority
                temp=${priority[i]}
                priority[i]=${priority[j]}
                priority[j]=$temp

                # Swap Process ID
                temp=${pid[i]}
                pid[i]=${pid[j]}
                pid[j]=$temp
            fi
        done
    done
}
```

```
# Function to arrange processes by Priority (for the same Arrival Time)
arrangePriority() {
    for ((i = 0; i < n; i++)); do
        for ((j = i + 1; j < n; j++)); do
            if [ ${arrival_time[i]} -eq ${arrival_time[j]} ] && [ ${priority[i]} -gt ${priority[j]} ];
then

                # Swap Arrival Time
                temp=${arrival_time[i]}
                arrival_time[i]=${arrival_time[j]}
                arrival_time[j]=$temp

                # Swap Burst Time
                temp=${burst_time[i]}
                burst_time[i]=${burst_time[j]}
                burst_time[j]=$temp

                # Swap Priority
                temp=${priority[i]}
                priority[i]=${priority[j]}
                priority[j]=$temp

                # Swap Process ID
                temp=${pid[i]}
                pid[i]=${pid[j]}
                pid[j]=$temp
            fi
        done
    done
}


# Function to calculate Waiting Time
```

```
findWaitingTime() {
    service_time[0]=0
    waiting_time[0]=0

    for ((i = 1; i < n; i++)); do
        service_time[i]=$((service_time[i - 1] + burst_time[i - 1]))
        waiting_time[i]=$((service_time[i] - arrival_time[i]))
        if [ ${waiting_time[i]} -lt 0 ]; then
            waiting_time[i]=0
        fi
    done
}

# Function to calculate Turnaround Time
findTurnAroundTime() {
    for ((i = 0; i < n; i++)); do
        tat[i]=$((waiting_time[i] + burst_time[i]))
    done
}

# Read the number of processes
echo -n "Enter the number of processes: "
read n

# Read process details
for ((i = 0; i < n; i++)); do
    echo -n "Enter Process Id: "
    read pid[i]
    echo -n "Enter Arrival Time: "
    read arrival_time[i]
```

```
    echo -n "Enter Burst Time: "
    read burst_time[i]
    echo -n "Enter Priority: "
    read priority[i]
done

# Sort processes by Arrival Time and Priority
arrangeArrival
arrangePriority

# Calculate Waiting Time and Turnaround Time
findWaitingTime
findTurnAroundTime

# Calculate totals for averages
total_wt=0
total_tat=0

# Print table headers
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" \
    "Process Id" "Burst Time" "Arrival Time" "Waiting Time" "Turnaround Time"
"Completion Time"
border

# Print process details
for ((i = 0; i < n; i++)); do
    total_wt=$((total_wt + waiting_time[i]))
    total_tat=$((total_tat + tat[i]))
    completion_time=$((arrival_time[i] + tat[i]))
```

```
    printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" \
        ${pid[i]} ${burst_time[i]} ${arrival_time[i]} ${waiting_time[i]} ${tat[i]}
$completion_time
done
border

# Calculate and print averages
avg_wt=$(echo "scale=3; $total_wt / $n" | bc)
avg_tat=$(echo "scale=3; $total_tat / $n" | bc)
echo "Average Waiting Time = $avg_wt"
echo "Average Turnaround Time = $avg_tat"

# Print Gantt chart
for ((i = 0; i < 8 * n + n + 1; i++)); do
    echo -n "-"
done
echo ""
for ((i = 0; i < n; i++)); do
    echo -n "| P${pid[i]} "
done
echo "|"
for ((i = 0; i < 8 * n + n + 1; i++)); do
    echo -n "-"
done
echo ""
echo -n "0 "
for ((i = 0; i < n; i++)); do
    completion_time=$((arrival_time[i] + tat[i]))
    echo -n "$completion_time "
done
echo ""
```

OUTPUT:

```
~$ nano ps.sh
~$ chmod +x ps.sh
~$ ./ps.sh
Enter the number of processes: 3
Enter Process Id: 0
Enter Arrival Time: 2
Enter Burst Time: 5
Enter Priority: 1
Enter Process Id: 1
Enter Arrival Time: 5
Enter Burst Time: 8
Enter Priority: 0
Enter Process Id: 2
Enter Arrival Time: 4
Enter Burst Time: 5
Enter Priority: 0
-----------------------------------------------------------------------------------------------------------
|Process Id        |Burst Time          |Arrival Time      |Waiting Time        |Turnaround Time   |Completion Time     |
-----------------------------------------------------------------------------------------------------------
|0                 |5                   |2                 |0                   |5                 |7                   |
|2                 |5                   |4                 |1                   |6                 |10                  |
|1                 |8                   |5                 |5                   |13                |18                  |
-----------------------------------------------------------------------------------------------------------
Average Waiting Time = 2.000
Average Turnaround Time = 8.000
---------------------------
| P0 | P2 | P1 |
---------------------------
0 7 10 18
~$
```

Q 3) Implement Round Robin CPU Scheduling Algorithm in Shell Script programming language to determine the average waiting time and average turnaround time given n processes and their burst times.

```
sort() {
  # Sort processes based on arrival time
  for ((i=0; i<$n; i++))
  do
    for ((j=0; j<$n-i-1; j++))
    do
      if [ ${arrival_time[j]} -gt ${arrival_time[$((j+1))]} ]; then
        # Swap arrival times
        temp=${arrival_time[j]}
        arrival_time[$j]=${arrival_time[$((j+1))]}
        arrival_time[$((j+1))]=$temp

        # Swap burst times
        temp=${burst_time[j]}
```

```
        burst_time[$j]=${burst_time[$((j+1))]}
        burst_time[$((j+1))]=$temp

        # Swap process IDs
        temp=${pid[j]}
        pid[$j]=${pid[$((j+1))]}
        pid[$((j+1))]=$temp
      fi
    done
  done
}

calcWaitingtime() {
  t=0
  arrival=0
  is_completed=0
  chart=() # Array to store the Gantt chart
  gantt_line=() # Array to store the Gantt chart line (process execution)

  while [ $is_completed -eq 0 ]; do
    is_completed=1
    for ((i=0; i<$n; i++)); do
      if [ ${burst_time[$i]} -gt 0 ] && [ ${arrival_time[$i]} -le $arrival ]; then
        is_completed=0
        if [ ${burst_time[$i]} -gt $quantum ]; then
          # Process the quantum time
          for ((j=0; j<$quantum; j++)); do
            chart[$t]=$((i+1))  # Track which process is running
            gantt_line[$t]="P${pid[$i]}" # Record the process ID in the gantt line
            ((t++))
```

```
        done
        burst_time[$i]=$((burst_time[$i] - quantum))
      else
        # Process the remaining burst time
        for ((j=0; j<${burst_time[$i]}; j++)); do
        chart[$t]=$((i+1))
        gantt_line[$t]="P${pid[$i]}"
          ((t++))
        done
        burst_time[$i]=0
        completion_time[$i]=$t
      fi
    fi
  done
arrival=$t
done

for ((i=0; i<$n; i++)); do
tat[$i]=$((completion_time[$i] - arrival_time[$i]))
waiting_time[$i]=$((tat[$i] - burst_time_copy[$i]))
done

total_wt=0
total_tat=0
border
printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" "Process Id" "Burst time"
"Arrival time" "Waiting time" "Turn around time" "Completion time"
border
for ((i=0; i<$n; i++)); do
  total_wt=$((total_wt + waiting_time[$i]))
  total_tat=$((total_tat + tat[$i]))
```

```
    completion_time[$i]=$((arrival_time[$i] + tat[$i]))
    printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]}
${burst_time_copy[$i]} ${arrival_time_copy[$i]} ${waiting_time[$i]} ${tat[$i]}
${completion_time[$i]}
  done
  border

  avgwt=$(echo "scale=3; $total_wt / $n" | bc)
  echo "Average waiting time = $avgwt"
  avgtat=$(echo "scale=3; $total_tat / $n" | bc)
  echo "Average turn around time = $avgtat"

  # Gantt Chart Display
  echo -e "\nGantt Chart:"

  # Print the Gantt chart header
  for ((i=0; i<$t; i++)); do
    if [ $((i % 4)) -eq 0 ]; then
      echo -n " ---"
    fi
  done
  echo ""

  # Print the Gantt chart line (Process IDs)
  for ((i=0; i<$t; i++)); do
    echo -n "|${gantt_line[$i]} "
  done
  echo "|"

  # Print the Gantt chart footer (Time units)
  for ((i=0; i<$t; i++)); do
```

```
    if [ $((i % 4)) -eq 0 ]; then
      echo -n " ---"
    fi
  done
  echo ""

  # Print the time units below the Gantt chart
  for ((i=0; i<$t; i++)); do
    if [ $((i % 4)) -eq 0 ]; then
      echo -n "$i    "
    fi
  done
  echo "$t"
}

border() {
  z=121
  for ((i=0; i<$z; i++)); do
    echo -n "-"
  done
  echo ""
}

# Main execution starts here
echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++)); do
  echo -n "Enter Process Id: "
  read pid[$i]
  echo -n "Enter arrival time: "
```

```
  read arrival_time[$i]
  arrival_time_copy[$i]=${arrival_time[$i]}
  echo -n "Enter burst time: "
  read burst_time[$i]
burst_time_copy[$i]=${burst_time[$i]} done

echo -n "Enter quantum size: " read
quantum

sort calcWaitingtime
```

OUTPUT:

```
~$ nano ROUND_ROBIN.sh
~$ chmod +x ROUND_ROBIN.sh
~$ ./ROUND_ROBIN.sh
Enter the number of processes: 4
Enter Process Id: 0
Enter arrival time: 2
Enter burst time: 5
Enter Process Id: 1
Enter arrival time: 0
Enter burst time: 3
Enter Process Id: 2
Enter arrival time: 5
Enter burst time: 4
Enter Process Id: 3
Enter arrival time: 6
Enter burst time: 8
Enter quantum size: 3
-------------------------------------------------------------------------------------------------------------------
|Process Id        |Burst time          |Arrival time      |Waiting time        |Turn around time  |Completion time     |
-------------------------------------------------------------------------------------------------------------------
|1                 |5                   |2                 |-2                  |3                 |3                   |
|0                 |3                   |0                 |3                   |6                 |8                   |
|2                 |4                   |5                 |6                   |10                |15                  |
|3                 |8                   |6                 |6                   |14                |20                  |
-------------------------------------------------------------------------------------------------------------------
Average waiting time = 3.250
Average turn around time = 8.250

Gantt Chart:
--------------------
|P1 |P1 |P1 |P0 |P0 |P0 |P0 |P0 |P2 |P2 |P2 |P3 |P3 |P3 |P2 |P3 |P3 |P3 |P3 |P3 |
--------------------
0    4    8    12    16    20
~$
```