Ex No: 6 Date:	STUDY & USE OF COMMANDS FOR PERFORMING ARITHMETIC OPERATIONS WITH UNIX/LINUX

AIM

To Study & use of commands for performing arithmetic operations with Unix/Linux.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License). We can do arithmetic operations in shell script in a serval way (using let command, using expr command).

PROGRAM

1. Using echo Command or Double Parenthesis

```
#!/bin/sh

a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
```

```
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

if [ $a == $b ]
then
   echo "a is equal to b"
fi

if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

OUTPUT

2. Using let Command

```
#!/bin/bash

x=10
y=3

let "z = $(( x * y ))"  # multiplication
echo $z
let z=$((x*y))
echo $z

let "z = $(( x / y ))"  # division
```

```
echo $z
let z=$((x/y))
echo $z
```

OUTPUT

3. Using expr Command

```
#!/bin/bash
a=10
b=3

# there must be spaces before/after the operator
sum=`expr $a + $b`
echo $sum

sub=`expr $a - $b`
echo $sub

mul=`expr $a \* $b`
echo $mul

div=`expr $a / $b`
echo $div
```

OUTPUT

Pre and Post Lab Questions

1. What is the 'expr' command used for? Give an example.
2. How can you perform floating-point arithmetic in the shell?
3. Explain the difference between $((...)) and $(...) in bash.
4. How do you use the 'bc' command for complex calculations?
5. What is the purpose of the 'scale' variable in 'bc'?
6. How can you generate random numbers in bash?
7. Explain how to use variables in arithmetic operations within a shell script.
8. What is the significance of the '$?' variable after an arithmetic operation?
9. How do you perform exponentiation in bash?
10. Explain the use of the 'let' command for arithmetic operations.

RESULT:

Thus the Study & use of commands for performing arithmetic operations with Unix/Linux are Executed and the output is obtained successfully.

Ex No: 7 Date:	EXECUTE SHELL COMMANDS THROUGH VI EDITOR

AIM

To Execute shell commands through VI editor.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

The default editor that comes with the Linux/UNIX operating system is called vi (visual editor). Using vi editor, we can edit an existing file or create a new file from scratch. we can also use this editor to just read a text file. The advanced version of the vi editor is the vim editor.

Vi Command Mode :

When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the window. This mode allows us to move through a file, and delete, copy, or paste a piece of text. Enter into Command Mode from any other mode, requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.

Vi Insert mode:

This mode enables you to insert text into the file. Everything that’s typed in this mode is interpreted as input and finally, it is put in the file. The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode, you simply type

i. To get out of insert mode, press the Esc key, which will put you back into command mode.

Vi Last Line Mode (Escape Mode):

Line Mode is invoked by typing a colon [:], while vi is in Command Mode. The cursor will jump to the last line of the screen and vi will wait for a command. This mode enables you to perform tasks such as saving files and executing commands.

A shell is a special user program that provides an interface to the user to use operating system services. Shell accepts human-readable commands from the user and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

Vi COMMANDS

$ vi <filename>	:	Open or edit a file
I	:	Switch to Insert mode
Esc	:	Switch to Command mode
:w	:	Save and continue editing
:wq or ZZ	:	Save and quit/exit vi
:q!	:	Quit vi and do not save changes
Yy	:	Yank (copy) a line of text
P	:	Paste a line of yanked text below the current line
o	:	Open a new line under the current line
O	:	Open a new line above the current line
A	:	Append to the end of the line
a	:	Append after the cursor's current position
I	:	Insert text at the beginning of the current line
b	:	Go to the beginning of the word
E	:	Go to the end of the word
x	:	Delete a single character
dd	:	Delete an entire line
Xdd	:	Delete X number of lines
Xyy	:	Yank X number of lines
G	:	Go to the last line in a file
XG	:	Go to line X in a file
gg	:	Go to the first line in a file
:num	:	Display the current line's line number
h	:	Move left one character
j	:	Move down one line
k	:	Move up one line
L	:	Move right one character

SHELL COMMANDS

1). Displaying the file contents on the terminal:

cat: It is generally used to concatenate the files. It gives the output on the standard output.

more: It is a filter for paging through text one screenful at a time.

less: It is used to viewing the files instead of opening the file.Similar to more command but it allows backward as well as forward movement.

head : Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.

tail : Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

2). File and Directory Manipulation Commands:

mkdir : Used to create a directory if not already exist. It accepts the directory name as an input parameter.

cp : This command will copy the files and directories from the source path to the destination path. It can copy a file/directory with the new name to the destination path. It accepts the source file/directory and destination file/directory.

mv : Used to move the files or directories. This command's working is almost similar to cp command but it deletes a copy of the file or directory from the source path.

rm : Used to remove files or directories.

touch : Used to create or update a file.

3). Extract, sort, and filter data Commands:

grep : This command is used to search for the specified text in a file.

grep with Regular Expressions: Used to search for text using specific regular expressions in file.

sort : This command is used to sort the contents of files.

wc : Used to count the number of characters, words in a file.

cut : Used to cut a specified part of a file.

4). Basic Terminal Navigation Commands:

ls : To get the list of all the files or folders.

ls -l: Optional flags are added to ls to modify default behavior, listing contents in extended form -l is used for "long" output

ls -a: Lists of all files including the hidden files, add -a flag
cd: Used to change the directory.
du: Show disk usage.
pwd: Show the present working directory.
man: Used to show the manual of any command present in Linux.
rmdir: It is used to delete a directory if it is empty.
ln file1 file2: Creates a physical link.
ln -s file1 file2: Creates a symbolic link.
locate: It is used to locate a file in Linux System
echo: This command helps us move some data, usually text into a file.
df: It is used to see the available disk space in each of the partitions in your system.
tar: Used to work with tarballs (or files compressed in a tarball archive)

5). File Permissions Commands:

chown : Used to change the owner of the file.
chgrp : Used to change the group owner of the file.
chmod : Used to modify the access/permission of a user.

Pre and Post Lab Questions

1. How do you enter command mode in Vi to execute shell commands?
2. What is the syntax for executing a single shell command from within Vi?
3. How can you execute multiple shell commands in sequence from Vi?
4. Explain how to insert the output of a shell command into your current file in Vi.
5. What's the difference between using '!' and ':!' for executing shell commands in Vi?
6. How do you use Vi to edit a command's output before inserting it into your file?
7. Can you execute a shell script from within Vi? If so, how?
8. How would you use a Vi command to sort the lines in your current file?
9. Explain how to use Vi to search and replace text using sed or awk commands.
10. How can you save the output of a shell command executed in Vi to a new file?

RESULT:

Thus the shell commands through VI editor are Executed and the output is obtained successfully.

Ex No: 8 Date:	STUDY AND USE OF THE COMMAND FOR CHANGING FILE PERMISSIONS

AIM

To Study and use of the Command for changing file permissions.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

Using Linux as your operating system allows you to easily provide access to many users simultaneously. However, that access also presents potential security risks. Understanding the variety and types of Linux file permissions for users and groups will ensure that your system is optimally secure.

There are three options for permission groups available in Linux. These are

owners: these permissions will only apply to owners and will not affect other groups.

groups: you can assign a group of users specific permissions, which will only impact users within the group.

all users: these permissions will apply to all users, and as a result, they present the greatest security risk and should be assigned with caution.

There are three kinds of file permissions in Linux:

Read (r): Allows a user or group to view a file.

Write (w): Permits the user to write or modify a file or directory.

Execute (x): A user or grup with execute permissions can execute a file or view a directory.

PERMISSION COMMANDS

1. Change directory permissions

chmod +rwx filename	**:**	to add permissions
chmod -rwx directoryname	**:**	to remove permissions.
chmod +x filename	**:**	to allow executable permissions.
chmod -wx filename	**:**	to take out write and executable permissions.

2. Change directory permissions for the Group Owners and Others

chmod g+w filename	**:**	to add permissions to group
chmod g-wx filename	**:**	to remove write and executable permissions to group.
chmod o+w filename	**:**	to allow write permissions to others.
chmod o-rwx foldername	**:**	to take out read, write and executable permissions to others.
chmod ugo+rwx foldername	**:**	to give read, write, and execute to everyone.
chmod a=r foldername	**:**	to give only read permission for everyone.

3. Change Groups of Files and Directories

chgrp groupname filename	**:**	To change the group ownership of a file or directory
chgrp groupname foldername	**:**	To change the group ownership of a folder

4. Changing ownership

chown name filename	**:**	Change ownership to file
chown name foldername	**:**	Change ownership to folder.
chown -R name:filename /home/name/directoryname	**:**	combine the group and ownership.

5. Changing Linux permissions in numeric code

0 =	No Permission	+	Add permissions
1 =	Execute	-	Remove permissions
2 =	Write	=	Set the permissions to the specified values
3 =	Write execute	u	User
4 =	Read	g	Group
5 =	Read execute	o	Others
6 =	Read write	a	All three
7 =	Read write execute		

Pre and Post Lab Questions

1. What is the primary command used to change file permissions in Unix/Linux?
2. Explain the meaning of the three sets of rwx in file permissions.
3. How do you use numeric (octal) notation to set file permissions?
4. What's the difference between chmod +x and chmod 755?
5. How can you recursively change permissions for a directory and its contents?
6. Explain the concept of SUID, SGID, and sticky bit. How do you set them?
7. What does the command "chmod go-rwx filename" do?
8. How do you view the current permissions of a file?
9. What's the difference between chmod and chown commands?
10. How would you give read and write permissions to the owner, and only read permissions to others?

RESULT:

Thus the Study and use of the Command for changing file permission are Executed and the output is obtained successfully.

Ex No: 9 Date:	DEVELOP SCHEDULING ALGORITHM FOR REAL TIME APPLICATIONS

AIM

To Develop scheduling algorithm for Real time Applications.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

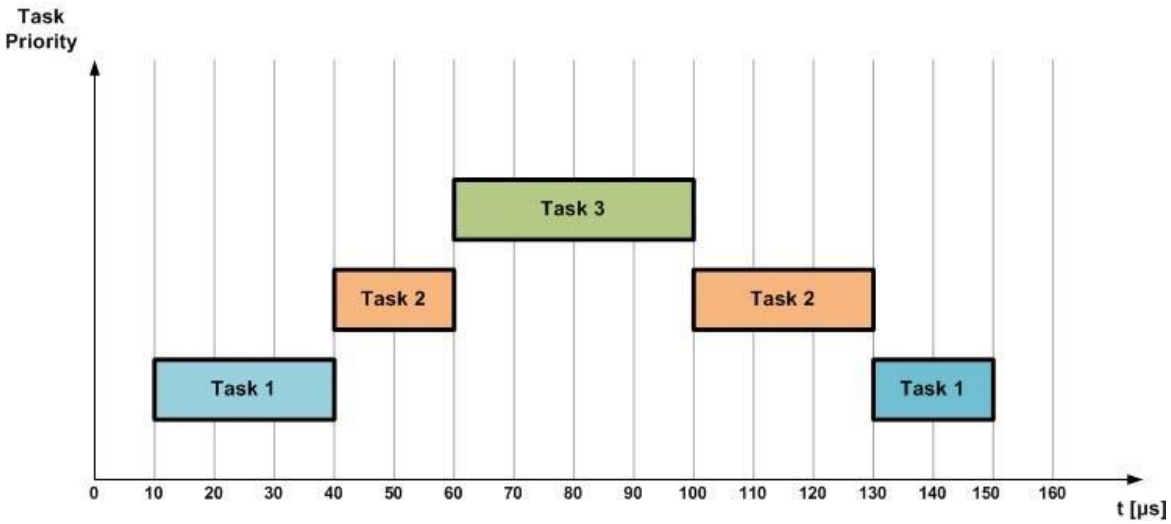
THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

There are many scheduling algorithms that can be used for scheduling task execution on a CPU. They can be classified into two main types: preemptive scheduling algorithms and non-preemptive scheduling algorithms.

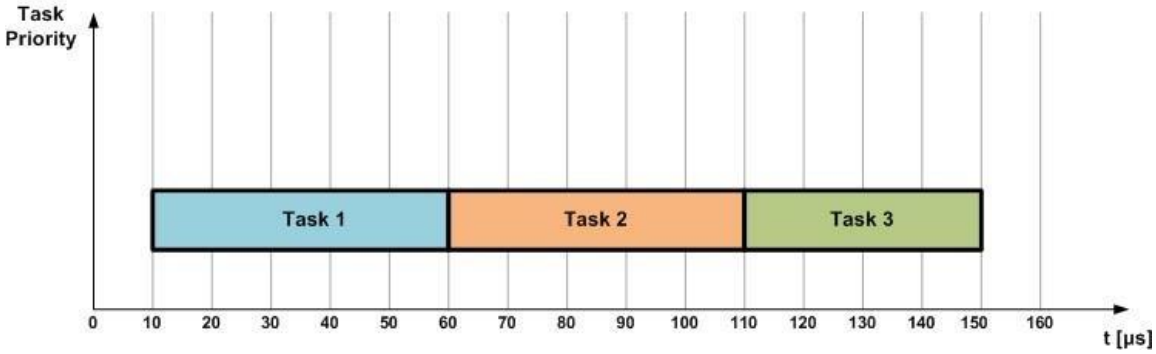
Preemptive Scheduling

Preemptive scheduling allows the interruption of a currently running task, so another one with more “urgent” status can be run. The interrupted task is involuntarily moved by the scheduler from running state to ready state. This dynamic switching between tasks that this algorithm employs is, in fact, a form of multitasking. It requires assigning a priority level for each task. A running task can be interrupted if a task with a higher priority enters the queue.



Non-preemptive Scheduling (a.k.a Co-Operative Scheduling)

In non-preemptive scheduling, the scheduler has more restricted control over the tasks. It can only start a task and then it has to wait for the task to finish or for the task to voluntarily return the control. A running task can't be stopped by the scheduler. The non-preemptive scheduling can simplify the synchronization of the tasks, but that is at the cost of increased response times to events. This reduces its practical use in complex real-time systems.



The most used algorithms in practical RTOS are non-preemptive scheduling, round-robin scheduling, and preemptive priority scheduling.

First Come, First Served (FCFS)

FCFS is a non-preemptive scheduling algorithm that has no priority levels assigned to the tasks. The task that arrives first into the scheduling queue (i.e enters ready state), gets put into the running state first and starts utilizing the CPU. It is a relatively simple scheduling algorithm where all the tasks will get executed eventually. The response time is high as this is a non-preemptive type of algorithm.

Shortest Job First (SJF)

In the shortest job first scheduling algorithm, the scheduler must obtain information about the execution time of each task and it then schedules the one with the shortest execution time to run next. SJF is a non-preemptive algorithm, but it also has a preemptive version. In the preemptive version of the algorithm (aka shortest remaining time) the parameter on

which the scheduling is based is the remaining execution time of a task. If a task is running it can be interrupted if another task with shorter remaining execution time enters the queue. A disadvantage of this algorithm is that it requires the total execution time of a task to be known before it is run.

Priority Scheduling

Priority scheduling is one of the most popular scheduling algorithms. Each task is assigned a priority level. The basic principle is that the task with the highest priority will be given the opportunity to use the CPU. In the preemptive version of the algorithm, a running task can be stopped if a higher priority task enters the scheduling queue. In the non-preemptive version of the algorithm once a task is started it can't be interrupted by a higher priority task. Of course, not all tasks can have unique priority levels and there will always be tasks that have the same priority. Different approaches can be used for handling the scheduling of those tasks (e.g FCFS scheduling or round-robin scheduling).

Round-Robin Scheduling

Round-robin is a preemptive type of scheduling algorithm. There are no priorities assigned to the tasks. Each task is put into a running state for a fixed predefined time. This time is commonly referred to as time-slice (aka quantum). A task can not run longer than the time-slice. In case a task has not completed by the end of its dedicated time-slice, it is interrupted, so the next task from the scheduling queue can be run in the following time slice. A pre-emptied task has an opportunity to complete its operation once it's again its turn to use a time-slice. An advantage of this type of scheduling is its simplicity and relatively easy implementation.

PROGRAMS

First Come, First Served (FCFS)

```
#!/bin/bash sort() {
   context_switches=0
   for ((i = 0; i < $n; i++)); do
      for ((j = 0; j < `expr $n - $i - 1`; j++)); do
         if [ ${arrival_time[j]} -gt ${arrival_time[$((j+1))]} ]; then
            # swap
            temp=${arrival_time[j]}
            arrival_time[$j]=${arrival_time[$((j+1))]}
            arrival_time[$((j+1))]=$temp
            temp=${burst_time[j]}
            burst_time[$j]=${burst_time [$((j+1))]}
            burst_time[$((j+1))]=$temp
            temp=${pid[j]}
            pid[$j]=${pid[$((j+1))]}
            pid[$((j+1))]=$temp
            context_switches=$((context_switches+1))
```

```
                elif [ ${arrival_time[j]} -eq ${arrival_time[$((j+1))]} ]; then
                    if [ ${pid[j]} -eq ${pid[$((j+1))]} ]; then
                        temp=${arrival_time[j]}
                fi
                fi done
            done
        }
arrival_time[$j]=${arrival_time[$((j+1))]}
arrival_time[$((j+1))]=$temp temp=${burst_time[j]}
burst_time[$j]=${burst_time[$((j+1))]}
burst_time[$((j+1))]=$temp temp=${pid[j]}
pid[$j]=${pid[$((j+1))]} pid[$((j+1))]=$temp
context_switches=$((context_switches+1)border(){ z=121
            for ((i=0; i<$z; i++))
            do
            echo -n "-"
            done
            echo ""
            }

            findWaitingTime(){
                service_time[0]=0
                waiting_time[0]=0
                for ((i=1; i<$n; i++))
                do
                    z=1
                    y=`expr $i - $z`
                    service_time[$i]=`expr ${service_time[$y]} + ${burst_time[$y]} `
                    waiting_time[$i]=`expr ${service_time[$i]} - ${arrival_time[$i]}`
                    if [ ${waiting_time[$i]} -lt 0 ]
                    then
                        waiting_time[$i]=0
                    fi
                done
            }
            findTurnAroundTime(){
                for ((i=0; i<$n; i++))
                do
                    tat[$i]=`expr ${waiting_time[$i]} + ${burst_time[$i]}`
                done
            }

        findAverageTime() {
            sort
```

```
    findWaitingTime
    findTurnAroundTime
    total_wt=0
    total_tat=0
    echo ""
    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=$((tat[$i] + total_tat))
        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
    echo ""
    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"
    echo ""
    echo "GANTT CHART: "
    echo ""
    for ((i = 0; i < 8 * n + n + 1; i++)); do
        echo -n "-"
    done
    echo ""

    for ((i = 0; i < $n; i++)); do
        echo -n "|  "
        echo -n "P${pid[$i]}"
        echo -n "  "
    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
        echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`"
```

```
        echo -n "       "
    done
    echo ""
    echo "Number of context switches: $context_switches"
}

echo ""
echo "--FIRST COME FIRST SERVED SCHEDULING--"
echo ""
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[$i]
    echo -n "Enter burst time: "
    read burst_time[$i]
done
findAverageTime
```

Shortest Job First (SJF)

```
#!/bin/bash
 border() {
    z=121
    for ((i=0; i<$z; i++)); do
        echo -n "-"
    done
    echo ""
}

arrangeArrival() {
    z=1
    for ((i=0; i<$n; i++)); do
        for ((j=i+1; j<$n; j++)); do
            if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]; then
                temp=${arrival_time[$j]}
                arrival_time[$j]=${arrival_time[$i]}
                arrival_time[$i]=$temp

                temp=${burst_time[$j]}
                burst_time[$j]=${burst_time[$i]}
                burst_time[$i]=$temp
```

```
            temp=${burst_time_copy[$j]}
            burst_time_copy[$j]=${burst_time_copy[$i]}
            burst_time_copy[$i]=$temp

            temp=${pid[$j]}
            pid[$j]=${pid[$i]}
            pid[$i]=$temp
        fi
    done
done
}


arrangeBurst() {
    z=1
    for ((i=0; i<$n; i++)); do
        for ((j=i+1; j<$n; j++)); do
            if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]; then
                if [ ${burst_time[$i]} -gt ${burst_time[$j]} ]; then
                    temp=${arrival_time[$j]}
                    arrival_time[$j]=${arrival_time[$i]}
                    arrival_time[$i]=$temp

                    temp=${burst_time[$j]}
                    burst_time[$j]=${burst_time[$i]}
                    burst_time[$i]=$temp

                    temp=${burst_time_copy[$j]}
                    burst_time_copy[$j]=${burst_time_copy[$i]}
                    burst_time_copy[$i]=$temp

                    temp=${pid[$j]}
                    pid[$j]=${pid[$i]}
                    pid[$i]=$temp
                fi
            fi
        done
    done
}
```

```
timecalc() { is_completed=0
current_time=0 cp=0
        count=0
        max=1000
        context_switches=0
        for ((i=0; i<$n; i++)); do
            if [ ${arrival_time[$i]} -le $current_time ]; then
                if [ ${burst_time[$i]} -lt $max ]; then
                    if [ ${burst_time[$i]} -ne 0 ]; then
                        cp=$i
                        max=${burst_time[$i]}
                        if [ ${burst_time[$i]} -eq ${burst_time_copy[$i]} ]; then
                            waiting_time[$i]=$current_time
                        fi
                    fi
                fi
            fi
        done
        while [ $is_completed -eq 0 ]; do
            if [ $count -eq $n ]; then
                is_completed=1
                h=$current_time
            fi
            chart[$current_time]=`expr $cp + 1`
            ((current_time++))
            if [ ${burst_time[$cp]} -gt 0 ]; then
                burst_time[$cp]=`expr ${burst_time[$cp]} - 1`
                max=${burst_time[$cp]}
                if [ ${burst_time[$cp]} -eq 0 ]; then
                    ((count++))
                    completion_time[$cp]=$current_time
                    max=1000
                fi
            fi
            prevcp=$cp
            for ((i=0; i<$n; i++)); do
                if [ ${arrival_time[$i]} -le $current_time ]; then
                    if [ ${burst_time[$i]} -lt $max ]; then
                        if [ ${burst_time[$i]} -ne 0 ]; then
                            cp=$i
                            max=${burst_time[$i]}
                        fi
                    fi
```

```
            fi
        done
        if [ $prevcp -ne $cp ]; then
            waiting_time[$i]=$current_time
            context_switches=$((context_switches+1))
        fi
    done

    for ((i=0; i<$n; i++)); do
        waiting_time[$i]=`expr ${completion_time[$i]} - ${arrival_time[$i]} -
${burst_time_copy[$i]}`
        if [ ${waiting_time[$i]} -lt 0 ]; then
            waiting_time[$i]=0
        fi
        tat[$i]=`expr ${waiting_time[$i]} + ${burst_time_copy[$i]}`
    done
    total_wt=0
    total_tat=0
    echo ""
    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=`expr $total_wt + ${waiting_time[$i]}`
        total_tat=`expr ${tat[$i]} + $total_tat`
        completion_time=`expr ${arrival_time[$i]} + ${tat[$i]}`
        printf "|%-18s|%-20s|%-18s|%-20s|%-18s|%-20s|\n" ${pid[$i]}
${burst_time_copy[$i]} ${arrival_time[$i]} ${waiting_time[$i]} ${tat[$i]}
$completion_time
    done
    border
    echo ""
    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 = $avgtat"
    echo ""
    echo "GANTT CHART:"
    echo ""
    count_cols=1
    cols_id[0]=${chart[0]}
    cols[0]=0
```

```
    j=1
    for ((i=1; i<$h; i++)); do
        if [ ${chart[$i]} -ne ${chart[`expr $i - 1`]} ]; then
            ((count_cols++))
            cols[$j]=$i
            cols_id[$j]=${chart[$i]}
            ((j++))
        fi
    done
    echo ""

    for ((i=0; i<8*count_cols+count_cols+1; i++)); do
        echo -n "-"
    done
    echo ""

    for ((i=0; i<$count_cols; i++)); do
        echo -n "|   "
        echo -n "P${cols_id[$i]}"
        echo -n "   "
    done
    echo "|"
    for ((i=0; i<8*count_cols+count_cols+1; i++)); do
        echo -n "-"
    done
    echo ""
    echo -n "0     "
    for ((i=1; i<$count_cols; i++)); do
        echo -n "${cols[$i]}"
        echo -n "        "
    done
    echo -n "$h"
    echo ""
    echo "Number of context switches: $context_switches"
}
echo ""
echo "--PRE-EMPTIVE SHORTEST JOB FIRST SCHEDULING--"
echo ""
echo -n "Enter the number of processes: "
read n
for ((i=0; i<$n; i++)); do
    echo -n "Enter Process Id: "
    read pid[$i]
```

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

Priority Scheduling

```
# Function to implement Priority Scheduling Algorithm (non pre-emptive)
    border(){
        z=121
        for ((i=0; i<$z; i++))
        do
        echo -n "-"
        done
        echo ""
    }

    arrangeArrival(){
        z=1
        for ((i=0; i<$n; i++))
        do
            for ((j=i+1; j<$n; j++))
            do
                if [ ${arrival_time[$i]} -gt ${arrival_time[$j]} ]
                then
                    temp=${arrival_time[$j]}
                    arrival_time[$j]=${arrival_time[$i]}
                    arrival_time[$i]=$temp

                    temp=${burst_time[$j]}
                    burst_time[$j]=${burst_time[$i]}
                    burst_time[$i]=$temp

                    temp=${priority[$j]}
                    priority[$j]=${priority[$i]}
                    priority[$i]=$temp
```

```
            fi done
        done
    }
    temp=${pid[$j]} pid[$j]=${pid[$i]} pid[$i]=$temp


    arrangePriority(){
        z=1
        for ((i=0; i<$n; i++))
        do
            for ((j=i+1; j<$n; j++))
            do
                if [ ${arrival_time[$i]} -eq ${arrival_time[$j]} ]
                then
                    if [ ${priority[$i]} -gt ${priority[$j]} ]
                    then
                        temp=${arrival_time[$j]}
                        arrival_time[$j]=${arrival_time[$i]}
                        arrival_time[$i]=$temp

                        temp=${burst_time[$j]}
                        burst_time[$j]=${burst_time[$i]}
                        burst_time[$i]=$temp

                        temp=${priority[$j]}
                        priority[$j]=${priority[$i]}
                        priority[$i]=$temp

                        temp=${pid[$j]}
                        pid[$j]=${pid[$i]}
                        pid[$i]=$temp
                    fi
                fi
            done
        done
    }

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

```
    context_switches=0
    current_process=${pid[0]}

    for ((i=1; i<$n; i++)); do
        z=1
        y=`expr $i - $z`
        service_time[$i]=`expr ${service_time[$y]} + ${burst_time[$y]} `
        waiting_time[$i]=`expr ${service_time[$i]} - ${arrival_time[$i]}`
        if [ ${waiting_time[$i]} -lt 0 ]; then
            waiting_time[$i]=0
        fi
        # Check for context switch
        if [ "${pid[$i]}" != "$current_process" ]; then
            context_switches=$((context_switches+1))
            current_process=${pid[$i]}
        fi
    done

    echo "Number of context switches: $context_switches"
}

    findTurnAroundTime(){
        for ((i=0; i<$n; i++))
        do
            tat[$i]=`expr ${waiting_time[$i]} + ${burst_time[$i]}`
        done
    }

    echo ""
    echo "--NON PRE-EMPTIVE PRIORITY SCHEDULING--"
    echo ""
    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[$i]
    echo -n "Enter burst time: "
    read burst_time[$i]
    echo -n "Enter priority: "
    read priority[$i]
```

```
    done
    arrangeArrival
    arrangePriority
    findWaitingTime
    findTurnAroundTime
    total_wt=0
    total_tat=0
    echo ""
    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=`expr $total_wt + ${waiting_time[$i]}`
        total_tat=`expr ${tat[$i]} + $total_tat`
        completion_time=`expr ${arrival_time[$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

        #echo "${burst_time[$i]}     ${arrival_time[$i]}     ${waiting_time[$i]}       ${tat[$i]} $completion_time"
    done
    border
    echo ""
    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"
    echo ""

    echo "GANTT CHART:"
    echo ""
    for ((i=0; i<8*n+n+1; i++))
    do
        echo -n "-"
        done
        echo ""
    for ((i=0; i<$n; i++))
    do
        echo -n "|   "
        echo -n "P${pid[$i]}"
        echo -n "   "
    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
       echo -n "`expr ${arrival_time[$i]} + ${tat[$i]}`                "
    done
    echo ""
rr(){
    # Function to implement Round Robin CPU scheduling algorithm.
sort() {
    for ((i = 0; i<$n; i++)); do
       for ((j = 0; j<`expr $n - $i - 1`; j++)); do
          if [ ${arrival_time[j]} -gt ${arrival_time[$((j+1))]} ]; then
             # swap
             temp=${arrival_time[j]}
             arrival_time[$j]=${arrival_time[$((j+1))]}

             arrival_time[$((j+1))]=$temp

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

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

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

             temp=${arrival_time_copy[j]}
             arrival_time_copy[$j]=${arrival_time_copy[$((j+1))]}
             arrival_time_copy[$((j+1))]=$temp
          elif [ ${arrival_time[j]} -eq ${arrival_time[$((j+1))]} ]; then
             if [ ${pid[j]} -eq ${pid[$((j+1))]} ]; then
                temp=${arrival_time[j]}
                arrival_time[$j]=${arrival_time[$((j+1))]}
```

```
                arrival_time[$((j+1))]=$temp

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

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

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




            fi
        fi
    done
done
}
}
temp=${arrival_time_copy[j]}
arrival_time_copy[$j]=${arrival_time_copy[$((j+1))]}
arrival_time_copy[$((j+1))]=$temp
```

Round-Robin Scheduling

```
#!/bin/bash
cnt=0
j=0
n=0
t=0
remain=0
flag=0
tq=0
wt=0
tat=0
context_switches=0
declare -a at bt rt
echo -n "Enter Total Process: "
read n
remain=$n
for ((cnt = 0; cnt < n; cnt++)); do
   echo -n "Enter Arrival Time and Burst Time for Process Process Number $((cnt + 1)): "
   read at[cnt]
   read bt[cnt]
   rt[cnt]=${bt[cnt]}
done

echo -n "Enter Time Quantum: "
read tq
echo -e "\n\nProcess\t|Turnaround Time|Waiting Time\n\n"
for ((t = 0, cnt = 0; remain != 0; )); do
   if ((rt[cnt] <= tq && rt[cnt] > 0)); then
      t=$((t + rt[cnt]))
      rt[cnt]=0
      flag=1
   elif ((rt[cnt] > 0)); then
      rt[cnt]=$((rt[cnt] - tq))
      t=$((t + tq))
      ((context_switches++))
   fi
   if ((rt[cnt] == 0 && flag == 1)); then
      remain=$((remain - 1))
      echo "P[$((cnt + 1))]" $'\t|\t' "$((t - at[cnt]))" $'\t|\t' "$((t - at[cnt] - bt[cnt]))"
      wt=$((wt + t - at[cnt] - bt[cnt]))
      tat=$((tat + t - at[cnt]))
      flag=0
   fi
```

```
    if ((cnt == n - 1)); then

        cnt=0
    elif ((at[cnt + 1] <= t)); then
        cnt=$((cnt + 1))
    else
        cnt=0
    fi
done
echo -e "\nAverage Waiting Time= $(awk 'BEGIN {printf "%.2f", '$wt'/'$n'}')"
echo -e "\nAvg Turnaround Time = $(awk 'BEGIN {printf "%.2f", '$tat'/'$n'}')"
echo -e "\nNumber of Context Switches: $context_switches"
exit 0
```

Pre and Post Lab Questions

1. What are the real-time scheduling policies available in Linux, and how do they differ from standard scheduling policies?
2. Explain the purpose and usage of the SCHED_FIFO and SCHED_RR scheduling classes in Linux.
3. How can you set and modify the priority of a real-time process in Linux using system calls?
4. What is the role of the Linux kernel's completely fair scheduler (CFS) in real-time applications?
5. How does the PREEMPT_RT patch modify Linux for improved real-time performance?
6. Explain how to use the chrt command to set real-time attributes for a process in Linux.
7. What are the limitations of using standard Linux for hard real-time applications?
8. How can you implement periodic tasks in Linux using timer functions like timer_create()?
9. Explain the concept of CPU affinity and how it can be used to improve real-time performance in multi-core systems.
10. How can you measure and analyze the scheduling behavior of real-time tasks in Linux using tools like ftrace or perf?

RESULT:

Thus the programs for scheduling algorithm for Real time Applications are Executed and the output is obtained successfully.

Ex No: 10 Date:	MINI PROJECT USING EMBEDDED LINUX PLATFORM

AIM

To Develop a Mini project using Embedded Linux Platform.

APPARATUS REQUIRED

1. Personal Computer with Linux
2. Keyboard

THEORY

Linux is an open-source Unix-like operating system-based family on the Linux kernel, and the OS kernel was first published on 17 September 1991 by Linus Torvalds. Typically, Linux is packaged as the Linux distribution, which contains the supporting libraries and system software and kernel, several of which are offered by the GNU Project. Several Linux distributions use the term "Linux" in the title, but the Free Software Foundation uses the "GNU/Linux" title to focus on the necessity of GNU software, causing a few controversies. Originally, Linux was designed for personal computers that were Intel x86 architecture-based, but it has since been moved to more environments than other operating systems. Including Android, Linux has the biggest installed base of every general-purpose operating system because of the control of the Linux-based Android over smartphones as of May 2022. Linux executes on many embedded systems, i.e., devices whose OS is typically designed into the firmware and is extremely customized to the system. It includes spacecraft (Perseverance rover, Dragon crew capsule, and Falcon 9 rocket), automobiles (Toyota, Hyundai, Mercedes-Benz, Audi, and Tesla), televisions (LG and Samsung Smart TVs), video game consoles, smart home devices, automation controls, and routers. Linux is one of the most outstanding examples of open-source and free software collaboration. The source code may be distributed, modified, and used non-commercially or commercially by everyone under the conditions of its respective licenses, like the GNU GPL (General Public License).

TENTATIVE PROJECT TITLES

1. Smart Home Automation System
    - Control lights, temperature, and appliances
    - Implement voice control or smartphone app interface
2. Weather Station
    - Collect temperature, humidity, and pressure data
    - Display on LCD and upload to a web server
3. Network Attached Storage (NAS) Device
    - Create a simple file server
    - Implement user authentication and file sharing
4. Surveillance Camera System

- Capture video and stream over network
- Implement motion detection and alert system

5. Digital Signage Player
    - Display dynamic content on a screen
    - Support remote content updates
6. Raspberry Pi-based Robot
    - Implement movement control and sensor integration
    - Add computer vision capabilities
7. IoT Gateway
    - Collect data from various sensors
    - Process and forward data to cloud services
8. Embedded Music Player
    - Play audio files from local storage or streaming services
    - Implement playlist management and equalizer
9. Vehicle Tracking System
    - Use GPS module for location tracking
    - Transmit data over cellular network
10. Smart Mirror
    - Display time, weather, and personalized information
    - Implement gesture or voice control
11. Industrial Process Monitor
    - Collect and display data from industrial sensors
    - Implement alarm system for out-of-range values
12. Automatic Plant Watering System
    - Monitor soil moisture and control water pump
    - Implement scheduling and remote monitoring
13. Energy Monitoring System
    - Measure and log power consumption
    - Provide analytics and suggestions for energy saving
14. Facial Recognition Door Lock
    - Use camera and ML for face recognition
    - Control electronic door lock
15. Portable Game Console
    - Emulate retro games
    - Implement custom controller interface

Pre and Post Lab Questions

1. What embedded Linux platform did you choose for your project, and why?
2. Explain the boot process of your embedded Linux system. How does it differ from a standard PC boot process?
3. How did you cross-compile your application for the target embedded platform?
4. What challenges did you face in integrating hardware peripherals with your embedded Linux system?
5. Explain how you implemented real-time features in your project using Linux. Did you use any specific scheduling policies?
6. How did you manage power consumption in your embedded Linux project?
7. What method did you use for inter-process communication in your project? Why did you choose this method?
8. Explain how you implemented error handling and system recovery in your embedded Linux application.
9. How did you ensure the security of your embedded Linux system, especially if it's network-connected?
10. What tools and techniques did you use for debugging your application on the embedded Linux platform?

RESULT:

Thus the Mini project using Embedded Linux Platform is completed and the output is obtained successfully.