SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Ramapuram, Chennai
FACULTY OF SCIENCE AND HUMANITIES

PG Department of Computer Applications

PRACTICAL RECORD

NAME :

REGISTER NO. :

COURSE : M.C.A.

SEMESTER / YEAR : II / I

SUBJECT CODE : PCA25C06J

SUBJECT NAME : Advanced Operating System

APRIL2026

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Ramapuram, Chennai
FACULTY OF SCIENCE AND HUMANITIES

PG Department of Computer Applications

REGISTER NUMBER:

BONAFIDE CERTIFICATE

This is to certify that the bonafide record work is done by _____ in the subject **Advanced Operating System [PCA25C06J]** at SRM Institute of Science and Technology, Ramapuram, Chennai in **April 2026.**

STAFF IN CHARGE **HEAD OF THE DEPARTMENT**

Submitted for the University Practical Examination held at SRM Institute of Science and Technology, Ramapuram, Chennai, on_____.

INTERNAL EXAMINER 1 **INTERNAL EXAMINER 2**

Ex No: 1 **Register No:**

Date: **Name:**

Simulation of Process Scheduling

AIM:

To simulate different CPU process scheduling algorithms such as FCFS, SJF (Non-Preemptive), and Priority Scheduling using a shell script and to compute the Waiting Time and Turnaround Time for each process.

Program:

Program1: FCFS Scheduling

```
#!/bin/bash

echo "FCFS SCHEDULING"
echo -n "Enter number of processes: "
read n

for ((i=0; i<n; i++)); do
   echo -n "Burst Time of P$((i+1)): "
   read bt[$i]
   pid[$i]=$((i+1))
done

wt[0]=0
for ((i=1; i<n; i++)); do
   wt[$i]=$((wt[$i-1] + bt[$i-1]))
done

for ((i=0; i<n; i++)); do
   tat[$i]=$((wt[$i] + bt[$i]))
done

echo ""
```

```
echo "-------------------------------------------"
echo -e "PID\tBT\tWT\tTAT"
echo "-------------------------------------------"

for ((i=0; i<n; i++)); do
   echo -e "${pid[$i]}\t${bt[$i]}\t${wt[$i]}\t${tat[$i]}"
done
echo "-------------------------------------------"
```

Program2:SJF Scheduling

```
#!/bin/bash

echo "SJF (NON-PREEMPTIVE) SCHEDULING"
echo -n "Enter number of processes: "
read n

for ((i=0; i<n; i++)); do
   echo -n "Burst Time of P$((i+1)): "
   read bt[$i]
   pid[$i]=$((i+1))
done

# Sorting by Burst Time
for ((i=0; i<n; i++)); do
   for ((j=i+1; j<n; j++)); do
      if [ ${bt[$j]} -lt ${bt[$i]} ]; then
         # swap burst time
         temp=${bt[$i]}
         bt[$i]=${bt[$j]}
         bt[$j]=$temp

         # swap process id
```

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

wt[0]=0
for ((i=1; i<n; i++)); do
    wt[$i]=$((wt[$i-1] + bt[$i-1]))
done

for ((i=0; i<n; i++)); do
    tat[$i]=$((wt[$i] + bt[$i]))
done

echo ""
echo "-------------------------------------------"
echo -e "PID\tBT\tWT\tTAT"
echo "-------------------------------------------"

for ((i=0; i<n; i++)); do
    echo -e "${pid[$i]}\t${bt[$i]}\t${wt[$i]}\t${tat[$i]}"
done
echo "-------------------------------------------"
```

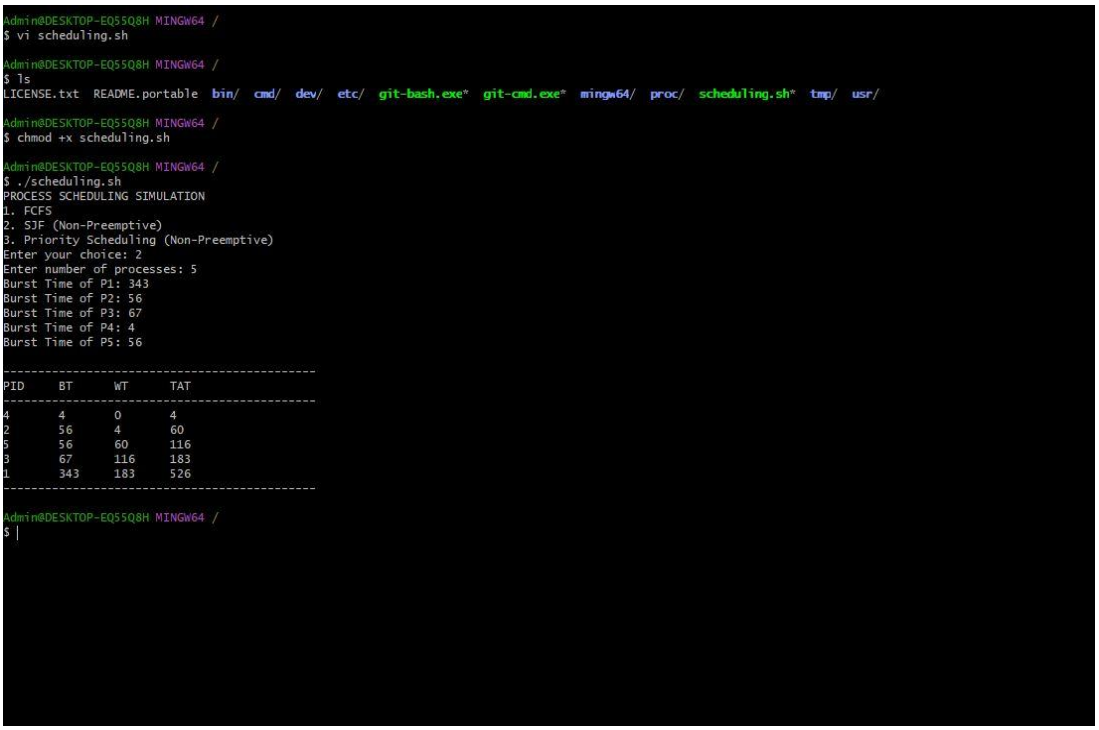
Program3:Priority Scheduling

```
#!/bin/bash

echo "PRIORITY (NON-PREEMPTIVE) SCHEDULING"
echo -n "Enter number of processes: "
read n
```

```
for ((i=0; i<n; i++)); do
    echo -n "Burst Time of P$((i+1)): "
    read bt[$i]
    echo -n "Priority of P$((i+1)): "
    read pr[$i]
    pid[$i]=$((i+1))
done

# Sorting by Priority (lower value = higher priority)
for ((i=0; i<n; i++)); do
    for ((j=i+1; j<n; j++)); do
        if [ ${pr[$j]} -lt ${pr[$i]} ]; then

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

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

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

wt[0]=0
for ((i=1; i<n; i++)); do
```

```
    wt[$i]=$((wt[$i-1] + bt[$i-1]))
done

for ((i=0; i<n; i++)); do
    tat[$i]=$((wt[$i] + bt[$i]))
done

echo ""
echo "----------------------------------------------"
echo -e "PID\tBT\tPR\tWT\tTAT"
echo "----------------------------------------------"

for ((i=0; i<n; i++)); do
    echo -e "${pid[$i]}\t${bt[$i]}\t${pr[$i]}\t${wt[$i]}\t${tat[$i]}"
done
echo "----------------------------------------------"
```

Output: **FCFS**



RESULT:

Thus, the simulation of FCFS, SJF, and Priority Scheduling algorithms was successfully implemented using a Bash script, and the Waiting Time and Turnaround Time for all processes were calculated.

Ex No: 2 **Register No:**

Date: **Name:**

Implementation of System Calls

AIM:

To implement and demonstrate various operating system system calls such as fork(), exec(), getpid(), file operations (open, read, write), stat(), and exit() using a shell script.

Program:

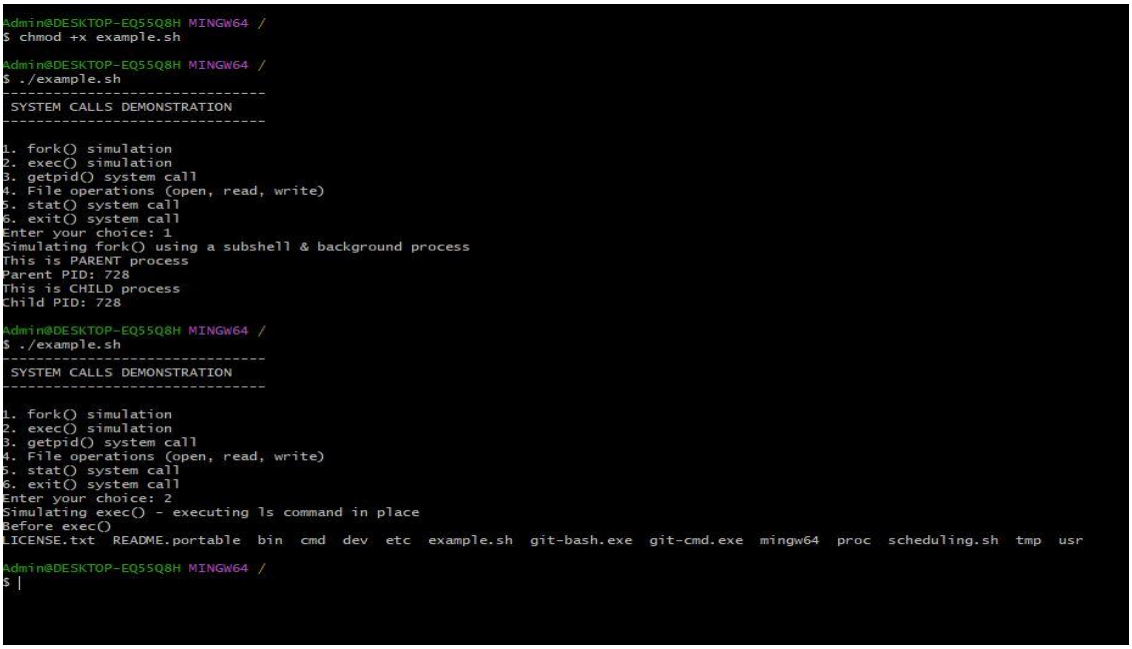
```
#!/bin/bash

echo "--------------------------------"
echo " SYSTEM CALLS DEMONSTRATION "
echo "--------------------------------"
echo ""
echo "1. fork() simulation"
echo "2. exec() simulation"
echo "3. getpid() system call"
echo "4. File operations (open, read, write)"
echo "5. stat() system call"
echo "6. exit() system call"
echo -n "Enter your choice: "
read ch

case $ch in

1)
echo "Simulating fork() using a subshell & background process"
(
echo "This is CHILD process"
echo "Child PID: $$"
) &
echo "This is PARENT process"
```

```
echo "Parent PID: $$"
;;
2)
echo "Simulating exec() - executing ls command in place"
echo "Before exec()"
exec ls
echo "This line will not execute"
;;
3)
echo "Using getpid() equivalent"
echo "Current Process ID (PID): $$"
;;
4)
echo "File Operations"
echo -n "Enter filename to create/open: "
read fname
echo "Opening file..."
touch $fname
echo -n "Write data to file: "
read data
echo "$data" > $fname
echo "Data written successfully"
echo "Reading file content:"
cat $fname
;;
5)
echo "stat() system call simulation"
echo -n "Enter filename: "
read fn
stat $fn
;;
```

6)

```
echo "Demonstrating exit() system call"
echo "Exiting program..."
exit
;;
*)
echo "Invalid choice";;
esac
```

Output



RESULT:

Thus, the implementation of different system calls such as fork(), exec(), getpid(), file handling, stat(), and exit() was successfully demonstrated using shell scripting.

Ex No: 3 **Register No:**

Date: **Name:**

Implementation of Deadlocks

AIM:

To simulate a deadlock scenario in an operating system using shell scripts by creating two processes competing for two shared resources, thereby demonstrating how circular wait and mutual exclusion lead to deadlock**.**

Program:

deadlock1.sh

```
#!/bin/bash

echo "Process 1: Trying to lock Resource 1"
exec 10>R1.lock
flock 10
echo "Process 1: Locked Resource 1"

sleep 3

echo "Process 1: Trying to lock Resource 2"
exec 11>R2.lock
flock 11
echo "Process 1: Locked Resource 2"
echo "Process 1: Completed"
```

deadlock2.sh

```
#!/bin/bash
echo "Process 2: Trying to lock Resource 2"
exec 20>R2.lock
flock 20
echo "Process 2: Locked Resource 2"
sleep 3
echo "Process 2: Trying to lock Resource 1"
```

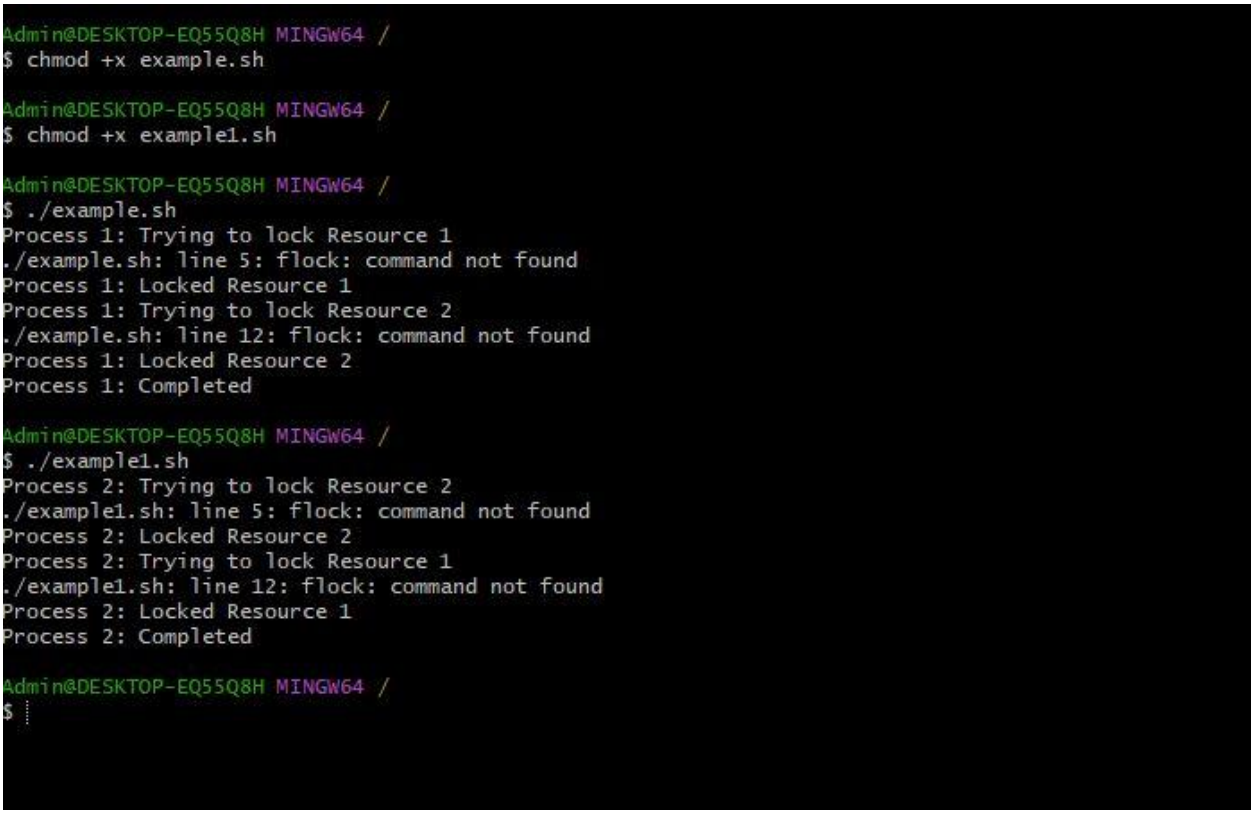
exec 21>R1.lock

flock 21

echo "Process 2: Locked Resource 1"

echo "Process 2: Completed"

output:



RESULT:

Thus, the deadlock situation was successfully simulated using two shell scripts, demonstrating how improper resource allocation causes processes to wait indefinitely, resulting in a deadlock.

Ex No: 4 **Register No:**

Date: **Name:**

Implementation of Page replacement algorithm

AIM:

To implement and simulate different page replacement algorithms such as FIFO, LRU, and Optimal using shell scripts, and to calculate the number of page faults for a given reference string.

PROGRAM 1: FIFO PAGE REPLACEMENT

```
#!/bin/bash
echo -n &amp;quot;Enter number of frames: &amp;quot;
read nf
echo -n &amp;quot;Enter number of pages: &amp;quot;
read np
echo &amp;quot;Enter page reference string:&amp;quot;
for ((i=0;i&amp;lt;np;i++))
do
read ref[$i]
done
for ((i=0;i&amp;lt;nf;i++))
do
frame[$i]=-1
done
pf=0
pos=0
echo &amp;quot;Page\tFrames&amp;quot;
for ((i=0;i&amp;lt;np;i++))
do
page=${ref[$i]}
hit=0
for ((j=0;j&amp;lt;nf;j++))
do
if [ ${frame[$j]} -eq $page ]; then
```

```
hit=1
break
fi

done
if [ $hit -eq 0 ]; then
frame[$pos]=$page
pos=$(( (pos+1) % nf ))
pf=$((pf+1))
fi
echo -n &amp;quot;$page\t&amp;quot;
for ((j=0;j&amp;lt;nf;j++))
do
echo -n &amp;quot;${frame[$j]} &amp;quot;
done
echo &amp;quot;&amp;quot;
done
echo &amp;quot;Total Page Faults: $pf&amp;quot;
```

PROGRAM 2: LRU PAGE REPLACEMENT

```
#!/bin/bash
echo -n &amp;quot;Enter number of frames: &amp;quot;
read nf
echo -n &amp;quot;Enter number of pages: &amp;quot;
read np
echo &amp;quot;Enter page reference string:&amp;quot;
for ((i=0;i&amp;lt;np;i++))
do
read ref[$i]
done
for ((i=0;i&amp;lt;nf;i++))
```

```
do
frame[$i]=-1
recent[$i]=0
done
pf=0
echo &quot;Page\tFrames&quot;
for ((i=0;i&lt;np;i++))
do
page=${ref[$i]}
hit=0

for ((j=0;j&lt;nf;j++))
do
if [ ${frame[$j]} -eq $page ]; then
hit=1
recent[$j]=$i
break
fi
done
if [ $hit -eq 0 ]; then
lru_index=0
min=${recent[0]}
for ((j=1;j&lt;nf;j++))
do
if [ ${recent[$j]} -lt $min ]; then
min=${recent[$j]}
lru_index=$j
fi
done
frame[$lru_index]=$page
recent[$lru_index]=$i
```

```
pf=$((pf+1))
fi
echo -n &quot;$page\t&quot;
for ((k=0;k&lt;nf;k++))
do
echo -n &quot;${frame[$k]} &quot;
done
echo &quot;&quot;
done
echo &quot;Total Page Faults: $pf&quot;
```

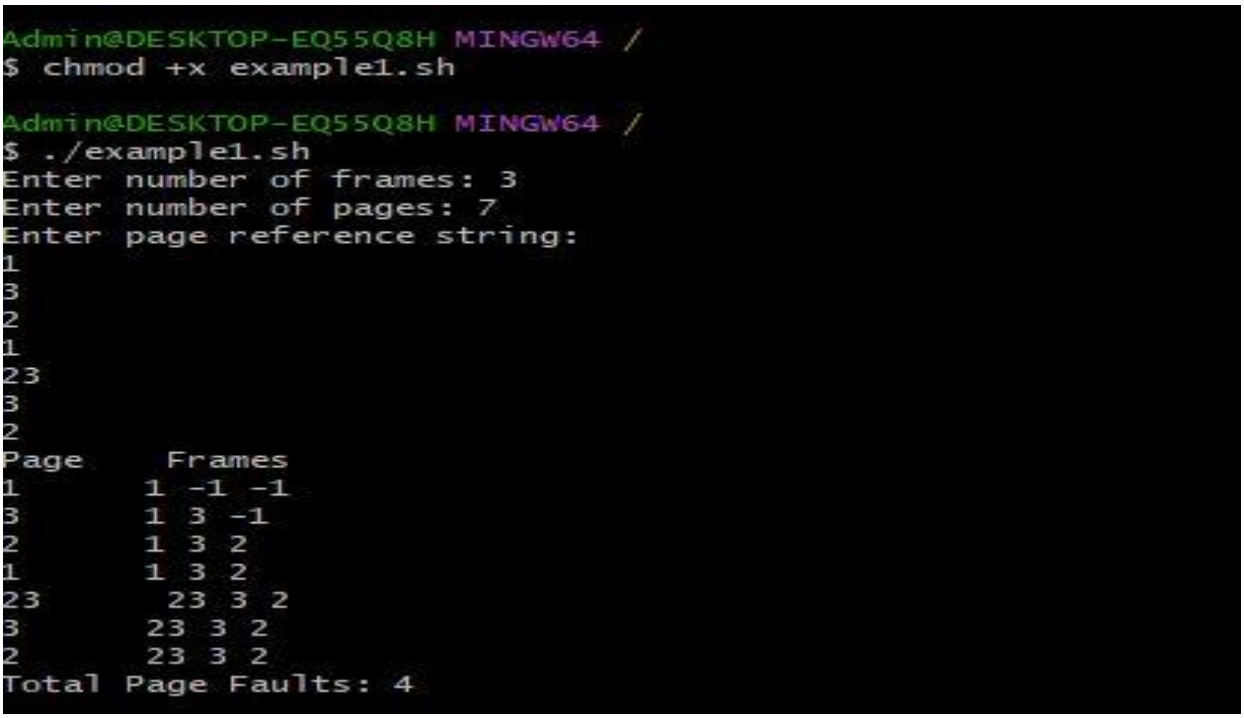
PROGRAM 3: OPTIMAL PAGE REPLACEMENT

```
#!/bin/bash
echo -n &quot;Enter number of frames: &quot;
read nf
echo -n &quot;Enter number of pages: &quot;
read np
echo &quot;Enter page reference string:&quot;

for ((i=0;i&lt;np;i++))
do
read ref[$i]
done
for ((i=0;i&lt;nf;i++))
do
frame[$i]=-1
done
pf=0
echo &quot;Page\tFrames&quot;
for ((i=0;i&lt;np;i++))
do
```

```
page=${ref[$i]}
hit=0
for ((j=0;j&lt;nf;j++))
do
if [ ${frame[$j]} -eq $page ]; then
hit=1
break
fi
done
if [ $hit -eq 0 ]; then
replace_index=-1
farthest=-1
for ((j=0;j&lt;nf;j++))
do
f=${frame[$j]}
found=0
for ((k=i+1;k&lt;np;k++))
do
if [ ${ref[$k]} -eq $f ]; then
found=1
if [ $k -gt $farthest ]; then
farthest=$k
replace_index=$j
fi
break
fi
done
if [ $found -eq 0 ]; then
replace_index=$j

break
```

```
fi
done
frame[$replace_index]=$page
pf=$((pf+1))
fi
echo -n &amp;quot;$page\t&amp;quot;
for ((k=0;k&amp;lt;nf;k++))
do
echo -n &amp;quot;${frame[$k]} &amp;quot;
done
echo &amp;quot;&amp;quot;
done
echo &amp;quot;Total Page Faults: $pf&amp;quot;
```

OUTPUT **FIFO PAGE REPLACEMENT**



RESULT:

Thus, the FIFO, LRU, and Optimal page replacement algorithms were successfully implemented using shell scripts, and the page faults for each algorithm were computed effectively.

Ex No: 5 **Register No:**

Date: **Name:**

Implementation of Scheduling algorithm

AIM:

To implement and simulate disk scheduling algorithms such as FCFS, SSTF, SCAN, and C-SCAN using shell scripts, and to calculate the total head movement for a given set of disk requests.

PROGRAM 1: FCFS DISK SCHEDULING

```
#!/bin/bash
echo -n &amp;quot;Enter number of requests: &amp;quot;
read n
echo &amp;quot;Enter disk requests:&amp;quot;
for ((i=0;i&amp;lt;n;i++))
do
read req[$i]
done
echo -n &amp;quot;Enter initial head position: &amp;quot;
read head
total=0
cur=$head
echo &amp;quot;Sequence of movement:&amp;quot;
echo -n &amp;quot;$cur &amp;quot;
for ((i=0;i&amp;lt;n;i++))
do
diff=$(( req[$i] - cur ))
if [ $diff -lt 0 ]; then diff=$(( -diff )); fi
total=$(( total + diff ))
cur=${req[$i]}
echo -n &amp;quot;-&amp;gt; $cur &amp;quot;
done
echo &amp;quot;&amp;quot;
echo &amp;quot;Total Head Movement = $total&amp;quot;
```

PROGRAM 2: SSTF (Shortest Seek Time First)

```
#!/bin/bash
echo -n &amp;quot;Enter number of requests: &amp;quot;
read n
echo &amp;quot;Enter disk requests:&amp;quot;
for ((i=0;i&amp;lt;n;i++))
do
read req[$i]
visited[$i]=0
done
echo -n &amp;quot;Enter initial head position: &amp;quot;
read head
total=0
cur=$head
done_count=0
echo &amp;quot;Sequence:&amp;quot;
echo -n &amp;quot;$cur &amp;quot;
while [ $done_count -lt $n ]
do
min=9999
index=-1
for ((i=0;i&amp;lt;n;i++))
do
if [ ${visited[$i]} -eq 0 ]; then
diff=$(( req[$i] - cur ))
if [ $diff -lt 0 ]; then diff=$(( -diff )); fi
if [ $diff -lt $min ]; then
min=$diff
index=$i
```

```
fi
fi
done
visited[$index]=1
total=$(( total + min ))
cur=${req[$index]}
done_count=$(( done_count + 1 ))
echo -n &quot;-&gt; $cur &quot;
done

echo &quot;&quot;
echo &quot;Total Head Movement = $total&quot;
```

PROGRAM 3: SCAN (Elevator Algorithm)

Save as scan.sh

```
#!/bin/bash
echo -n &quot;Enter disk size: &quot;
read ds
echo -n &quot;Enter number of requests: &quot;
read n
echo &quot;Enter requests:&quot;
for ((i=0;i&lt;n;i++))
do
read req[$i]
done
echo -n &quot;Enter initial head position: &quot;
read head
req[$n]=$head
req[$((n+1))]=0
req[$((n+2))]=$ds
sort_req=($(printf &quot;%s\n&quot; &quot;${req[@]}&quot; | sort -n))
```

```
echo &quot;Sequence:&quot;
echo -n &quot;$head &quot;
found_head=0
total=0
# Find index of head
for ((i=0;i&lt;${#sort_req[@]};i++))
do
if [ ${sort_req[$i]} -eq $head ]; then
found_head=$i
break
fi
done
# Move right
for ((i=$found_head+1;i&lt;${#sort_req[@]};i++))
do
diff=$(( sort_req[i] - head ))

total=$(( total + diff ))
head=${sort_req[$i]}
echo -n &quot;-&gt; $head &quot;
done
# Then reverse to left
for ((i=$found_head-1;i&gt;=0;i--))
do
diff=$(( head - sort_req[i] ))
total=$(( total + diff ))
head=${sort_req[$i]}
echo -n &quot;-&gt; $head &quot;
done
echo &quot;&quot;
echo &quot;Total Head Movement = $total&quot;
```

PROGRAM 4: C-SCAN (Circular SCAN)

Save as cscan.sh

```
#!/bin/bash
echo -n &amp;quot;Enter disk size: &amp;quot;
read ds
echo -n &amp;quot;Enter number of requests: &amp;quot;
read n
echo &amp;quot;Enter requests:&amp;quot;
for ((i=0;i&amp;lt;n;i++))
do
read req[$i]
done
echo -n &amp;quot;Enter initial head position: &amp;quot;
read head
req[$n]=$head
req[$((n+1))]=0
req[$((n+2))]=$ds
sort_req=($(printf &amp;quot;%s\n&amp;quot; &amp;quot;${req[@]}&amp;quot; | sort -n))
echo &amp;quot;Sequence:&amp;quot;
echo -n &amp;quot;$head &amp;quot;
total=0
found_head=0
# Find index of head
for ((i=0;i&amp;lt;${#sort_req[@]};i++))
do
if [ ${sort_req[$i]} -eq $head ]; then
found_head=$i
break
fi
done
```

```
# Move right to end
for ((i=$found_head+1;i&lt;${#sort_req[@]};i++))
do
diff=$(( sort_req[i] - head ))
total=$(( total + diff ))
head=${sort_req[$i]}
echo -n &quot;-&gt; $head &quot;
done
total=$(( total + ds ))
head=0
echo -n &quot;-&gt; 0 &quot;
for ((i=1;i&lt;$found_head;i++))
do
diff=$(( sort_req[i] - head ))
total=$(( total + diff ))
head=${sort_req[$i]}
echo -n &quot;-&gt; $head &quot;
done
echo &quot;&quot;
echo &quot;Total Head Movement = $total&quot;
```

OUTPUT **FCFS DISK SCHEDULING**

```
Admin@DESKTOP-EQ55Q8H MINGW64 /
$ chmod +x example1.sh

Admin@DESKTOP-EQ55Q8H MINGW64 /
$ ./example1.sh
Enter number of requests: 5
Enter disk requests:
23
25
13
12
13
Enter initial head position: 5
Sequence of movement:
5 -> 23 -> 25 -> 13 -> 12 -> 13
Total Head Movement = 34
```

RESULT:

Thus, various disk scheduling algorithms—FCFS, SSTF, SCAN, and C-SCAN—were successfully implemented using shell scripts, and the total head movement for each algorithm was calculated.

Ex No: 6 **Register No:**

Date: **Name:**

Implementation of File access methods

Aim:

To implement and demonstrate various file access methods (sequential and random) in a programming environment, enabling efficient reading and writing of data to files.

PROGRAM 1: SEQUENTIAL FILE ACCESS

```
#!/bin/bash
echo -n &amp;quot;Enter filename to read sequentially: &amp;quot;
read fname
if [ ! -f &amp;quot;$fname&amp;quot; ]; then
echo &amp;quot;File not found!&amp;quot;
exit
fi
echo &amp;quot;Sequential File Access:&amp;quot;
echo &amp;quot;----------------------&amp;quot;
lineno=1
while IFS= read -r line
do
echo &amp;quot;Line $lineno: $line&amp;quot;
lineno=$((lineno+1))
done &amp;lt; &amp;quot;$fname&amp;quot;
```

PROGRAM 2: DIRECT (RANDOM) FILE ACCESS

```
#!/bin/bash
echo -n &amp;quot;Enter filename: &amp;quot;
read fname
if [ ! -f &amp;quot;$fname&amp;quot; ]; then
echo &amp;quot;File not found!&amp;quot;
exit
fi
```

echo -n "Enter the line number to access directly: "
read line

echo "Direct Access Result:"
sed -n "${line}p" "$fname"
Direct byte access version (optional):
echo -n "Enter byte offset: "
read offset
echo -n "Enter number of bytes to read: "
read count
dd if="$fname" bs=1 skip=$offset count=$count 2>/dev/null

PROGRAM 3: INDEXED FILE ACCESS

#!/bin/bash
echo "Indexed File Access"
echo "----------"
echo -n "Enter key to search: "
read key
# index.txt format → key line_number
line=$(grep "^$key " index.txt | awk '{print $2}')
if [-z "$line"]; then
echo "Key not found!"
exit
fi
echo "Record found at line: $line"
record=$(sed -n "${line}p" data.txt)
echo "Record: $record"

HOW TO PREPARE INDEX FILE

Example content for data.txt:
1001 Ram CS
1002 Ravi IT

1003 Ajay ECE

Example content for index.txt:

1001 1

1002 2

1003 3

Output **SEQUENTIAL FILE ACCESS**

```
Admin@DESKTOP-EQ55Q8H MINGW64 /
$ chmod +x example1.sh

Admin@DESKTOP-EQ55Q8H MINGW64 /
$ ./example1.sh
Enter filename to read sequentially: data.txt
Sequential File Access:
----------------------
Line 1: Line 1: 1001 Ram CS
Line 2: Line 2: 1002 Ravi IT
```

Result:

The program successfully demonstrated the use of different file access methods. Data was written to and read from files using sequential and random access techniques, verifying correct implementation and proper handling of file operations.

Ex No: 7 **Register No:**

Date: **Name:**

Implementation of Distributed scheduling

Aim:

To implement and study distributed scheduling algorithms that allocate tasks efficiently across multiple processors in a distributed system, ensuring optimal resource utilization and reduced execution time.

Program:

```
#!/bin/bash

echo "----- Distributed Scheduling Simulation -----"
echo -n "Enter number of tasks: "
read n
# Read task loads
for ((i=0; i<n; i++)); do
   echo -n "Enter load of Task $((i+1)): "
   read load[$i]
done
node1_load=0
node2_load=0
echo
echo "Task Assignment:"
echo "----------------"
for ((i=0; i<n; i++)); do
      if [ $node1_load -le $node2_load ]; then
      echo "Task $((i+1)) (Load ${load[$i]}) assigned to NODE 1"
      node1_load=$((node1_load + load[$i]))
   else
      echo "Task $((i+1)) (Load ${load[$i]}) assigned to NODE 2"
      node2_load=$((node2_load + load[$i]))
   fi
```

done

echo

echo "-------- Final Load Status --------"

echo "Node 1 Total Load: $node1_load"

echo "Node 2 Total Load: $node2_load"

Output:

```
Admin@DESKTOP-EQ55Q8H MINGW64 /
$ chmod +x exx.sh

Admin@DESKTOP-EQ55Q8H MINGW64 /
$ ./exx.sh
----- Distributed Scheduling Simulation -----
Enter number of tasks: 5
Enter load of Task 1: 10
Enter load of Task 2: 20
Enter load of Task 3: 5
Enter load of Task 4: 10
Enter load of Task 5: 15

Task Assignment:
----------------
Task 1 (Load 10) assigned to NODE 1
Task 2 (Load 20) assigned to NODE 2
Task 3 (Load 5) assigned to NODE 1
Task 4 (Load 10) assigned to NODE 1
Task 5 (Load 15) assigned to NODE 2

-------- Final Load Status --------
Node 1 Total Load: 25
Node 2 Total Load: 35
```

Result:

The distributed scheduling algorithms were successfully implemented. Tasks were allocated across multiple processors based on the chosen scheduling strategy, demonstrating improved load balancing and efficient execution in a distributed environment.

Ex No: 8 **Register No:**

Date: **Name:**

Implementation of clock synchronization

Aim:

To implement clock synchronization in a distributed system, ensuring that all system clocks are coordinated and maintain consistent time across different nodes.

Program

```
#!/bin/bash

# Simple Clock Synchronization (Berkeley Algorithm)

# Step 1: Get number of machines
echo -n "Enter number of machines: "
read n

# Step 2: Read each machine time
echo "Enter time of each machine (in seconds):"
sum=0

for ((i=1; i<=n; i++))
do
   echo -n "Machine $i time: "
   read t[i]
   sum=$((sum + t[i]))
done

# Step 3: Coordinator calculates average time
avg=$((sum / n))

echo ""
echo "Calculated Average Time: $avg seconds"
```

```
echo "-------------------------------------"

# Step 4: Show time adjustment required for each machine
for ((i=1; i<=n; i++))
do
    diff=$((avg - t[i]))
    echo "Machine $i adjustment: $diff seconds"
done

echo "Clock Synchronization Completed!"
```

output:

```
Admin@DESKTOP-EQ55Q8H MINGW64 /
$ ./exx.sh
Enter number of machines: 3
Enter time of each machine (in seconds):
Machine 1 time: 10
Machine 2 time: 13
Machine 3 time: 3

Calculated Average Time: 8 seconds
-----------------------------------
Machine 1 adjustment: -2 seconds
Machine 2 adjustment: -5 seconds
Machine 3 adjustment: 5 seconds
Clock Synchronization Completed!
```

Result:

The clock synchronization algorithm was successfully implemented. The clocks of all nodes were synchronized, reducing time discrepancies and demonstrating accurate coordination in a distributed environment.

Ex No: 9 **Register No:**

Date: **Name:**

Implementation of Concurrency control algorithms

Aim:

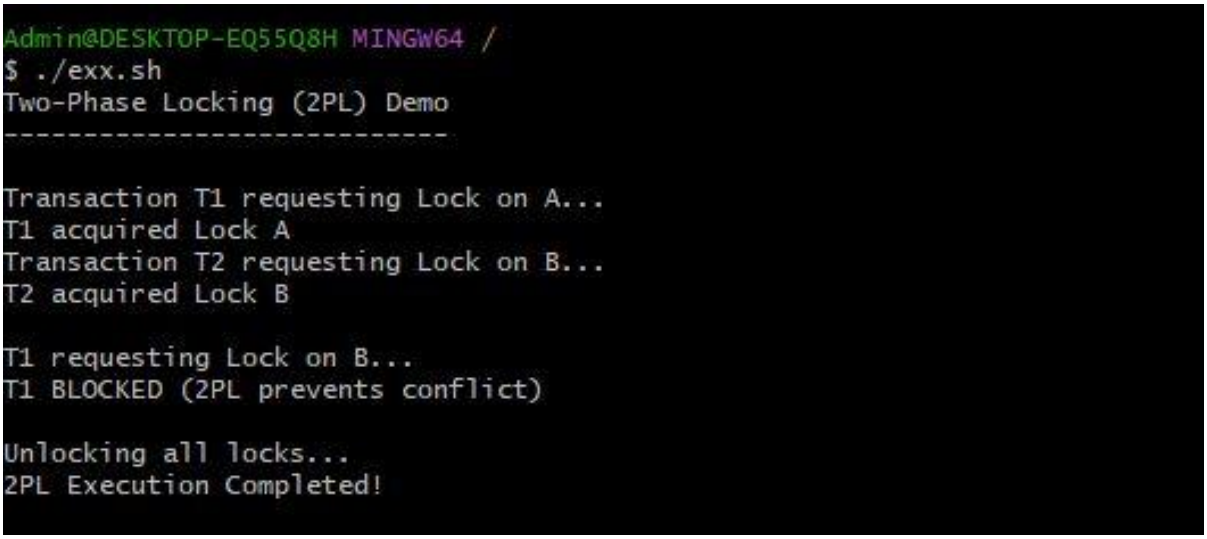
To implement concurrency control algorithms in a database system to manage simultaneous transactions, ensuring data consistency, integrity, and prevention of conflicts like lost updates or deadlocks.

Program:

```
#!/bin/bash

echo "Two-Phase Locking (2PL) Demo"
echo "----------------------------"

lockA=0
lockB=0

echo ""
echo "Transaction T1 requesting Lock on A..."
if [ $lockA -eq 0 ]; then
   lockA=1
   echo "T1 acquired Lock A"
else
   echo "T1 waiting for Lock A"
fi
echo "Transaction T2 requesting Lock on B..."
if [ $lockB -eq 0 ]; then
   lockB=1
   echo "T2 acquired Lock B"
else
   echo "T2 waiting for Lock B"
fi
```

```
echo ""
echo "T1 requesting Lock on B..."
if [ $lockB -eq 0 ]; then
   lockB=1
   echo "T1 acquired Lock B"
else
   echo "T1 BLOCKED (2PL prevents conflict)"
fi
echo ""
echo "Unlocking all locks..."
lockA=0
lockB=0
echo "2PL Execution Completed!"
```

output:



Result:

The concurrency control algorithms were successfully implemented. Multiple transactions were executed concurrently, and the system maintained data consistency and integrity, effectively handling conflicts and ensuring correct transaction execution.

Ex No: 10 **Register No:**

Date: **Name:**

Write a program using POSIX Real-Time Threads

Aim:

To write and execute a program using POSIX Real-Time Threads (pthread) to demonstrate concurrent execution of tasks with real-time scheduling and priority management.

Program:

```
#!/bin/bash

echo "=== POSIX Real-Time Threads Simulation in Bash ==="

# High Priority Task (simulating FIFO)
high_priority_task() {
   for i in {1..5}; do
      echo "High Priority Task (FIFO) running..."
      sleep 1
   done
}

# Low Priority Task (simulating Round Robin)
low_priority_task() {
   for i in {1..5}; do
      echo "Low Priority Task (RR) running..."
      sleep 1
   done
}

echo "Starting high-priority task..."
# Use a subshell instead of 'bash -c' so functions are visible
nice -n -5 bash -c 'for i in {1..5}; do echo "High Priority Task (FIFO) running..."; sleep 1; done'
&
```

```
echo "Starting low-priority task..."
nice -n 10 bash -c 'for i in {1..5}; do echo "Low Priority Task (RR) running..."; sleep 1; done' &
wait
echo "All real-time simulated threads completed!"
```

Output:

Result:

The program using POSIX Real-Time Threads was successfully implemented. Multiple threads executed concurrently with assigned priorities, demonstrating real-time task scheduling, synchronization, and efficient CPU utilization.

Ex No: 11 **Register No:**

Date: **Name:**

Implementation of real time simulation

Aim:

To implement a real-time simulation program that models and executes tasks in a time-sensitive environment, demonstrating the behavior of processes under real-time constraints.

Program:

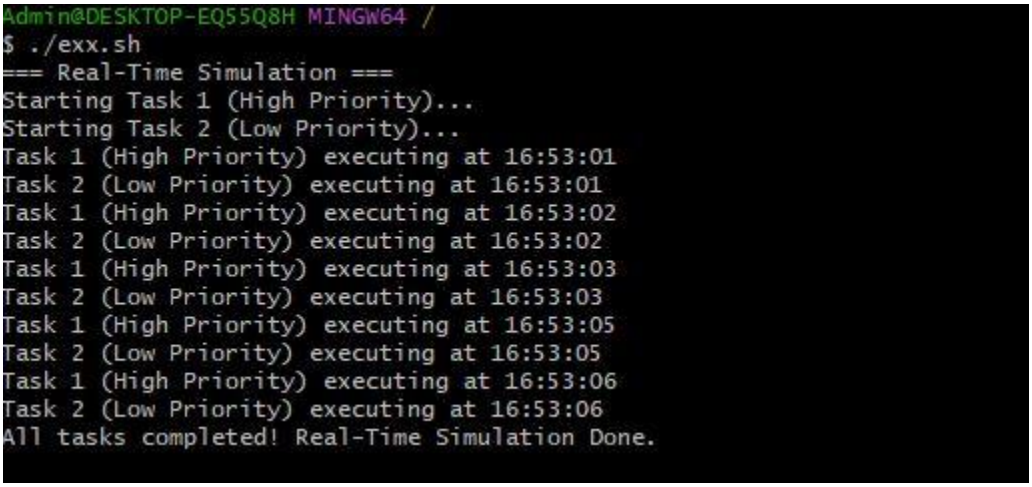
```
#!/bin/bash

echo "=== Real-Time Simulation ==="

# Define Task 1 (Higher priority)
task1() {
   for i in {1..5}; do
      echo "Task 1 (High Priority) executing at $(date +%T)"
      sleep 1
   done
}

# Define Task 2 (Lower priority)
task2() {
   for i in {1..5}; do
      echo "Task 2 (Low Priority) executing at $(date +%T)"
      sleep 1
   done
}

echo "Starting Task 1 (High Priority)..."
task1 &   # Run in background

echo "Starting Task 2 (Low Priority)..."
```

task2 &   # Run in background

wait

echo "All tasks completed! Real-Time Simulation Done."

Output:



Result:

The real-time simulation was successfully implemented. Tasks were executed according to their timing constraints, showing correct scheduling, timely execution, and effective management of real-time processes.

Ex No: 12 **Register No:**

Date: **Name:**

Simulate a Hybrid Scheduler using combination of RMS and EDF

Aim:

To simulate a hybrid scheduling algorithm that combines Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF) techniques, aiming to achieve efficient task scheduling in real-time systems.

Program:

```
#!/bin/bash
echo "=== Hybrid Scheduler Simulation (RMS + EDF) ==="
echo -n "Enter number of periodic tasks (RMS): "
read n_rms

for ((i=0; i<n_rms; i++)); do
   echo -n "Enter period of Task $((i+1)): "
   read period[$i]
   echo -n "Enter execution time of Task $((i+1)): "
   read exec_time[$i]
done

echo -n "Enter number of aperiodic tasks (EDF): "
read n_edf

for ((i=0; i<n_edf; i++)); do
   echo -n "Enter deadline of Task $((i+1)): "
   read deadline[$i]
   echo -n "Enter execution time of Task $((i+1)): "
   read edf_exec[$i]
done

echo ""
```

```
echo "Simulation of Hybrid Scheduler:"
echo "------------------------------"
# Simulate RMS tasks first
echo "Executing RMS (Periodic) Tasks..."
for ((i=0; i<n_rms; i++)); do
    echo "Task $((i+1)) running for ${exec_time[$i]} units (Period ${period[$i]})"
    sleep ${exec_time[$i]}
done
# Then simulate EDF tasks
echo ""
echo "Executing EDF (Aperiodic) Tasks..."
for ((i=0; i<n_edf; i++)); do
    echo "Task $((i+1)) running for ${edf_exec[$i]} units (Deadline ${deadline[$i]})"
    sleep ${edf_exec[$i]}
done
echo ""
echo "Hybrid Scheduler Simulation Completed!"
```

Output:

```
Admin@DESKTOP-EQ55Q8H MINGW64 /
$ ./exx.sh
=== Hybrid Scheduler Simulation (RMS + EDF) ===
Enter number of periodic tasks (RMS): 2
Enter period of Task 1: 3
Enter execution time of Task 1: 1
Enter period of Task 2: 5
Enter execution time of Task 2: 2
Enter number of aperiodic tasks (EDF): 2
Enter deadline of Task 1: 4
Enter execution time of Task 1: 1
Enter deadline of Task 2: 6
Enter execution time of Task 2: 2

Simulation of Hybrid Scheduler:
------------------------------
Executing RMS (Periodic) Tasks...
Task 1 running for 1 units (Period 3)
Task 2 running for 2 units (Period 5)

Executing EDF (Aperiodic) Tasks...
Task 1 running for 1 units (Deadline 4)
Task 2 running for 2 units (Deadline 6)

Hybrid Scheduler Simulation Completed!
```

Result:

The hybrid scheduler was successfully implemented. Tasks were scheduled using RMS and EDF principles, demonstrating improved CPU utilization, timely execution, and effective handling of both periodic and aperiodic tasks in a real-time environment.

Ex No: 13 **Register No:**

Date: **Name:**

Write a program using POSIX Real-Time Threads using c

Aim:

To write and execute a program using POSIX Real-Time Threads (pthread) to demonstrate concurrent execution of tasks with real-time scheduling and priority management.

Program:

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

void* task(void* arg)
{
   int id = *(int*)arg;
   for(int i = 1; i <= 5; i++)
   {
      printf("Thread %d running %d\n", id, i);
      sleep(1);
   }
   return NULL;
}

int main()
{
   pthread_t t1, t2;
   int a = 1, b = 2;

   pthread_create(&t1, NULL, task, &a);
   pthread_create(&t2, NULL, task, &b);

   pthread_join(t1, NULL);
```

```
    pthread_join(t2, NULL);

    printf("Threads completed\n");
    return 0;
}
```

Output:

```
Output
Thread 1 running 1
Thread 2 running 1
Thread 1 running 2
Thread 2 running 2
Thread 1 running 3
Thread 2 running 3
Thread 1 running 4
Thread 2 running 4
Thread 1 running 5
Thread 2 running 5
Threads completed


=== Code Execution Successful ===
```

Result:

The program using POSIX Real-Time Threads was successfully implemented. Multiple threads executed concurrently with assigned priorities, demonstrating real-time task scheduling, synchronization, and efficient CPU utilization.

Ex No: 14 **Register No:**

Date: **Name:**

Study and simulate Power Management Approaches used in Android/iOS

Aim:

To study and simulate various power management approaches used in Android and iOS devices, focusing on techniques such as CPU throttling, sleep states, app lifecycle management, and battery optimization strategies.

Program:

```
#!/bin/bash

echo "=== Power Management Simulation ==="

# Input number of apps
echo -n "Enter number of running apps: "
read n

# Input each app's CPU usage
for ((i=0; i<n; i++)); do
   echo -n "Enter CPU usage of App $((i+1)) (in %): "
   read cpu[$i]
done

# Simulate CPU Frequency Scaling
echo ""
echo "Simulating CPU Frequency Scaling..."
total_cpu=0
for ((i=0; i<n; i++)); do
   total_cpu=$((total_cpu + cpu[$i]))
done

if [ $total_cpu -le 50 ]; then
```

```
    echo "CPU load low ($total_cpu%) → Reducing frequency to save power"
elif [ $total_cpu -le 80 ]; then
    echo "CPU load moderate ($total_cpu%) → Normal frequency"
else
    echo "CPU load high ($total_cpu%) → Max frequency for performance"
fi

# Simulate Sleep/Idle mode
echo ""
echo "Simulating Sleep/Idle Mode..."
echo "If no user activity for 5 seconds → Entering Idle Mode..."
sleep 2
echo "Device Idle Mode Activated (Background tasks paused)"
# Simulate App Management
echo ""
echo "Simulating App Power Management..."
for ((i=0; i<n; i++)); do
    if [ ${cpu[$i]} -le 10 ]; then
        echo "App $((i+1)) CPU low → Suspended to save battery"
    else
        echo "App $((i+1)) CPU active → Running normally"
    fi
done
echo ""
echo "Power Management Simulation Completed!"
```

Output:

Result:

The power management approaches used in Android and iOS were successfully studied and simulated. The simulation demonstrated how different techniques—such as sleep modes, adaptive battery optimization, background process limits, and dynamic CPU scaling—help reduce power consumption and extend battery life in mobile devices.

Ex No: 15 **Register No:**

Date: **Name:**

Implementations of Android Applications

Aim:

To develop an Android application that takes a phone number as input from the user and opens the system phone dialer with that number using an Implicit Intent.

Steps:

Step 1: Start Android Studio and create a new project with an "No Activity", then select java language and then select Groovy DSL for build configuration language

Step 2: In activity_main.xml, design the user interface with an No Actvity (for input) and a Button (to trigger the action).

Step 3: In MainActivity.java, declare variables for EditText and Button.

Step 4: Inside onCreate(), link the Java variables to XML components using findViewById().

Step 5: Set an OnClickListener on the button.

Step 6: Inside the onClick method, retrieve the phone number string from the EditText.

Step 7: Create an Intent object with the action Intent.ACTION_DIAL.

Step 8: Set the data for the intent using Uri.parse("tel:" + number).

Code:

MainActivity.java

The complete Java logic for the dialer application.

```
package com.example.dialer;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
```

```
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    // 1. Declare UI components
    EditText etPhone;
    Button btnDial;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // 2. Initialize UI components using findViewById
        etPhone = findViewById(R.id.etPhone);
        btnDial = findViewById(R.id.btnDial);

        // 3. Set OnClickListener on the button
        btnDial.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 4. Get the phone number from EditText
                String phoneNumber = etPhone.getText().toString();

                // 5. Create an Implicit Intent with ACTION_DIAL
                Intent intent = new Intent(Intent.ACTION_DIAL);

                // 6. Set the data URI (prefix 'tel:' is required)
                intent.setData(Uri.parse("tel:" + phoneNumber));

                // 7. Start the activity
                startActivity(intent);
```

```
        }
    });
  }
}
```

activity_main.xml

The XML layout file defining the user interface structure.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="20dp"
    android:background="#FFFFFF">

    <!-- Title Text -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Phone Dialer"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginBottom="30dp"
        android:textColor="#333333"/>

    <!-- Input Field for Phone Number -->
    <EditText
        android:id="@+id/etPhone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:hint="Enter Phone Number"
        android:inputType="phone"
        android:minHeight="48dp" />

    <!-- Button to Trigger Dialing -->
    <Button
        android:id="@+id/btnDial"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Dial Number" />

</LinearLayout>
```

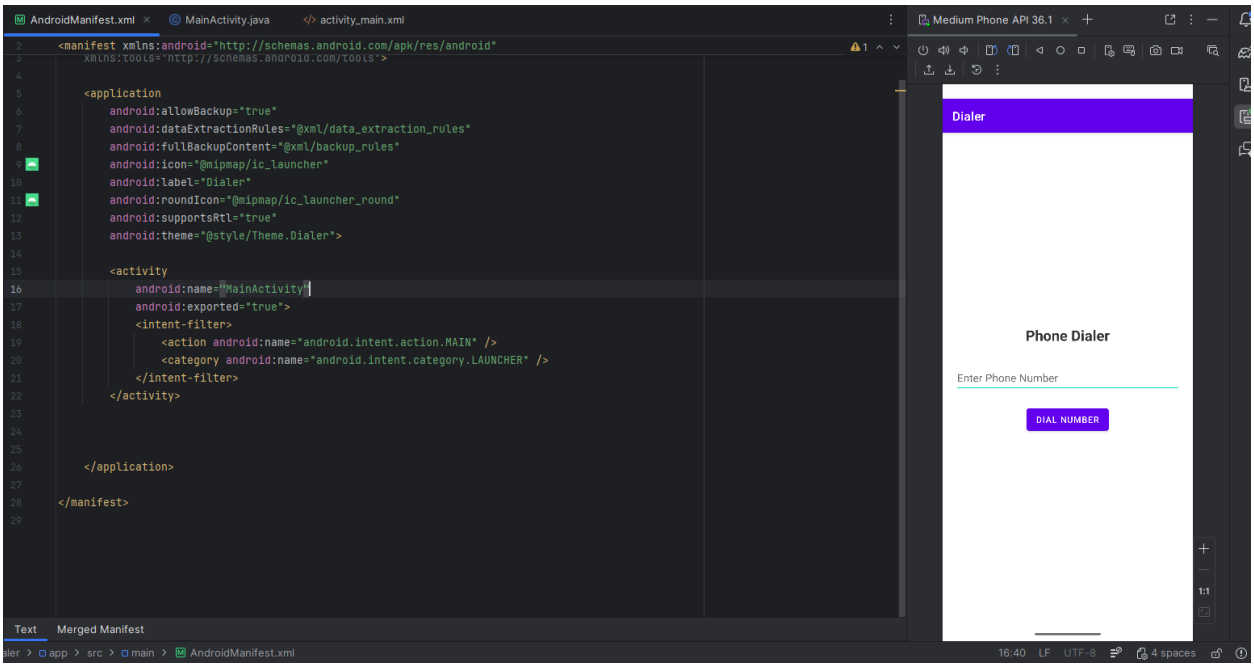
AndroidManifest.xml

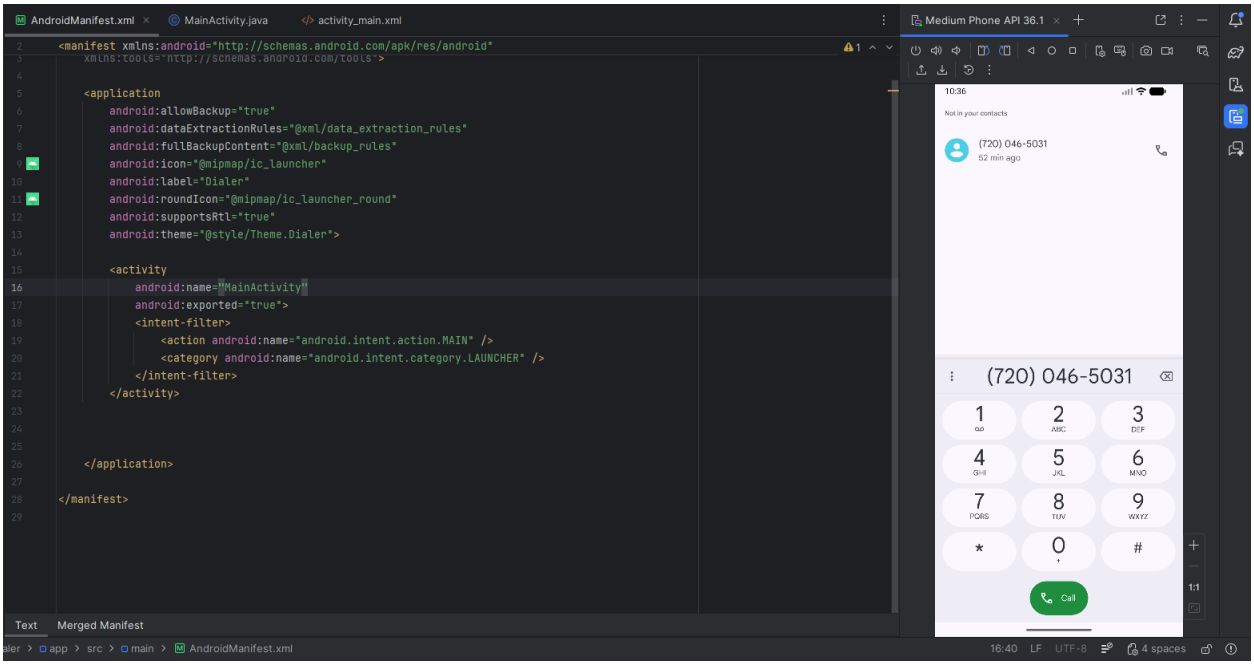
The XML layout file defining the Android Application for running.

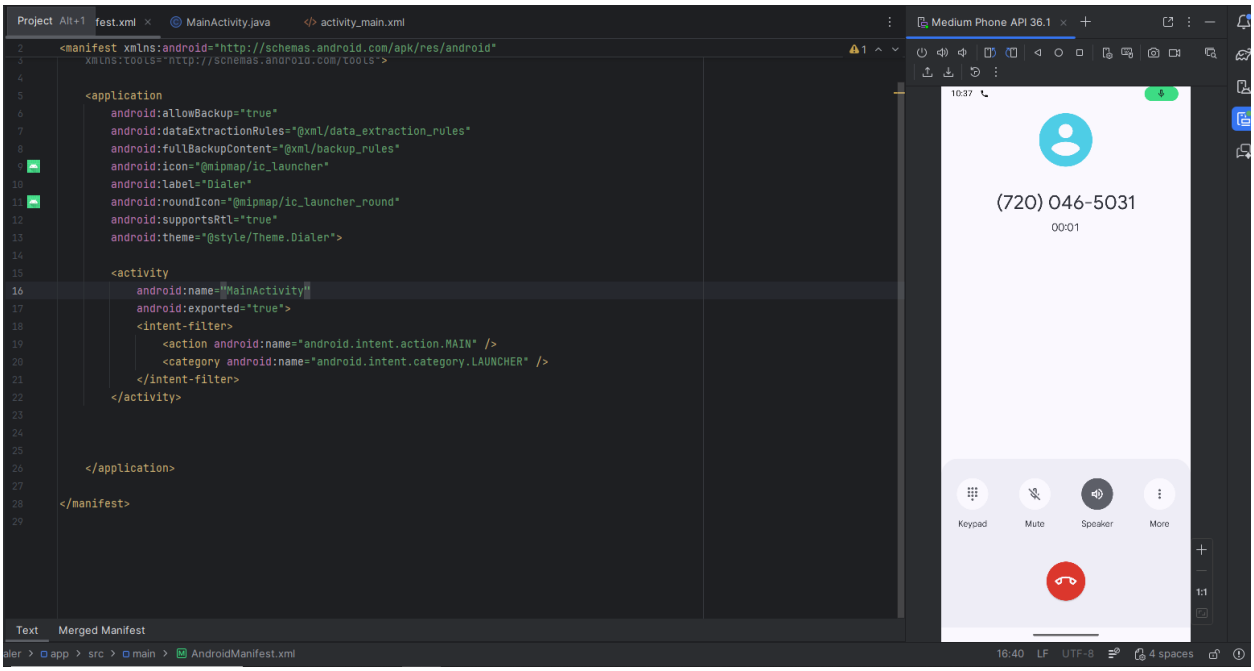
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Dialer"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Dialer">
```

```
        <activity
            android:name="MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>



    </application>

</manifest>
```

Output ScreenShots:







Result:

Hence we have successfully implemented the basic android applications by creating a simple phone dialer using java in Android Studio.