

## **System Fundamentals Experiment List**

**Explore the internal commands of Linux and Write shell scripts to do the following:**

1. Display top 10 processes in descending order  
`ps aux --sort -%cpu`
2. Display processes with highest memory usage.  
`ps aux --sort -%mem`
3. Display current logged in user and logname.  
`Whoami`  
`id`
4. Display current shell, home directory, operating system type, current path setting, current working directory.  
`echo $SHELL`  
`echo $HOME`  
`uname -s`  
`echo $PATH`  
`pwd`
5. Display OS version, release number, kernel version.  
`uname -s`  
`uname -r`  
`uname -v`
6. Write a command to display the first 15 columns from each line in the file  
`cut -c 1-15 pikachu.txt`
7. cut specified columns from a file and display them  
`cut -d " " -f 2,3 pikachu.txt`
8. Sort given file ignoring upper and lower case  
`sort -f pikachu.txt`
9. Displays only directories in current working directory.  
`ls -dF *`
10. copying files from one place to another,  
`cp pikachu.txt raichu.txt`
11. moving files from one place to another.
12. Removing specific directory with various options  
`rm -r /home/savioratharv/animals`  
`rm -rv /home/savioratharv/pokemon`  
`rm -rf /home/savioratharv/gpt`
13. list the numbers of users currently login in the system and then sort it.  
`w`
14. Merge two files into one file  
`cat squirtle.txt charmander.txt > water_and_fire.txt`
15. changes the access mode of one file  
`chmod u+w charmander.txt`

16. display the last ten lines of the file.

```
tail -10 water_and_fire.txt
```

17. to locate files in a directory and in a subdirectory.

```
find /home/savioratharv/pracs -name "squirtle.txt"
```

18. This displays the contents of all files having a name starting with ap followed by any number of characters.

```
ls sq*
```

19. Rename any file aaa to aaa.aa1, where aa1 is the user login name.

```
mv aaa aaa.$(whoami)
```

### Illustrate the use of sort, grep, awk, etc.

20. Write a command to search the word 'picture' in the file and if found, the lines containing it would be displayed on the screen.

```
grep "pictures" picturesque.txt
```

21. Write a command to search for all occurrences of 'Rebecca' as well as 'rebecca' in file and display the lines which contain one of these words.

```
grep -i "pictures" picturesque.txt
```

22. Write a command to search all four-letter words whose first letter is a 'b' and last letter, a 'k'.

```
grep '\<b[a-z][a-z]ks\>' books
```

23. Write a command to see only those lines which do not contain the search patterns

```
grep -v '\<b[a-z][a-z]k\>' books
```

### System fundamentals Algorithm

24. Implement Booth's multiplication algorithm.

25. Implement Restoring division algorithm.

26. Implement Non-Restoring division algorithm.

27. Implement fully associative memory mapped cache organization.

28. Implement various LRU cache/page replacement policy

29. Implement various optimal cache/page replacement policy
30. Implement various FIFO cache/page replacement policy
31. Implement FCFS CPU scheduling algorithm.
32. Implement SJF CPU scheduling algorithm.
33. Implement Non Prremptive Priority CPU scheduling algorithm.
34. Implement Prremptive Priority CPU scheduling algorithm.
35. Implement SRTF CPU scheduling algorithm.
36. Implement Round Robin CPU scheduling algorithm.
37. Implement Best Fit Memory allocation policy.
38. Implement First Fit Memory allocation policy.
39. Implement Worst Fit Memory allocation policy.
40. Implement Producer -Consumer problem with Semaphore.
41. Implement order scheduling in supply chain using Banker's Algorithm
42. Implement FIFO Disk Scheduling Algorithms.
43. Implement SSTF Disk Scheduling Algorithms.
44. Implement SCAN Disk Scheduling Algorithms.
45. Implement C-SCAN Disk Scheduling Algorithms.
46. Implement Look Disk Scheduling Algorithms.
47. Implement Look Disk Scheduling Algorithms.

## Shell Scripting

48. Write Shell script to copy files from one folder to another

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

```
dst_dir="/home/savioratharv/pokemons"
```

```
src_dir="/home/savioratharv/animals"
```

```
cp "$src_dir"/* "$dst_dir"/
```

```
echo "Files copied successfully"
```

49. Write Shell script Count number of words, characters and lines.

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

```
file="/home/savioratharv/pokemons/pikachu.txt"
```

```
c=$(cat "$file"|wc -c)
```

```
w=$(cat "$file"|wc -w)
```

```
l=$(cat "$file"|wc -l)
```

```
echo "Number of characters: $c"
```

```
echo "Number of words: $w"
```

```
echo "Number of lines: $l"
```

50. Write Shell script To describe files in different format.

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

```
file="/home/savioratharv/pokemons/pikachu.txt"
```

```
c=$(stat $file)
```

```
echo "$c"
```

**51.** Write Shell script to find factorial of given number using bash script

```
#!/bin/bash
echo "Enter the number"
read num
fact=1
for ((i=1;i<=$num;i++))
do
    fact=$((fact*i))
done
echo "The factorial of a number is $fact"
NOTE: Do chmod +x fourth.sh and use ./fourth.sh instead of sh fourth.sh
```

**52.** Display first 10 natural numbers using bash script

```
#!/bin/bash
for ((i=1;i<=10;i++))
do
    echo $i
done
```

**53.** Display Fibonacci series using bash script

```
#!/bin/bash
echo "How long Fibbo"
read num
a=1
b=1
echo "$a"
echo "$b"
for((i=1;i<=num;i++))
do
    c=$((a+b))
    echo "$c "
    a=$b
    b=$c
done
```

**54.** Find given number is prime or nor using bash script

```
#!/bin/bash
echo "Enter the number"
read num
count=0
for((i=2;i<num;i++))
do
    if (($num%i==0))
    then
        count=$((count+1))
    fi
done
if (($count==0))
then
    echo "Prime number"
else
    echo "Not a prime number"
fi
```

**55.** Write shell script to find biggest of three numbers

```
#!/bin/bash
echo "Enter first number"
read num1
echo "Enter second number"
read num2
echo "Enter third number"
read num3
if(($num1>$num2 && $num1>$num3))
then
    echo "Biggest number is $num1"
elif(($num2>$num1 && $num2>$num3))
then
    echo "Biggest number is $num2"
else
    echo "Biggest number is $num3"
fi
```

**56.** Write shell script to reverse a given number

```
#!/bin/bash
echo "Enter the number"
read num
new=0
while(($num>0))
do
    rem=$((num%10))
    new=$((new*10 + rem))
    num=$((num/10))
done
echo "Reverse number is: $new"
```

**57.** Write shell script to find Sum of individual digits (1234 => 1+2+3+4=10)

```
#!/bin/bash
echo "Enter number"
read num
sum=0
while(($num>0))
do
    rem=$num%10
    sum=$((sum+rem))
    num=$num/10
done
echo "The total sum of digits is $sum"
```

**58.** Write a shell script to display a list of users currently logged in.

```
#!/bin/bash
c=$(w)
echo "Users logged in:"
echo "$c"
```

**59.** Write a shell script to perform arithmetic operations.

```
#!/bin/bash
echo "1. Add, 2. Subtract, 3.Multiply, 4.Divide, 5.Exponent"
read num
echo "Enter two numbers"
read a b
case $num in
    1) echo "The sum is $((a+b))"
        ;;
    2) echo "The difference is $((a-b))"
        ;;
    3) echo "The product is $((a*b))"
        ;;
    4) echo "The division is $((a/b))"
        ;;
    5) echo "The exponent is $((a**b))"
        ;;
    *) echo "Invalid choice! Please try again!"
        ;;
esac
```

**60.** Write a shell script to copy contents of one file to another.

repeat

**61.** Write a shell program to generate multiplication table of a number upto a given range.

```
echo "Enter max range"
read range
echo "Enter number"
read num
for((i=1;i<=$range;i++))
do
    echo "$i * $num = $((i*num))"
done
```

**62.** Write a shell program to count the number of files in a directory.

```
#!/bin/bash
file="/home/savioratharv/animals"
c=$(ls -lp $file | wc -l)
echo "Number of files in directory: $c"
```

**63.** WAS to find the number of matched characters, words and lines in a file.

```
#!/bin/bash
file="/home/savioratharv/pracs/picturesque.txt"
c=$(grep -o "pictures" $file | wc -c)
w=$(grep -o "pictures" $file | wc -w)
l=$(grep -o "pictures" $file | wc -l)
echo "Number of matched characters: $c"
echo "Number of matched words: $w"
echo "Number of matched lines: $l"
```

**64.** Write a script to find the number of characters, words and lines in a file.

Repeat

**65.** Write a script to display list of files starting with particular letter in the directory.

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

```
c=$(ls -1p "/home/savioratharv/pokemons" pi*)
```

```
echo "List of files: "
```

```
echo "$c"
```

**66.** Write a script to develop a Fibonacci series.

Repeat

**67.** Write a shell script to replace the Nth occurrence of a pattern.

**68.** Write a shell script to convert temperature from Centigrade to Fahrenheit.

```
#!/bin/bash
echo "Enter fahrenheit"
read num
cel=$(( (num-32)*5/9 ))
echo "Celsius is $cel"
```

**69.** Write a shell script to compute the power of a given number.

```
#!/bin/bash
echo "Enter number and exponent"
read num exp
echo "Exponent is $((num**exp))"
```

**70.** Write a shell script to check whether the entered number is prime or not.

repeat

**71.** Write a shell script to check whether the year is leap year or not.

```
#!/bin/bash
echo "Enter year"
read num
if(( ($num%4==0 && $num%100!=0) || ($num%400==0) ))
then
    echo "Leap year"
else
    echo "Not Leap year"
fi
```

**72.** Write a shell script to check whether a number is even or odd.

```
#!/bin/bash
echo "Enter number"
read num
if(($num%2==0))
then
    echo "Even number"
else
    echo "Odd number"
fi
```



## System Fundamentals Experiment List

Explore the internal commands of Linux and Write shell scripts to do the following:

1. Display top 10 processes in descending order

```
bash
```

[Copy code](#)

```
ps aux --sort=-%cpu | head -n 11
```

2. Display processes with highest memory usage.

```
bash
```

[Copy code](#)

```
ps aux --sort=-%mem | head -n 11
```

3. Display current logged in user and logname.

```
bash
```

[Copy code](#)

```
whoami
```

```
logname
```

4. Display current shell, home directory, operating system type, current path setting, current working directory.

```
bash
```

[Copy code](#)

```
$ echo $SHELL
```

```
/bin/bash
```

```
$ echo $HOME
```

```
/home/john.doe
```

```
$ uname -s
```

```
Linux
```

```
$ echo $PATH
```

```
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

```
$ pwd
```

```
/home/john.doe/Documents
```

5. Display OS version, release number, kernel version.

```
makefile Copy code

$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal

$ uname -r
5.4.0-80-generic
```

6. Write a command to display the first 15 columns from each line in the file

```
shell Copy code

cut -f 1-15 <filename>
```

7. cut specified columns from a file and display them

```
shell Copy code

cut -f 1,3,5 <filename>
```

8. Sort given file ignoring upper and lower case

```
shell Copy code

sort -f myfile.txt
```


9. Displays only directories in current working directory.

```
shell Copy code

ls -d */
```

10. copying files from one place to another,


shell

 Copy code

```
cp file.txt /path/to/destination/
```

### 11. moving files from one place to another.


shell

 Copy code

```
mv file.txt /path/to/destination/
```

### 12. Removing specific directory with various options


shell

 Copy code

```
rm -r <directory>
```

### 13. list the numbers of users currently login in the system and then sort it.


shell

 Copy code

```
who | cut -d' ' -f1 | sort | uniq | wc -l
```

### 14. Merge two files into one file


shell

 Copy code

```
cat file1.txt file2.txt > merged.txt
```

### 15. changes the access mode of one file


shell

 Copy code

```
chmod 744 file.txt
```

2. To add execute permission for all users (owner, group, and others):


shell

 Copy code

```
chmod +x file.sh
```

3. To remove write permission for the group and others:


shell

 Copy code

```
chmod go-w file.txt
```

### 16. display the last ten lines of the file.

shell


 Copy code

```
tail -n 10 myfile.txt
```

**17. to locate files in a directory and in a subdirectory.**

Here's the command to locate files in a directory and its subdirectories:

shell


 Copy code

```
find <directory> -type f
```

Replace `**<directory>**` with the path to the directory you want to search for files in.

Here's an example command:


shell

 Copy code

```
find /path/to/directory -type f
```

**18. This displays the contents of all files having a name starting with ap followed by any number of characters.**

shell


 Copy code

```
cat ap*
```

This command uses the asterisk (\*) wildcard character after "ap" to match any number of characters following it. It will display the contents of all files in the current directory that have names starting with "ap".

If you want to search for these files in a specific directory, you can provide the directory path before the wildcard:

shell

 Copy code

```
cat /path/to/directory/ap*
```

**19. Rename any file aaa to aaa.aa1, where aa1 is the user login name.**

```
shell Copy code  
  
mv aaa "aaa.$LOGNAME"
```

### Illustrate the use of sort, grep, awk, etc.

20. Write a command to search the word 'picture' in the file and if found, the lines containing it would be displayed on the screen.

```
shell Copy code  
  
grep "picture" filename.txt
```

Replace `filename.txt` with the actual name of the file you want to search in. When you run this command, `grep` will scan the file for lines containing the word "picture" and display those lines on the screen. If the word appears multiple times in a line, the entire line will be displayed.

If you want to perform a case-insensitive search, meaning it will match both "picture" and "Picture," you can add the `-i` flag:

```
shell Copy code  
  
grep -i "picture" filename.txt
```

21. Write a command to search for all occurrences of 'Rebecca' as well as 'rebecca' in file and display the lines which contain one of these words.

```
shell Copy code  
  
grep -i "Rebecca\|rebecca" filename.txt
```

22. Write a command to search all four-letter words whose first letter is a 'b' and last letter, a 'k'.

```
shell Copy code  
  
grep -i '^b[a-z]k$' filename.txt
```

23. Write a command to see only those lines which do not contain the search patterns

```
shell Copy code  
  
grep -v "pattern" filename.txt
```

### System fundamentals Algorithm

24. Implement Booth's multiplication algorithm.

25. Implement various LRU cache/page replacement policy

✓  
0s



#LRU - SF - Ref String

```
capacity = 3  
processList = [ 7, 0, 2,1,0,5,4,6,0,1]  
  
s = []  
  
pageFaults = 0  
  
for i in processList:  
    if i not in s:  
        if(len(s) == capacity):  
            s.remove(s[0])  
            s.append(i)  
        else:  
            s.append(i)  
        pageFaults +=1  
    else:  
        s.remove(i)  
        s.append(i)  
  
print("Page Faults : {}".format(pageFaults))
```



Page Faults : 9

## 26. Implement various optimal cache/page replacement policy

## ▼ OPTIMAL

```

frames = 3
ref_strings = "31216513"
n = len(ref_strings)
hit = 0
miss = 0
rank = [0,0,0]
Queue = ["", "", ""]


for i in range(n):
    item = ref_strings[i]
    if item not in Queue:
        #if no page hit
        miss += 1
        if "" not in Queue:
            #if queue is full
            for item_index, queue_item in enumerate(Queue):
                if queue_item not in ref_strings[i:]:
                    #if queue item does not exist in ref string
                    rank[item_index] = n+1
                else:
                    #if queue item does exist in ref string
                    rank[item_index] = ref_strings[i:].index(queue_item)
            #index of max rank
            insert_index = rank.index(max(rank))
            Queue[insert_index] = item
        else:
            #found empty space
            insert_index = Queue.index("")
            Queue[insert_index] = item

        print(f"{item} -> {Queue}")
    else:
        #page hit
        hit += 1
        print(f"{item} -> hit")

print(f"Hit ratio : {round(hit/n, 3)}")
print(f"Page Fault ratio : {round(miss/n, 3)}")

```

## 27. Implement various FIFO cache/page replacement policy

```
✓ 16s  fr = int(input("Enter no of frames : "))
ref = input("Enter ref string : ")
hit = 0
miss = 0
n = len(ref)
que = []
ix = 0

for i in range(0,n):

    if len(que) < fr:
        que.append(ref[i])
        print(que)
        miss+=1

    else:
        if (ref[i] in que):
            hit+=1
            print("Hit")
        else:
            que[ix] = ref[i]
            miss+=1

            if ix==fr-1:
                ix=0
            else:
                ix+=1
            print(que)

print(f"Hit ratio : {hit/n}")
print(f"Page Fault ratio : {miss/n}")
```

**28. Implement FCFS CPU scheduling algorithm.**



## ▼ FCFS scpu scheduling

```

class Process:
    def __init__(self, process_id, arrival_time, burst_time):
        self.process_id = process_id
        self.arrival_time = arrival_time
        self.burst_time = burst_time

def fcfs_scheduling(processes):
    n = len(processes)
    completion_time = [0] * n
    turnaround_time = [0] * n
    waiting_time = [0] * n

    # Calculate completion time, turnaround time, and waiting time for each process
    completion_time[0] = processes[0].burst_time
    for i in range(1, n):
        completion_time[i] = completion_time[i - 1] + processes[i].burst_time

    for i in range(n):
        turnaround_time[i] = completion_time[i] - processes[i].arrival_time
        waiting_time[i] = turnaround_time[i] - processes[i].burst_time

    total_turnaround_time = sum(turnaround_time)
    total_waiting_time = sum(waiting_time)
    average_turnaround_time = total_turnaround_time / n
    average_waiting_time = total_waiting_time / n

    # Print the results
    print("Process\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time")
    for i in range(n):
        print(f"{processes[i].process_id}\t\t\t{processes[i].arrival_time}\t\t\t{processes[i].burst_time}\t\t\t"
              f"{completion_time[i]}\t\t\t{turnaround_time[i]}\t\t\t{waiting_time[i]}")
    print("Average Turnaround Time:", average_turnaround_time)
    print("Average Waiting Time:", average_waiting_time)

# Example usage
processes = [
    Process(1, 0, 6),
    Process(2, 2, 4),
    Process(3, 4, 8),
    Process(4, 6, 1),
]

fcfs_scheduling(processes)

```

## 29. Implement SJF CPU scheduling algorithm.

```

class Process:
    def __init__(self, process_id, arrival_time, burst_time):
        self.process_id = process_id
        self.arrival_time = arrival_time
        self.burst_time = burst_time

def sjf_scheduling(processes):
    # Sort the processes based on arrival time (if arrival times are equal, based on burst time)
    processes.sort(key=lambda x: (x.arrival_time, x.burst_time))

    n = len(processes)
    completion_time = [0] * n
    turnaround_time = [0] * n
    waiting_time = [0] * n

    # Calculate completion time, turnaround time, and waiting time for each process
    completion_time[0] = processes[0].burst_time
    for i in range(1, n):
        completion_time[i] = completion_time[i - 1] + processes[i].burst_time

    for i in range(n):
        turnaround_time[i] = completion_time[i] - processes[i].arrival_time
        waiting_time[i] = turnaround_time[i] - processes[i].burst_time

    total_turnaround_time = sum(turnaround_time)
    total_waiting_time = sum(waiting_time)
    average_turnaround_time = total_turnaround_time / n
    average_waiting_time = total_waiting_time / n

    # Print the results
    print("Process\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time")
    for i in range(n):
        print(f"{processes[i].process_id}\t\t\t{processes[i].arrival_time}\t\t\t{processes[i].burst_time}\t\t\t"
              f"{completion_time[i]}\t\t\t{turnaround_time[i]}\t\t\t{waiting_time[i]}")
    print("Average Turnaround Time:", average_turnaround_time)
    print("Average Waiting Time:", average_waiting_time)

# Example usage
processes = [
    Process(1, 0, 6),
    Process(2, 1, 4),
    Process(3, 2, 8),
    Process(4, 3, 1),
]

sjf_scheduling(processes)

```

### 30. Implement Non Preemptive Priority CPU scheduling algorithm.

## ▼ non preemptive priority

```

class Process:
    def __init__(self, process_id, burst_time, priority):
        self.process_id = process_id
        self.burst_time = burst_time
        self.priority = priority

def non_preemptive_priority_scheduling(processes):
    # Sort the processes based on priority (lower number = higher priority)
    processes.sort(key=lambda x: x.priority)

    completion_time = [0] * len(processes)
    turnaround_time = [0] * len(processes)
    waiting_time = [0] * len(processes)

    completion_time[0] = processes[0].burst_time
    for i in range(1, len(processes)):
        completion_time[i] = completion_time[i - 1] + processes[i].burst_time

    for i in range(len(processes)):
        turnaround_time[i] = completion_time[i]
        waiting_time[i] = turnaround_time[i] - processes[i].burst_time

    total_turnaround_time = sum(turnaround_time)
    total_waiting_time = sum(waiting_time)
    average_turnaround_time = total_turnaround_time / len(processes)
    average_waiting_time = total_waiting_time / len(processes)

    # Print the results
    print("Process\tBurst Time\tPriority\tCompletion Time\tTurnaround Time\tWaiting Time")
    for i in range(len(processes)):
        print(f"{processes[i].process_id}\t\t\t{processes[i].burst_time}\t\t\t{processes[i].priority}\t\t\t"
              f"{completion_time[i]}\t\t\t{turnaround_time[i]}\t\t\t{waiting_time[i]}")
    print("Average Turnaround Time:", average_turnaround_time)
    print("Average Waiting Time:", average_waiting_time)

# Example usage
processes = [
    Process(1, 10, 3),
    Process(2, 6, 1),
    Process(3, 8, 2),
    Process(4, 3, 4),
    Process(5, 4, 2)
]

non_preemptive_priority_scheduling(processes)

```

### 31. Implement Best Fit Memory allocation policy.

```

def bestFit(blockSize,processSize):
    n = len(blockSize)
    m = len(processSize)
    allocation = [-1]*m
    for i in range(m):
        bestIndex = -1
        minDiff = float('inf')
        for j in range(n):
            if blockSize[j] >= processSize[i]:
                diff = blockSize[j]-processSize[i]
                if diff < minDiff:
                    bestIndex = j
                    minDiff = diff
        if bestIndex != -1:
            allocation[i] = bestIndex
            blockSize[bestIndex] -= processSize[i]
    print("Process No. \t Process Size \t Block No.")
    for i in range(m):
        print(i+1,"\t\t",processSize[i],end='\t\t\t')
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("No block allocated")

blockSize = list(map(int, input("Enter the block size list: ").split()))
processSize = list(map(int, input("Enter the process size list: ").split()))
bestFit(blockSize,processSize)

```

Enter the block size list: 10 20  
 Enter the process size list: 10 20  

Process No.	Process Size	Block No.
1	10	1
2	20	2

### 32. Implement First Fit Memory allocation policy.

```

✓ [13] def firstFit(blockSize,processSize):
0s
    n = len(blockSize)
    m = len(processSize)
    allocation = [-1]*m
    for i in range(m):
        for j in range(n):
            if blockSize[j] >= processSize[i]:
                allocation[i] = j
                blockSize[j] -= processSize[i]
                break

    print("Process No. \t Process Size \t Block No.")
    for i in range(m):
        print(i+1,"\t\t",processSize[i],end='\t\t\t')
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("No block allocated")

```

```

✓ [14s] ▶ blockSize = list(map(int, input("Enter the block size list: ").split()))
    processSize = list(map(int, input("Enter the process size list: ").split()))
    firstFit(blockSize,processSize)

```

```

↳ Enter the block size list: 1
Enter the process size list: 2
Process No.      Process Size      Block No.
1                2                      No block allocated

```

### 33. Implement Worst Fit Memory allocation policy.

```

def worstFit(blockSize,processSize):
    n = len(blockSize)
    m = len(processSize)
    allocation = [-1]*m
    for i in range(m):
        worstIndex = -1
        for j in range(n):
            if blockSize[j] >= processSize[i]:
                if worstIndex == -1:
                    worstIndex = j
                elif blockSize[worstIndex] < blockSize[j]:
                    worstIndex = j
        if worstIndex != -1:
            allocation[i] = worstIndex
            blockSize[worstIndex] -= processSize[i]
    print("Process No. \t Process Size \t Block No.")
    for i in range(m):
        print(i+1,"\t\t",processSize[i],end='\t\t\t')
        if allocation[i] != -1:
            print(allocation[i] + 1)
        else:
            print("No block allocated")

blockSize = list(map(int, input("Enter the block size list: ").split()))
processSize = list(map(int, input("Enter the process size list: ").split()))
worstFit(blockSize,processSize)

```

Enter the block size list: 10 20  
 Enter the process size list: 10 10  
 Process No.      Process Size      Block No.  
 1                  10                  2  
 2                  10                  1

**34. Implement order scheduling in supply chain using Banker's Algorithm**✓  
0s

```
# Banker's Algorithm

# Number of processes
P = 5

# Number of resources
R = 3

# Function to find the need of each process
def calculateNeed(need, maxm, allot):

    # Calculating Need of each P
    for i in range(P):
        for j in range(R):

            # Need of instance = maxm instance -
            # allocated instance
            need[i][j] = maxm[i][j] - allot[i][j]
```

✓  
0s

```
# Function to find the system is in
# safe state or not
def isSafe(processes, avail, maxm, allot):
    need = []
    for i in range(P):
        l = []
        for j in range(R):
            l.append(0)
        need.append(l)

    # Function to calculate need matrix
    calculateNeed(need, maxm, allot)

    # Mark all processes as in finish
    finish = [0] * P

    # To store safe sequence
    safeSeq = [0] * P

    # Make a copy of available resources
    work = [0] * R
    for i in range(R):
        work[i] = avail[i]
```

  
0s

```
# While all processes are not finished
# or system is not in safe state.
count = 0
while (count < P):

    # Find a process which is not finish
    # and whose needs can be satisfied
    # with current work[] resources.
    found = False
    for p in range(P):

        # First check if a process is finished,
        # if no, go for next condition
        if (finish[p] == 0):

            # Check if for all resources
            # of current P need is less
            # than work
            for j in range(R):
                if (need[p][j] > work[j]):
                    break
```





```
# If all needs of p were satisfied.
if (j == R - 1):

    # Add the allocated resources of
    # current P to the available/work
    # resources i.e.free the resources
    for k in range(R):
        work[k] += allot[p][k]

    # Add this process to safe sequence.
    safeSeq[count] = p
    count += 1

    # Mark this p as finished
    finish[p] = 1

    found = True

# If we could not find a next process
# in safe sequence.
if (found == False):
    print("System is not in safe state")
    return False
```



0s



```
# If system is in safe state then
# safe sequence will be as below
print("System is in safe state.",
      "\nSafe sequence is: ", end = " ")
print(*safeSeq)

return True

# Driver code
if __name__ == "__main__":

    processes = [0, 1, 2, 3, 4]

    # Available instances of resources
    avail = [3, 3, 2]

    # Maximum R that can be allocated
    # to processes
    maxm = [[7, 5, 3], [3, 2, 2],
            [9, 0, 2], [2, 2, 2],
            [4, 3, 3]]

    # Resources allocated to processes
    allot = [[0, 1, 0], [2, 0, 0],
            [3, 0, 2], [2, 1, 1],
            [0, 0, 2]]

    # Check system is in safe state or not
    isSafe(processes, avail, maxm, allot)
```

### 35. Implement FIFO Disk Scheduling Algorithms.

## FCFS Disk Scheduling

```

✓ [1] def FCFS(requests, head):
0s     sequence = [head]+requests
     seekTime = 0

     for i in range(len(sequence)-1):
         distance = abs(sequence[i+1]-sequence[i])
         seekTime += distance


     # seekTime=sum(abs(sequence[i+1]-sequence[i])for i in range(len(sequence)-1))
     return seekTime, sequence

```

```

✓ [2] n = 200
0s     head = 55
     requests = [93, 176, 42, 148, 27, 14, 180]
     seekTime, seekSequence = FCFS(requests, head)
     print("Total seek time:", seekTime)
     print("Seek sequence:", seekSequence)

```

 Total seek time: 661  
 Seek sequence: [55, 93, 176, 42, 148, 27, 14, 180]

## 36. Implement SSTF Disk Scheduling Algorithms.

## SSTF Disk Scheduling

```

✓ [3] def SSTF(requests, head):
0s     total_seek_time = 0
     sequence = []

     while requests:
         closest_request = min(requests, key=lambda x: abs(x-head))
         sequence.append(head)
         total_seek_time += abs(closest_request - head)
         head = closest_request
         requests.remove(closest_request)

     return total_seek_time, sequence

```

```

✓ [4] n = 200
0s     head = 50
     requests = [82, 170, 43, 140, 24, 16, 190]
     seekTime, seekSequence = SSTF(requests, head)
     print("Total seek time:", seekTime)
     print("Seek sequence:", seekSequence)

```

Total seek time: 208  
 Seek sequence: [50, 43, 24, 16, 82, 140, 170]

### 37. Implement SCAN Disk Scheduling Algorithms.

#### ▼ SCAN Disk Scheduling

```

0s [6] def SCAN(head,n,request):
    sequence = sorted(requests)
    if head in sequence:
        sequence.remove(head)
    sequence = [head]+[i for i in sequence if i>head]+[n]+[i for i in reversed(sequence) if i<head]
    seekTime=sum(abs(sequence[i+1]-sequence[i])for i in range(len(sequence)-1))
    return seekTime,sequence

0s [6] requests = [95, 180, 34, 119, 11, 123, 62, 64]
    head = 50
    n = 199

    seekTime, seekSequence = SCAN(head,n,requests)

    print("Total seek time:", seekTime)
    print("Seek sequence:", seekSequence)

Total seek time: 337
Seek sequence: [50, 62, 64, 95, 119, 123, 180, 199, 34, 11]

```

### 38. Implement C-SCAN Disk Scheduling Algorithms.

#### C-SCAN Disk Scheduling

```

0s [7] def C_SCAN(head,n,request):
    sequence = sorted(requests)
    if head in sequence:
        sequence.remove(head)
    sequence = [head]+[i for i in sequence if i>head]+[n]+[0]+[i for i in sequence if i<head]
    seekTime=sum(abs(sequence[i+1]-sequence[i])for i in range(len(sequence)-1))
    return seekTime,sequence

0s [7] requests = [82, 170, 43, 140, 24, 16, 190]
    head = 50
    n = 199

    seekTime, seekSequence = C_SCAN(head,n,requests)

    print("Total seek time:", seekTime)
    print("Seek sequence:", seekSequence)

Total seek time: 391
Seek sequence: [50, 82, 140, 170, 190, 199, 0, 16, 24, 43]

```

### 39. Implement Look Disk Scheduling Algorithms.

## LOOK Disk Scheduling

```

✓ [9] def LOOK(head,n,request):
0s     sequence = sorted(requests)
        if head in sequence:
            sequence.remove(head)
        sequence=[head]+[i for i in sequence if i > head] + [i for i in reversed(sequence) if i < head]
        seekTime=sum(abs(sequence[i+1]-sequence[i])for i in range(len(sequence)-1))
        return seekTime,sequence

```

```

✓ requests = [82, 170, 43, 140, 24, 16, 190]
0s head = 50
    n = 200

    seekTime, seekSequence = LOOK(head,n,requests)

    print("Total seek time:", seekTime)
    print("Seek sequence:", seekSequence)

```

```

📄 Total seek time: 314
    Seek sequence: [50, 82, 140, 170, 190, 43, 24, 16]

```

**40. Implement C-Look Disk Scheduling Algorithms.**

## C-LOOK Disk Scheduling

```

🔍 def C_LOOK(head,n,request):
    sequence = sorted(requests)
    if head in sequence:
        sequence.remove(head)
    sequence=[head]+[i for i in sequence if i > head] + [i for i in sequence if i < head]
    seekTime=sum(abs(sequence[i+1]-sequence[i])for i in range(len(sequence)-1))
    return seekTime,sequence

```

```

[ ] requests = [93, 176, 42, 148, 27, 14, 183]
    head = 55
    n = 200

    seekTime, seekSequence = C_LOOK(head,n,requests)

    print("Total seek time:", seekTime)
    print("Seek sequence:", seekSequence)

```

```

    Total seek time: 325
    Seek sequence: [55, 93, 148, 176, 183, 14, 27, 42]

```

**Shell Scripting****41. Write Shell script to copy files from one folder to another**

```
bash Copy code

#!/bin/bash

# Source and destination folders
source_folder="/path/to/source/folder"
destination_folder="/path/to/destination/folder"

# Copy files from source to destination
cp -r "$source_folder"/* "$destination_folder"

# Verify if the copy was successful
if [ $? -eq 0 ]; then
    echo "Files copied successfully."
else
    echo "Error occurred while copying files."
fi
```

```
bash Copy code

chmod +x copy_files.sh
```


To execute the script, simply run:

```
bash Copy code

./copy_files.sh
```

42. Write Shell script Count number of words, characters and lines.

bash

 Copy code

```
#!/bin/bash

# File to be analyzed
file_path="/path/to/file"


# Count the number of words
word_count=$(wc -w < "$file_path")

# Count the number of characters
character_count=$(wc -m < "$file_path")

# Count the number of lines
line_count=$(wc -l < "$file_path")

# Display the counts
echo "Word count: $word_count"
echo "Character count: $character_count"
echo "Line count: $line_count"
```


bash

 Copy code

```
chmod +x count_stats.sh
```

To execute the script, simply run:


bash

 Copy code

```
./count_stats.sh
```

43. Write Shell script To describe files in different format.

bash

 Copy code

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

```
# Function to describe a file in long format
```

```
describe_file_long() {
```

```
    file_path="$1"
```

```
    file_info=$(ls -l "$file_path")
```

```
    echo "File Information (Long Format):"
```

```
    echo "$file_info"
```

```
}
```

```
# Function to describe a file in short format
```

```
describe_file_short() {
```

```
    file_path="$1"
```

```
    file_name=$(basename "$file_path")
```

```
    file_size=$(du -h "$file_path" | awk '{print $1}')
```

```
    file_type=$(file -b --mime-type "$file_path")
```

```
    echo "File Information (Short Format):"
```

```
    echo "Name: $file_name"
```

```
    echo "Size: $file_size"
```

```
    echo "Type: $file_type"
```

```
# File to be described
```

```
file_path="/path/to/file"
```

```
# Describe the file in long format
```

```
describe_file_long "$file_path"
```

```
echo
```

```
# Describe the file in short format
```

```
describe_file_short "$file_path"
```



```
bash Copy code  
  
chmod +x describe_file.sh  
  
To execute the script, simply run:  
  
bash Copy code  
  
./describe_file.sh
```

44. Write Shell script to find factorial of given number using bash script

```
bash Copy code  
  
#!/bin/bash  
  
# Function to calculate factorial  
factorial() {  
    num=$1  
    fact=1  
  
    # Iterate from 1 to num  
    for ((i=1; i<=num; i++))  
    do  
        fact=$((fact * i))  
    done  
  
    echo "Factorial of $num is: $fact"  
}  
  
# Read input from user  
echo -n "Enter a number: "  
read number  
  
# Call the factorial function  
factorial "$number"
```

45. Display first 10 natural numbers using bash script

```
bash Copy code

#!/bin/bash

echo "First 10 Natural Numbers:"

# Loop from 1 to 10
for ((i=1; i<=10; i++))
do
    echo "$i"
done
```

46. Display Fibonacci series using bash script

```
bash Copy code

#!/bin/bash

echo "Fibonacci Series:"

# Function to calculate Fibonacci series
fibonacci() {
    limit=$1

    # First two numbers of the series
    a=0
    b=1

    echo -n "$a "
}
```

```
# Loop to generate Fibonacci series
for ((i=1; i<limit; i++))
do
    echo -n "$b "
    next=$((a + b))
    a=$b
    b=$next
done

echo

# Read input from user
echo -n "Enter the limit: "
read limit

# Call the Fibonacci function
fibonacci "$limit"
```

47. Find given number is prime or nor using bash script

```
# Function to check if a number is prime
is_prime() {
    num=$1

    # Check if num is less than 2
    if [ $num -lt 2 ]; then
        echo "$num is not a prime number."
        return
    fi

    # Check for factors from 2 to sqrt(num)
    for ((i=2; i*i<=num; i++))
    do
        # If num is divisible by i, it is not prime
        if [ $((num % i)) -eq 0 ]; then
            echo "$num is not a prime number."
            return
        fi
    done

    echo "$num is a prime number."
}

# Read input from user
echo -n "Enter a number: "
read number

# Call the is_prime function
is_prime "$number"
```

☐ Stop generating

48. Write shell script to find biggest of three numbers

```
bash Copy code

#!/bin/bash

# Read three numbers from the user
echo "Enter three numbers:"

echo -n "Number 1: "
read num1

echo -n "Number 2: "
read num2


echo -n "Number 3: "
read num3

# Check which number is the biggest
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]; then
    biggest=$num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]; then
    biggest=$num2
else
    biggest=$num3
fi

# Display the biggest number
echo "The biggest number is: $biggest"
```

49. Write shell script to reverse a given number

bash

 Copy code

```
#!/bin/bash


# Read the number from the user
echo -n "Enter a number: "
read number

# Variable to store the reversed number
reversed=0

# Reverse the number
while [ $number -gt 0 ]
do
    remainder=$((number % 10))
    reversed=$((reversed * 10 + remainder))
    number=$((number / 10))
done

# Display the reversed number
echo "Reversed number: $reversed"
```

50. Write shell script to find Sum of individual digits (1234 => 1+2+3+4=10)

```
bash  Copy code

#!/bin/bash

# Function to calculate the sum of individual digits
sum_digits() {
    local number=$1
    local sum=0


    # Loop through each digit in the number
    while [ $number -gt 0 ]; do
        digit=$((number % 10)) # Get the last digit
        sum=$((sum + digit))   # Add the digit to the sum
        number=$((number / 10)) # Remove the last digit
    done

    echo $sum
}

# Prompt the user to enter a number
read -p "Enter a number: " input_number

# Call the function and display the result
result=$(sum_digits $input_number)
echo "Sum of individual digits: $result"
```

51. Write a shell script to display a list of users currently logged in.

```
bash  Copy code


#!/bin/bash

# Get the list of currently logged in users
users=$(who | awk '{print $1}')

# Display the list of users
echo "Currently logged in users:"
echo "$users"
```

52. Write a shell script to perform arithmetic operations.

bash

 Copy code

```
#!/bin/bash

# Function to perform arithmetic operations
perform_arithmetic() {
    local num1=$1
    local operator=$2
    local num2=$3
    local result=0

    case $operator in
        "+")
            result=$((num1 + num2))
            ;;
        "-")
            result=$((num1 - num2))
            ;;
        "*")
            result=$((num1 * num2))
            ;;
        "/")
            result=$((num1 / num2))
            ;;
        "%")
            result=$((num1 % num2))
            ;;
        *)

```



```

        ;;
    *)
        echo "Invalid operator: $operator"
        exit 1
    ;;
esac

echo $result
}

# Prompt the user to enter the numbers and operator
read -p "Enter the first number: " number1
read -p "Enter the operator (+, -, *, /, %): " operator
read -p "Enter the second number: " number2

# Call the function and display the result
result=$(perform_arithmetic $number1 $operator $number2)
echo "Result: $result"

```

53. Write a shell script to copy contents of one file to another.

```

bash Copy code

#!/bin/bash

# Function to copy contents of one file to another
copy_file_contents() {
    local source_file=$1
    local destination_file=$2

    # Copy the contents of the source file to the destination file
    cp $source_file $destination_file

    echo "Contents copied successfully from $source_file to $destination_file"
}

# Prompt the user to enter the source and destination file names
read -p "Enter the source file name: " source_file
read -p "Enter the destination file name: " destination_file

# Call the function to copy the file contents
copy_file_contents $source_file $destination_file

```

54. Write a shell program to generate multiplication table of a number upto a given range.

```
bash Copy code

#!/bin/bash

# Function to generate the multiplication table
generate_multiplication_table() {
    local number=$1
    local range=$2

    echo "Multiplication table of $number up to $range:"

    for ((i=1; i<=range; i++)); do
        result=$((number * i))
        echo "$number x $i = $result"
    done
}

# Prompt the user to enter the number and range
read -p "Enter the number: " input_number
read -p "Enter the range: " input_range

# Call the function to generate the multiplication table
generate_multiplication_table $input_number $input_range
```

55. Write a shell program to count the number of files in a directory.

```
bash Copy code

#!/bin/bash

# Function to count the number of files in a directory
count_files_in_directory() {
    local directory=$1

    # Check if the directory exists
    if [ -d "$directory" ]; then
        file_count=$(ls -l "$directory" | grep "^-" | wc -l)
        echo "Number of files in $directory: $file_count"
    else
        echo "Directory $directory does not exist."
    fi
}

# Prompt the user to enter the directory path
read -p "Enter the directory path: " input_directory

# Call the function to count the files in the directory
count_files_in_directory "$input_directory"
```

56. WAS to find the number of matched characters, words and lines in a file.

```
#!/bin/bash

# Function to find the number of matched characters, words, and lines in a file
find_matched_counts() {
    local file=$1
    local search_pattern=$2

    # Find the matched lines and store them in a temporary file
    matched_lines=$(grep -n "$search_pattern" "$file")

    # Count the number of matched characters
    matched_characters=$(echo -n "$matched_lines" | wc -m)

    # Count the number of matched words
    matched_words=$(echo -n "$matched_lines" | wc -w)

    # Count the number of matched lines
    matched_lines_count=$(echo -n "$matched_lines" | wc -l)

    # Count the number of matched words
    matched_words=$(echo -n "$matched_lines" | wc -w)

    # Count the number of matched lines
    matched_lines_count=$(echo -n "$matched_lines" | wc -l)

    # Display the results
    echo "Number of matched characters: $matched_characters"
    echo "Number of matched words: $matched_words"
    echo "Number of matched lines: $matched_lines_count"
}

# Prompt the user to enter the file path and search pattern
read -p "Enter the file path: " input_file
read -p "Enter the search pattern: " input_pattern

# Call the function to find the matched counts
find_matched_counts "$input_file" "$input_pattern"
```

57. Write a script to find the number of characters, words and lines in a file.

```
#!/bin/bash

# Function to find the number of characters, words, and lines in a file
find_counts() {
    local file=$1

    # Count the number of characters
    characters=$(wc -m < "$file")

    # Count the number of words
    words=$(wc -w < "$file")

    # Count the number of lines
    lines=$(wc -l < "$file")

    # Display the results
    echo "Number of characters: $characters"
    echo "Number of words: $words"
    echo "Number of lines: $lines"
}

# Prompt the user to enter the file path
read -p "Enter the file path: " input_file

# Call the function to find the counts
find_counts "$input_file"
```

58. Write a script to display list of files starting with particular letter in the directory.

```
bash Copy code

#!/bin/bash

# Function to display a list of files starting with a particular letter
display_files_starting_with_letter() {
    local directory=$1
    local letter=$2

    # Check if the directory exists
    if [ -d "$directory" ]; then
        # Find and display the list of files starting with the specified letter
        files=$(ls "$directory"/"$letter"*)
        echo "List of files starting with letter '$letter':"
        echo "$files"
    else
        echo "Directory $directory does not exist."
    fi
}

# Prompt the user to enter the directory path and the starting letter
read -p "Enter the directory path: " input_directory
read -p "Enter the starting letter: " input_letter

# Call the function to display the list of files
display_files_starting_with_letter "$input_directory" "$input_letter"
```

59. Write a script to develop a Fibonacci series.

```
# Function to generate Fibonacci series
generate_fibonacci_series() {
    local n=$1

    # First two numbers in the series
    num1=0
    num2=1

    echo "Fibonacci series up to $n terms:"
    echo "$num1"

    # Generate the Fibonacci series
    for ((i=1; i<n; i++)); do
        echo "$num2"

        # Calculate the next number in the series
        next=$((num1 + num2))
        num1=$num2
        num2=$next
    done
}

# Prompt the user to enter the number of terms in the series
read -p "Enter the number of terms in the Fibonacci series: " input_terms

# Call the function to generate the Fibonacci series
generate_fibonacci_series $input_terms
```

60. Write a shell script to replace the Nth occurrence of a pattern.

```
#!/bin/bash

# Function to replace the Nth occurrence of a pattern in a file
replace_nth_occurrence() {
    local file=$1
    local pattern=$2
    local replace_text=$3
    local n=$4

    # Check if the file exists
    if [ -f "$file" ]; then
        # Replace the Nth occurrence of the pattern
        sed -i "0,/$pattern/{s//$replace_text/}" "$file"
        echo "Pattern '$pattern' replaced with '$replace_text' at the $n occurrence"
    else
        echo "File $file does not exist."
    fi
}

# Prompt the user to enter the file path, pattern, replacement text, and the Nth occurrence
read -p "Enter the file path: " input_file
read -p "Enter the pattern to replace: " input_pattern
read -p "Enter the replacement text: " input_replace
read -p "Enter the Nth occurrence to replace: " input_occurrence

# Call the function to replace the Nth occurrence of the pattern
replace_nth_occurrence "$input_file" "$input_pattern" "$input_replace" "$input_occurrence"
```



61. Write a shell script to convert temperature from Centigrade to Fahrenheit.

```
bash Copy

#!/bin/bash

# Function to convert temperature from Centigrade to Fahrenheit
convert_celsius_to_fahrenheit() {
    local celsius=$1

    # Convert Centigrade to Fahrenheit
    fahrenheit=$(echo "scale=2; ($celsius * 9/5) + 32" | bc)

    echo "$celsius°C is equivalent to $fahrenheit°F"
}

# Prompt the user to enter the temperature in Centigrade
read -p "Enter the temperature in Centigrade: " input_celsius

# Call the function to convert Centigrade to Fahrenheit
convert_celsius_to_fahrenheit $input_celsius
```

62. Write a shell script to compute the power of a given number.

```
bash Copy code

#!/bin/bash

# Function to compute the power of a given number
compute_power() {
    local number=$1
    local exponent=$2

    # Initialize the result
    result=1

    # Compute the power using a loop
    for ((i=1; i<=exponent; i++)); do
        result=$((result * number))
    done

    echo "$number raised to the power of $exponent is $result"
}

# Prompt the user to enter the number and the exponent
read -p "Enter the number: " input_number
read -p "Enter the exponent: " input_exponent

# Call the function to compute the power
compute_power $input_number $input_exponent
```

63. Write a shell script to check whether the entered number is prime or not.

```
bash Copy code

#!/bin/bash

# Function to check if a number is prime
is_prime() {
    local number=$1

    if [ $number -lt 2 ]; then
        echo "$number is not a prime number."
        return
    fi

    is_prime=true
```


```
# Check divisibility from 2 to the square root of the number
for ((i=2; i*i<=number; i++)); do
    if [ $(number % i) -eq 0 ]; then
        is_prime=false
        break
    fi
done

if $is_prime; then
    echo "$number is a prime number."
else
    echo "$number is not a prime number."
fi
}

# Prompt the user to enter the number
read -p "Enter a number: " input_number

# Call the function to check if the number is prime
is_prime $input_number
```

64. Write a shell script to check whether the year is leap year or not.

```
bash  Copy code

#!/bin/bash

# Function to check if a year is a leap year
is_leap_year() {
    local year=$1

    if ((year % 4 == 0 && year % 100 != 0)) || ((year % 400 == 0)); then
        echo "$year is a leap year."
    else
        echo "$year is not a leap year."
    fi
}

# Prompt the user to enter the year
read -p "Enter a year: " input_year

# Call the function to check if the year is a leap year
is_leap_year $input_year
```

65. Write a shell script to check whether a number is even or odd.

```
bash Copy code

#!/bin/bash

# Function to check if a number is even or odd
check_even_odd() {
    local number=$1

    if ((number % 2 == 0)); then
        echo "$number is an even number."
    else
        echo "$number is an odd number."
    fi
}

# Prompt the user to enter the number
read -p "Enter a number: " input_number

# Call the function to check if the number is even or odd
check_even_odd $input_number
```