# Program 1

## Introduction To Linux

Linux is a free and open-source family of operating systems that is resilient and flexible. In 1991, an individual by the name as Linus Torvalds constructed it. The system's source code is accessible to everyone for anyone to look at and change, making it cool that anyone can see how the system works. People from across the world are urged to work together and keep developing Linux due to its openness. Since the beginning, Linux has grown into a dependable and safe OS that is used in an array of gadgets, including PCs, cell phones, and huge supercomputers. It is well-known for being cost-effective, which implies that employing it doesn't cost a lot, and efficient, which indicates it can complete a lot of jobs quickly.

## Linux Operating System

Developed by Linus Torvalds in 1991, the Linux operating system is a powerful and flexible open-source software platform. It acts as the basis for a variety of devices, such as embedded systems, cell phones, servers, and personal computers. Linux, which is well-known for its reliability, safety, and flexibility, allows users to customize and improve their environment to suit specific needs. With an extensive and active community supporting it, Linux is an appealing choice for people as well as companies due to its wealth of resources and constant developments.

## History

Linus Torvalds designed the free and open-source Linux operating system kernel in 1991. Torvalds set out to develop a free and flexible system for personal computers, drawing ideas from the UNIX operating system and the MINIX operating system. Teamwork in development was encouraged with the initial release of the Linux kernel, which attracted developers and enthusiasts globally quickly. Various open-source software packages integrated with the Linux kernel created fully operational operating systems, occasionally referred to as Linux distributions. Over the years, Linux has become known as a key component of modern computing, powering everything from servers and personal computers to supercomputers and smartphones. Due to its flexibility, durability, and strong community support, developers, businesses, and educational institutions frequently opt for it.

## Types Of Linux Operating System

One of the most popular operating systems being utilised on computers and other devices is Linux. Although numerous Linux variants are also used on desktop, laptop, and mainframe machines in addition to other obscure devices, they are arguably best recognised for their use on commercial computer servers. Both the Chrome OS operating system for laptop computers known as Chromebooks and the Android mobile and tablet operating system from Google are based on Linux. Different Linux system types are best adapted for certain uses.

- Ubuntu
- Debian
- FedoraCentOS
- Red Hat Enterprise Linux (RHEL)
- Arch Linux

- Manjaro
- openSUSE
- Linux Mint
- Kali Linux
- Elementary OS
- Zorin OS
- Puppy Linux
- Solus

## Installation

Linux is an operating system, similar to Windows, but with many different versions due to the nature of being open source and fully customizable. To install Linux, you must choose an install method and choose a Linux distribution.

**To install Linux:**

1. Choose an install method: Windows Subsystem for Linux (WSL), Bare metal Linux; or create a Virtual
Machine (VM) to run Linux locally or in the cloud.
2. Choose a Linux distribution: Ubuntu, Debian, Kali Linux, openSUSE, etc.
3. Follow the steps for your preferred install method:
   Use the install Linux command with Windows Subsystem for Linux (WSL)
   Create a Linux Virtual Machine (VM) in the cloud
   Create a Linux Virtual Machine (VM) on your local machine
   Create a bootable USB to install bare-metal Linux
4. After installing Linux: Get familiar with your distribution's package manager, update and upgrade the packages available, and get familiar with the other Linux resources at Microsoft, such as training courses, Linux versions of popular tools, news, and Open Source events.

## Types Of Linux Commands

**1. File and Directory Commands**

These commands are used to manage files and directories.

**1.1 ls: Lists directory contents**

Input: ls

Output: file1.txt file2.txt directory1 directory2

**1.2 cd: Changes the current directory.**

**Input: cd /home/user1.3 pwd: Prints the current working directory.**

**Input**: pwd

**Ouput**: /home/user

**1.4 cp: Copies files or directories.**

**Input**: cp file1.txt /home/user/backup/

**1.5 rm: Removes files or directories.**

**Input**: rm file1.txt

## 2. File Viewing and Manipulation Commands

Commands for viewing and editing file contents.

**2.1 cat: Concatenates and displays file content.**

**Input**: cat file1.txt

**Output**: This is the content of file1.

**2.2 less: Views file content one page at a time.**
**Input**: less file1.txt
**2.3 head: Displays the first part of a file.**
**Input**: head -n 5 file1.txt
**Output**: First line
Second line
Third line
Fourth line
Fifth line

**2.4 tail: Displays the last part of a file.**
**Input**: tail -n 5 file1.txt
**Output**: Fifth last line
Fourth last line
Third last line
Second last lineLast line
**2.5 grep: Searches for a pattern in files.**
**Input**: grep "pattern" file1.txt
**Output**: This line contains the pattern.

## 3. System Information Commands

Commands to get system information and manage processes.

**3.1 top: Displays real-time system processes.**
**Input**: top

**3.2 ps: Displays information about active processes.**
**Input**: ps aux

**Output**: USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
user 1234 0.0 0.1 24568 2044 ? Ss 12:00 0:00 /bin/bash

**3.3 df: Reports disk space usage.**
**Input**: df –h
**Output**: Filesystem Size Used Avail Use% Mounted on
/dev/sda1 50G 20G 28G 43% /

**3.4 du: Estimates file and directory space usage.**
**Input**: du -sh /home/user
**Output**: 1.2G /home/user

**3.5 free: Displays memory usage.**
**Input**: free –h
**Output**: total used free shared buff/cache available
Mem: 16G 2.5G 12G 100M 1.5G 13G
Swap: 2.0G 0B 2.0G

## 4. File Permissions Commands

Commands to manage file and directory permissions.4.1 chmod: Changes file permissions.
**Input**: chmod 755 script.sh

**4.2 chown: Changes file owner and group.**
**Input**: chown user:group file1.txt

**4.3 chgrp: Changes the group ownership of a file.**
**Input**: chgrp group file1.txt

**4.4 umask: Sets default file creation permissions.**

Input: umask 022

**4.5 ls -l: Lists files with detailed permissions.**

**Input**: ls –l

**Output**: -rwxr-xr-x 1 user group 1234 Aug 6 12:00 file1.txt

## 5. Networking Commands

Commands to manage network connections and configurations.

**5.1 ifconfig: Displays or configures network interfaces.**

**Input**: ifconfig

**5.2 ping: Sends ICMP ECHO_REQUEST to network hosts.**

**Input**: ping google.com

**Output**: PING google.com (172.217.14.206) 56(84) bytes of data.

64 bytes from 172.217.14.206: icmp_seq=1 ttl=117 time=12.3 ms

**5.3 netstat: Displays network connections and statistics.**

**Input**: netstat –tuln

**Output**: Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN

**5.4 ss: Utility to investigate sockets.**

**Input**: ss –tuln

**Output**: Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port

**TCP LISTEN 0 128 0.0.0.0:22 0.0.0.0:*5.5 traceroute: Traces the route packets take to a network host.**

**Input**: traceroute google.com

**Output**: 1 router.local (192.168.1.1) 1.200 ms 1.124 ms 1.097 ms

2 10.10.10.1 (10.10.10.1) 2.389 ms 2.163 ms 2.103 ms

3 172.217.14.206 (172.217.14.206) 12.345 ms 12.567 ms 12.678 ms

# Program 2

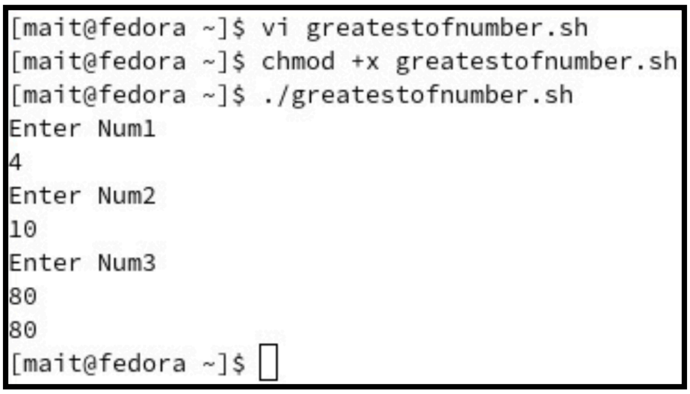
**Aim:** Write a program to find the greatest of three numbers (numbers passed as command lines.)

## Code:

```
echo "Enter num1"
read num1
echo "Enter num2"
read num2
echo "Enter num2"
read num2

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]; then
   echo $num1
elif [ $num2 -gt $num3 ] && [ $num2 -gt $num1 ]; then
   echo $num2
else
   echo $num3
fi
```

## Output:



```
[mait@fedora ~]$ vi greatestofnumber.sh
[mait@fedora ~]$ chmod +x greatestofnumber.sh
[mait@fedora ~]$ ./greatestofnumber.sh
Enter Num1
4
Enter Num2
10
Enter Num3
80
80
[mait@fedora ~]$
```

# Program 3

**Aim:** Write a script to check whether the given no. is even / odd

### Code:

```
echo "Enter the Number"
read num

if [$((num % 2)) -eq 0 ]
then
echo "The number is Even”

else
echo "The number is Odd"
fi
```

### Output:

```
[mait@fedora ~]$ chmod 777 evenOddProgram.sh
[mait@fedora ~]$ ./evenOddProgram.sh
Enter the Number
45
The number is Odd
```

# Program 4

**Aim:** Write a script to check whether the given number is prime or not

## Code:

```
echo -e "Enter Number :  "
read n
for ((i=2; i<=$n/2; i++))
do
ans=$(( n%i ))
if [ $ans -eq 0 1
then
echo "$n is not a prime number."
exit o
fi done
echo "$n is a prime number."
```

## Output:

```
[mait@fedora ~]$ vi prime.sh
[mait@fedora ~]$ ./prime.sh
Enter Number : 4
4 is not a prime number.
[mait@fedora ~]$ ./prime.sh
Enter Number : 5
5 is a prime number.
[mait@fedora ~]$ 
```

# Program 5

**Aim:** Write a program to check whether the given input is a number or a string.

## Code:

```
read -p "Enter input: " input

if [[ $input =~ ^[0-9]+$ ]]; then
   echo "It's a number"
else
   echo "It's a string"
fi
```

## Output:

```
[mait@fedora ~]$ vi numberorstring.sh
[mait@fedora ~]$ chmod +x numberorstring.sh
[mait@fedora ~]$ ./numberorstring.sh
Enter something: helloworld
The input is a string.
[mait@fedora ~]$ ./numberorstring.sh
Enter something: 4
The input is a number.
[mait@fedora ~]$
```

# Program 6

**Aim:** Write a script to compute no. of characters and words in each line of given file

**Code:**

```
file_path=" /home/mait/testfile.txt"

number_of_lines= wc --lines < §file_path
number_of_words= wc --word ‹ $file_path

echo "Number of lines: $number_of_lines" echo "Number of words: $number_of_words"
```

```
Linux has been around since the mid-1990s and has since reached a user-base that
 spans the globe.
Linux is actually everywhere: It’s in your phones, your thermostats, in your car
s, refrigerators, Roku devices, and televisions.
It also runs most of the Internet, all of the world’s top 500 supercomputers, an
d the world’s stock exchanges.
~
~
```

**Output:**

```
[mait@fedora ~]$ vi testfile.txt
[mait@fedora ~]$ vi countNoOfWordsAndLines.sh
[mait@fedora ~]$ ./countNoOfWordsAndLines.sh
Number of lines: 3
Number of words: 54
[mait@fedora ~]$
```

# Program 7

**Aim: Write a script to calculate average of n numbers.**

**Code:**

```
sum=0
count=0
echo "Enter numbers separated by spaces (e.g., 1 2 3 4):"
read -a numbers
for num in "${numbers[@]}"; do
   sum=$(echo "$sum + $num" | bc)
   count=$((count + 1))
done

if [ $count -ne 0 ]; then
   average=$(echo "scale=2; $sum / $count" | bc)
   echo "Average: $average"
else
   echo "No numbers provided."
fi
```

**Output:**

```
[mait@fedora ~]$ chmod +x prog1.sh
[mait@fedora ~]$ ./prog1.sh
hello world
Enter Numbers:
1 2 3 4
Average : 2.50
[mait@fedora ~]$ 
```

# Program 8

**Aim: Write a script to print Fibonacci series up to n terms**

**Code:**

```
echo "Enter the number of terms in the Fibonacci series:"
read n

a=0
b=1

for (( i=0; i<n; i++ )); do
   echo -n "$a "
   # Update the terms
   fn=$((a + b))
   a=$b
   b=$fn
done

echo
```

**Output:**

```
[mait@fedora ~]$ vi prog2.sh
[mait@fedora ~]$ chmod +x prog2.sh
[mait@fedora ~]$ ./prog2.sh
Enter the number of terms in fibonnaci sequence
5
0 1 1 2 3 [mait@fedora ~]$ vi prog3.sh
```

# Program 9

## Aim: Write a script to print factorial of a given number

## Code:

```
echo "Enter a number to calculate its factorial:"
read n

factorial=1

for (( i=1; i<=n; i++ )); do
   factorial=$((factorial * i))
done

echo "Factorial of $n is $factorial"
```

## Output:

```
[mait@fedora ~]$ chmod +x prog3.sh
[mait@fedora ~]$ ./prog3.sh
Enter the number: 5
factorial of 5 is 120
[mait@fedora ~]$ 
```

# Program 10

**Aim: Write a script to print sum of digits of a given number.**

**Code:**

```
echo -n "Enter a number: "
read number

sum=0

while [ $number -gt 0 ]; do
	digit=$((number%10))
	sum=$((sum+digit))
	number=$((number/10))
done

echo "Sum of digits of entered number is $sum"
```

**Output:**

```
[mait@fedora ~]$ vi prog4.sh
[mait@fedora ~]$ chmod +x prog4.sh
[mait@fedora ~]$ ./prog4.sh
Enter a number: 123
Sum of digits of entered number is 6
```

# Program 11

**Aim: Write a script to check whether the given string is a palindrome**

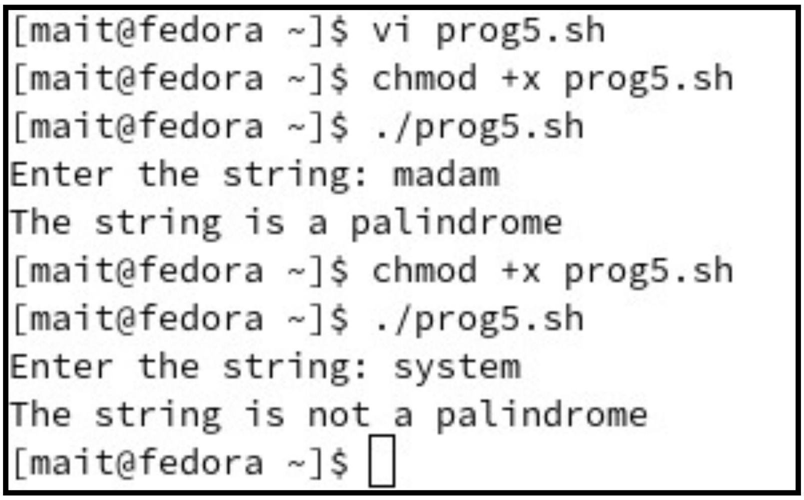
**Code:**

```
echo -n "Enter the string: "
read str

reversed=$(echo "$str" | rev)

if [ "$str" = "$reversed" ]; then
	echo "The string is a palindrome"
else
	echo "The string is not a palindrome"
fi
```

**Output:**



```
[mait@fedora ~]$ vi prog5.sh
[mait@fedora ~]$ chmod +x prog5.sh
[mait@fedora ~]$ ./prog5.sh
Enter the string: madam
The string is a palindrome
[mait@fedora ~]$ chmod +x prog5.sh
[mait@fedora ~]$ ./prog5.sh
Enter the string: system
The string is not a palindrome
[mait@fedora ~]$
```

# Program 12

**Aim: Write a program to implement CPU scheduling for first come first serve.**

**Code:**

```
processes=(
    "P1 0 5"
    "P2 1 3"
    "P3 2 8"
    "P4 3 6"
)

total_processes=${#processes[@]}
current_time=0

echo -e "Process\tArrival Time\tBurst Time\tWaiting Time"

for process in "${processes[@]}";do
    IFS=' ' read -r pid arrival burst <<< "$processes"
    if [ "$current_time" -lt "$arrival" ]; then
        current_time="$arrival"
fi
waiting_time=$((current_time - arrival))

echo -e "$pid\t$arrival\t$burst\t$waiting_time"

current_time=$((current_time + burst))
done
```

**Output:**

```
[mait@fedora Sarthak]$ ./fcfs.sh
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P1      0               5               0               5
P2      1               3               4               7
P3      2               8               6               14
P4      3               6               13              19
[mait@fedora Sarthak]$
```

# Program 13

**Aim: Write a program to implement CPU scheduling for shortest job first.**

**Code:**

```
processes=("P1 6" "P2 8" "P3 7" "P4 3")
sort_processes() {
  tmp_file=$(mktemp)
  for process in "${processes[@]}"; do
    echo "$process" >> "$tmp_file"
  done

  sorted_processes=$(sort -k2 -n "$tmp_file")
  rm "$tmp_file"
}

execute_processes() {
  echo "Executing processes in Shortest Job First order:"
  while read -r process; do
    process_id=$(echo "$process" | awk '{print $1}')
    burst_time=$(echo "$process" | awk '{print $2}')

    echo "Process $process_id with burst time $burst_time"
    sleep "$burst_time" # Simulate burst time with sleep
  done <<< "$sorted_processes"
}

sort_processes
execute_processes
```

**Output:**

```
[mait@fedora Sarthak]$ chmod +x ./SJF.sh
[mait@fedora Sarthak]$ ./SJF.sh
Executing processes in Shortest Job First order:
Process P4 with burst time 3
Process P1 with burst time 6
Process P3 with burst time 7
Process P2 with burst time 8
```

# Program 14

## Aim: Write a program to perform priority scheduling

## Code:

```
processes=(
  "3 Process A 10"
  "1 Process B 5"
  "2 Process C 7"
  "4 Process D 8"
)

display_scheduled_processes() {
  echo "Scheduled Processes (from highest to lowest priority):"
  # Sort processes by priority (highest first) and print them
  printf "%s\n" "${processes[@]}" | sort -nr | while read -r line; do
    priority=$(echo $line | awk '{print $1}')
    process_name=$(echo $line | awk '{print $2}')
    burst_time=$(echo $line | awk '{print $3}')
    echo "Process Name: $process_name, Priority: $priority, Burst Time: $burst_time"
  done
}

# Main script execution
if [ ${#processes[@]} -eq 0 ]; then
  echo "No processes to schedule."
else
  display_scheduled_processes
fi
```

## Output:

```
[mait@fedora Sarthak]$ vi ps.sh
[mait@fedora Sarthak]$ ./ps.sh
Scheduled Processes (from highest to lowest priority):
Process Name: Process, Priority: 4, Burst Time: D
Process Name: Process, Priority: 3, Burst Time: A
Process Name: Process, Priority: 2, Burst Time: C
Process Name: Process, Priority: 1, Burst Time: B
```

# Program 15

**Aim: Write a program to implement CPU scheduling for Round Robin.**

**Code:**

```
print_table_header() {
  echo "Process   Burst Time   Waiting Time   Turnaround Time"
  echo "---------------------------------------------------"
}

print_table_row() {
  printf "%-10s%-12d%-14d%-16d\n" $1 $2 $3 $4
}

n=4
quantum=4
burst_time=(10 5 8 6)

declare -a waiting_time
declare -a turnaround_time
declare -a remaining_time

for ((i=0; i<n; i++)); do
  remaining_time[$i]=${burst_time[$i]}
  waiting_time[$i]=0
done

time=0

while true; do
  done=true
  for ((i=0; i<n; i++)); do
    if [[ ${remaining_time[$i]} -gt 0 ]]; then
      done=false
      if [[ ${remaining_time[$i]} -gt $quantum ]]; then
        time=$((time + quantum))
        remaining_time[$i]=$((remaining_time[$i] - quantum))
      else
        time=$((time + remaining_time[$i]))
        waiting_time[$i]=$((time - burst_time[$i]))
        remaining_time[$i]=0
      fi
    fi
  done
  if $done; then
    break
```

```
    fi
done

for ((i=0; i<n; i++)); do
    turnaround_time[$i]=$((burst_time[$i] + waiting_time[$i]))
done
```

## Output:

```
sarthakjain@Sarthaks-MacBook-Air ~ % vi roundrobin.sh
sarthakjain@Sarthaks-MacBook-Air ~ % chmod +x ./roundrobin.sh
sarthakjain@Sarthaks-MacBook-Air ~ % ./roundrobin.sh
Process   Burst Time   Waiting Time   Turnaround Time
-----------------------------------------------------
P1        10           19             29
P2        5            16             21
P3        8            17             25
P4        6            21             27
```

# Program 16(a)

**Aim: Write a program for page replacement policy using a) LRU**

**Code:**

```
lru() {
   pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
   capacity=3
   frame=()
   page_faults=0
   echo "Page  | Page Fault"
   echo "------------------"
   for page in "${pages[@]}"; do
      if [[ ! " ${frame[@]} " =~ " $page " ]]; then
         if [[ ${#frame[@]} -eq $capacity ]]; then
            frame=("${frame[@]:1}")
         fi
         frame+=($page)
         ((page_faults++))
         echo "$page   | Yes"
      else
         echo "$page   | No"
      fi
   done
   echo "------------------"
   echo "Total Page Faults: $page_faults"
}

lru
```

**Output:**

```
sarthakjain@Sarthaks-MacBook-Air ~ % vi lru.sh
sarthakjain@Sarthaks-MacBook-Air ~ % chmod +x ./lru.sh
sarthakjain@Sarthaks-MacBook-Air ~ % ./lru.sh
Page  | Page Fault
------------------
7   | Yes
0   | Yes
1   | Yes
2   | Yes
0   | No
3   | Yes
0   | Yes
4   | Yes
2   | Yes
3   | Yes
0   | Yes
3   | No
2   | No
------------------
Total Page Faults: 10
```

# Program 16(b)

**Aim: Write a program for page replacement policy using b) FIFO**

**Code:**

```
fifo() {
   pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
   capacity=3
   frame=()
   page_faults=0
   echo "Page  | Page Fault"
   echo "------------------"
   for page in "${pages[@]}"; do
      if [[ ! " ${frame[@]} " =~ " $page " ]]; then
         if [[ ${#frame[@]} -eq $capacity ]]; then
            frame=("${frame[@]:1}")
         fi
         frame+=($page)
         ((page_faults++))
         echo "$page   | Yes"
      else
         echo "$page   | No"
      fi
   done
   echo "------------------"
   echo "Total Page Faults: $page_faults"
}

fifo
```

**Output:**

```
sarthakjain@Sarthaks-MacBook-Air ~ % vi fifo.sh
sarthakjain@Sarthaks-MacBook-Air ~ % chmod +x ./fifo.sh
sarthakjain@Sarthaks-MacBook-Air ~ % ./fifo.sh
Page  | Page Fault
------------------
7   | Yes
0   | Yes
1   | Yes
2   | Yes
0   | No
3   | Yes
0   | Yes
4   | Yes
2   | Yes
3   | Yes
0   | Yes
3   | No
2   | No
------------------
Total Page Faults: 10
```

# Program 16(c)

**Aim: Write a program for page replacement policy using c) Optimal**

**Code:**

```
optimal() {
   pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
   capacity=3
   frame=()
   page_faults=0
   echo "Page  | Page Fault"
   echo "------------------"
   for ((i=0; i<${#pages[@]}; i++)); do
      page=${pages[i]}
      if [[ ! " ${frame[@]} " =~ " $page " ]]; then
         if [[ ${#frame[@]} -lt $capacity ]]; then
            frame+=($page)
         else
            future_use=()
            for f in "${frame[@]}"; do
               found=false
               for ((j=i+1; j<${#pages[@]}; j++)); do
                  if [[ ${pages[j]} -eq $f ]]; then
                     future_use+=($j)
                     found=true
                     break
                  fi
               done
               if [[ $found == false ]]; then
                  future_use+=(9999)
               fi
            done
            max_index=$(echo "${future_use[@]}" | tr ' ' '\n' | sort -nr | head -n1)
            for ((k=0; k<${#frame[@]}; k++)); do
               if [[ ${future_use[$k]} -eq $max_index ]]; then
                  unset frame[$k]
                  break
               fi
            done
            frame=("${frame[@]}")
            frame+=($page)
         fi
         ((page_faults++))
         echo "$page   | Yes"
      else
         echo "$page   | No"
```

```
        fi
    done
    echo "------------------"
    echo "Total Page Faults: $page_faults"
}

optimal
```

## Output:

```
sarthakjain@Sarthaks-MacBook-Air ~ % vi opt.sh
sarthakjain@Sarthaks-MacBook-Air ~ % chmod +x ./opt.sh
sarthakjain@Sarthaks-MacBook-Air ~ % ./opt.sh
Page  | Page Fault
------------------
7    | Yes
0    | Yes
1    | Yes
2    | Yes
0    | No
3    | Yes
0    | No
4    | Yes
2    | No
3    | No
0    | Yes
3    | No
2    | No
------------------
Total Page Faults: 7
```

# Program 17

**Aim: Write a program to implement first fit, best fit and worst fit algorithm for memory management.**

**Code:**

```
#!/bin/bash

declare -a memory_blocks=(100 200 300)
declare -a process_sizes=(75 125 250 50)
declare -a block_status

display_allocation() {
    echo "Memory Allocation:"
    for i in "${!process_sizes[@]}"; do
        if [ "${block_status[$i]}" != "-1" ]; then
            echo "Process ${i} of size ${process_sizes[$i]} allocated to Block ${block_status[$i]} of
size ${memory_blocks[${block_status[$i]}]}"
        else
            echo "Process ${i} of size ${process_sizes[$i]} is not allocated"
        fi
    done
}

first_fit() {
    echo "Using First Fit Algorithm"
    for i in "${!process_sizes[@]}"; do
        allocated=false
        for j in "${!memory_blocks[@]}"; do
            if [ "${memory_blocks[$j]}" -ge "${process_sizes[$i]}" ] && [ "${block_status[$j]}" ==
"-1" ]; then
                block_status[$i]=$j
                memory_blocks[$j]=$((memory_blocks[$j] - process_sizes[$i]))
                allocated=true
                break
            fi
        done

        if [ "$allocated" == true ]; then
            echo "Process ${i} of size ${process_sizes[$i]} allocated to Block ${block_status[$i]}
(First Fit)"
        else
            echo "Process ${i} of size ${process_sizes[$i]} could not be allocated (First Fit)"
        fi
    done
}
```

```
best_fit() {
  echo "Using Best Fit Algorithm"
  for i in "${!process_sizes[@]}"; do
    best_fit_block=-1
    min_diff=-1
    for j in "${!memory_blocks[@]}"; do
      if [ "${memory_blocks[$j]}" -ge "${process_sizes[$i]}" ] && [ "${block_status[$j]}" ==
"-1" ]; then
        diff=$((memory_blocks[$j] - process_sizes[$i]))
        if [ "$min_diff" == -1 ] || [ "$diff" -lt "$min_diff" ]; then
          min_diff=$diff
          best_fit_block=$j
        fi
      fi
    done

    if [ "$best_fit_block" -ne -1 ]; then
      block_status[$i]=$best_fit_block
      memory_blocks[$best_fit_block]=$((memory_blocks[$best_fit_block] - process_sizes[$i]))
      echo "Process ${i} of size ${process_sizes[$i]} allocated to Block ${block_status[$i]} (Best
Fit)"
    else
      echo "Process ${i} of size ${process_sizes[$i]} could not be allocated (Best Fit)"
    fi
  done
}

worst_fit() {
  echo "Using Worst Fit Algorithm"
  for i in "${!process_sizes[@]}"; do
    worst_fit_block=-1
    max_diff=-1
    for j in "${!memory_blocks[@]}"; do
      if [ "${memory_blocks[$j]}" -ge "${process_sizes[$i]}" ] && [ "${block_status[$j]}" ==
"-1" ]; then
        diff=$((memory_blocks[$j] - process_sizes[$i]))
        if [ "$max_diff" == -1 ] || [ "$diff" -gt "$max_diff" ]; then
          max_diff=$diff
          worst_fit_block=$j
        fi
      fi
    done

    if [ "$worst_fit_block" -ne -1 ]; then
      block_status[$i]=$worst_fit_block
      memory_blocks[$worst_fit_block]=$((memory_blocks[$worst_fit_block] -
process_sizes[$i]))
```

```
            echo "Process ${i} of size ${process_sizes[$i]} allocated to Block ${block_status[$i]} (Worst Fit)"
        else
            echo "Process ${i} of size ${process_sizes[$i]} could not be allocated (Worst Fit)"
        fi
    done
}

main() {
    block_status=()
    for ((i=0; i<${#memory_blocks[@]}; i++)); do
        block_status[$i]=-1
    done

    first_fit
    echo "------------------------------"

    block_status=()
    for ((i=0; i<${#memory_blocks[@]}; i++)); do
        block_status[$i]=-1
    done

    best_fit
    echo "------------------------------"

    block_status=()
    for ((i=0; i<${#memory_blocks[@]}; i++)); do
        block_status[$i]=-1
    done

    worst_fit
}
```

## Output:

```
[mait@fedora ~]$ vi mem.sh
[mait@fedora ~]$ chmod +x mem.sh
[mait@fedora ~]$ ./mem.sh
Using First Fit Algorithm
Process 0 of size 75 allocated to Block 0 (First Fit)
Process 1 of size 125 allocated to Block 1 (First Fit)
Process 2 of size 250 allocated to Block 2 (First Fit)
Process 3 of size 50 could not be allocated (First Fit)
------------------------------
Using Best Fit Algorithm
Process 0 of size 75 allocated to Block 1 (Best Fit)
Process 1 of size 125 could not be allocated (Best Fit)
Process 2 of size 250 could not be allocated (Best Fit)
Process 3 of size 50 allocated to Block 2 (Best Fit)
------------------------------
Using Worst Fit Algorithm
Process 0 of size 75 could not be allocated (Worst Fit)
Process 1 of size 125 could not be allocated (Worst Fit)
Process 2 of size 250 could not be allocated (Worst Fit)
Process 3 of size 50 could not be allocated (Worst Fit)
```

# Program 18

**Aim: Write a program to implement reader/writer problem using semaphore.**

**Code:**

```
#!/bin/bash

read_count=0
write_count=0
resource="Shared Resource"

reader_mutex="reader_mutex.lock"
writer_mutex="writer_mutex.lock"
read_mutex="read_mutex.lock"

increase_readers() {
  exec 200>$read_mutex
  flock -n 200 || exit 1
  ((read_count++))
  if [ $read_count -eq 1 ]; then
    exec 201>$writer_mutex
    flock -n 201 || exit 1
  fi
  flock -u 200
}

decrease_readers() {
  exec 200>$read_mutex
  flock -n 200 || exit 1
  ((read_count--))
  if [ $read_count -eq 0 ]; then
    flock -u 201
  fi
  flock -u 200
}

start_reading() {
  echo "Reader $1 is reading $resource"
  sleep 1
  echo "Reader $1 finished reading $resource"
}

start_writing() {
  echo "Writer $1 is writing to $resource"
  sleep 2
```

```
    echo "Writer $1 finished writing to $resource"
}

reader() {
    while true; do
        sleep $((RANDOM % 5))
        increase_readers
        start_reading $1
        decrease_readers
    done
}

writer() {
    while true; do
        sleep $((RANDOM % 5))
        exec 201>$writer_mutex
        flock -n 201 || exit 1
        start_writing $1
        flock -u 201
    done
}

main() {
    for i in {1..5}; do
        reader $i &
    done

    for i in {1..3}; do
        writer $i &
    done

    wait
}

main
```

## Output:

```
[mait@fedora ~]$ vi r_w.sh
[mait@fedora ~]$ chmod +x r_w.sh
[mait@fedora ~]$ ./r_w.sh
Writer 1 is writing to Shared Resource
Writer 1 finished writing to Shared Resource
Writer 2 is writing to Shared Resource
Writer 2 finished writing to Shared Resource
Writer 3 is writing to Shared Resource
Writer 3 finished writing to Shared Resource
```

# Program 19

**Aim: Write a program to implement Producer-Consumer problem using semaphores.**

**Code:**

```
#!/bin/bash

BUFFER_SIZE=5
MAX_ITEMS=10
echo 0 > full
echo $BUFFER_SIZE > empty
echo 1 > mutex
buffer=()

producer() {
  local items_produced=0
  while [ $items_produced -lt $MAX_ITEMS ]; do
    sleep $((RANDOM % 3 + 1))
    item=$((RANDOM % 100))
    while [ "$(cat empty)" -le 0 ]; do sleep 1; done
    empty_value=$(cat empty)
    echo $((empty_value - 1)) > empty
    while [ "$(cat mutex)" -le 0 ]; do sleep 1; done
    echo 0 > mutex
    buffer+=($item)
    echo "Producer produced: $item. Buffer size: ${#buffer[@]}/$BUFFER_SIZE"
    echo 1 > mutex
    full_value=$(cat full)
    echo $((full_value + 1)) > full
    items_produced=$((items_produced + 1))
  done
}

consumer() {
  local items_consumed=0
  while [ $items_consumed -lt $MAX_ITEMS ]; do
    sleep $((RANDOM % 3 + 1))
    while [ "$(cat full)" -le 0 ]; do sleep 1; done
    full_value=$(cat full)
    echo $((full_value - 1)) > full
    while [ "$(cat mutex)" -le 0 ]; do sleep 1; done
    echo 0 > mutex
    item=${buffer[0]}
    buffer=("${buffer[@]:1}")
```

```
    echo "Consumer consumed: $item. Buffer size: ${#buffer[@]}/$BUFFER_SIZE"
    echo 1 > mutex
    empty_value=$(cat empty)
    echo $((empty_value + 1)) > empty
    items_consumed=$((items_consumed + 1))
  done
}

producer &
consumer &

wait
```

## Output:

```
[mait@fedora ~]$  vi prodcons.sh
[mait@fedora ~]$ ./prodcons.sh
Producer produced: 66. Buffer size: 1/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 97. Buffer size: 2/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 37. Buffer size: 3/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 97. Buffer size: 4/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 81. Buffer size: 5/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 0. Buffer size: 6/5
Producer produced: 82. Buffer size: 7/5
Producer produced: 85. Buffer size: 8/5
Consumer consumed: . Buffer size: 0/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 58. Buffer size: 9/5
Consumer consumed: . Buffer size: 0/5
Consumer consumed: . Buffer size: 0/5
Producer produced: 84. Buffer size: 10/5
Consumer consumed: . Buffer size: 0/5
```

# Program 20

**Aim: Write a program to implement Banker's algorithm for deadlock avoidance.**

**Code:**

```
#!/bin/bash

check_safety() {
 finish=()
 safe_sequence=()
 work=("${available[@]}")

 for ((i=0; i<$num_processes; i++)); do
  finish[$i]=0
 done

 while :; do
  found=false

  for ((i=0; i<$num_processes; i++)); do
   if [ ${finish[$i]} -eq 0 ]; then
    can_proceed=true
    for ((j=0; j<$num_resources; j++)); do
     if [ ${need[$i*$num_resources + $j]} -gt ${work[$j]} ]; then
      can_proceed=false
      break
     fi
    done

    if [ "$can_proceed" = true ]; then
     for ((j=0; j<$num_resources; j++)); do
      work[$j]=$((work[$j] + ${allocation[$i*$num_resources + $j]}))
     done
     finish[$i]=1
     safe_sequence+=($i)
     found=true
    fi
   fi
  done

  if [ "$found" = false ]; then
   break
  fi
 done
```

```
  for ((i=0; i<$num_processes; i++)); do
    if [ ${finish[$i]} -eq 0 ]; then
      echo "System is not in a safe state."
      return 1
    fi
  done

  echo "System is in a safe state."
  echo "Safe sequence: ${safe_sequence[@]}"
  return 0
}

num_processes=5
num_resources=3

allocation=(
  0 1 0
  2 0 0
  3 0 2
  2 1 1
  0 0 2
)

max=(
  7 5 3
  3 2 2
  9 0 2
  4 2 2
  5 3 3
)

available=(3 3 2)

need=()
for ((i=0; i<num_processes; i++)); do
  for ((j=0; j<num_resources; j++)); do
    need[$i*$num_resources + $j]=$((max[$i*$num_resources + $j] - allocation[$i*$num_resources
+ $j]))
  done
done

echo "Need matrix:"
for ((i=0; i<num_processes; i++)); do
  for ((j=0; j<num_resources; j++)); do
    echo -n "${need[$i*$num_resources + $j]} "
  done
  echo
```

done

check_safety

## Output:

```
[mait@fedora ~]$ vi banker.sh
[mait@fedora ~]$ chmod +x banker.sh
[mait@fedora ~]$ ./banker.sh
Need matrix:
7 4 3
1 2 2
6 0 0
2 1 1
5 3 1
System is in a safe state.
Safe sequence: 1 3 4 0 2
```

# Program 21(a)

**Aim: Write C programs to implement the various File Organization Techniques.**
**(a) Sequential File Organization**

**Code:**

```
#include <stdio.h>
#include <stdlib.h>

struct record {
    int id;
    char name[20];
    int age;
};

void writeRecords(const char *filename) {
    FILE *file = fopen(filename, "w");
    if (file == NULL) {
        return;
    }

    struct record recs[3] = {
        {1, "Alice", 25},
        {2, "Bob", 30},
        {3, "Charlie", 22}
    };

    for (int i = 0; i < 3; i++) {
        fwrite(&recs[i], sizeof(struct record), 1, file);
    }

    fclose(file);
}

void readRecords(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        return;
    }

    struct record rec;
    while (fread(&rec, sizeof(struct record), 1, file)) {
        printf("ID: %d, Name: %s, Age: %d\n", rec.id, rec.name, rec.age);
    }
```

```
    fclose(file);
}

int main() {
    const char *filename = "sequential.dat";
    writeRecords(filename);
    readRecords(filename);
    return 0;
}
```

**Output:**

```
ID: 1, Name: Alice, Age: 25
ID: 2, Name: Bob, Age: 30
ID: 3, Name: Charlie, Age: 22
sarthakjain@Sarthaks-MacBook-Air
```

# Program 21(b)

**Aim: Write C programs to implement the various File Organization Techniques.**
**(b) Indexed File Organization**

**Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct record {
    int id;
    char name[20];
    int age;
};

void writeRecords(const char *filename, const char *indexfile) {
    FILE *file = fopen(filename, "w");
    FILE *indexFile = fopen(indexfile, "w");
    if (file == NULL || indexFile == NULL) {
        return;
    }

    struct record recs[3] = {
        {1, "Alice", 25},
        {2, "Bob", 30},
        {3, "Charlie", 22}
    };

    for (int i = 0; i < 3; i++) {
        fwrite(&recs[i], sizeof(struct record), 1, file);
        fprintf(indexFile, "%d %ld\n", recs[i].id, ftell(file) - sizeof(struct record));
    }

    fclose(file);
    fclose(indexFile);
}

void searchById(const char *filename, const char *indexfile, int searchId) {
    FILE *file = fopen(filename, "r");
    FILE *indexFile = fopen(indexfile, "r");
    if (file == NULL || indexFile == NULL) {
        return;
    }
```

```
    int id;
    long position;
    struct record rec;

    while (fscanf(indexFile, "%d %ld", &id, &position) != EOF) {
        if (id == searchId) {
            fseek(file, position, SEEK_SET);
            fread(&rec, sizeof(struct record), 1, file);
            printf("Record found - ID: %d, Name: %s, Age: %d\n", rec.id, rec.name, rec.age);
            fclose(file);
            fclose(indexFile);
            return;
        }
    }

    fclose(file);
    fclose(indexFile);
}

int main() {
    const char *filename = "indexed.dat";
    const char *indexfile = "index.txt";
    writeRecords(filename, indexfile);
    searchById(filename, indexfile, 2);
    return 0;
}
```

## Output:

```
Record found - ID: 2, Name: Bob, Age: 30
sarthakjain@Sarthaks-MacBook-Air os lab %
```

# Program 21(c)

**Aim: Write C programs to implement the various File Organization Techniques.**
**(b) Hashed File Organization**

**Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABLE_SIZE 10

struct record {
    int id;
    char name[20];
    int age;
};

struct record hashTable[TABLE_SIZE];

int hashFunction(int id) {
    return id % TABLE_SIZE;
}

void insertRecords() {
    struct record recs[3] = {
        {1, "Alice", 25},
        {2, "Bob", 30},
        {12, "Charlie", 22}
    };

    for (int i = 0; i < 3; i++) {
        int index = hashFunction(recs[i].id);
        while (hashTable[index].id != 0) {
            index = (index + 1) % TABLE_SIZE;
        }
        hashTable[index] = recs[i];
    }
}

void searchRecord(int id) {
    int index = hashFunction(id);
    int startIndex = index;
    while (hashTable[index].id != 0) {
        if (hashTable[index].id == id) {
```

```
            printf("Record found - ID: %d, Name: %s, Age: %d\n", hashTable[index].id,
hashTable[index].name, hashTable[index].age);
            return;
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == startIndex) break;
    }
}

void displayTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i].id != 0) {
            printf("Index %d -> ID: %d, Name: %s, Age: %d\n", i, hashTable[i].id, hashTable[i].name,
hashTable[i].age);
        }
    }
}

int main() {
    insertRecords();
    searchRecord(12);
    displayTable();
    return 0;
}
```

## Output:

```
Record found - ID: 12, Name: Charlie, Age: 22
Index 1 -> ID: 1, Name: Alice, Age: 25
Index 2 -> ID: 2, Name: Bob, Age: 30
Index 3 -> ID: 12, Name: Charlie, Age: 22
sarthakjain@Sarthaks-MacBook-Air os lab %
```