MANIPAL UNIVERSITY JAIPUR

SCHOOL OF BASIC SCIENCE

DEPARTMENT OF COMPUTER APPLICATIONS

CA7132 UNIX & Shell Programming LAB

SESSION: JULY24 – NOV24

LAB ASSIGNMENT

SUBMITTED BY: -

Vaibhav Gupta

23FS20MCA00068

MCA (3rd Semester)

SUBMITTED TO: -

ARPANA SINHAL

Faculty of Science, School of Basic Sciences

Department of Computer Applications MCA

1. A) Introduction and Characteristics of the Linux Operating System

Introduction: Linux is a powerful open-source operating system that was inspired by UNIX. It was created by Linus Torvalds in 1991 and has since grown into a versatile and widely used system. Linux is employed in a wide range of devices, including servers, supercomputers, smartphones, and more. Distributed under the GNU General Public License (GPL), Linux can be freely used, modified, and redistributed. Its reputation for security, stability, and adaptability has made it especially popular among developers, system administrators, and businesses worldwide.

Characteristics:

- **Shell and Command-Line Interface (CLI):** Linux offers a robust command-line interface, commonly known as the shell, which allows users to run commands and automate processes using scripts.
- **File System Structure:** Linux organizes its files in a hierarchical file system, which helps in managing data efficiently and logically.
- **Security and Stability:** With strong file permissions and excellent system stability, Linux is considered a secure and reliable choice for servers and mission-critical applications.
- **Multi-User Support:** Linux enables multiple users to work simultaneously on a system, each with their own account and specific permissions, facilitating efficient resource sharing.

---

1. B) Write and Execute the Following Directory and File-Related Commands

i. mkdir

- mkdir new_folder: Creates a new directory.
- mkdir -p parent/child: Creates a parent directory and a nested child directory.

ii. ls

- ls: Lists the files and directories in the current directory.
- ls -l: Shows detailed information about the files (including permissions, owners, sizes, and timestamps).
- ls -a: Displays all files, including hidden ones.
- ls -lh: Lists files with human-readable sizes.

iii. cd

- cd folder: Changes the current directory to "folder."
- cd ..: Moves one directory up.
- cd /path: Changes the current directory to a specific path.

iv. pwd

- pwd: Prints the full path of the current working directory.

v. rmdir

- rmdir folder: Removes an empty directory.

vi. rm

- rm file.txt: Deletes the file "file.txt."
- rm -r folder: Deletes the folder and all of its contents.
- rm -i file.txt: Asks for confirmation before deleting the file.

vii. cp

- cp file1 file2: Copies the contents of "file1" to "file2."
- cp -r folder1 folder2: Recursively copies the contents of "folder1" to "folder2."

viii. head

- head file.txt: Displays the first 10 lines of the file.

- head -n 5 file.txt: Shows the first 5 lines of the file.

ix. tail

- tail file.txt: Displays the last 10 lines of the file.
- tail -n 5 file.txt: Shows the last 5 lines of the file.

x. mv

- mv file1 file2: Renames or moves "file1" to "file2."
- mv folder1 /path: Moves "folder1" to the specified path.

xi. who

- who: Displays a list of users currently logged in.

xii. ln

- ln file link_name: Creates a hard link to the file.
- ln -s file link_name: Creates a symbolic (soft) link to the file.

xiii. wc

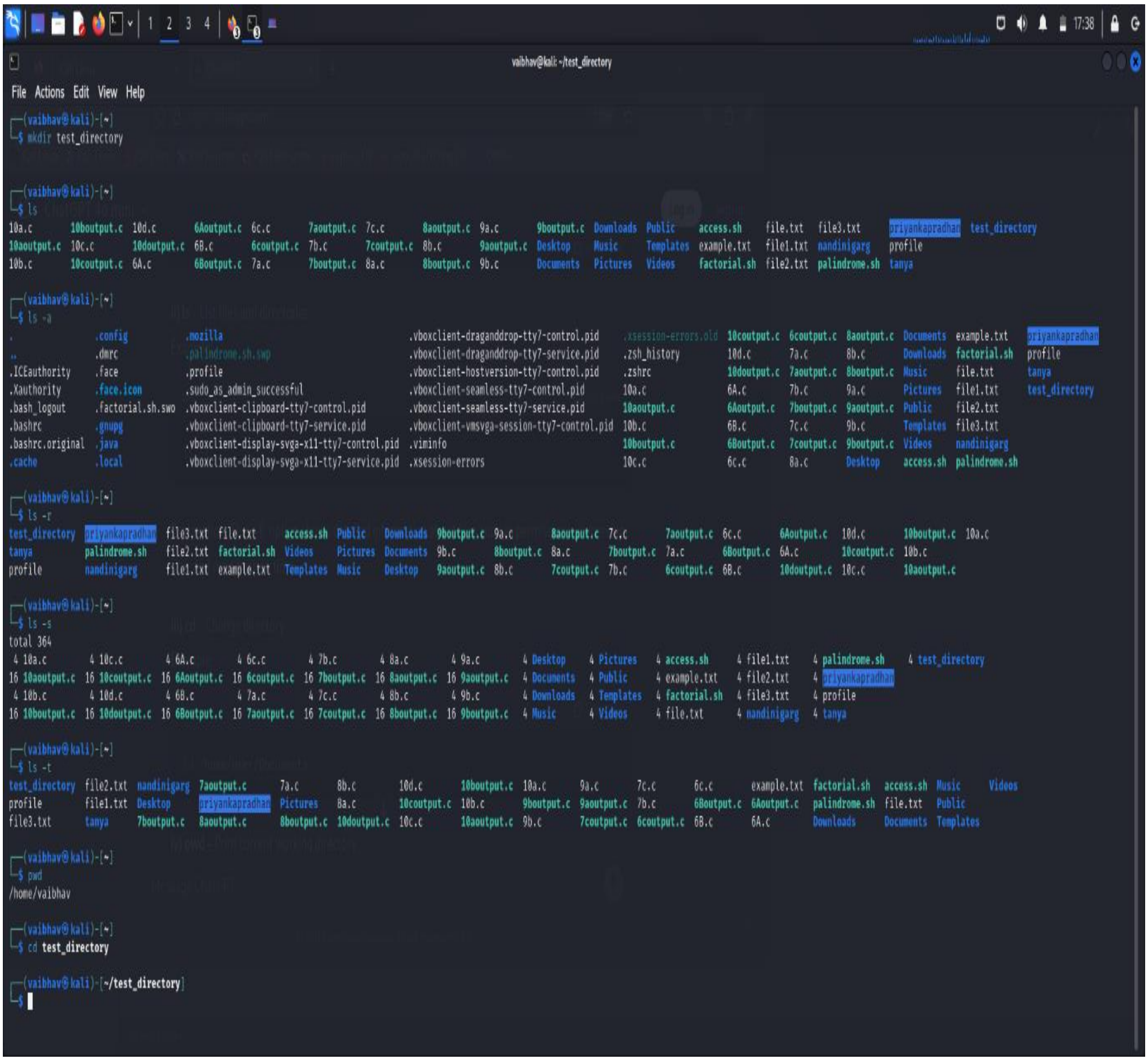
- wc file.txt: Counts the number of lines, words, and characters in the file.
- wc -l file.txt: Counts only the number of lines in the file.

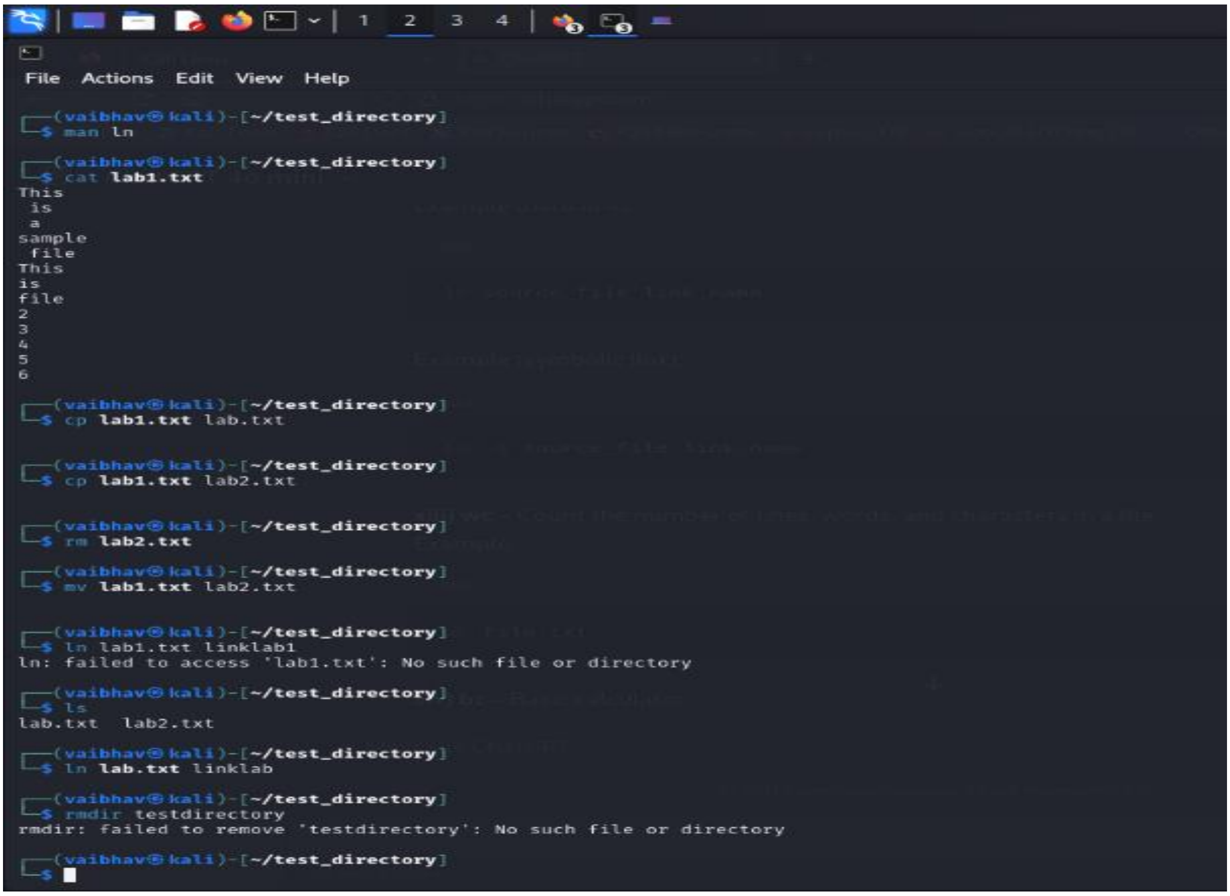
xiv. bc

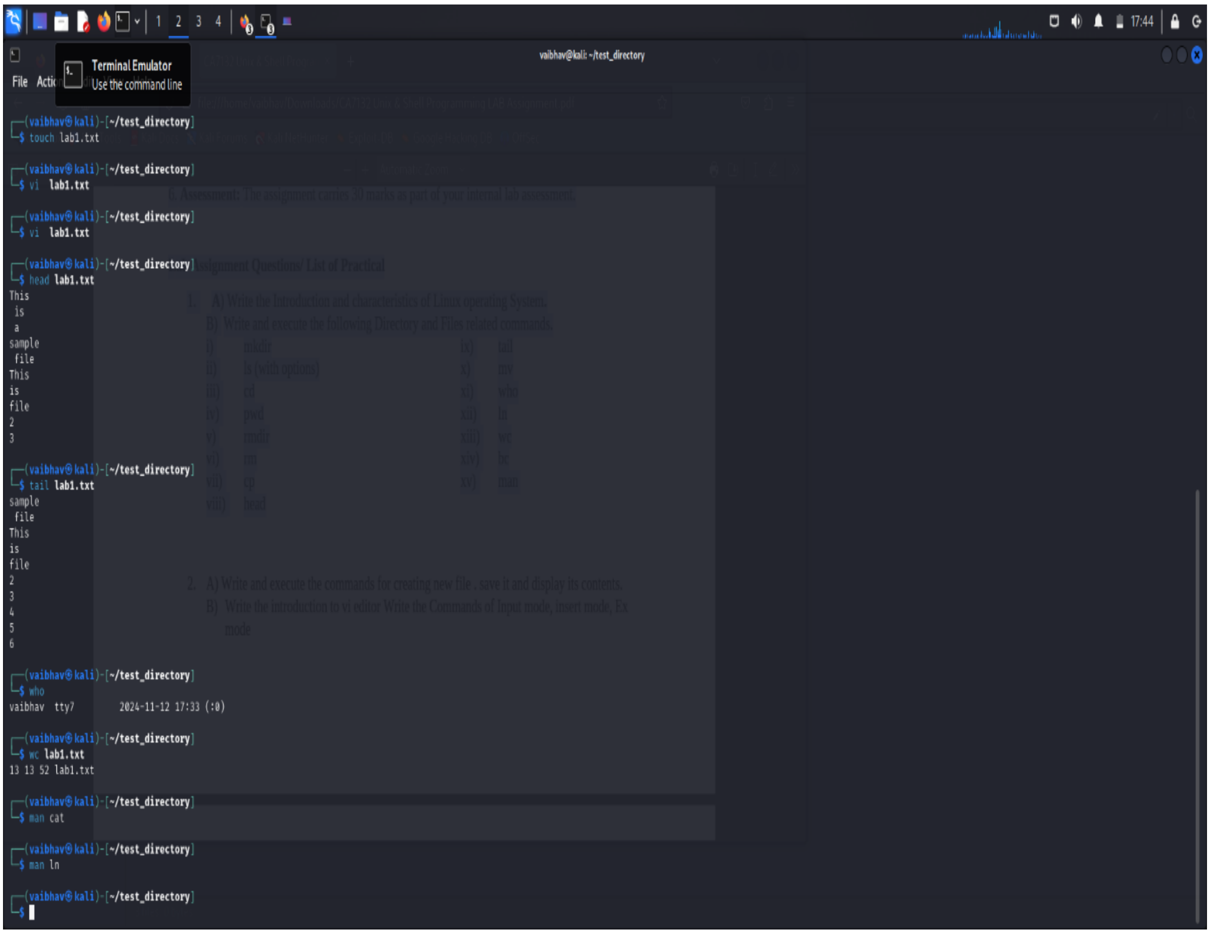
- bc: Starts the command-line calculator for arithmetic operations.

xv. man

- man command: Displays the manual page for a specified command.




File Actions Edit View Help
(vaibhav㉿kali)-[~/test_directory]
$ man ln
(vaibhav㉿kali)-[~/test_directory]
$ cat lab1.txt
This
is
a
sample
file
This
is
file
2
3
4
5
6
(vaibhav㉿kali)-[~/test_directory]
$ cp lab1.txt lab.txt
(vaibhav㉿kali)-[~/test_directory]
$ cp lab1.txt lab2.txt
(vaibhav㉿kali)-[~/test_directory]
$ rm lab2.txt
(vaibhav㉿kali)-[~/test_directory]
$ mv lab1.txt lab2.txt
(vaibhav㉿kali)-[~/test_directory]
$ ln lab1.txt linklab1
ln: failed to access 'lab1.txt': No such file or directory
(vaibhav㉿kali)-[~/test_directory]
$ ls
lab.txt lab2.txt
(vaibhav㉿kali)-[~/test_directory]
$ ln lab.txt linklab
(vaibhav㉿kali)-[~/test_directory]
$ rmdir testdirectory
rmdir: failed to remove 'testdirectory': No such file or directory
(vaibhav㉿kali)-[~/test_directory]
$


Terminal Emulator
Use the command line
vaibhav@kali: ~/test_directory
(vaibhav㉿kali)-[~/test_directory]
$ touch lab1.txt
(vaibhav㉿kali)-[~/test_directory]
$ vi lab1.txt
(vaibhav㉿kali)-[~/test_directory]
$ vi lab1.txt
(vaibhav㉿kali)-[~/test_directory]
$ head lab1.txt
This
is
a
sample
file
This
is
file
2
3
(vaibhav㉿kali)-[~/test_directory]
$ tail lab1.txt
sample
file
This
is
file
2
3
4
5
6
(vaibhav㉿kali)-[~/test_directory]
$ who
vaibhav tty7 2024-11-12 17:33 (:0)
(vaibhav㉿kali)-[~/test_directory]
$ wc lab1.txt
13 13 52 lab1.txt
(vaibhav㉿kali)-[~/test_directory]
$ man cat
(vaibhav㉿kali)-[~/test_directory]
$ man ln
(vaibhav㉿kali)-[~/test_directory]
$

2. A) Write and Execute the Commands for Creating a New File, Saving It, and Displaying Its Contents

Step 1: Create a New File

- To create an empty file, use the touch command:

bash
Copy code
touch newfile.txt

- To edit the file, you can use a text editor like nano or vi:

bash
Copy code
nano newfile.txt
vi newfile.txt

Step 2: Add Content and Save the File

- In **nano**:
  - Type the content you want.
  - Press Ctrl + O to save and Ctrl + X to exit.
- In **vi**:
  - Press i to enter insert mode and type the content.
  - Save and exit by pressing Esc, then type :wq.

Step 3: Display File Content

- Using cat:

bash
Copy code
cat newfile.txt

- Using less:

bash
Copy code
less newfile.txt

---

2. B) Introduction to the vi Editor

The **vi** editor is a widely used text editor in Unix-based systems, known for its efficiency and minimalistic design. It operates in different modes, including **Command Mode**, **Insert Mode**, and **Ex Mode**, each suited for specific tasks.

Modes in vi Editor:

1. **Command Mode:**
   - The default mode in which you can navigate through the text and manipulate it (e.g., delete, copy, paste).
   - Press Esc to return to Command Mode from any other mode.
2. **Insert Mode:**
   - This mode is used for inserting or editing text.
   - Commands in Insert Mode:
     - i: Insert text before the cursor.
     - a: Append text after the cursor.
     - o: Open a new line below the cursor.
3. **Ex Mode:**
   - This mode is used for advanced editing tasks, such as saving, quitting, and searching/replacing text.
   - Common Ex Mode commands:
     - :w: Save the file.
     - :q: Quit vi.

- :%s/old/new/g: Replace all occurrences of "old" with "new."

---

3. A) Write a Program to Check File Access Permissions
Check File Permissions

bash

Copy code

```
#!/bin/bash
filename=$1
if [ -r "$filename" ]; then
  echo "You have read permission on $filename"
else
  echo "You do NOT have read permission on $filename"
fi

if [ -w "$filename" ]; then
  echo "You have write permission on $filename"
else
  echo "You do NOT have write permission on $filename"
fi

if [ -x "$filename" ]; then
  echo "You have execute permission on $filename"
else
  echo "You do NOT have execute permission on $filename"
fi
```

---

3. B) Change File Permissions

bash

Copy code

```
chmod u+rwx example.txt   # Grant read, write, and execute permissions to the user.
chmod g-w example.txt     # Remove write permission from the group.
chmod o+r example.txt     # Grant read permission to others.
```

---

3. C) Change File Ownership

bash

Copy code

```
sudo chown username example.txt  # Change the ownership of the file to "username."
```

---

3. D) Change File Group

bash

Copy code

```
sudo chgrp groupname example.txt  # Change the group ownership of the file to "groupname."
```

---

4. A) Write a Program to Check Whether a Given String is a Palindrome
Check Palindrome

bash

Copy code

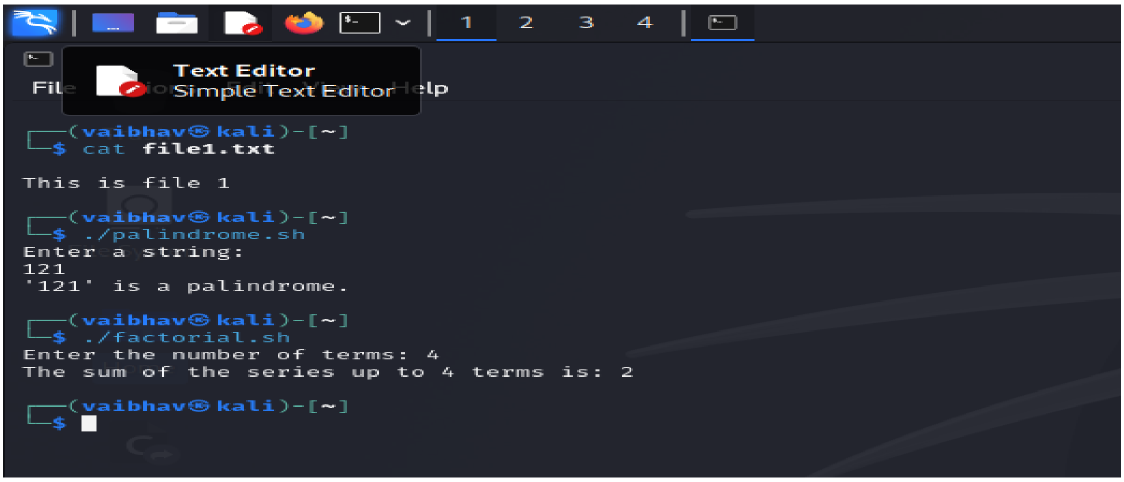
```
#!/bin/bash
echo "Enter a string:"
read str
reverse_str=$(echo "$str" | rev)
if [ "$str" == "$reverse_str" ]; then
  echo "$str is a palindrome"
else
  echo "$str is not a palindrome"
fi
```

---

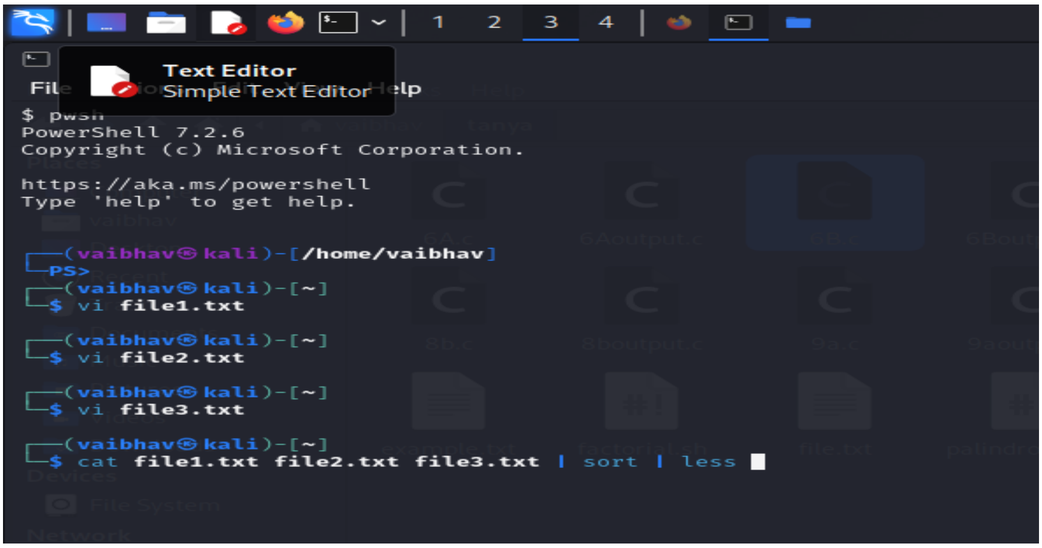
4. B) Write a Shell Program to Sum Up the Following Series

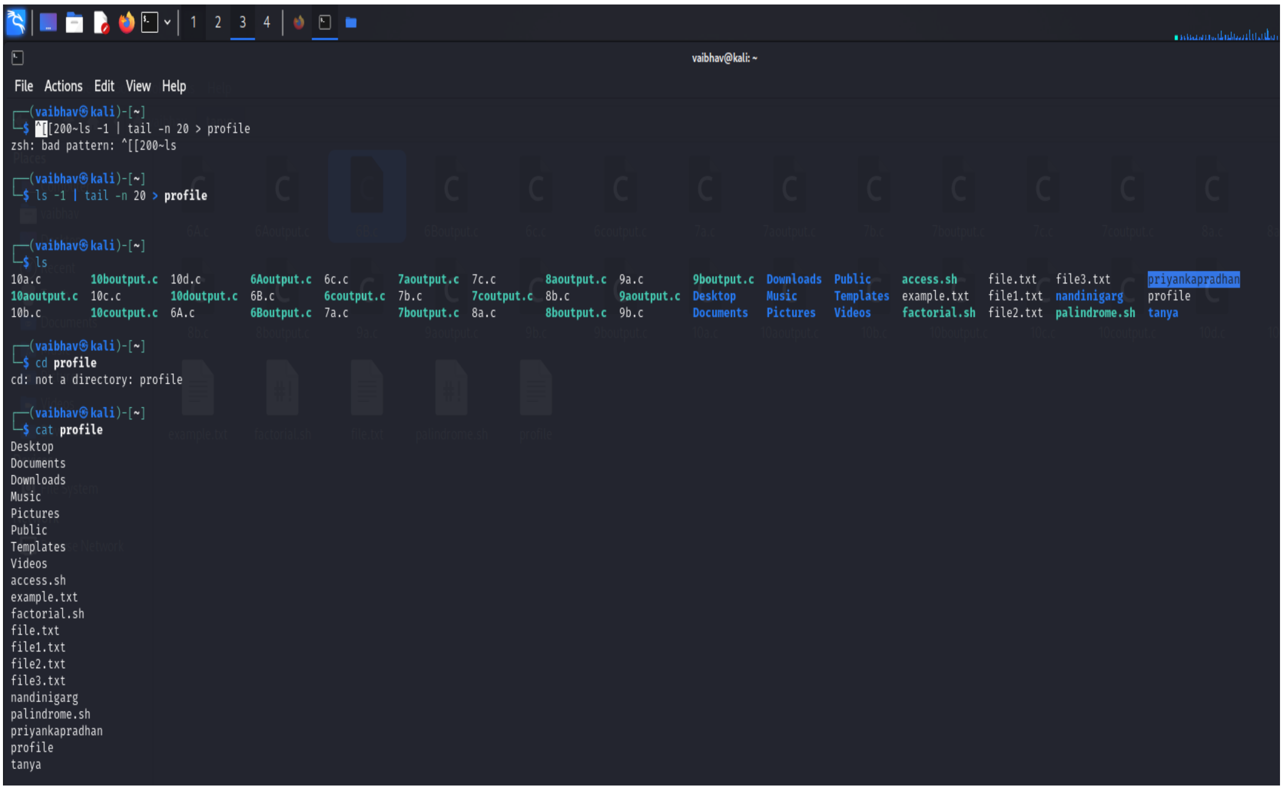
```
#!/bin/bash
factorial() {
  n=$1
  fact=1
  for (( i=1; i<=n; i++ )); do
    fact=$((fact * i))
  done
  echo $fact
}

echo "Enter the number of terms:"
read terms
sum=0
for (( i=1; i<=terms; i++ )); do
  fact=$(factorial $i)
  term=$(echo "scale=4; $i / $fact" | bc -l)
  sum=$(echo "scale=4; $sum + $term" | bc -l)
done
```

echo "The sum of the series up to $terms terms is: $sum"



5 Merge the contents of three files, sort them and display the sorted output on screen page by page.
Command:
bash
Copy code
cat file1.txt file2.txt file3.txt | sort | less
Steps:
1. Use cat to merge files.
2. Pipe output to sort for alphabetical sorting.
3. Use less for paginated display.





6. UNIX System Calls: File Operations:

A) Write a C program to implement basic file operations using system calls (open, read, write, close).

Open a file (create it if it doesn't exist).

Write some text to the file.

Read the text from the file.

Close the file.

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

   int file;

char buffer[100];

   file = open("example.txt", O_RDWR | O_CREAT, 0644);

   if (file < 0) {

      perror("Failed to open file");

      exit(1);

   }
```

```
    char *text = "Hello";
    if (write(file, text, strlen(text)) < 0) {
        perror("Failed to write to file");
        close(file);
        exit(1);
    }
    lseek(file, 0, SEEK_SET);
    // Read from file
    ssize_t bytesRead = read(file, buffer, sizeof(buffer) - 1);
    if (bytesRead < 0) {
        perror("Failed to read from file");
        close(file);
        exit(1);
    }
    buffer[bytesRead] = '\0';
    printf("Content of file: %s\n", buffer);
    close(file);
    return 0;
}
```

<u>B. Implement a program to create a child process using fork() and show parent-child process interaction.</u>

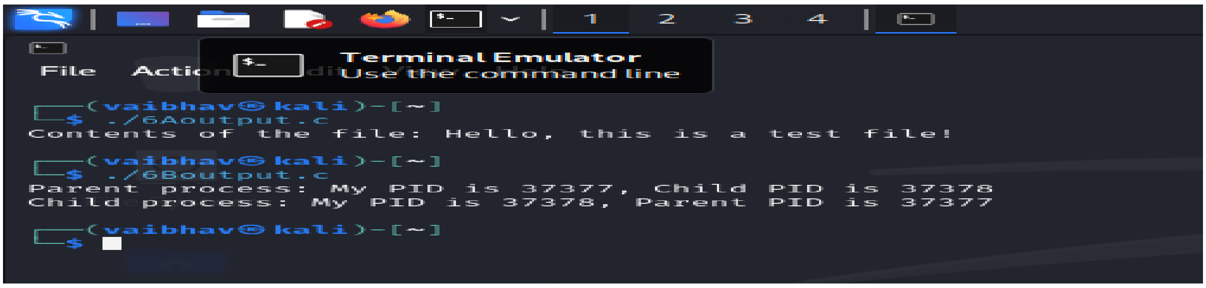
- The parent process creates a child process using **fork().**
- Both the parent and child processes print messages showing their **process IDs** (PIDs).
- The parent waits for the child process to complete using **wait().**

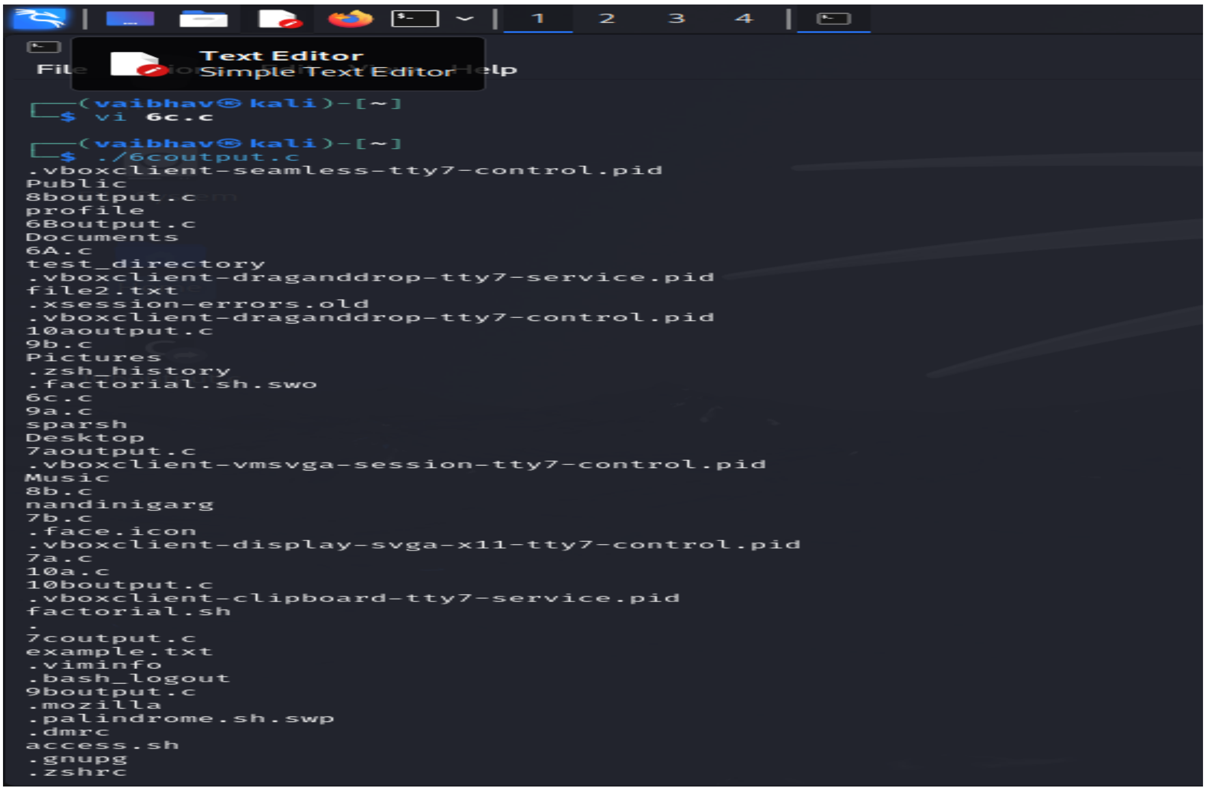
```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
    pid_t pid = fork();
    if (pid < 0) {
        perror("Fork failed");
```

```
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("Child Process: PID = %d, Parent PID = %d\n", getpid(), getppid());
        printf("Child Process: Doing some work...\n");
        sleep(2); // Simulate work
        printf("Child Process: Work done\n");
    } else {
        // Parent process
        printf("Parent Process: PID = %d, Waiting for child to finish...\n", getpid());
        wait(NULL); // Wait for child process to complete
        printf("Parent Process: Child finished\n");
    }
    return 0;
}
```

C) Write a C program to simulate the ls command using opendir(), readdir(), and closedir() system calls.

```
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>
int main() {
    DIR *dir;
    struct dirent *entry;
    dir = opendir(".");
    if (dir == NULL) {
        perror("Unable to open directory");
        return 1;
    }
    printf("Contents of the current directory:\n");
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }
    closedir(dir);
    return 0;
}
```





7. Implement the following CPU scheduling algorithms using C:

A. First-Come, First-Served (FCFS)

```
#include <stdio.h
struct Process {
    int pid;        // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};
void fcfs(struct Process p[], int n) {
    p[0].waiting = 0;  // First process has 0 waiting time
    for (int i = 1; i < n; i++) {
        p[i].waiting = p[i - 1].waiting + p[i - 1].burst;
```

```
    }
    for (int i = 0; i < n; i++) {
        p[i].turnaround = p[i].waiting + p[i].burst;
    }
    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += p[i].waiting;
        total_tat += p[i].turnaround;
    }
    printf("Average Waiting Time: %.2f\n", total_wt / n);
    printf("Average Turnaround Time: %.2f\n", total_tat / n);
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Arrival Time and Burst Time for Process %d: ", p[i].pid);
        scanf("%d %d", &p[i].arrival, &p[i].burst);
    }
    fcfs(p, n); // Call FCFS function
    return 0;
}
```

B. Shortest Job First (SJF)

```
#include <stdio.h>
struct Process {
    int pid;        // Process ID
    int burst;      // Burst Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};
void sjf(struct Process p[], int n) {
```

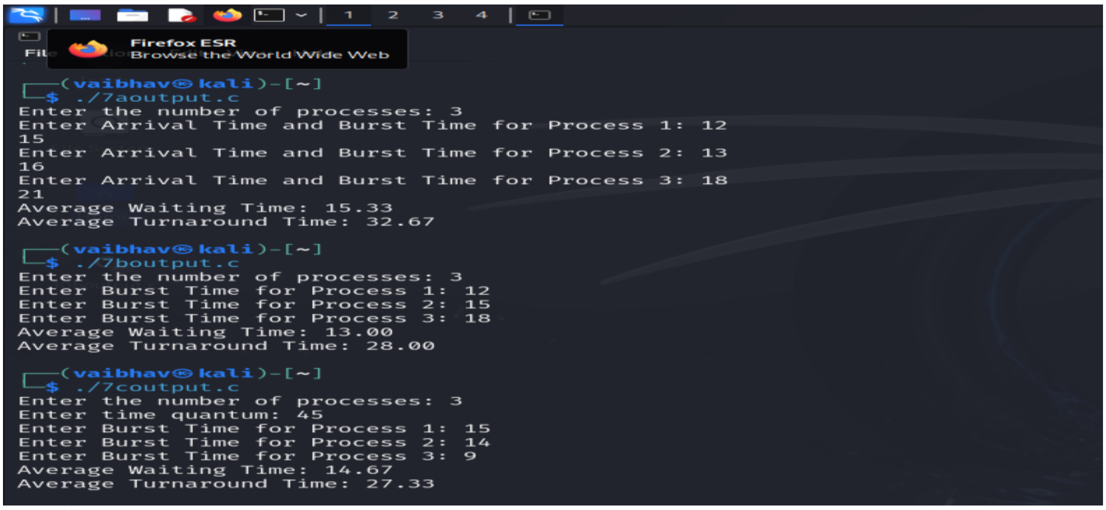
```
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].burst > p[j].burst) {
                struct Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
    p[0].waiting = 0;  // First process has 0 waiting time
    for (int i = 1; i < n; i++) {
        p[i].waiting = p[i - 1].waiting + p[i - 1].burst;
    }
    for (int i = 0; i < n; i++) {
        p[i].turnaround = p[i].waiting + p[i].burst;
    }
    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += p[i].waiting;
        total_tat += p[i].turnaround;
    }
    printf("Average Waiting Time: %.2f\n", total_wt / n);
    printf("Average Turnaround Time: %.2f\n", total_tat / n);
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Burst Time for Process %d: ", p[i].pid);
        scanf("%d", &p[i].burst);
    }
```

```
sjf(p, n); // Call SJF function
    return 0;
}
```

C. Round Robin (RR)

```
#include <stdio.h>
struct Process {
    int pid;        // Process ID
    int burst;      // Burst Time
    int remaining;  // Remaining Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};
void roundRobin(struct Process p[], int n, int quantum) {
    int time = 0;
    int remaining_processes = n;
    while (remaining_processes > 0) {
        for (int i = 0; i < n; i++) {
            if (p[i].remaining > 0) {
                if (p[i].remaining > quantum) {
                    time += quantum;
                    p[i].remaining -= quantum;
                } else {
                    time += p[i].remaining;
                    p[i].waiting = time - p[i].burst;
                    p[i].turnaround = time;
                    p[i].remaining = 0;
                    remaining_processes--;
                }
            }
        }
    }
    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += p[i].waiting;
```

```
total_tat += p[i].turnaround;
  }
  printf("Average Waiting Time: %.2f\n", total_wt / n);
  printf("Average Turnaround Time: %.2f\n", total_tat / n);
}
int main() {
  int n, quantum;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  printf("Enter time quantum: ");
  scanf("%d", &quantum);
  struct Process p[n];
  for (int i = 0; i < n; i++) {
    p[i].pid = i + 1;
    printf("Enter Burst Time for Process %d: ", p[i].pid);
    scanf("%d", &p[i].burst);
    p[i].remaining = p[i].burst; // Initialize remaining time
  }
  roundRobin(p, n, quantum); // Call Round Robin function
  return 0;
}
```



8. Deadlock Detection and Deadlock Avoidance

A) Write a program to simulate the deadlock detection algorithm for a set of processes and resources.

```
#include <stdio.h>
#include <stdbool.h>
#define P 5  // Number of processes
#define R 3  // Number of resources
bool isDeadlocked(int alloc[][R], int max[][R], int avail[], int n, int m) {
    int finish[n];
    int work[m];
    for (int i = 0; i < n; i++) {
        finish[i] = 0;  // Initially, no process has finished
    }
    for (int i = 0; i < m; i++) {
        work[i] = avail[i];  // Available resources
    }
    for (int count = 0; count < n; count++) {
        bool progressMade = false;
        for (int i = 0; i < n; i++) {
            if (finish[i] == 0) {  // If process i is not finished
                bool canFinish = true;
                for (int j = 0; j < m; j++) {
                    if (max[i][j] - alloc[i][j] > work[j]) {
                        canFinish = false;
                        break;
                    }
                }
                if (canFinish) {
                    for (int j = 0; j < m; j++) {
                        work[j] += alloc[i][j];
                    }
                    finish[i] = 1;  // Mark process i as finished
                    progressMade = true;
                    break;
                }
            }
```

```
        }
        if (!progressMade) {
            printf("Deadlock detected!\n");
            return true;
        }
    }
    printf("No deadlock detected.\n");
    return false;
}
int main() {
    int alloc[P][R] = {{0, 1, 0},
                       {2, 0, 0},
                       {3, 0, 2},
                       {2, 1, 1},
                       {0, 0, 2}};

    int max[P][R] = {{7, 5, 3},
                     {3, 2, 2},
                     {9, 0, 2},
                     {2, 2, 2},
                     {4, 3, 3}};
    int avail[R] = {3, 3, 2};
    isDeadlocked(alloc, max, avail, P, R);
    return 0;
}
```

B. Implement the Banker's algorithm for deadlock avoidance in a multi-process system.

```
#include <stdio.h>
#include <stdbool.h>
#define P 5  // Number of processes
#define R 3  // Number of resources
bool isSafeState(int alloc[][R], int max[][R], int avail[], int n, int m) {
    int need[n][m];
    int work[m];
```

```
bool finish[n];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        need[i][j] = max[i][j] - alloc[i][j];
    }
}
for (int i = 0; i < m; i++) {
    work[i] = avail[i];  // Available resources
}
for (int i = 0; i < n; i++) {
    finish[i] = false;  // Initially, no process is finished
}
int count = 0;
while (count < n) {
    bool progressMade = false;
    for (int i = 0; i < n; i++) {
        if (!finish[i]) {
            bool canFinish = true;
            for (int j = 0; j < m; j++) {
                if (need[i][j] > work[j]) {
                    canFinish = false;
                    break;
                }
            }
            if (canFinish) {
                // Add the allocated resources of process i to work[]
                for (int j = 0; j < m; j++) {
                    work[j] += alloc[i][j];
                }
                finish[i] = true;  // Mark process i as finished
                progressMade = true;
                count++;
                break;
```

```c
            }
        }
    }
    if (!progressMade) {
        printf("System is in an unsafe state!\n");
        return false;
    }
  }
  printf("System is in a safe state!\n");
  return true;
}
int main() {
  int alloc[P][R] = {{0, 1, 0},
                     {2, 0, 0},
                     {3, 0, 2},
                     {2, 1, 1},
                     {0, 0, 2}};
  int max[P][R] = {{7, 5, 3},
                   {3, 2, 2},
                   {9, 0, 2},
                   {2, 2, 2},
                   {4, 3, 3}};
  int avail[R] = {3, 3, 2};  // Available resources
  isSafeState(alloc, max, avail, P, R);
  return 0;
}
```

```
┌──(vaibhav㉿kali)-[~]
└─$ ./8aoutput.c
No deadlock detected.

┌──(vaibhav㉿kali)-[~]
└─$ ./8boutput.c
System is in a safe state!
```

9. Page Replacement Algorithms:

}

A) Implement the Least Recently Used (LRU) page replacement algorithm.

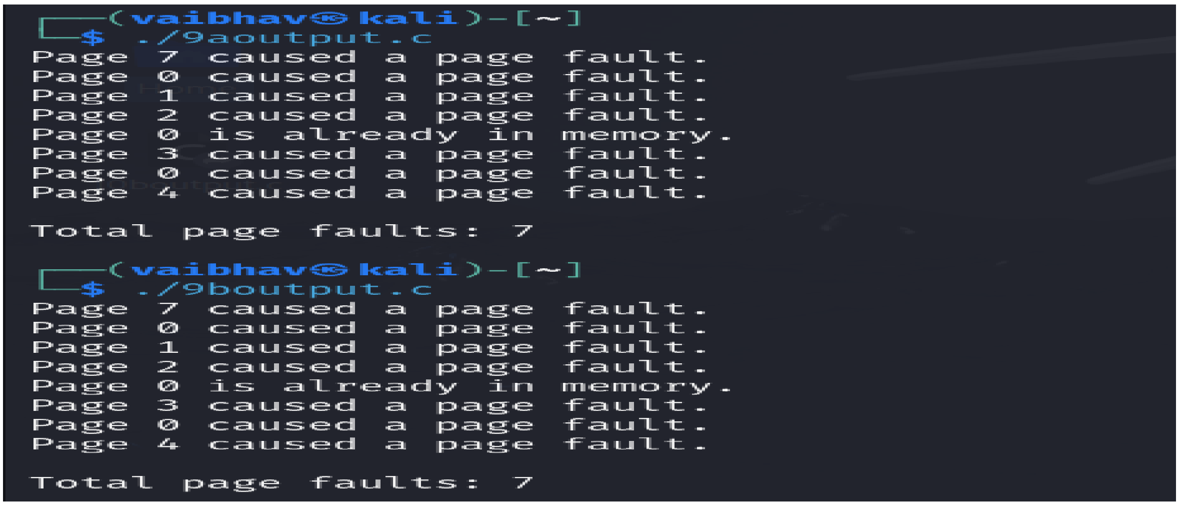
```
#include <stdio.h>
#define MAX_FRAMES 3  // Number of frames in memory
int isPageInMemory(int frames[], int page) {
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (frames[i] == page) {
            return 1;  // Page is in memory
        }
    }
    return 0;  // Page is not in memory
}
void lru(int pages[], int numPages) {
    int frames[MAX_FRAMES] = {-1, -1, -1};  // Initialize memory frames to -1 (empty)
    int pageFaults = 0;
    for (int i = 0; i < numPages; i++) {
        int page = pages[i];
        if (isPageInMemory(frames, page)) {
            printf("Page %d is already in memory.\n", page);
            continue;
        }
        pageFaults++;
        printf("Page %d caused a page fault.\n", page);
        for (int j = 0; j < MAX_FRAMES - 1; j++) {
            frames[j] = frames[j + 1];  // Shift pages to the left
        }
        frames[MAX_FRAMES - 1] = page;
    }
    printf("\nTotal page faults: %d\n", pageFaults);
}
int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4};  // Example page reference string
    int numPages = sizeof(pages) / sizeof(pages[0]);
```

```
    lru(pages, numPages);  // Call the LRU page replacement function
    return 0;
}
```

B. Write a program to simulate the First-In-First-Out (FIFO) page replacement algorithm.

```
#include <stdio.h>
#define MAX_PAGES 100
#define MAX_FRAMES 10
int pages[MAX_PAGES];  // Array to hold pages
int pageFrames[MAX_FRAMES];  // Array to hold page frames
int frameCount, pageCount;
void simulateFIFO() {
    int hits = 0, faults = 0;
    int nextFrame = 0; // To keep track of the next frame to replace
    for (int i = 0; i < pageCount; i++) {
        int j;
        for (j = 0; j < frameCount; j++) {
            if (pageFrames[j] == pages[i]) {
                hits++;
                break;
            }
        }
        if (j == frameCount) {
            faults++;
            pageFrames[nextFrame] = pages[i]; // Replace the page in the next frame
            nextFrame = (nextFrame + 1) % frameCount; // Move to the next frame
        }
    }
    printf("FIFO Page Faults: %d\n", faults);
    printf("FIFO Page Hits: %d\n", hits);
}
int main() {
    frameCount = 3; // Number of page frames
    pageCount = 12; // Number of pages
```

```
    int inputPages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3};
    for (int i = 0; i < pageCount; i++) {
        pages[i] = inputPages[i];
    }
    for (int i = 0; i < frameCount; i++) {
        pageFrames[i] = -1;
    }
    simulateFIFO();
    return 0;
}
```



10. Disk Scheduling Algorithms: Write a program to implement the following disk scheduling algorithms:

A) First-Come, First-Served (FCFS)

```
#include <stdio.h>
#include <stdlib.h>
void fcfs(int requests[], int n, int start) {
    int totalSeekTime = 0;
    int current = start;
    printf("FCFS Disk Scheduling\n");
    printf("Disk head starts at position %d\n", start);
    for (int i = 0; i < n; i++) {
        int seekTime = abs(requests[i] - current);
        totalSeekTime += seekTime;
```

```
        printf("Move from %d to %d, Seek time = %d\n", current, requests[i], seekTime);
        current = requests[i];
    }
    printf("\nTotal Seek Time = %d\n", totalSeekTime);
}
int main() {
    int requests[] = {55, 58, 60, 98, 70, 43, 84, 20, 15};
    int n = sizeof(requests) / sizeof(requests[0]);
    int start = 50;  // Disk head starting position
    fcfs(requests, n, start);
    return 0;
}
```

B. Shortest Seek Time First (SSTF) Disk Scheduling Algorithm

```
#include <stdio.h>
#include <stdlib.h>
int findClosestRequest(int requests[], int n, int current) {
    int minDistance = abs(requests[0] - current);
    int index = 0;
    for (int i = 1; i < n; i++) {
        int distance = abs(requests[i] - current);
        if (distance < minDistance) {
            minDistance = distance;
            index = i;
        }
    }
    return index;
}
void sstf(int requests[], int n, int start) {
    int totalSeekTime = 0;
    int current = start;
    int visited[n];
    for (int i = 0; i < n; i++) visited[i] = 0;
    printf("SSTF Disk Scheduling\n");
```

```
    printf("Disk head starts at position %d\n", start);
    for (int i = 0; i < n; i++) {
        int index = findClosestRequest(requests, n, current);
        int seekTime = abs(requests[index] - current);
        totalSeekTime += seekTime;
        printf("Move from %d to %d, Seek time = %d\n", current, requests[index], seekTime);
        current = requests[index];
        visited[index] = 1;
    }
    printf("\nTotal Seek Time = %d\n", totalSeekTime);
}
int main() {
    int requests[] = {55, 58, 60, 98, 70, 43, 84, 20, 15};
    int n = sizeof(requests) / sizeof(requests[0]);
    int start = 50;  // Disk head starting position
    sstf(requests, n, start);
    return 0;
}
```

C. Elevator (SCAN) Disk Scheduling Algorithm

```
#include <stdio.h>
#include <stdlib.h>
void scan(int requests[], int n, int start, int direction) {
    int totalSeekTime = 0;
    int current = start;
    int sortedRequests[n];
    for (int i = 0; i < n; i++) sortedRequests[i] = requests[i];
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (sortedRequests[i] > sortedRequests[j]) {
                int temp = sortedRequests[i];
                sortedRequests[i] = sortedRequests[j];
                sortedRequests[j] = temp;
```

```c
            }
        }
    }
    printf("SCAN Disk Scheduling\n");
    printf("Disk head starts at position %d\n", start);
    if (direction == 1) {  // Move right
        for (int i = 0; i < n; i++) {
            if (sortedRequests[i] >= start) {
                int seekTime = abs(sortedRequests[i] - current);
                totalSeekTime += seekTime;
                printf("Move from %d to %d, Seek time = %d\n", current, sortedRequests[i],
seekTime);
                current = sortedRequests[i];
            }
        }
        for (int i = n - 1; i >= 0; i--) {
            if (sortedRequests[i] < start) {
                int seekTime = abs(sortedRequests[i] - current);
                totalSeekTime += seekTime;
                printf("Move from %d to %d, Seek time = %d\n", current, sortedRequests[i],
seekTime);
                current = sortedRequests[i];
            }
        }
    } else {  // Move left
        for (int i = n - 1; i >= 0; i--) {
            if (sortedRequests[i] <= start) {
                int seekTime = abs(sortedRequests[i] - current);
                totalSeekTime += seekTime;
                printf("Move from %d to %d, Seek time = %d\n", current, sortedRequests[i],
seekTime);
                current = sortedRequests[i];
            }
        }
```

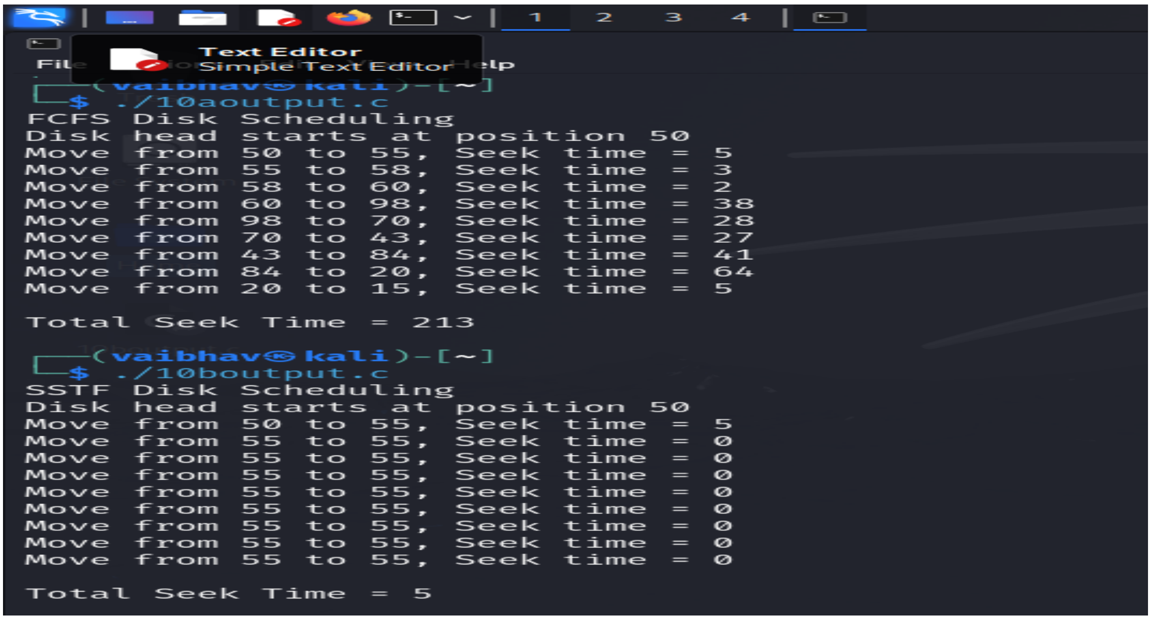
```
        for (int i = 0; i < n; i++) {
            if (sortedRequests[i] > start) {
                int seekTime = abs(sortedRequests[i] - current);
                totalSeekTime += seekTime;
                printf("Move from %d to %d, Seek time = %d\n", current, sortedRequests[i], seekTime);
                current = sortedRequests[i];
            }
        }
    }
    printf("\nTotal Seek Time = %d\n", totalSeekTime);
}
int main() {
    int requests[] = {55, 58, 60, 98, 70, 43, 84, 20, 15};
    int n = sizeof(requests) / sizeof(requests[0]);
    int start = 50;  // Disk head starting position
    int direction = 1;  // 1 for right, 0 for left
    scan(requests, n, start, direction);
    return 0;
}
```

D. Circular SCAN (C-SCAN) Disk Scheduling Algorithm

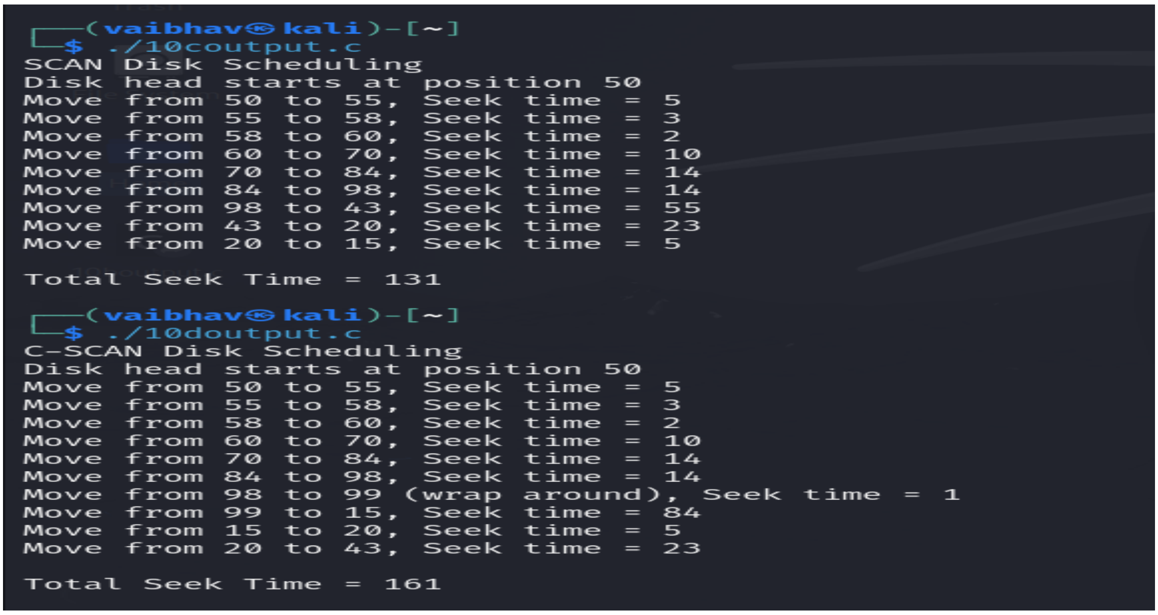
```
#include <stdio.h>
#include <stdlib.h>
void cscan(int requests[], int n, int start, int totalTracks) {
    int totalSeekTime = 0;
    int current = start;
    int sortedRequests[n];
    for (int i = 0; i < n; i++) sortedRequests[i] = requests[i];
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (sortedRequests[i] > sortedRequests[j]) {
                int temp = sortedRequests[i];
```

```
                sortedRequests[i] = sortedRequests[j];
                sortedRequests[j] = temp;
            }
        }
    }
    printf("C-SCAN Disk Scheduling\n");
    printf("Disk head starts at position %d\n", start);
    int i;
    for (i = 0; i < n; i++) {
        if (sortedRequests[i] >= start) {
            break;
        }
    }
    for (int j = i; j < n; j++) {
        int seekTime = abs(sortedRequests[j] - current);
        totalSeekTime += seekTime;
        printf("Move from %d to %d, Seek time = %d\n", current, sortedRequests[j], seekTime);
        current = sortedRequests[j];
    }
    if (current != totalTracks - 1) {
        totalSeekTime += abs(totalTracks - 1 - current);
        printf("Move from %d to %d (wrap around), Seek time = %d\n", current, totalTracks - 1,
abs(totalTracks - 1 - current));
        current = totalTracks - 1;
    }
    for (int j = 0; j < i; j++) {
        int seekTime = abs(sortedRequests[j] - current);
        totalSeekTime += seekTime;
        printf("Move from %d to %d, Seek time = %d\n", current, sortedRequests[j], seekTime);
        current = sortedRequests[j];
    }
    printf("\nTotal Seek Time = %d\n", totalSeekTime);
}
```

```c
int main() {
    int requests[] = {55, 58, 60, 98, 70, 43, 84, 20, 15};
    int n = sizeof(requests) / sizeof(requests[0]);
    int start = 50;  // Disk head starting position
    int totalTracks = 100;  // Total number of tracks


    cscan(requests, n, start, totalTracks);
    return 0;
}
```



```
┌──(vaibhav㉿kali)-[~]
└─$ ./10aoutput.c
FCFS Disk Scheduling
Disk head starts at position 50
Move from 50 to 55, Seek time = 5
Move from 55 to 58, Seek time = 3
Move from 58 to 60, Seek time = 2
Move from 60 to 98, Seek time = 38
Move from 98 to 70, Seek time = 28
Move from 70 to 43, Seek time = 27
Move from 43 to 84, Seek time = 41
Move from 84 to 20, Seek time = 64
Move from 20 to 15, Seek time = 5

Total Seek Time = 213
┌──(vaibhav㉿kali)-[~]
└─$ ./10boutput.c
SSTF Disk Scheduling
Disk head starts at position 50
Move from 50 to 55, Seek time = 5
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0
Move from 55 to 55, Seek time = 0

Total Seek Time = 5
```



```
┌──(vaibhav㉿kali)-[~]
└─$ ./10coutput.c
SCAN Disk Scheduling
Disk head starts at position 50
Move from 50 to 55, Seek time = 5
Move from 55 to 58, Seek time = 3
Move from 58 to 60, Seek time = 2
Move from 60 to 70, Seek time = 10
Move from 70 to 84, Seek time = 14
Move from 84 to 98, Seek time = 14
Move from 98 to 43, Seek time = 55
Move from 43 to 20, Seek time = 23
Move from 20 to 15, Seek time = 5

Total Seek Time = 131
┌──(vaibhav㉿kali)-[~]
└─$ ./10doutput.c
C-SCAN Disk Scheduling
Disk head starts at position 50
Move from 50 to 55, Seek time = 5
Move from 55 to 58, Seek time = 3
Move from 58 to 60, Seek time = 2
Move from 60 to 70, Seek time = 10
Move from 70 to 84, Seek time = 14
Move from 84 to 98, Seek time = 14
Move from 98 to 99 (wrap around), Seek time = 1
Move from 99 to 15, Seek time = 84
Move from 15 to 20, Seek time = 5
Move from 20 to 43, Seek time = 23

Total Seek Time = 161
```