LAB MANUAL

OF

314446: OPERATING SYSTEM

TE IT (2019 PATTERN)

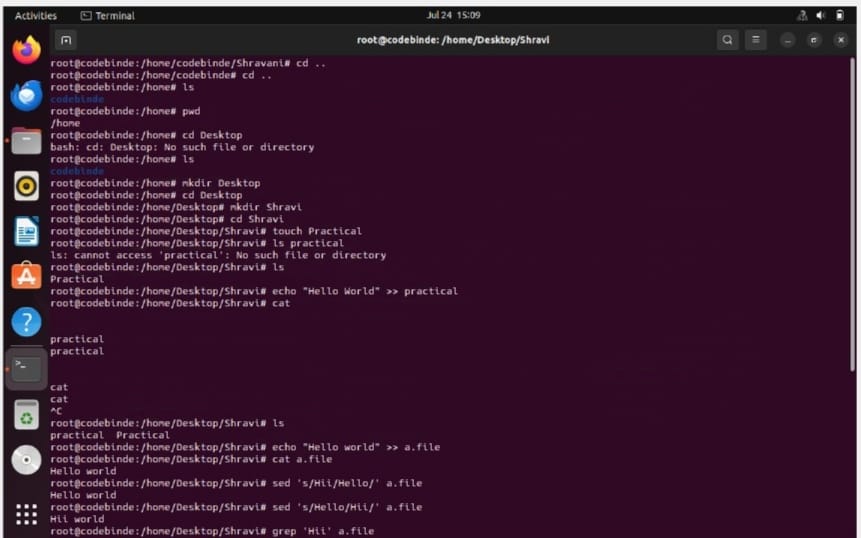
COURSE: IT ENGINEERING

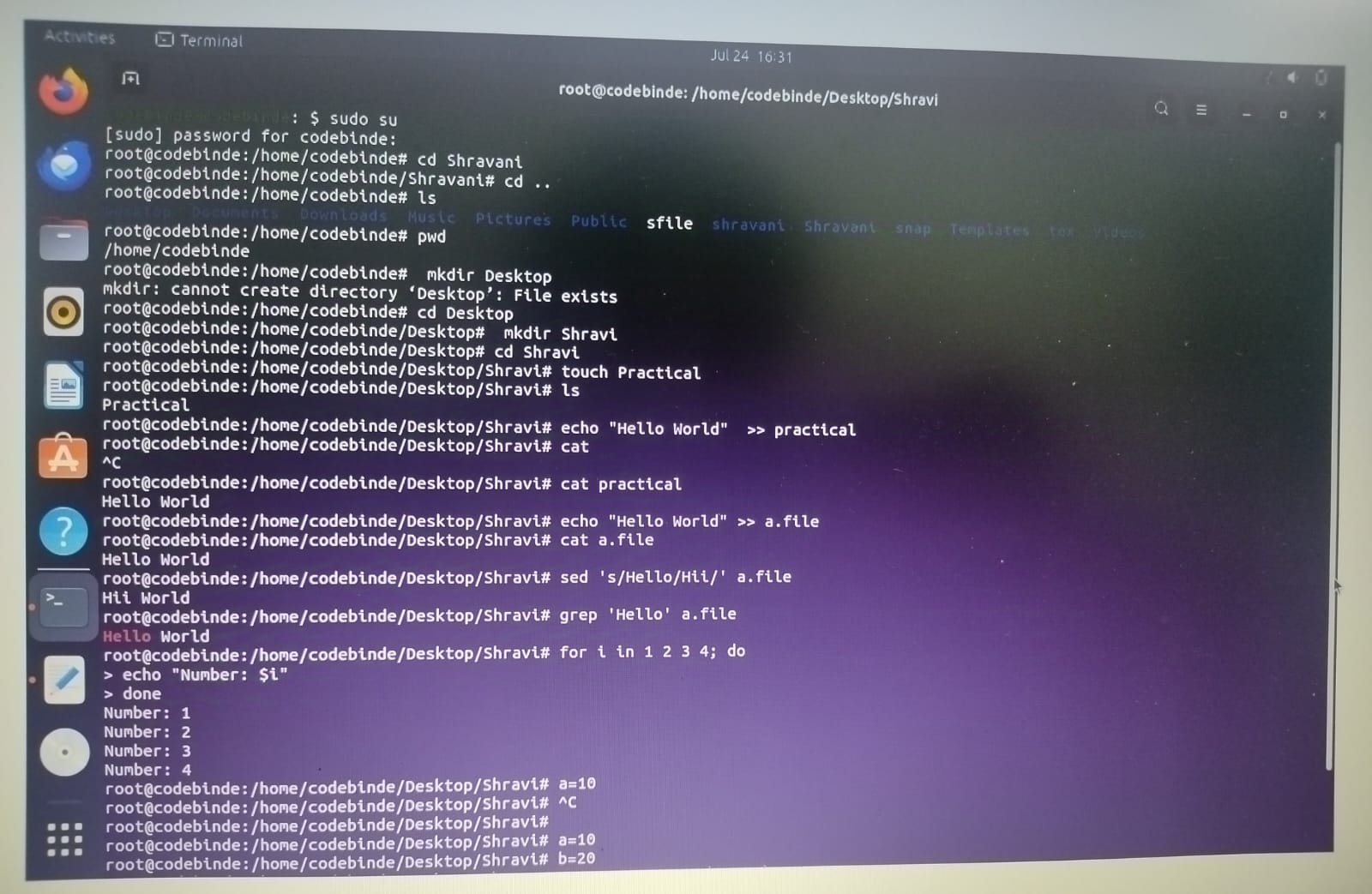
# Assignment No. 1

1. Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed etc.
2. Write a program to implement an address book with options given below:

a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit

# PART A





------------------------------------------------------------------------------------

# PART B

#!/bin/bash

# Function to create records in a speci ied text ile create() { echo "----- Welcome to the Address Book Application! -----" echo "" echo "Enter the name of the address book text ile (without extension): " read address\_book\_name address\_book\_ ile="${address\_book\_name}.txt"

# Check if ile already exists or create new if [ ! -f "$address\_book\_ ile" ]; then touch "$address\_book\_ ile" echo "Created new address book: $address\_book\_ ile" else

echo "Using existing address book: $address\_book\_ ile" i

echo "" echo "Enter the number of records you want to store: " read rec

record\_count=1 while [ $rec -ne 0 ]; do

echo ""

echo "Enter id, name, address, phone for record $record\_count: " read id name address phone

# Check if ID already exists if grep -q "^$id," "$address\_book\_ ile"; then echo ""

echo "ID $id already exists. Please enter a unique ID." else

echo "$id, $name, $address, $phone" >> "$address\_book\_ ile" rec=$((rec - 1)) record\_count=$((record\_count + 1)) i

done

echo "" echo "Records created successfully in $address\_book\_ ile!"

}

# Function to display records in tabular format display\_records() { echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" echo " ID | Name | Address | Phone" echo " --------------------" column -s ',' -t < "$address\_book\_ ile"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

}

# Function to search for a record by name

search() {

echo "" echo "Enter the name you want to search in records: " read name

if [ -f "$address\_book\_ ile" ]; then result=$(grep -i "$name" "$address\_book\_ ile") if [ -n "$result" ]; then

echo ""

echo " ==> $name found in the database!"

echo "$result"

else

echo ""

echo " No match found" i else

echo ""

echo "Address book ile not found!" i }

# Function to delete a record by name

delete() { echo "" echo "Enter the name you want to delete the record of: " read name

if [ -f "$address\_book\_ ile" ]; then result=$(grep -i "$name" "$address\_book\_ ile") if [ -z "$result" ]; then

echo ""

echo " No matches found" else

grep -iv "$name" "$address\_book\_ ile" > temp.txt && mv temp.txt "$address\_book\_ ile" echo ""

echo " Deletion done successfully!" i else

echo ""

echo "Address book ile not found!" i }

# Function to modify a record by name modify() { echo "" echo " Enter the name you want to modify: " read name

if [ -f "$address\_book\_ ile" ]; then result=$(grep -i "$name" "$address\_book\_ ile") if [ -z "$result" ]; then

echo ""

echo " No matches found" else echo ""

echo "Enter the new record (id, name, address, phone): " read new\_id new\_name new\_address new\_phone new\_record="$new\_id, $new\_name, $new\_address, $new\_phone" sed -i "s/$result/$new\_record/" "$address\_book\_ ile" echo ""

echo "Modi ication done successfully!" i else

echo ""

echo "Address book ile not found!" i }

# Main menu while true; do echo ""

echo " 1) Create record" echo " 2) Display records"

echo " 3) Search record" echo " 4) Delete record" echo " 5) Modify record"

echo " 6) Exit" echo "" echo -n " Enter your choice: "

read ch echo "" case $ch in

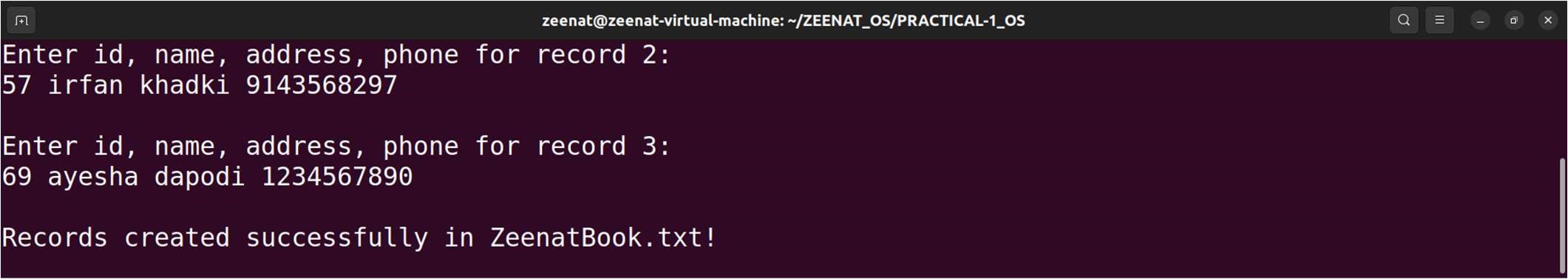
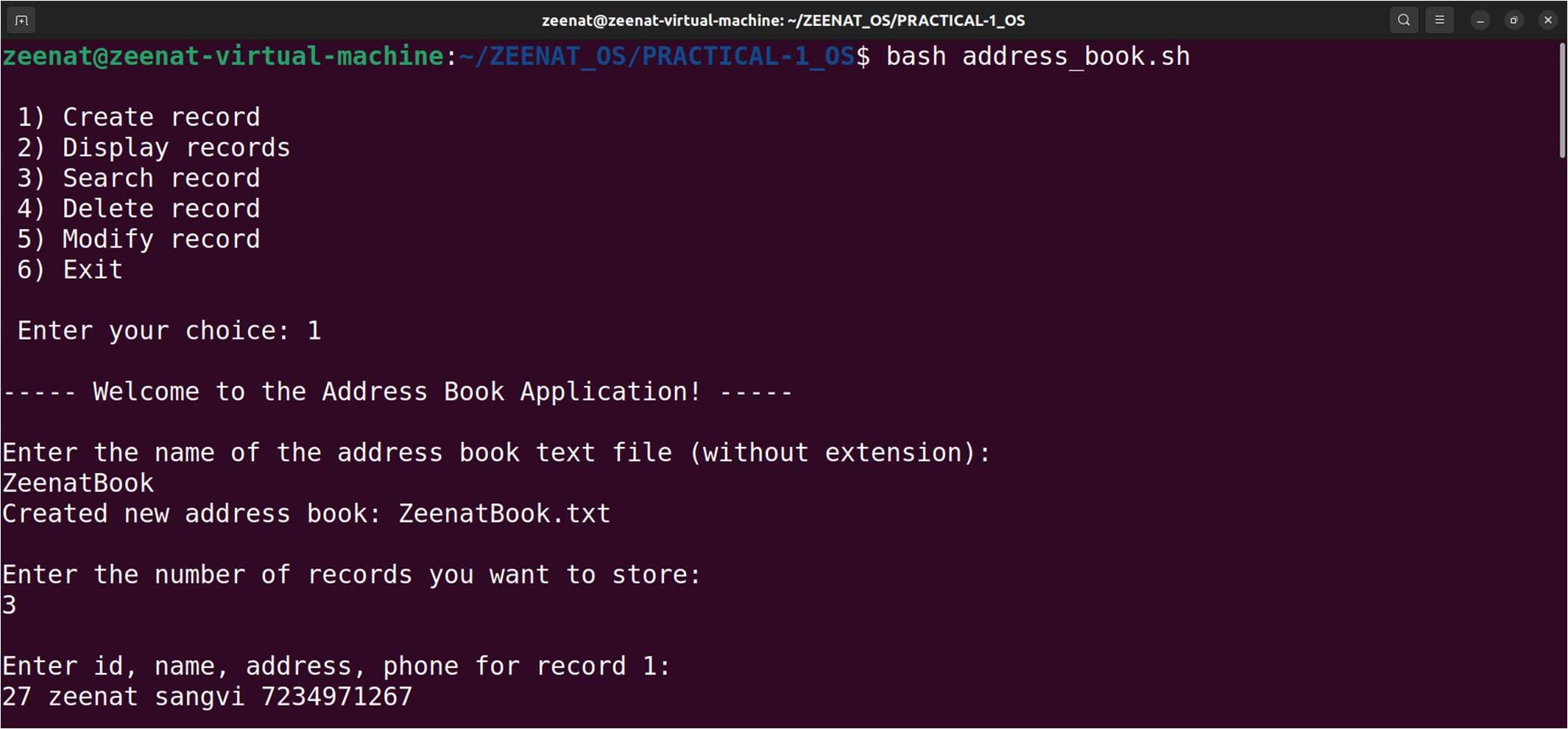
1. create ;;
2. display\_records ;;
3. search ;;
4. delete ;;
5. modify ;;
6. exit 0 ;;

\*) echo " Invalid option. Please try again." ;; esac

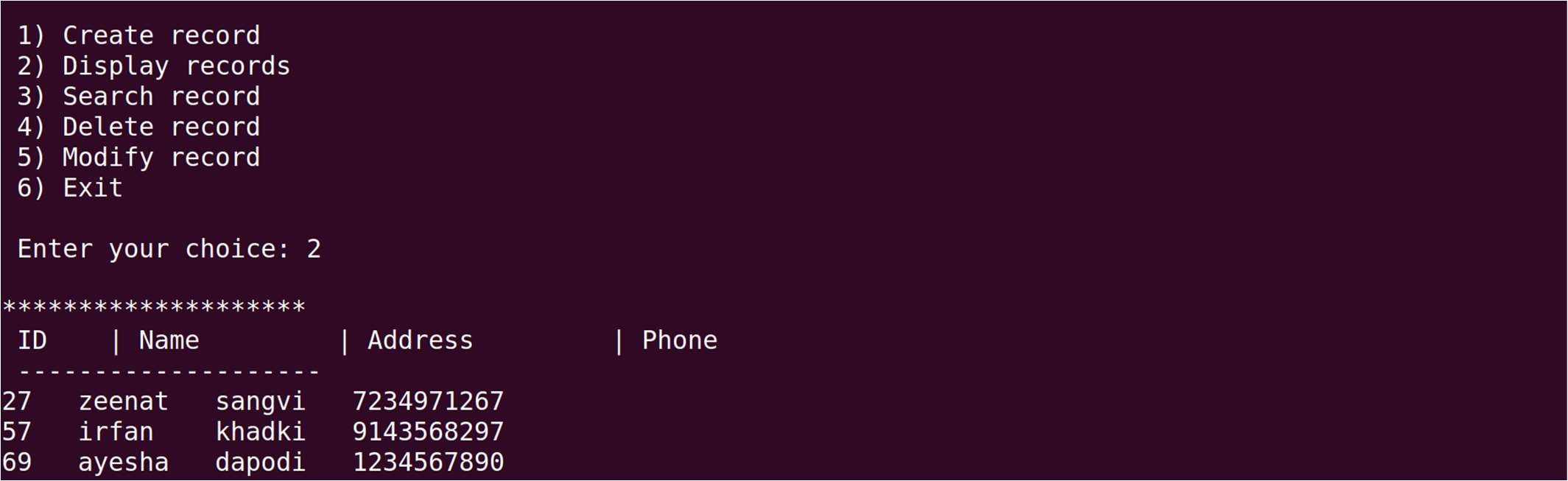
echo "" done

# OUTPUT: PART B

1. CREATE RECORD



1. DISPLAY RECORDS



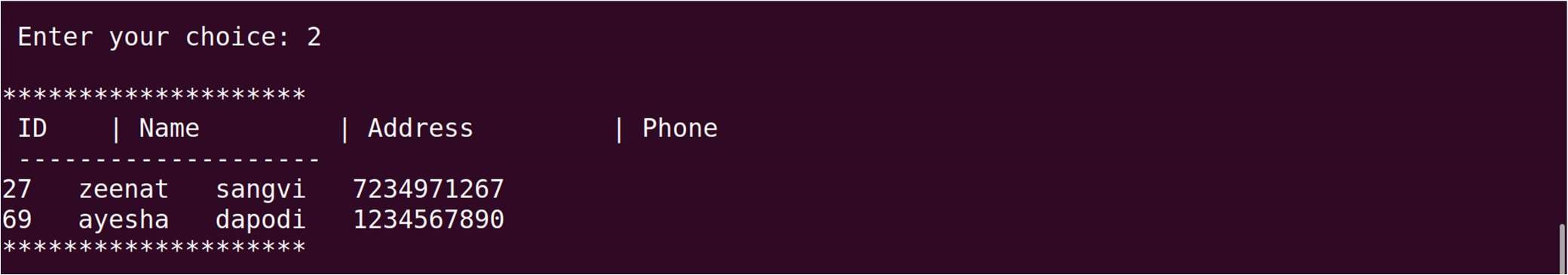
1. SEARCH RECORD



1. DELETE RECORD



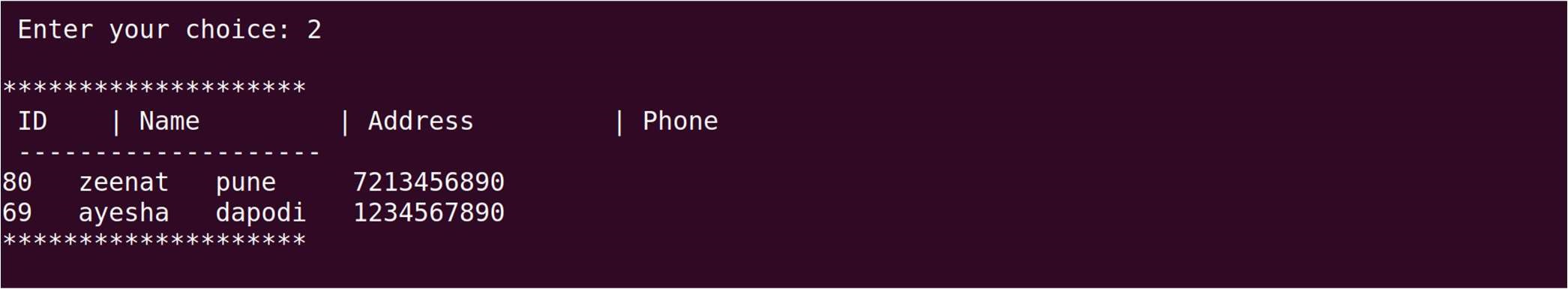
After delete operation check by displaying its successfully deleted or not



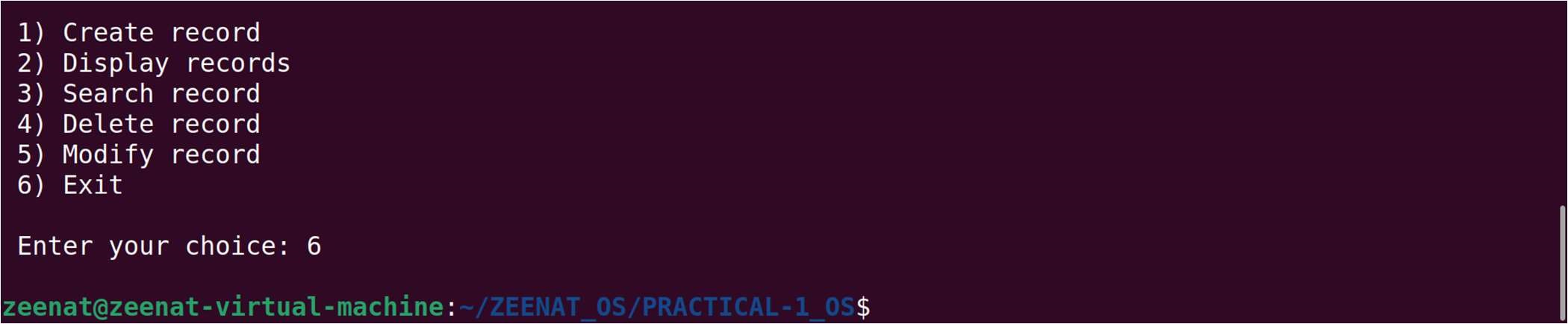
1. MODIFY RECORD



After modify operation check by displaying its successfully modified or not



1. EXIT



## Assignment No. 2

Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.

1. Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

1. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

# PART A

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

// Function to perform Bubble Sort

void bubbleSort(int arr[], int n) { int i, j, temp; for (i = 0; i < n-1; i++) { for (j = 0; j < n-i-1; j++) { if (arr[j] > arr[j+1]) { temp = arr[j]; arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

// Function to perform Insertion Sort

void insertionSort(int arr[], int n) {

int i, key, j; for (i = 1; i < n; i++) { key = arr[i]; j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j]; j = j - 1;

}

arr[j + 1] = key;

}

}

// Function to print the array

void printArray(int arr[], int n) { int i;

for (i = 0; i < n; i++) printf("%d ", arr[i]); printf("\n");

}

int main() { pid\_t pid; int status;

int n, i;

// Accepting input

printf("Enter the number of integers: "); scanf("%d", &n);

int arr[n];

printf("Enter the integers:\n");

for (i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

// Demonstrating Zombie State printf("\n--- Demonstrating Zombie State ---\n");

pid = fork();

if (pid < 0) { // Fork failed perror("fork"); exit(EXIT\_FAILURE); } else if (pid == 0) { // Child process

printf("Child process (PID: %d) is sorting the array for zombie state.\n", getpid()); bubbleSort(arr, n); printf("Child process sorted array:\n");

printArray(arr, n);

// Simulate some processing time printf("Child process will now sleep for 30 seconds...\n"); sleep(30);

printf("Child process is exiting now.\n"); exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process (PID: %d) is sorting the array for zombie state.\n", getpid()); bubbleSort(arr, n); printf("Parent process sorted array:\n");

printArray(arr, n);

// Wait before checking zombie state

sleep(10); // Ensure that child process has time to become a zombie

// Add space before checking zombie state

printf("\nChecking if the child process is a zombie...\n"); if (waitpid(pid, &status, WNOHANG) == 0) { printf("The child process is still alive and in a zombie state if parent exits next.\n");

} else {

printf("The child process is not a zombie.\n");

}

// Exit to observe the zombie state

printf("Parent process is exiting now. The child process should be a zombie.\n"); sleep(10); // Wait to observe the zombie state

}

// Clear screen

printf("\n--- Clearing Screen ---\n"); sleep(5); // Simulate some delay before the orphan demonstration

// Demonstrating Orphan State printf("\n--- Demonstrating Orphan State ---\n");

pid = fork();

if (pid < 0) { // Fork failed perror("fork"); exit(EXIT\_FAILURE); } else if (pid == 0) { // Child process printf("Child process (PID: %d) is sorting the array for orphan state.\n", getpid()); insertionSort(arr, n); printf("Child process sorted array:\n");

printArray(arr, n);

// Simulate some processing time printf("Child process will now sleep for 30 seconds...\n"); sleep(30);

printf("Child process is exiting now.\n"); exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process (PID: %d) is sorting the array for orphan state.\n", getpid());

insertionSort(arr, n); printf("Parent process sorted array:\n"); printArray(arr, n);

// Sleep before the parent process exits

printf("Parent process will now sleep for 5 seconds before exiting...\n"); sleep(5);

// Add space before checking orphan state

printf("\nChecking if the child process has become an orphan...\n");

// Print statement to indicate parent exit and orphan state

printf("Parent process is exiting now. The child process will become an orphan.\n");

// Check if child is still running (to con irm orphan state)

sleep(1); // Allow a moment for the child process to become an orphan if (waitpid(pid, &status, WNOHANG) == 0) { printf("The child process is still running, indicating it has become an orphan.\n");

} else {

printf("The child process is not running; it might have inished already.\n");

}

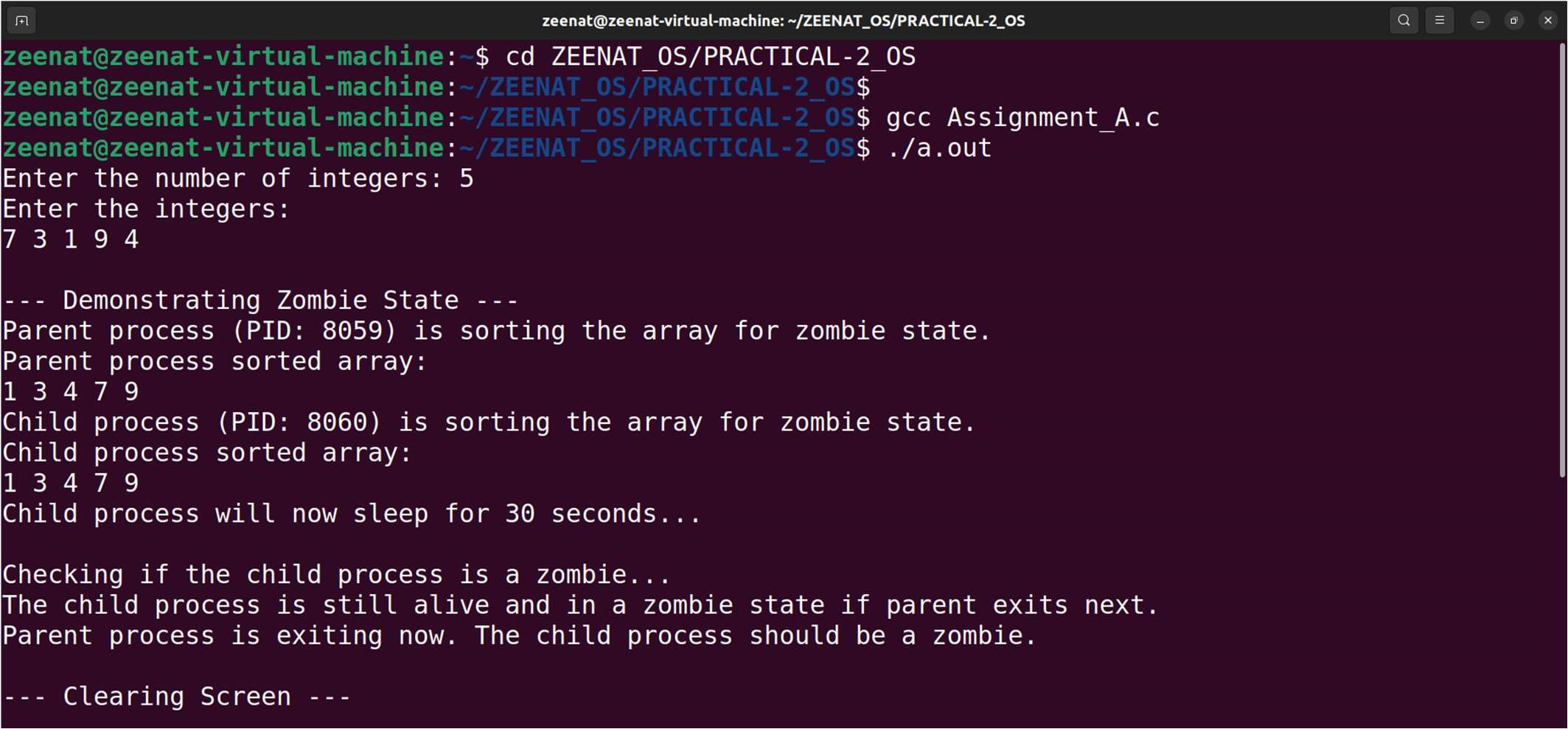
exit(EXIT\_SUCCESS);

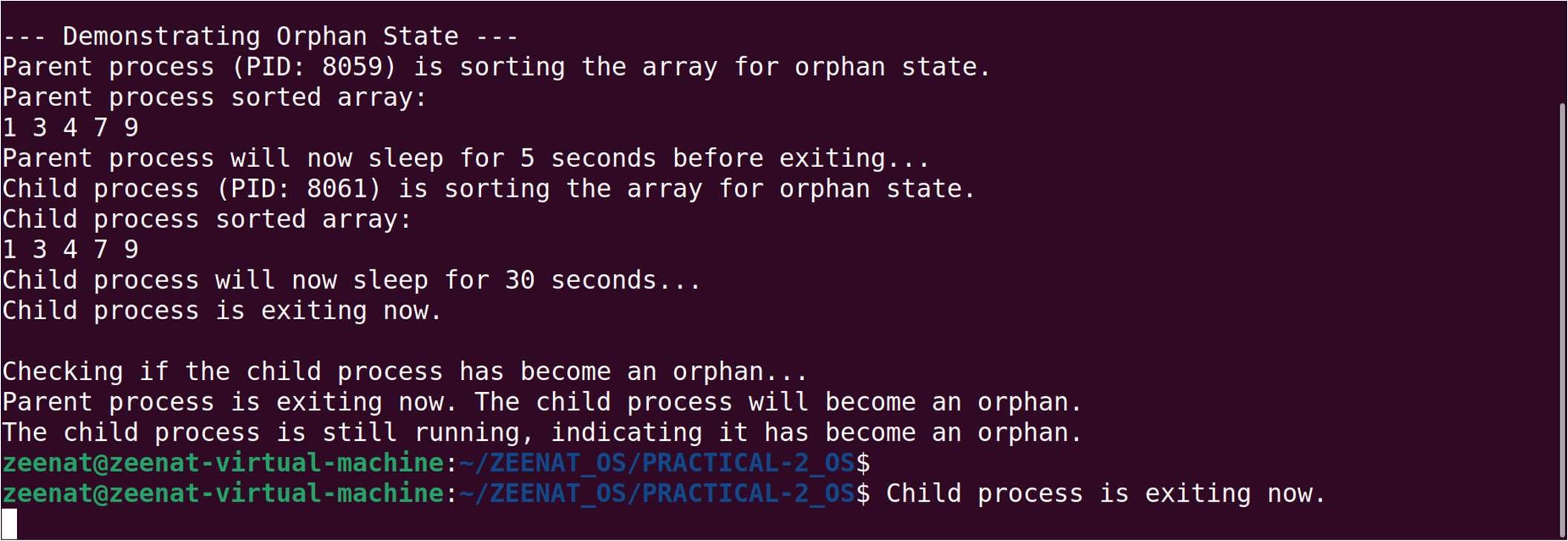
}

return 0;

}

# OUTPUT: PART A





## PART B: File 1 – parent.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#de ine MAX\_SIZE 100

void bubbleSort(int arr[], int n) { for (int i = 0; i < n-1; i++) { for (int j = 0; j < n-i-1; j++) { if (arr[j] > arr[j+1]) { int temp = arr[j]; arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

int main(int argc, char \*argv[]) { int arr[MAX\_SIZE]; int n;

printf("Enter the number of elements: "); scanf("%d", &n); printf("Enter the elements:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

// Sort the array bubbleSort(arr, n);

// Prepare arguments for execve char \*args[MAX\_SIZE + 2];

args[0] = "./child"; for (int i = 0; i < n; i++) { args[i + 1] = (char \*)malloc(10 \* sizeof(char)); sprintf(args[i + 1], "%d", arr[i]);

}

args[n + 1] = NULL;

// Fork the process pid\_t pid = fork(); if (pid < 0) { perror("Fork failed"); exit(1);

}

if (pid == 0) { // Child process printf("\nChild process (PID: %d) is executing the child program.\n", getpid()); execve("./child", args, NULL);

perror("execve failed"); exit(1);

} else {

// Parent process

printf("\nParent process (PID: %d) is sorting the array and will execute the child process.\n", getpid()); wait(NULL); printf("\nChild process completed.\n");

// Free allocated memory for (int i = 0; i < n; i++) {

free(args[i + 1]);

}

}

return 0;

}

## PART B: File 2 – child.c

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

printf("\nArray in reverse order:\n");

// Display the array in reverse order for (int i = argc - 1; i > 0; i--) {

printf("%s ", argv[i]);

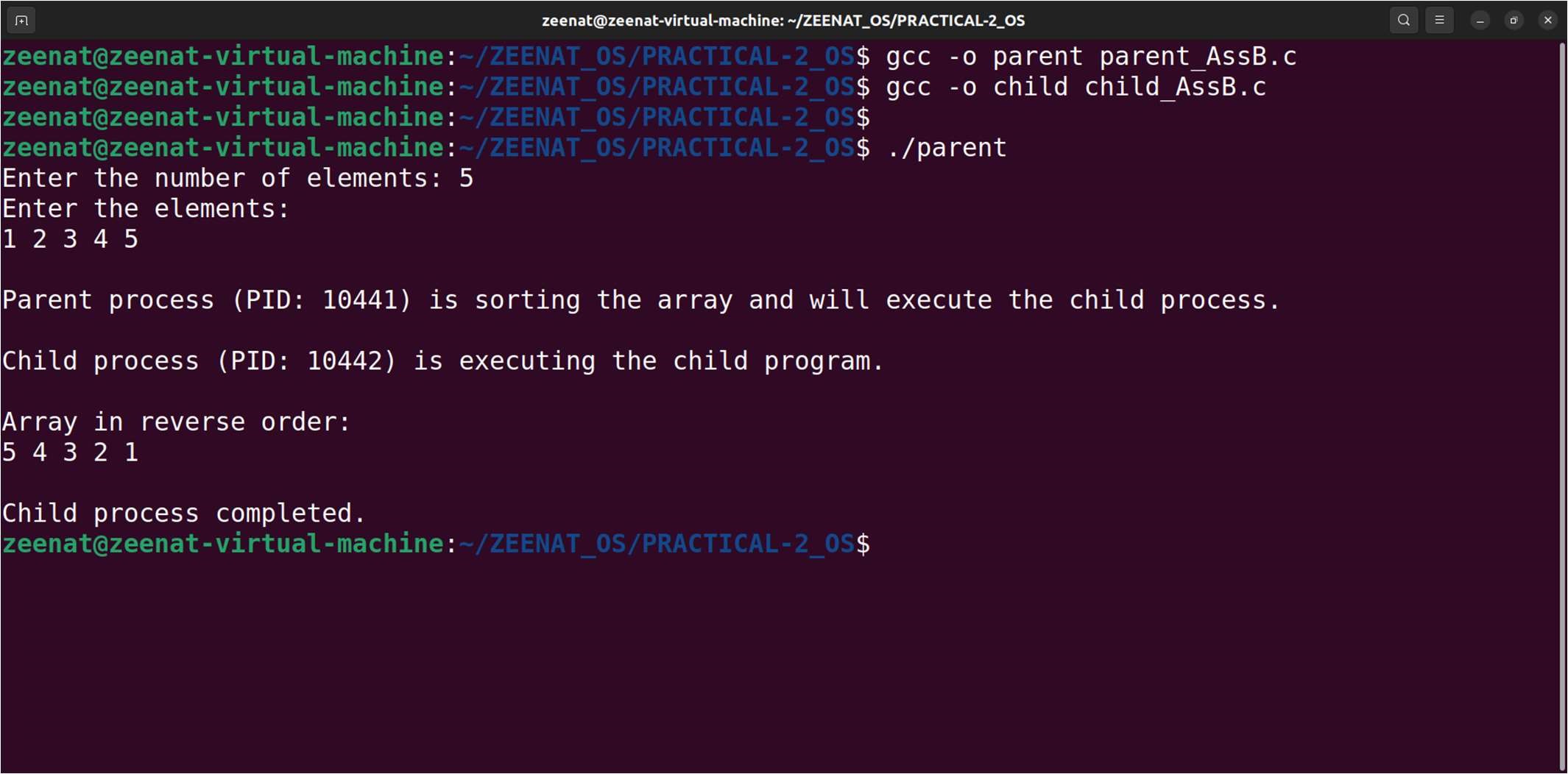
}

printf("\n");

return 0;

}

# OUTPUT: PART B



## Assignment No. 3

Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

### SRTF\_preemptive.c

#include <stdio.h>

#include <limits.h>

typedef struct { int id; // Process ID int at; // Arrival Time int bt; // Burst Time int rt; // Remaining Time int ct; // Completion Time int tat; // Turnaround Time int wt; // Waiting Time int start; // Start Time (for response time calculation)

} Process;

void srtf(Process proc[], int n); void calculate\_averages(Process proc[], int n);

int main() {

int n;

printf("Enter the number of processes: "); scanf("%d", &n);

Process proc[n];

printf("Enter Arrival Time and Burst Time for each process:\n"); for (int i = 0; i < n; i++) {

proc[i].id = i;

printf("Process %d: ", i + 1);

scanf("%d %d", &proc[i].at, &proc[i].bt); proc[i].rt = proc[i].bt; // Initialize remaining time with burst time proc[i].start = -1; // Start time is initially unde ined

}

printf("\nShortest Remaining Time First (SRTF) Scheduling:\n"); srtf(proc, n);

return 0;

}

void srtf(Process proc[], int n) { int time = 0, completed = 0, min\_rt, shortest; int is\_completed[n];

for (int i = 0; i < n; i++) {

is\_completed[i] = 0;

}

while (completed != n) { min\_rt = INT\_MAX;

shortest = -1;

for (int i = 0; i < n; i++) { if (proc[i].at <= time && is\_completed[i] == 0 && proc[i].rt < min\_rt) { min\_rt = proc[i].rt;

shortest = i;

}

}

if (shortest == -1) { time++;

continue;

}

if (proc[shortest].start == -1) { proc[shortest].start = time; // Set start time if not set

}

proc[shortest].rt--; time++;

if (proc[shortest].rt == 0) { proc[shortest].ct = time;

proc[shortest].tat = proc[shortest].ct - proc[shortest].at; proc[shortest].wt = proc[shortest].tat - proc[shortest].bt; is\_completed[shortest] = 1; completed++;

}

}

calculate\_averages(proc, n);

}

void calculate\_averages(Process proc[], int n) {

loat total\_wt = 0, total\_tat = 0, total\_rt = 0;

printf("ID\tAT\tBT\tCT\tTAT\tWT\tRT\n");

for (int i = 0; i < n; i++) {

proc[i].rt = proc[i].start - proc[i].at; // Calculate response time

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].id, proc[i].at, proc[i].bt, proc[i].ct, proc[i].tat, proc[i].wt, proc[i].rt); total\_wt += proc[i].wt; total\_tat += proc[i].tat;

total\_rt += proc[i].rt;

}

printf("\nAverage WT: %.2f\n", total\_wt / n); printf("Average TAT: %.2f\n", total\_tat / n); printf("Average RT: %.2f\n", total\_rt / n);

}

### OUTPUT: SRTF\_preemptive.c



### RoundRobin.c

#include <stdio.h>

#include <limits.h>

typedef struct { int id; // Process ID int at; // Arrival Time int bt; // Burst Time int rt; // Remaining Time int ct; // Completion Time int tat; // Turnaround Time int wt; // Waiting Time

int start; // Start Time (for response time calculation)

} Process;

void findTimes(Process proc[], int n, int quantum); void printResults(Process proc[], int n); void calculateAverages(Process proc[], int n); int main() { int n; int quantum;

printf("Enter the number of processes: "); scanf("%d", &n);

Process proc[n];

printf("Enter Arrival Time and Burst Time for each process:\n"); for (int i = 0; i < n; i++) { proc[i].id = i; printf("Process %d: ", i + 1); scanf("%d %d", &proc[i].at, &proc[i].bt); proc[i].rt = proc[i].bt; // Initialize remaining time with burst time proc[i].start = -1; // Start time is initially undefined

}

printf("Enter the time quantum: "); scanf("%d", &quantum); findTimes(proc, n, quantum); printResults(proc, n); calculateAverages(proc, n);

return 0;

}

void findTimes(Process proc[], int n, int quantum) { int time = 0, complete = 0; int rem\_bt[n];

// Initialize remaining burst time and other fields for (int i = 0; i < n; i++) { rem\_bt[i] = proc[i].bt; proc[i].wt = 0; proc[i].tat = 0; proc[i].rt = -1; // Response time is initially undefined proc[i].ct = 0;

}

// Scheduling while (complete < n) { int progress = 0;

for (int i = 0; i < n; i++) {

if (rem\_bt[i] > 0 && proc[i].at <= time) { if (proc[i].rt == -1) {

proc[i].rt = time - proc[i].at; // Record response time

}

if (rem\_bt[i] > quantum) { time += quantum; rem\_bt[i] -= quantum;

} else { time += rem\_bt[i]; proc[i].ct = time;

proc[i].tat = proc[i].ct - proc[i].at; proc[i].wt = proc[i].tat - proc[i].bt; rem\_bt[i] = 0; complete++;

}

progress = 1;

}

}

if (!progress) { time++;

}

}

}

void printResults(Process proc[], int n) { printf("ID\tAT\tBT\tCT\tTAT\tWT\tRT\n");

for (int i = 0; i < n; i++) { printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].id + 1, proc[i].at, proc[i].bt, proc[i].ct, proc[i].tat, proc[i].wt, proc[i].rt);

}

}

void calculateAverages(Process proc[], int n) { float total\_wt = 0, total\_tat = 0, total\_rt = 0;

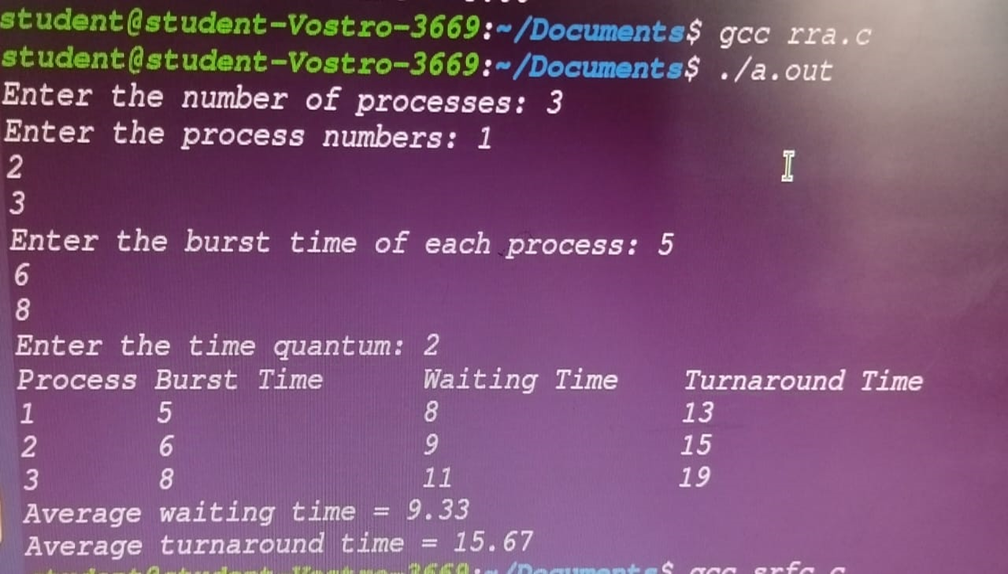
for (int i = 0; i < n; i++) { total\_wt += proc[i].wt; total\_tat += proc[i].tat; total\_rt += proc[i].rt;

}

printf("\nAverage WT: %.2f\n", total\_wt / n); printf("Average TAT: %.2f\n", total\_tat / n); printf("Average RT: %.2f\n", total\_rt / n);

}

### OUTPUT: RoundRobin.c



Assignment No. 4

1. Thread synchronization using counting semaphores. Application to demonstrate: producer- consumer problem with counting semaphores and mutex.

1. Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader- Writer problem with reader priority.

# PART A

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#de ine MAX 5 // Maximum buffer size

int buffer[MAX]; // Shared buffer int in = 0, out = 0; // Indices for in and out sem\_t empty, full; // Counting semaphores pthread\_mutex\_t mutex; // Mutex for critical section

// Producer function void\* producer(void\* arg) { int items\_to\_produce = \*(int\*)arg; // Get the number of items to produce for (int i = 0; i < items\_to\_produce; i++) { sem\_wait(&empty); // Wait for an empty slot pthread\_mutex\_lock(&mutex); // Lock the buffer buffer[in] = i; // Produce an item printf("Number of items Produced: %d\n", i); in = (in + 1) % MAX; // Update the in index pthread\_mutex\_unlock(&mutex); // Unlock the buffer sem\_post(&full); // Signal that an item is produced

}

return NULL;

}

// Consumer function void\* consumer(void\* arg) { int items\_to\_consume = \*(int\*)arg; // Get the number of items to consume for (int i = 0; i < items\_to\_consume; i++) { sem\_wait(&full); // Wait for a illed slot pthread\_mutex\_lock(&mutex); // Lock the buffer int item = buffer[out]; // Consume an item printf("Number of items Consumed: %d\n", item); out = (out + 1) % MAX; // Update the out index pthread\_mutex\_unlock(&mutex); // Unlock the buffer sem\_post(&empty); // Signal that an item is consumed

}

return NULL;

}

// Main function int main() { pthread\_t prod, cons;

int items\_to\_process;

// Initialize semaphores and mutex sem\_init(&empty, 0, MAX); // Set empty slots to MAX sem\_init(&full, 0, 0); // No illed slots initially

pthread\_mutex\_init(&mutex, NULL); // Initialize the mutex

// Get user input

printf("Enter the number of items to produce and consume: "); scanf("%d", &items\_to\_process);

// Create producer and consumer threads

pthread\_create(&prod, NULL, producer, &items\_to\_process); pthread\_create(&cons, NULL, consumer, &items\_to\_process);

// Wait for threads to inish pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

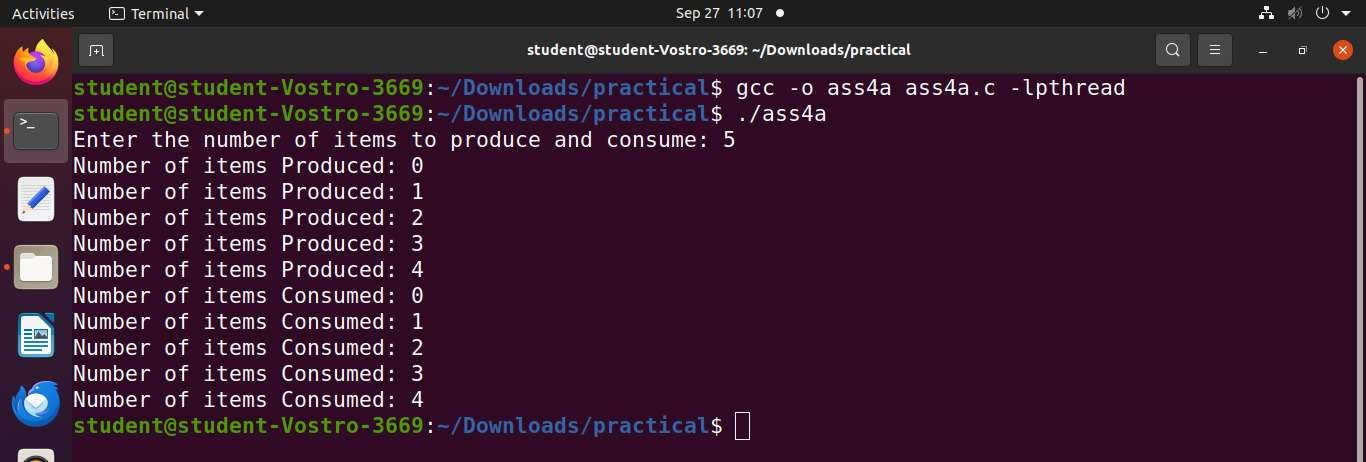
// Clean up sem\_destroy(&empty); sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

# OUTPUT: PART A



# PART B

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

pthread\_mutex\_t readWriteMutex; pthread\_mutex\_t mutex; int readCount = 0; int sharedData = 0; int iterations; // Number of iterations for readers and writers

void\* reader(void\* arg) {

int id = \*(int\*)arg;

for (int i = 0; i < iterations; i++) { // Run for a ixed number of iterations

pthread\_mutex\_lock(&mutex);

readCount++; if (readCount == 1) { pthread\_mutex\_lock(&readWriteMutex);

}

pthread\_mutex\_unlock(&mutex);

// Reading section printf("Reader %d: Reading data %d\n", id, sharedData);

sleep(1); // Simulate reading time pthread\_mutex\_lock(&mutex);

readCount--;

if (readCount == 0) { pthread\_mutex\_unlock(&readWriteMutex);

}

pthread\_mutex\_unlock(&mutex);

sleep(1); // Simulate some delay before next read

}

return NULL;

}

void\* writer(void\* arg) {

int id = \*(int\*)arg;

for (int i = 0; i < iterations; i++) { // Run for a ixed number of iterations

pthread\_mutex\_lock(&readWriteMutex);

// Writing section sharedData++; printf("Writer %d: Writing data %d\n", id, sharedData);

sleep(1); // Simulate writing time pthread\_mutex\_unlock(&readWriteMutex); sleep(3); // Simulate some delay before next write }

return NULL;

}

int main() {

int numReaders, numWriters;

printf("Enter the number of readers: "); scanf("%d", &numReaders); printf("Enter the number of writers: "); scanf("%d", &numWriters); printf("Enter the number of iterations for each reader/writer: "); scanf("%d", &iterations);

pthread\_t\* readers = malloc(numReaders \* sizeof(pthread\_t)); pthread\_t\* writers = malloc(numWriters \* sizeof(pthread\_t)); int\* readerIDs = malloc(numReaders \* sizeof(int)); int\* writerIDs = malloc(numWriters \* sizeof(int)); pthread\_mutex\_init(&readWriteMutex, NULL);

pthread\_mutex\_init(&mutex, NULL);

// Create reader threads for (int i = 0; i < numReaders; i++) {

readerIDs[i] = i + 1; pthread\_create(&readers[i], NULL, reader, &readerIDs[i]);

}

// Create writer threads for (int i = 0; i < numWriters; i++) {

writerIDs[i] = i + 1; pthread\_create(&writers[i], NULL, writer, &writerIDs[i]);

}

// Join threads

for (int i = 0; i < numReaders; i++) {

pthread\_join(readers[i], NULL);

}

for (int i = 0; i < numWriters; i++) {

pthread\_join(writers[i], NULL);

}

// Cleanup free(readers); free(writers); free(readerIDs);

free(writerIDs);

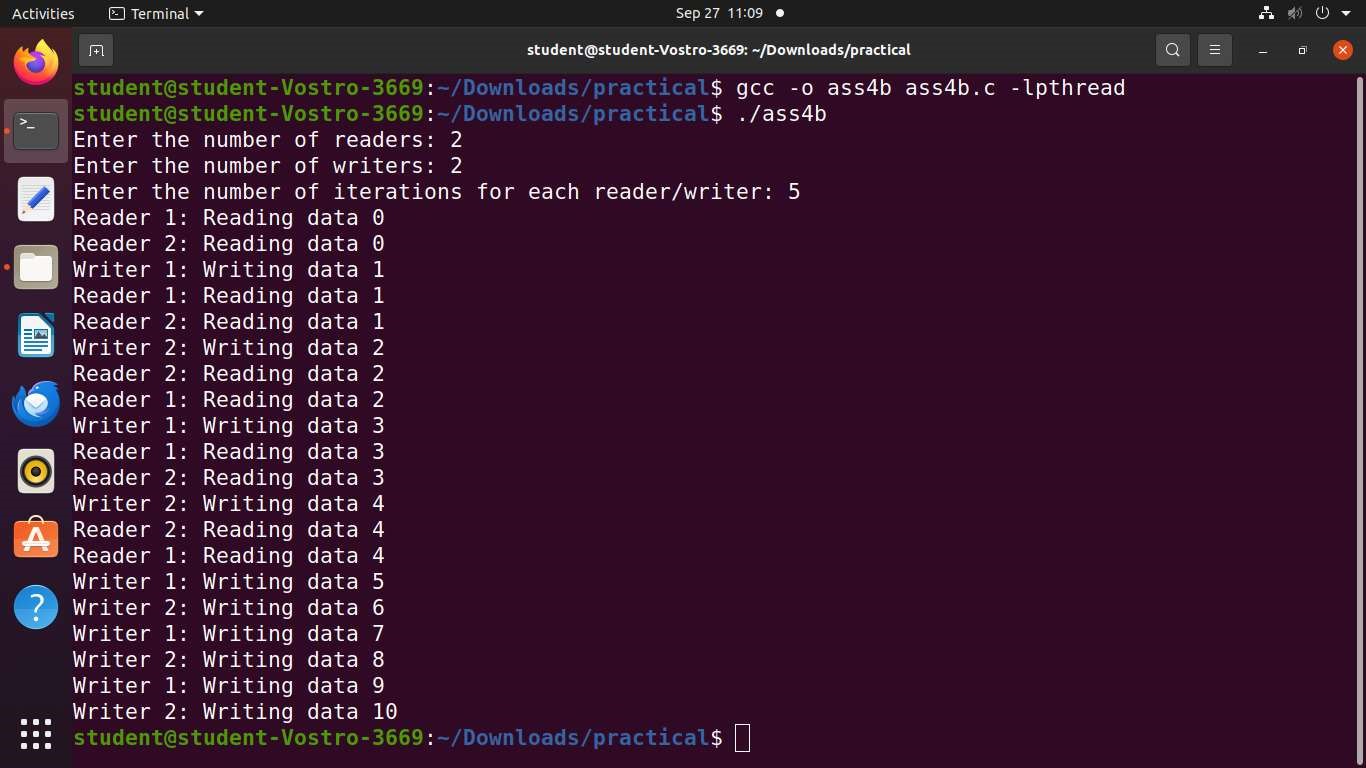
pthread\_mutex\_destroy(&readWriteMutex);

pthread\_mutex\_destroy(&mutex);

return 0;

}

# OUTPUT: PART B



Assignment No. 5

Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.

#include <stdio.h>

#de ine MAX\_PROCESSES 5

#de ine MAX\_RESOURCES 3

void calculateNeed(int need[MAX\_PROCESSES][MAX\_RESOURCES], int max[MAX\_PROCESSES][MAX\_RESOURCES], int alloc[MAX\_PROCESSES][MAX\_RESOURCES], int n, int m)

{

for (int i = 0; i < n; i++) { for (int j = 0; j < m; j++) {

need[i][j] = max[i][j] - alloc[i][j];

}

}

}

int isSafe(int alloc[MAX\_PROCESSES][MAX\_RESOURCES], int max[MAX\_PROCESSES][MAX\_RESOURCES], int avail[MAX\_RESOURCES], int n, int m) {

int f[MAX\_PROCESSES] = {0}, ans[MAX\_PROCESSES], ind = 0;

int need[MAX\_PROCESSES][MAX\_RESOURCES]; calculateNeed(need, max, alloc, n, m); int y = 0;

for (int k = 0; k < n; k++) { for (int i = 0; i < n; i++) {

if (f[i] == 0) { int lag = 0;

for (int j = 0; j < m; j++) { if (need[i][j] > avail[j]) {

lag = 1; break;

}

}

if ( lag == 0) { ans[ind++] = i; // Store the index of the process for (y = 0; y < m; y++) { avail[y] += alloc[i][y]; // Release allocated resources

}

f[i] = 1; // Mark the process as inished

}

}

}

}

for (int i = 0; i < n; i++) { if (f[i] == 0) { printf("The system is not in a safe state.\n"); return 0;

}

}

printf("The system is in a safe state.\n");

printf("Following is the SAFE sequence: ");

for (int i = 0; i < n; i++) {

printf("P%d ", ans[i]);

}

printf("\n");

return 1;

}

int main() { int n, m;

int alloc[MAX\_PROCESSES][MAX\_RESOURCES]; int max[MAX\_PROCESSES][MAX\_RESOURCES]; int avail[MAX\_RESOURCES];

// Input number of processes and resources printf("Enter the number of processes: "); scanf("%d", &n); printf("Enter the number of resources: "); scanf("%d", &m);

// Input allocation matrix printf("Enter the allocation matrix:\n");

for (int i = 0; i < n; i++) { for (int j = 0; j < m; j++) {

scanf("%d", &alloc[i][j]);

}

}

// Input maximum matrix printf("Enter the maximum matrix:\n"); for (int i = 0; i < n; i++) { for (int j = 0; j < m; j++) {

scanf("%d", &max[i][j]);

}

}

// Input available resources printf("Enter the available resources:\n");

for (int i = 0; i < m; i++) { scanf("%d", &avail[i]);

}

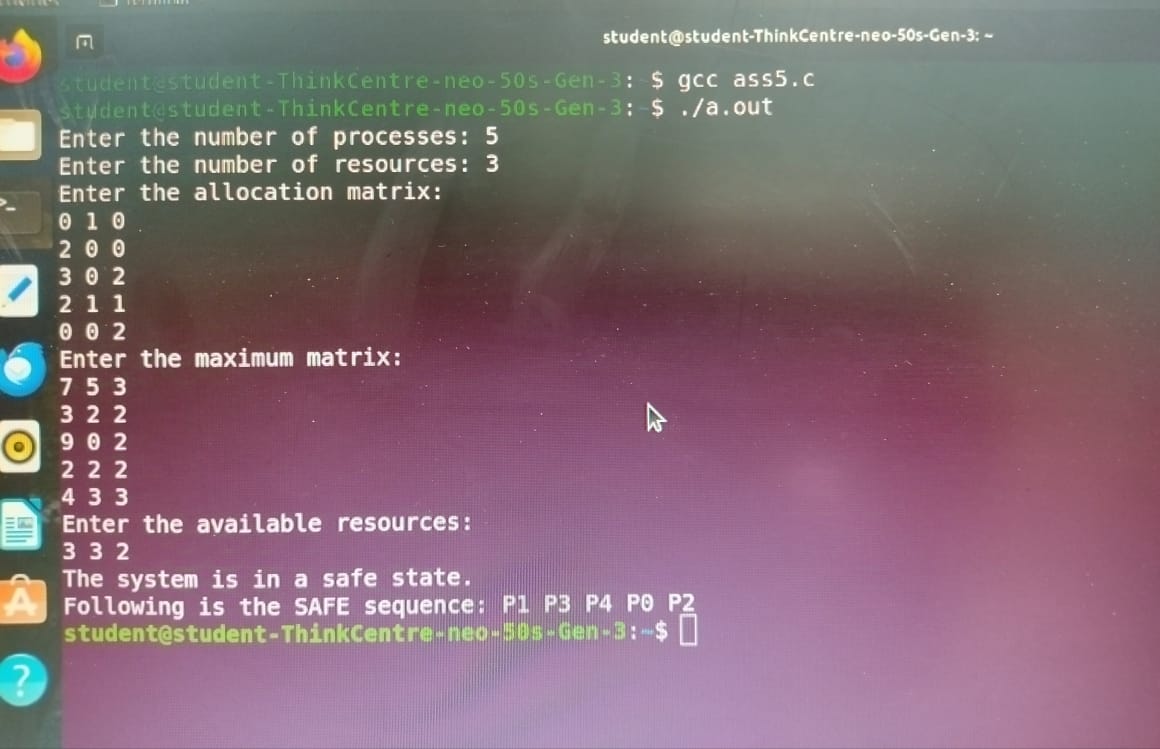
// Check if the system is in a safe state

isSafe(alloc, max, avail, n, m);

return 0;

}

# OUTPUT



Assignment No. 6

Implement the C program for Page Replacement Algorithms: FCFS, LRU, and Optimal for frame size as minimum three.

### FCFS.c

#include <stdio.h>

#include <stdlib.h>

#de ine MAX\_PAGES 100

#de ine MAX\_FRAMES 10

void printFrames(int frames[], int frame\_count) {

for (int i = 0; i < frame\_count; i++) {

if (frames[i] != -1) printf("%d ", frames[i]); else printf("\_ ");

}

printf("\n");

}

// FCFS Page Replacement void fcfs(int pages[], int page\_count, int frame\_count) { int frames[MAX\_FRAMES]; for (int i = 0; i < frame\_count; i++) frames[i] = -1;

int page\_faults = 0, hits = 0, next\_frame = 0;

for (int i = 0; i < page\_count; i++) { int found = 0; for (int j = 0; j < frame\_count; j++) { if (frames[j] == pages[i]) {

found = 1; hits++;

break;

}

}

if (!found) {

frames[next\_frame] = pages[i]; next\_frame = (next\_frame + 1) % frame\_count; // Move to next frame page\_faults++; printf("Page %d: MISS\n", pages[i]);

} else {

printf("Page %d: HIT\n", pages[i]);

}

printFrames(frames, frame\_count);

printf("\n"); // Add a new line after printing frames

}

printf("Total Hits: %d\n", hits); printf("Total Misses: %d\n", page\_faults); printf("Total Page Faults: %d\n", page\_faults);

}

int main() { int pages[MAX\_PAGES], page\_count, frame\_count;

printf("Enter number of pages (max %d): ", MAX\_PAGES); scanf("%d", &page\_count); printf("Enter page references: "); for (int i = 0; i < page\_count; i++) { scanf("%d", &pages[i]);

}

printf("Enter number of frames (min 3): "); scanf("%d", &frame\_count);

if (frame\_count < 3) { printf("Frame count must be at least 3.\n"); return 1;

}

printf("\nFCFS Page Replacement:\n");

fcfs(pages, page\_count, frame\_count);

return 0;

}

# OUTPUT: FCFS.c





## LRU.c

#include <stdio.h>

#include <stdlib.h>

#de ine MAX\_PAGES 100

#de ine MAX\_FRAMES 10

void printFrames(int frames[], int frame\_count) {

for (int i = 0; i < frame\_count; i++) {

if (frames[i] != -1) printf("%d ", frames[i]); else printf("\_ ");

}

printf("\n");

}

// LRU Page Replacement

void lru(int pages[], int page\_count, int frame\_count) { int frames[MAX\_FRAMES]; for (int i = 0; i < frame\_count; i++) frames[i] = -1;

int page\_faults = 0, hits = 0, misses = 0; // Added misses counter int time[MAX\_FRAMES] = {0}; // Track usage time for LRU

for (int i = 0; i < page\_count; i++) { int found = 0; for (int j = 0; j < frame\_count; j++) { if (frames[j] == pages[i]) {

found = 1; time[j] = i; // Update the usage time hits++;

printf("Page %d: HIT\n", pages[i]);

printFrames(frames, frame\_count); printf("\n");

break;

}

}

if (!found) {

// Find the LRU page to replace int lru\_index = 0, min\_time = time[0]; for (int j = 1; j < frame\_count; j++) { if (frames[j] == -1 || time[j] < min\_time) {

min\_time = time[j];

lru\_index = j;

}

}

frames[lru\_index] = pages[i]; time[lru\_index] = i; // Update usage time for the newly added page page\_faults++;

misses++; // Increment misses counter printf("Page %d: MISS\n", pages[i]);

printFrames(frames, frame\_count); printf("\n");

}

}

printf("Total Page Faults: %d\n", page\_faults); printf("Total Hits: %d\n", hits); printf("Total Misses: %d\n", misses); // Print total misses

}

int main() { int pages[MAX\_PAGES], page\_count, frame\_count;

printf("Enter number of pages (max %d): ", MAX\_PAGES); scanf("%d", &page\_count); printf("Enter page references: "); for (int i = 0; i < page\_count; i++) {

scanf("%d", &pages[i]);

}

printf("Enter number of frames (min 3): "); scanf("%d", &frame\_count);

if (frame\_count < 3) { printf("Frame count must be at least 3.\n"); return 1;

}

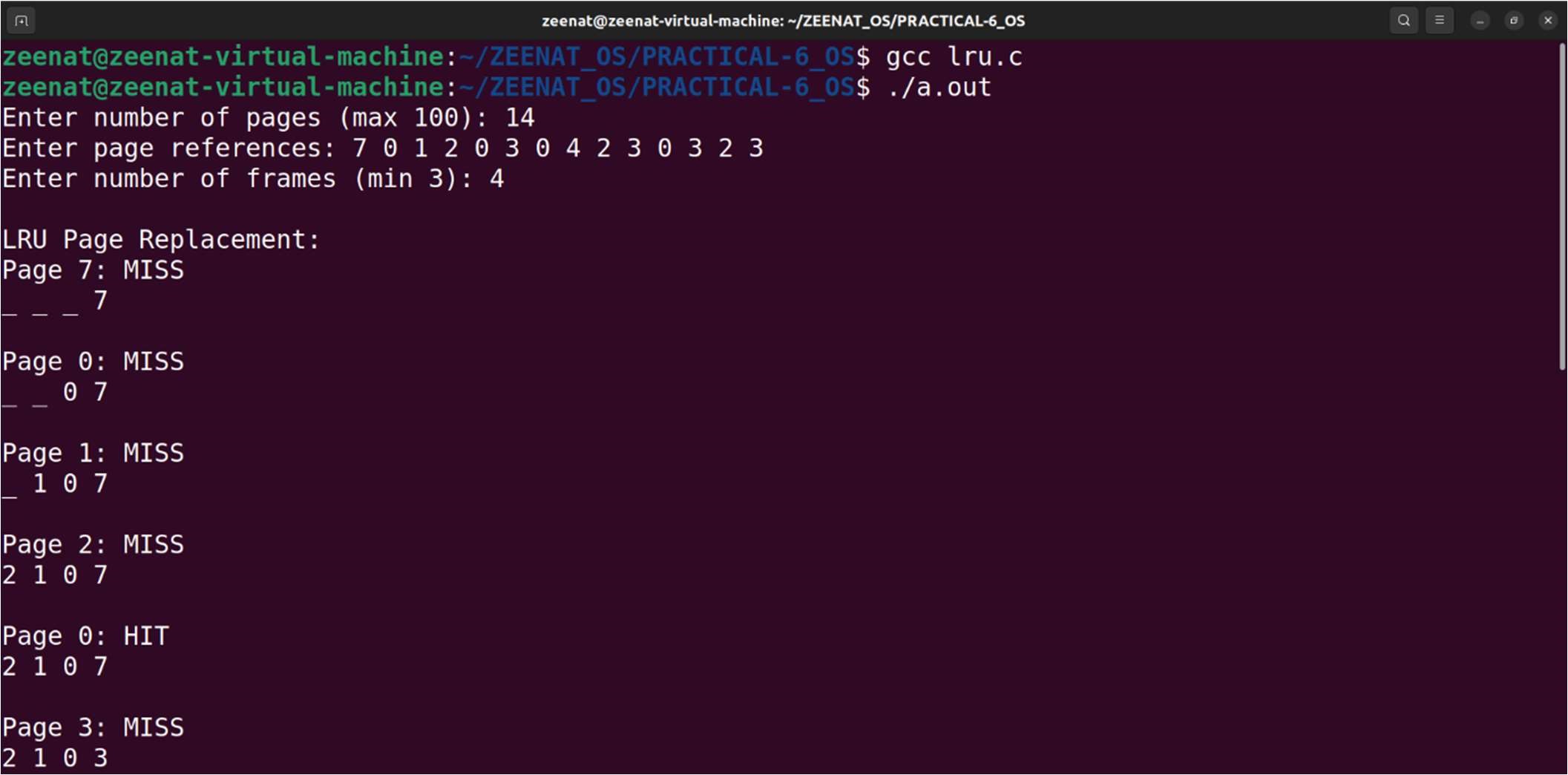
printf("\nLRU Page Replacement:\n");

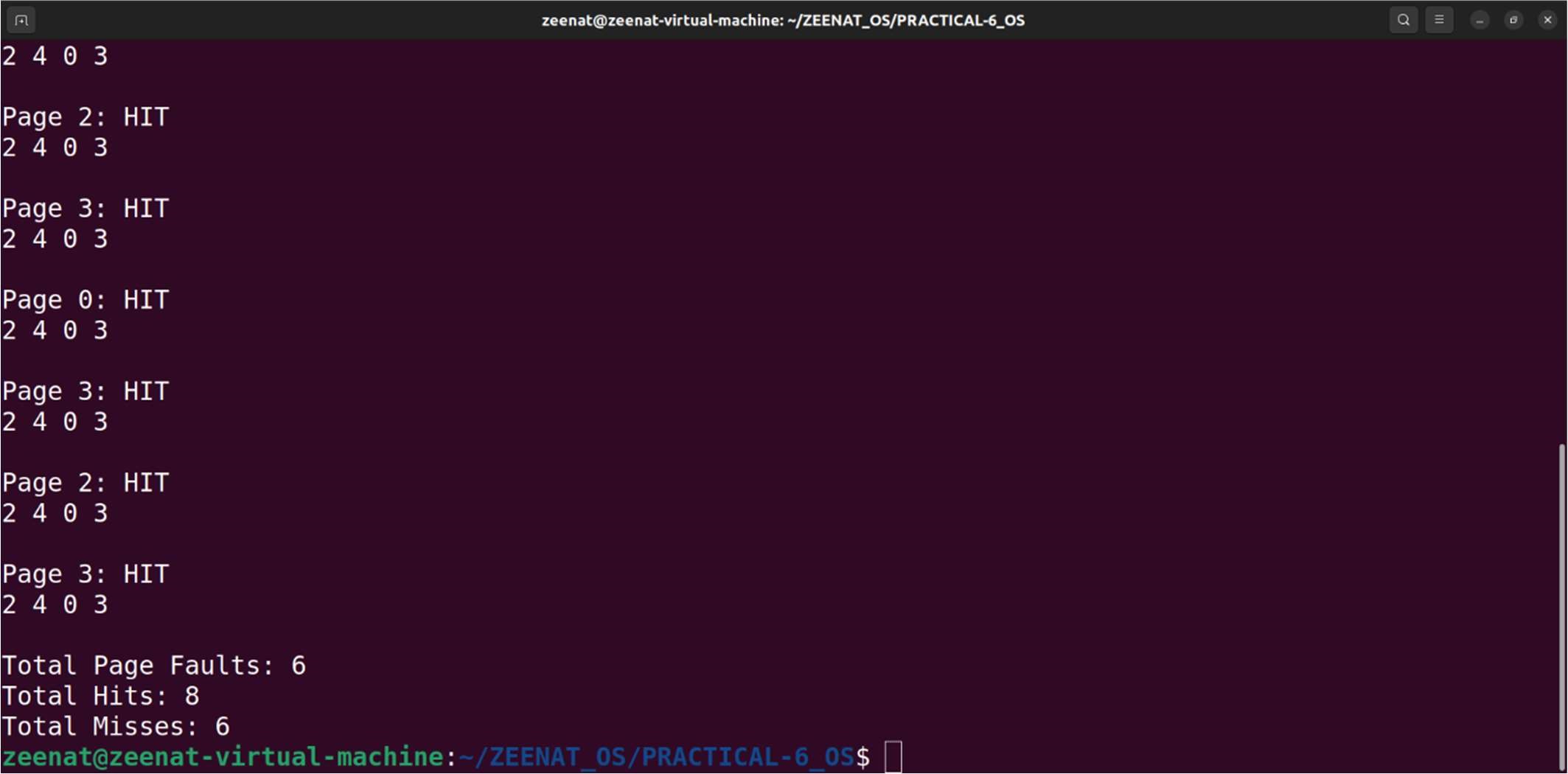
lru(pages, page\_count, frame\_count);

return 0;

}

# OUTPUT: LRU.c





# OPTIMAL.c

#include <stdio.h>

int main() {

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], lag1, lag2, lag3, i, j, k, pos, max, faults = 0, hits = 0, misses = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page reference string: "); for (i = 0; i < no\_of\_pages; ++i) {

scanf("%d", &pages[i]);

}

for (i = 0; i < no\_of\_frames; ++i) {

frames[i] = -1;

}

for (i = 0; i < no\_of\_pages; ++i) { lag1 = lag2 = 0;

for (j = 0; j < no\_of\_frames; ++j) { if (frames[j] == pages[i]) { lag1 = lag2 = 1; hits++; // Count hit

break;

}

}

if ( lag1 == 0) {

misses++; // Count miss for (j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == -1) { faults++; frames[j] = pages[i];

lag2 = 1;

break;

} }

}

if ( lag2 == 0) { lag3 = 0; for (j = 0; j < no\_of\_frames; ++j) {

temp[j] = -1; for (k = i + 1; k < no\_of\_pages; ++k) { if (frames[j] == pages[k]) {

temp[j] = k; break;

}

}

}

for (j = 0; j < no\_of\_frames; ++j) {

if (temp[j] == -1) { pos = j; lag3 = 1;

break;

}

}

if ( lag3 == 0) { max = temp[0];

pos = 0;

for (j = 1; j < no\_of\_frames; ++j) { if (temp[j] > max) { max = temp[j]; pos = j;

}

}

}

frames[pos] = pages[i]; faults++;

}

// Display the current state of frames for (j = 0; j < no\_of\_frames; ++j) { if (frames[j] == -1) { printf("- ");

} else {

printf("%d ", frames[j]);

}

}

// Display hit or miss

if ( lag1) { printf("\nPage %d: Hit\n", pages[i]);

} else {

printf("\nPage %d: Miss\n", pages[i]);

}

printf("\n"); // New line for spacing

}

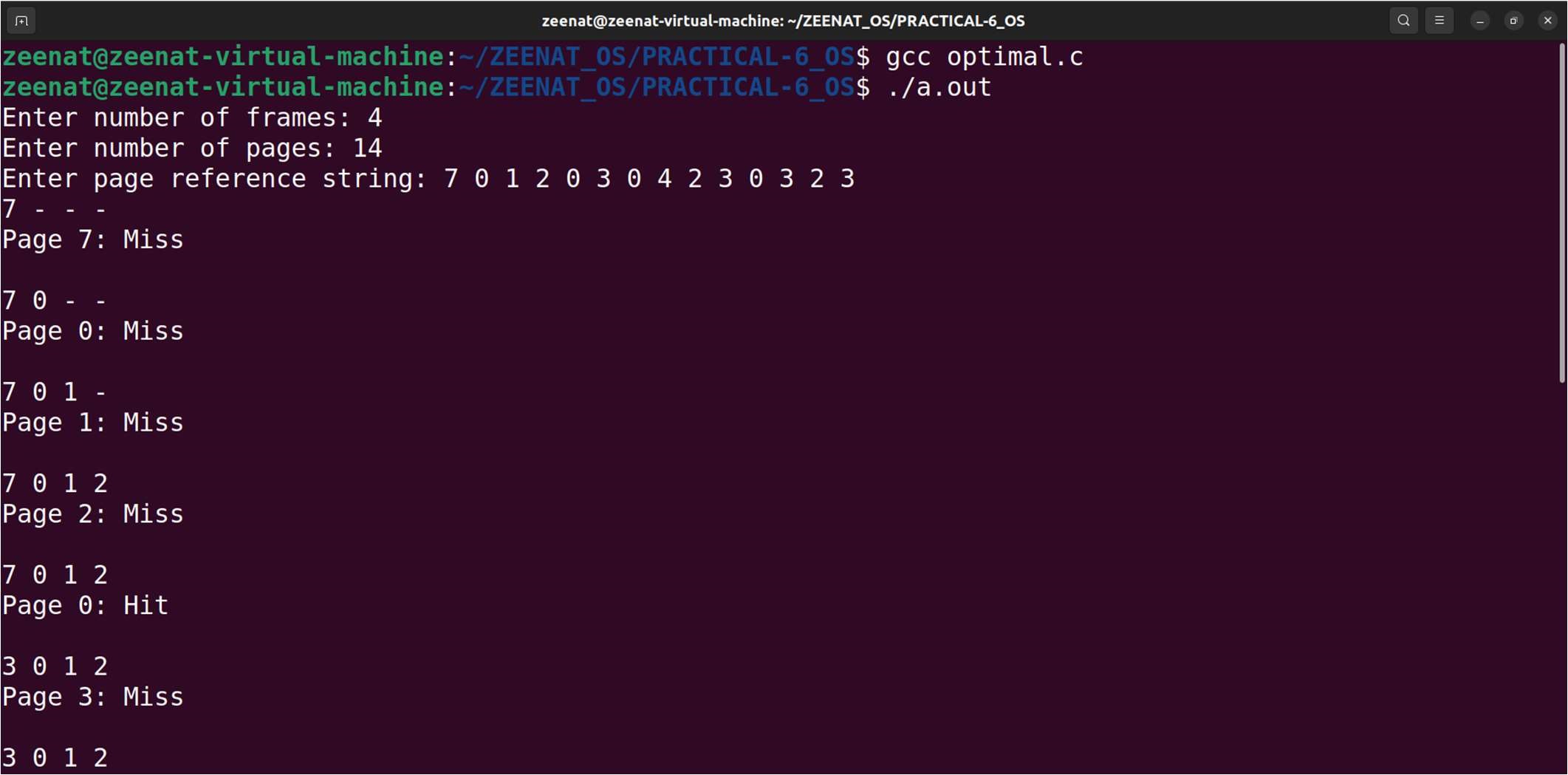
printf("Total Page Faults = %d\n", faults); printf("Total Hits = %d\n", hits);

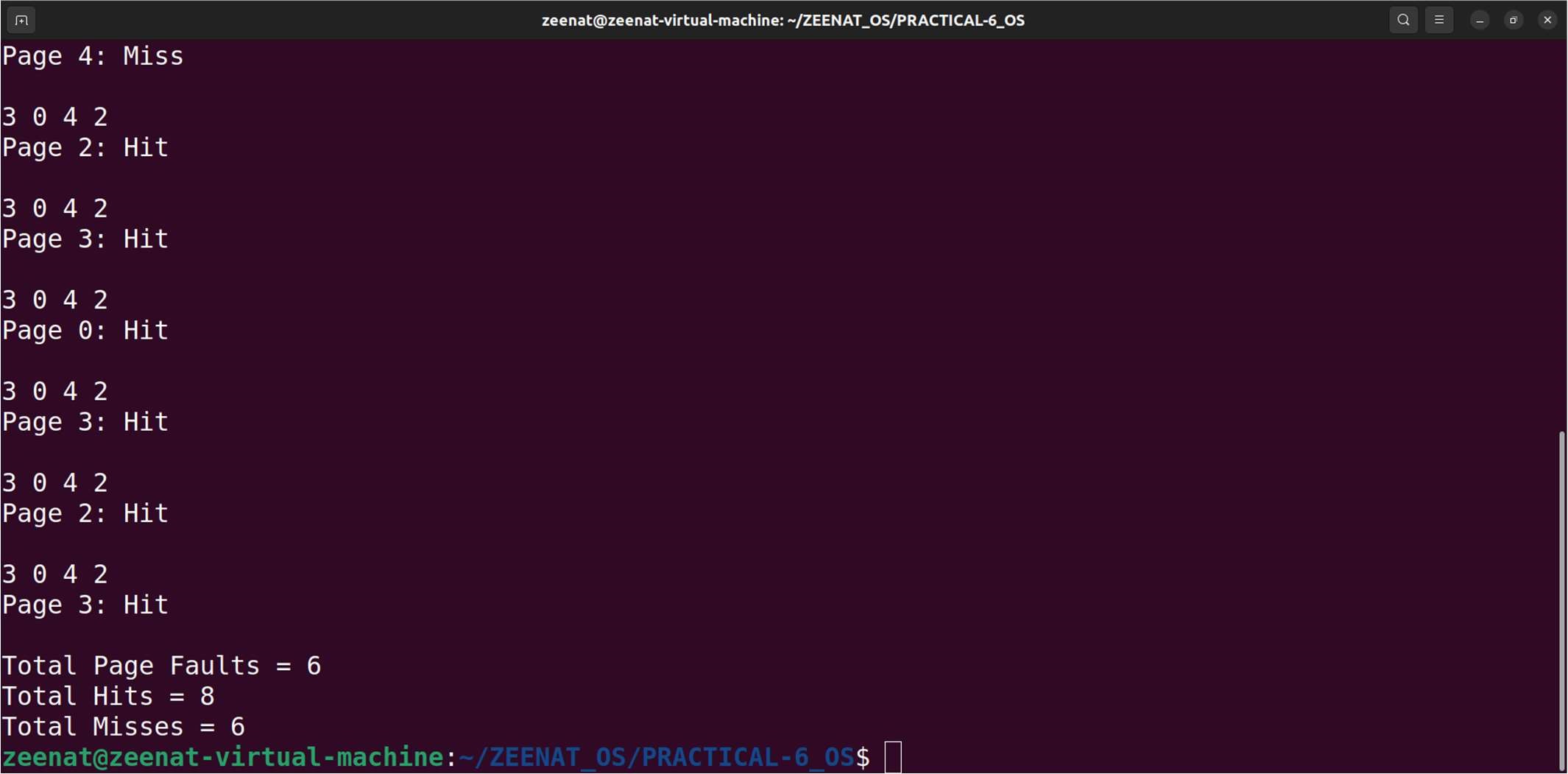
printf("Total Misses = %d\n", misses);

return 0;

}

# OUTPUT: OPTIMAL.c





## Assignment No. 7

Inter process communication in Linux using following.

1. FIFOS: Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays on standard output.
2. Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

### PART A: File 1 – writer.c

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

#de ine FIFO\_PATH "/home/student/text"

int main() {

int fd;

char buffer[100]; // Buffer for user input

// Create FIFO if it doesn't already exist if (mk ifo(FIFO\_PATH, 0666) == -1 && errno != EEXIST) {

perror("Error creating FIFO");

return 1;

}

// Open FIFO for writing fd = open(FIFO\_PATH, O\_WRONLY);

if (fd == -1) { perror("Error opening FIFO for writing"); return 1;

}

while (1) { printf("Enter a sentence (or 'exit' to quit): "); fgets(buffer, sizeof(buffer), stdin);

if (strcmp(buffer, "exit\n") == 0) {

// Write exit signal to FIFO

write(fd, buffer, strlen(buffer) + 1); break; // Exit condition

}

// Write data to FIFO

if (write(fd, buffer, strlen(buffer) + 1) == -1) { perror("Error writing to FIFO"); break;

}

}

close(fd);

return 0;

}

### PART A: File 2 – reader.c

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#de ine FIFO\_PATH "/home/student/text"

void countWordsLinesChars(char \*sentence, int \*wordCount, int \*lineCount, int \*charCount) { char \*token = strtok(sentence, " \n"); \*wordCount = 0;

\*lineCount = 1; // Start with one line

\*charCount = strlen(sentence);

while (token != NULL) { (\*wordCount)++;

token = strtok(NULL, " \n");

}

}

int main() {

int fd;

char buffer[100]; // Buffer for received input

// Open FIFO for reading fd = open(FIFO\_PATH, O\_RDONLY);

if (fd == -1) { perror("Error opening FIFO for reading"); return 1;

}

while (1) {

// Read data from FIFO if (read(fd, buffer, sizeof(buffer)) == -1) {

perror("Error reading from FIFO"); break;

}

// Check for exit condition if (strcmp(buffer, "exit\n") == 0) { printf("Exiting reader...\n");

break; // Exit condition

}

// Output the received data

printf("Received: %s\n", buffer);

// Count characters, words, and lines int wordCount, lineCount, charCount; countWordsLinesChars(buffer, &wordCount, &lineCount, &charCount);

// Print the counts printf("Characters: %d\n", charCount); printf("Words: %d\n", wordCount);

printf("Lines: %d\n", lineCount);

}

close(fd);

return 0;

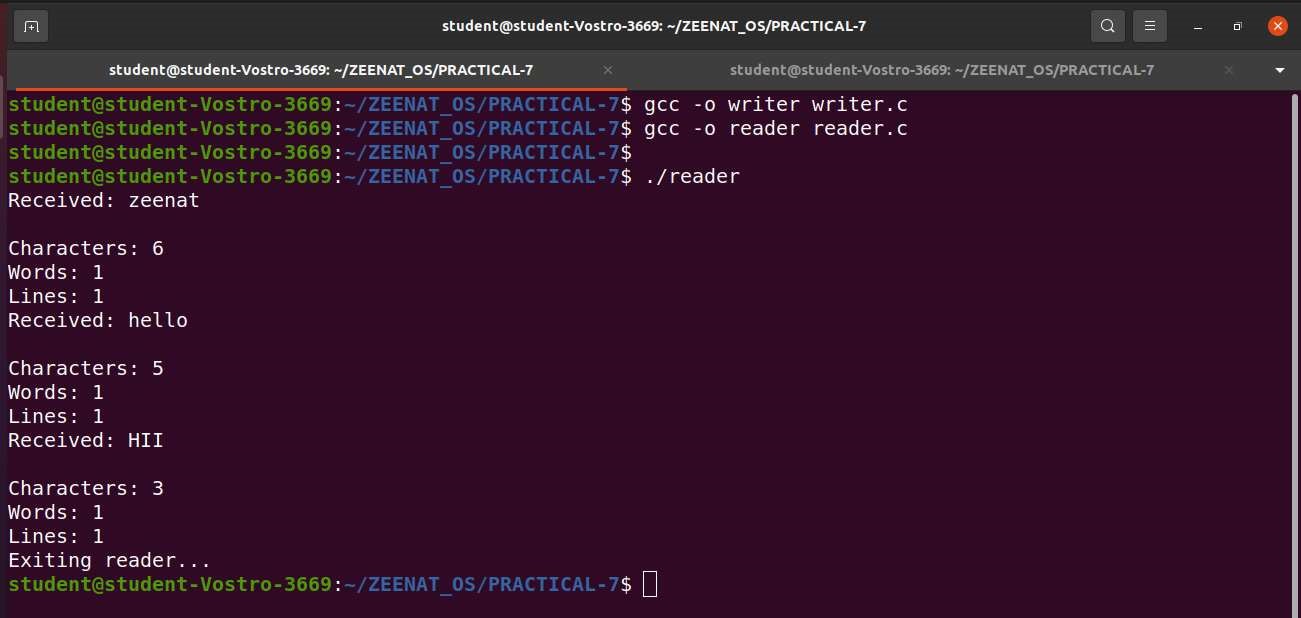
}

# OUTPUT: PART A

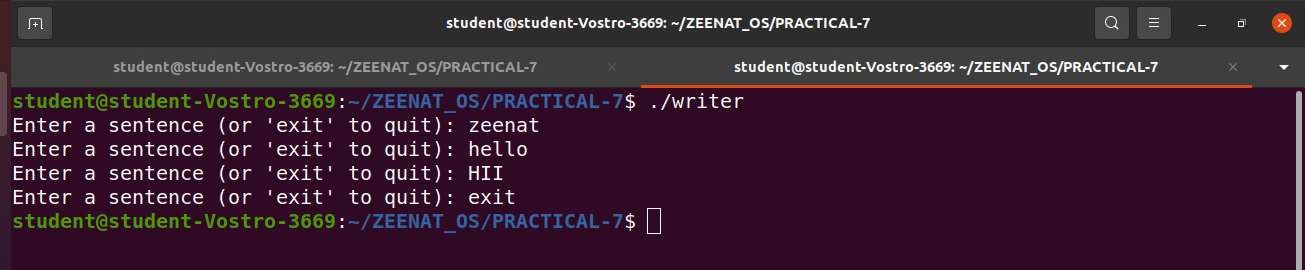
Important Note:

(Run this program at a time, u have to open two terminals on your screen then compile both the files and first run read file then quickly go to next terminal and run write file then after writing to the file go back to see read operation then after finishing in write exit it)

File 1 – reader.c



File 2 – writer.c



## PART B: File 1 – server.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

#include <unistd.h>

#de ine SHM\_SIZE 1024 // De ine the size of shared memory

int main() {

int shmid; key\_t key = 1234; // Shared memory key char \*shm;

// Create shared memory segment shmid = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666); if (shmid < 0) {

perror("shmget"); exit(1);

}

// Attach the shared memory segment shm = shmat(shmid, NULL, 0); if (shm == (char \*) -1) {

perror("shmat"); exit(1);

}

// Prompt user for a message

printf("Server: Enter a message to write to shared memory: "); fgets(shm, SHM\_SIZE, stdin); // Read user input

shm[strcspn(shm, "\n")] = 0; // Remove newline character

printf("Server: Wrote message to shared memory: \"%s\"\n", shm);

// Wait for the client to read the message printf("Server: Waiting for client to read...\n"); sleep(5); // Simulate some delay

// Detach and remove the shared memory segment

shmdt(shm);

shmctl(shmid, IPC\_RMID, NULL);

// Final con irmation message

printf("Server: Client has read the message.\n");

return 0;

}

PART B: File 2 – client.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <string.h>

#de ine SHM\_SIZE 1024 // De ine the size of shared memory

int main() {

int shmid; key\_t key = 1234; // Shared memory key char \*shm;

// Access the shared memory segment

shmid = shmget(key, SHM\_SIZE, 0666); if (shmid < 0) {

perror("shmget"); exit(1);

}

// Attach the shared memory segment shm = shmat(shmid, NULL, 0); if (shm == (char \*) -1) {

perror("shmat"); exit(1);

}

// Read the message from shared memory

printf("Client: Read message from shared memory: \"%s\"\n", shm);

// Detach from shared memory

shmdt(shm);

return 0;

}

# OUTPUT: PART B

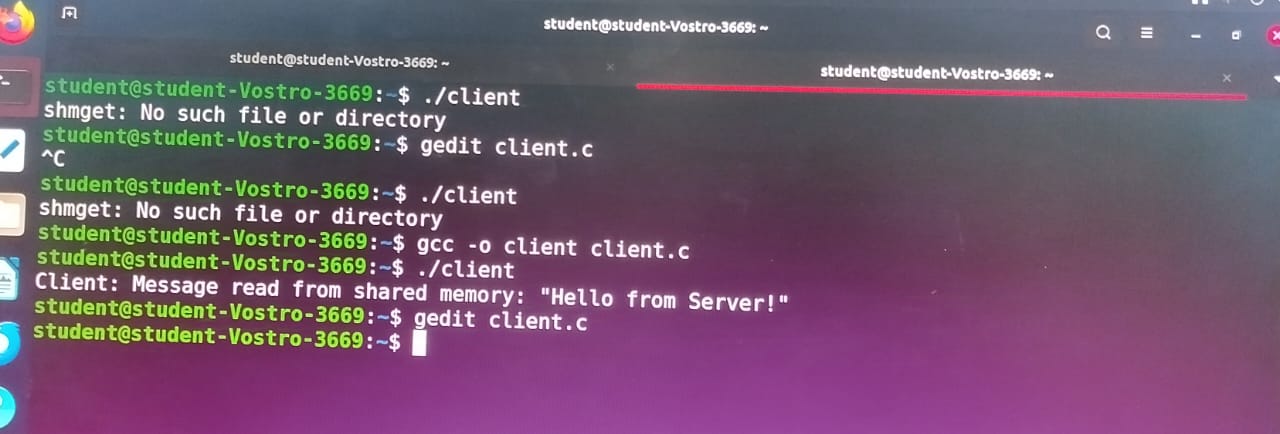
Important Note:

(Run this program at a time, u have to open two terminals on your screen then compile both the files and first run server file answer it then go to next terminal and run client file then see on first terminal u can see it shows msg that client read the msg successfully)

File 1 – server.c



File 2 – client.c



## 

## Assignment No:-8

Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.

### SSTF.c

#include <stdio.h>

#include <stdlib.h>

#de ine MAX\_REQUESTS 100

void SSTF(int requests[], int num\_requests, int initial\_head) { int completed[MAX\_REQUESTS] = {0}; int current\_head = initial\_head;

int total\_distance = 0;

printf("\n\n Seek Sequence: ");

for (int i = 0; i < num\_requests; i++) { int min\_index = -1; int min\_distance = 1000000; // Large number

// Find the closest request for (int j = 0; j < num\_requests; j++) {

if (!completed[j]) { int distance = abs(requests[j] - current\_head);

if (distance < min\_distance) { min\_distance = distance;

min\_index = j;

}

}

}

// Service the closest request if (min\_index != -1) { total\_distance += min\_distance; current\_head = requests[min\_index]; completed[min\_index] = 1;

printf("%d ", current\_head);

}

}

printf("\n Total Number of Seek Operations = %d\n", total\_distance);

}

int main() { int requests[MAX\_REQUESTS]; int num\_requests, initial\_head;

printf("\nEnter number of disk requests: "); scanf("%d", &num\_requests);

printf("Enter disk requests: "); for (int i = 0; i < num\_requests; i++) { scanf("%d", &requests[i]);

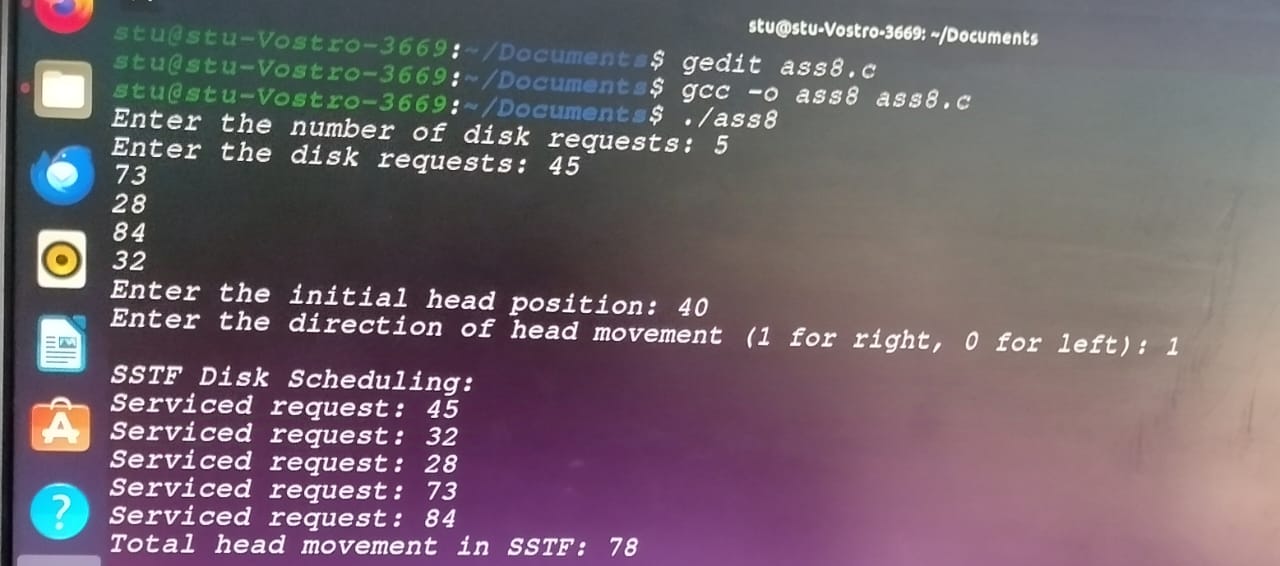
}

printf("Enter initial head position: "); scanf("%d", &initial\_head);

SSTF(requests, num\_requests, initial\_head); return 0;

}

# OUTPUT: SSTF.c



## SCAN.c

#include <stdio.h>

#include <stdlib.h>

#de ine MAX\_REQUESTS 100

void SCAN(int requests[], int num\_requests, int initial\_head, int direction) { int total\_distance = 0;

int current\_head = initial\_head;

// Sort the requests

for (int i = 0; i < num\_requests - 1; i++) { for (int j = i + 1; j < num\_requests; j++) { if (requests[i] > requests[j]) { int temp = requests[i]; requests[i] = requests[j];

requests[j] = temp;

}

}

}

printf("\n\n Seek Sequence: ");

if (direction == 0) { // Moving left // Service requests moving left for (int i = num\_requests - 1; i >= 0; i--) { if (requests[i] <= current\_head) { total\_distance += abs(requests[i] - current\_head);

current\_head = requests[i];

printf("%d ", current\_head);

}

}

// Move to the start of the disk (0) total\_distance += abs(current\_head - 0);

current\_head = 0;

printf("0 "); // Move to the start

// Then service all requests > 0 for (int i = 0; i < num\_requests; i++) {

if (requests[i] > 0) { total\_distance += abs(requests[i] - current\_head);

current\_head = requests[i];

printf("%d ", current\_head);

}

}

} else { // Moving right

// Service requests moving right for (int i = 0; i < num\_requests; i++) { if (requests[i] >= current\_head) { total\_distance += abs(requests[i] - current\_head);

current\_head = requests[i];

printf("%d ", current\_head);

}

}

// Move to the end of the disk (disk size) total\_distance += abs(current\_head - 200); // Assuming disk size is 200 current\_head = 200;

printf("200 "); // Move to the end

// Then service all requests < disk size for (int i = num\_requests - 1; i >= 0; i--) { if (requests[i] < current\_head) { total\_distance += abs(requests[i] - current\_head);

current\_head = requests[i];

printf("%d ", current\_head);

}

}

}

printf("\n Total number of seek operations = %d\n", total\_distance);

}

int main() {

int requests[MAX\_REQUESTS];

int num\_requests, initial\_head, direction;

printf("\nEnter number of disk requests: "); scanf("%d", &num\_requests);

printf("Enter disk requests: "); for (int i = 0; i < num\_requests; i++) { scanf("%d", &requests[i]);

}

printf("Enter initial head position: "); scanf("%d", &initial\_head);

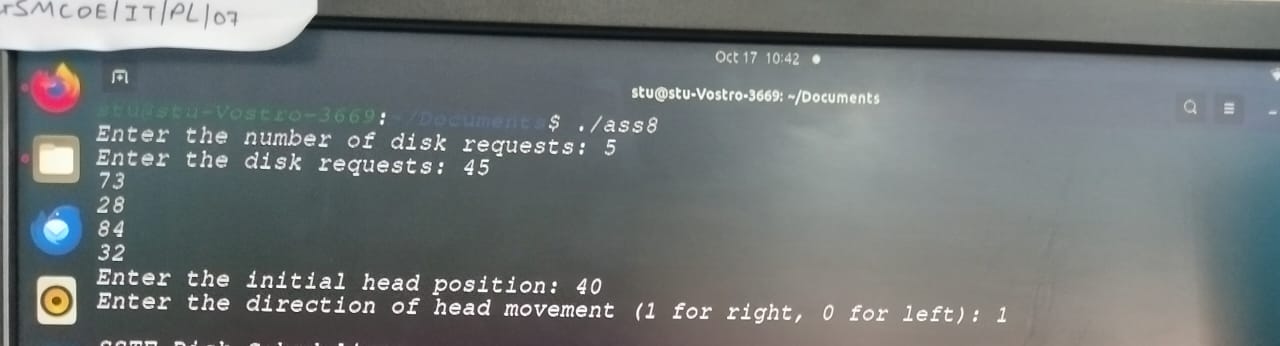
printf("Enter direction (0 for left, 1 for right): ");

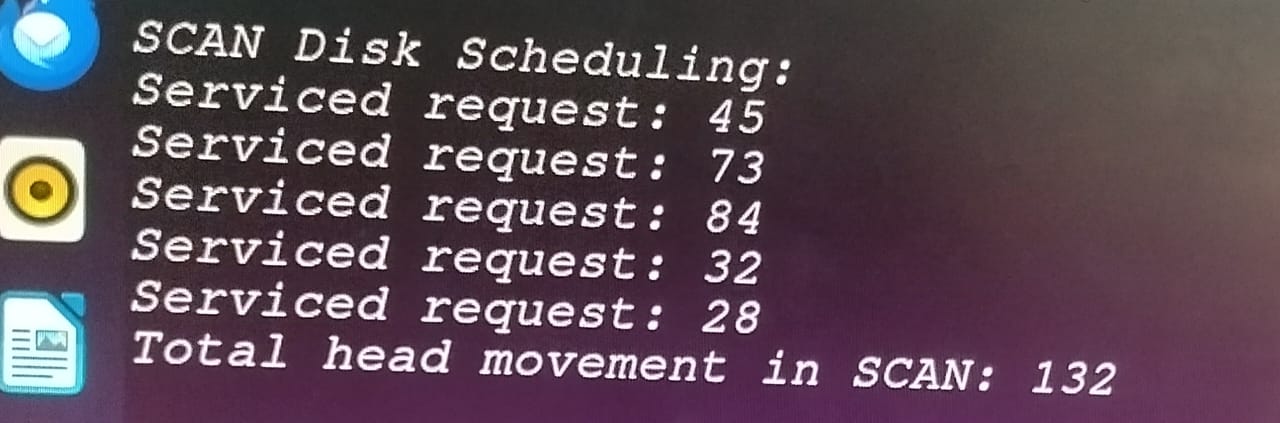
scanf("%d", &direction);

SCAN(requests, num\_requests, initial\_head, direction); return 0;

}

OUTPUT: SCAN.c





# C-LOOK.c

#include <stdio.h>

#include <stdlib.h>

void c\_look(int requests[], int num\_requests, int head, int direction) { int i, j;

// Sort the requests for (i = 0; i < num\_requests - 1; i++) { for (j = i + 1; j < num\_requests; j++) { if (requests[i] > requests[j]) { int temp = requests[i]; requests[i] = requests[j];

requests[j] = temp;

}

}

}

printf("\n\n Seek Sequence: "); int total\_seek\_operations = 0;

int current\_position = head;

// Move based on direction

if (direction == 1) { // Moving to the right

// Service requests greater than the current head position

for (i = 0; i < num\_requests; i++) { if (requests[i] >= current\_position) { total\_seek\_operations += abs(current\_position - requests[i]);

current\_position = requests[i];

printf("%d ", current\_position);

}

}

// Jump to the smallest request (circular movement)

if (current\_position != requests[0]) { total\_seek\_operations += abs(current\_position - requests[0]);

current\_position = requests[0];

printf("%d ", current\_position);

}

// Service the remaining smaller requests for (i = 0; i < num\_requests; i++) { if (requests[i] < head) { total\_seek\_operations += abs(current\_position - requests[i]);

current\_position = requests[i];

printf("%d ", current\_position);

}

}

} else { // Moving to the left

// Service requests smaller than the current head position

for (i = num\_requests - 1; i >= 0; i--) { if (requests[i] <= current\_position) { total\_seek\_operations += abs(current\_position - requests[i]); current\_position = requests[i];

printf("%d ", current\_position);

}

}

// Jump to the largest request (circular movement) if (current\_position != requests[num\_requests - 1]) { total\_seek\_operations += abs(current\_position - requests[num\_requests - 1]); current\_position = requests[num\_requests - 1];

printf("%d ", current\_position);

}

// Service the remaining larger requests for (i = num\_requests - 1; i >= 0; i--) { if (requests[i] > head) { total\_seek\_operations += abs(current\_position - requests[i]); current\_position = requests[i];

printf("%d ", current\_position);

}

}

}

printf("\n Total number of seek operations = %d\n", total\_seek\_operations);

}

int main() { int num\_requests, head, direction;

printf("\nEnter the number of requests: "); scanf("%d", &num\_requests);

int \*requests = (int \*)malloc(num\_requests \* sizeof(int));

printf("Enter the disk requests: "); for (int i = 0; i < num\_requests; i++) {

scanf("%d", &requests[i]);

}

printf("Enter the initial head position: ");

scanf("%d", &head);

printf("Enter the direction (0 for left, 1 for right): ");

scanf("%d", &direction);

c\_look(requests, num\_requests, head, direction);

free(requests);

return 0;

}

# OUTPUT: C-LOOK.c

