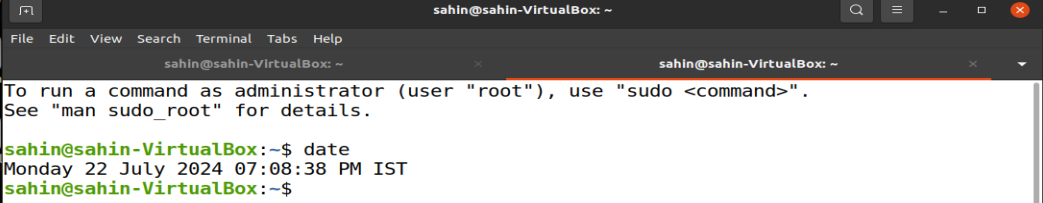
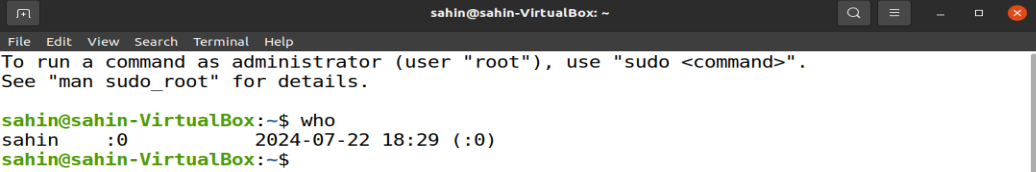
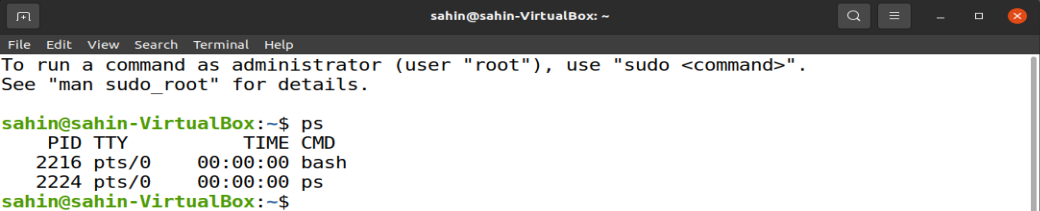
**1. Display the date using ‘date’ command.**



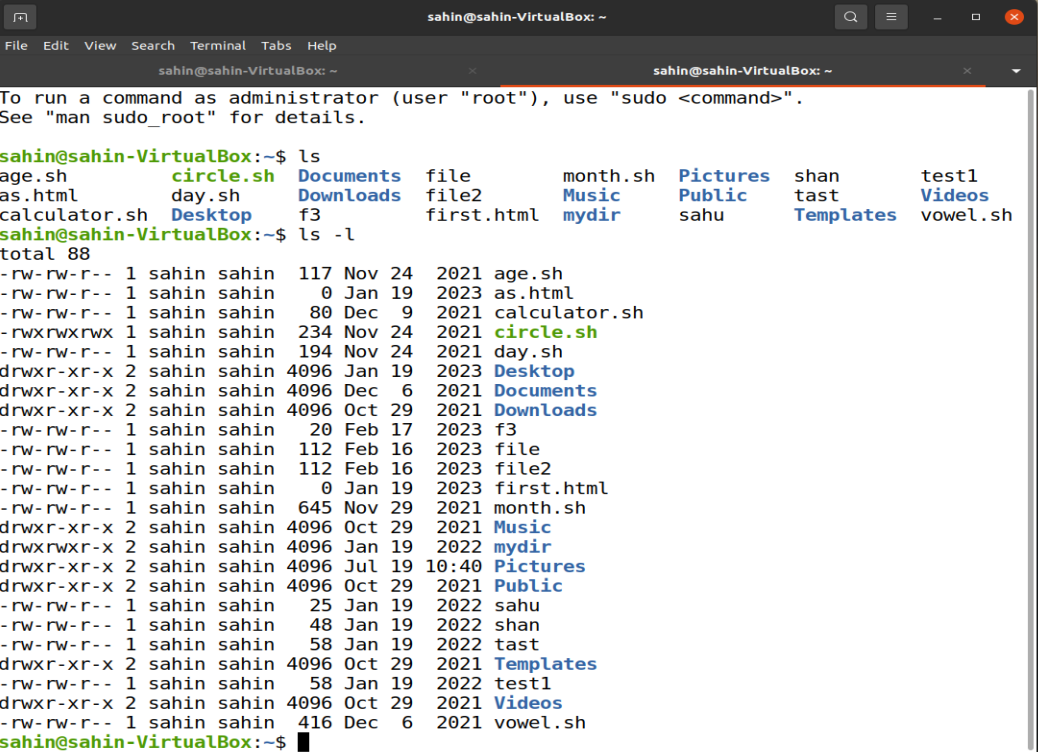
**2. Check who are the users logged in using the ‘who’ command.**



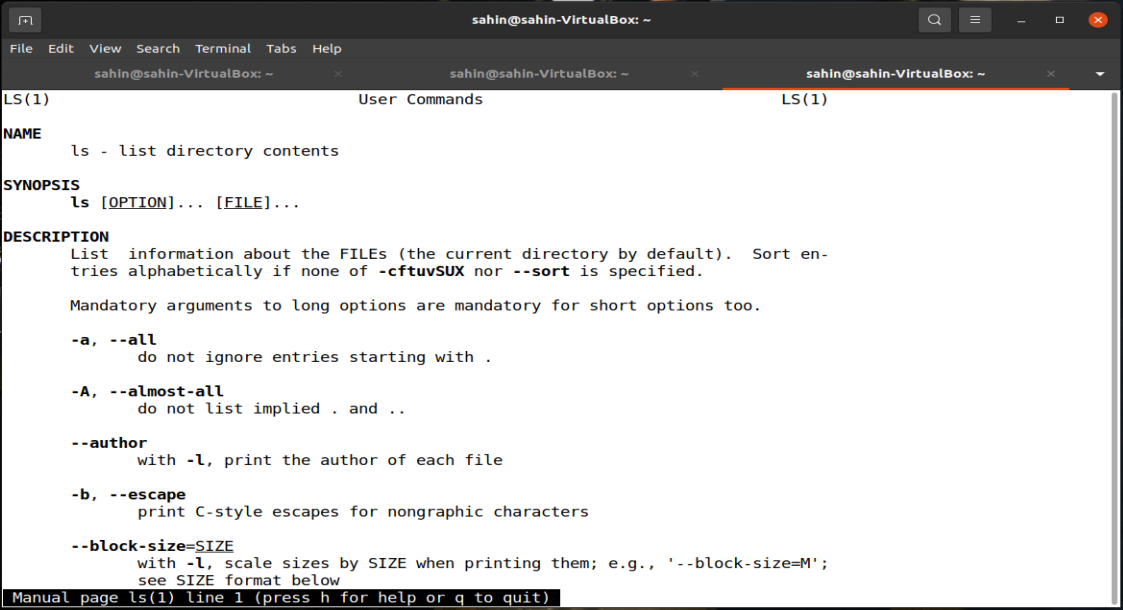
**3. Check the running processes using the ‘ps’ command.**



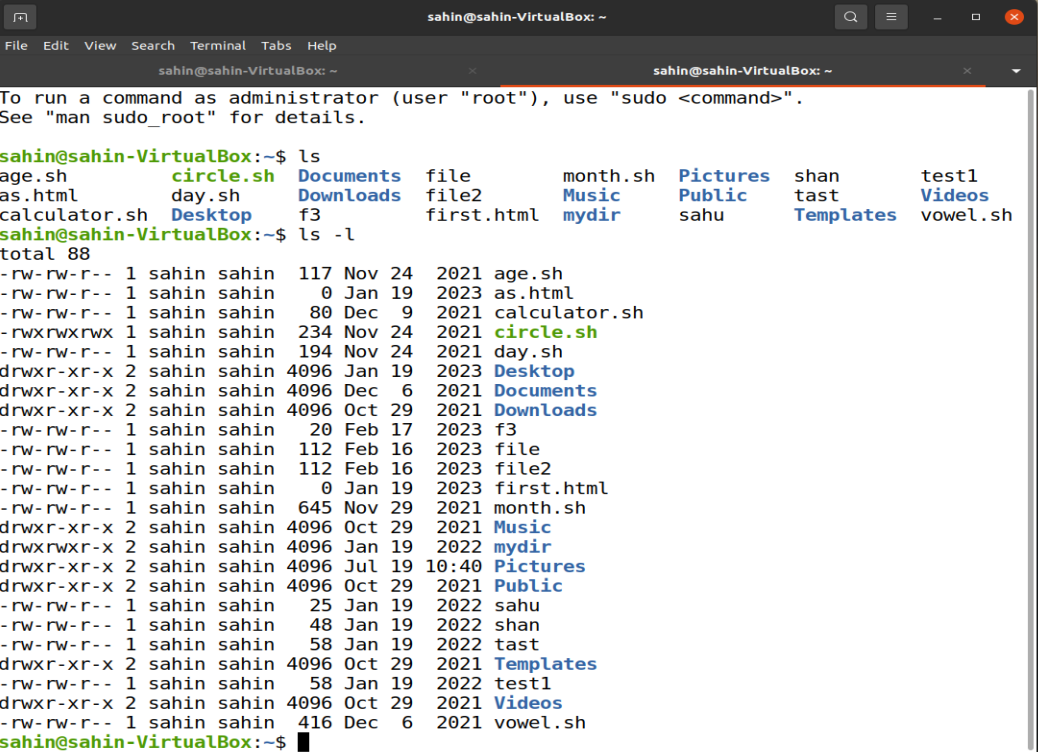
**4. List the files with ‘ls’ command with and without –l option.**

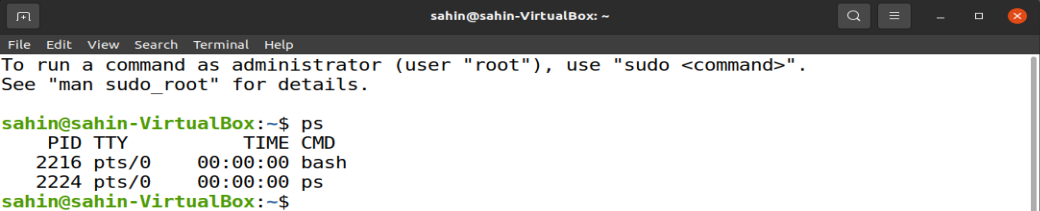


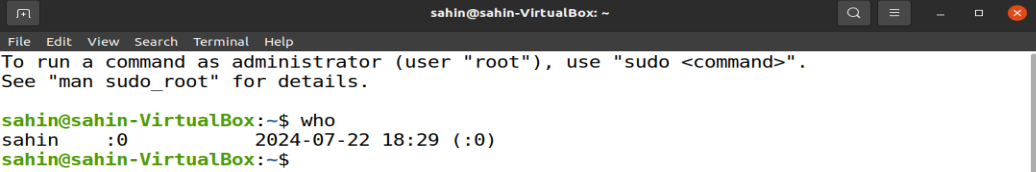
**5. Check the manual of ‘ls’ command.**



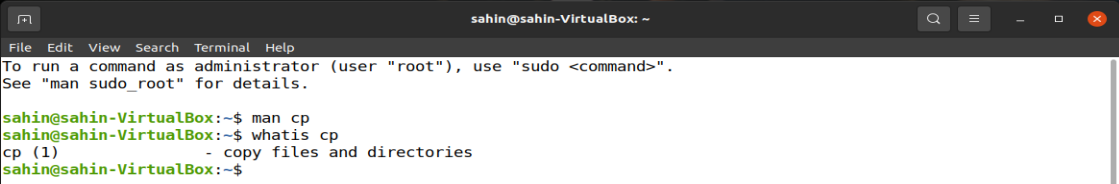
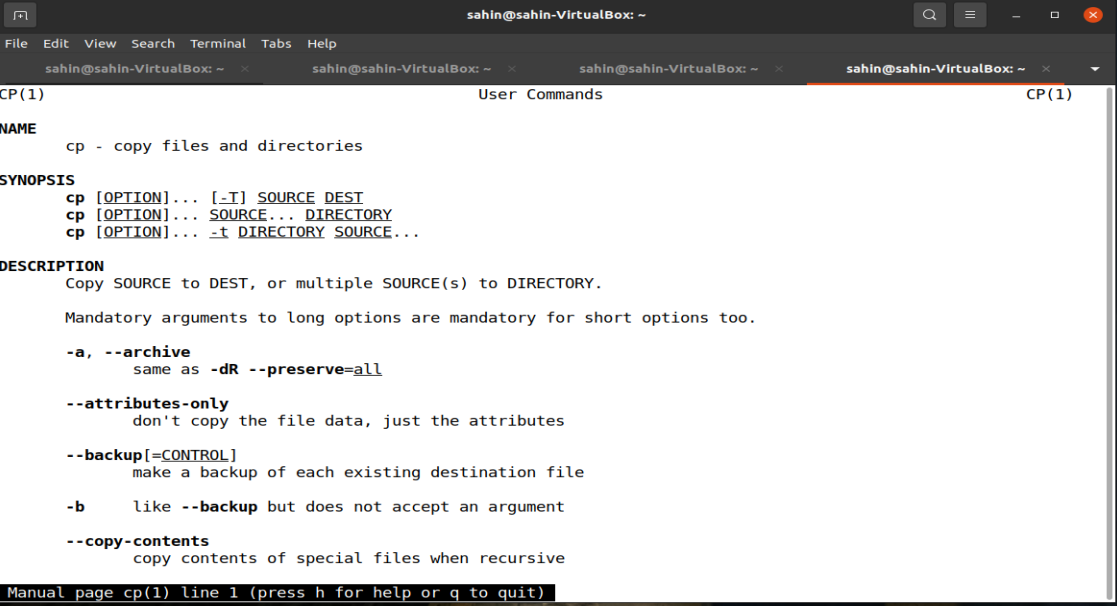
**6. Show the command used to display (i) filenames (ii) processes (iii) users.**



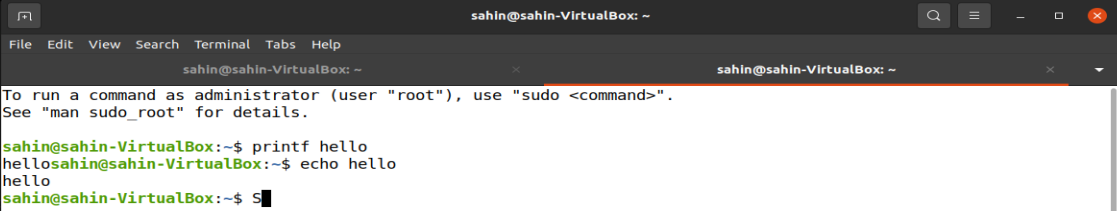




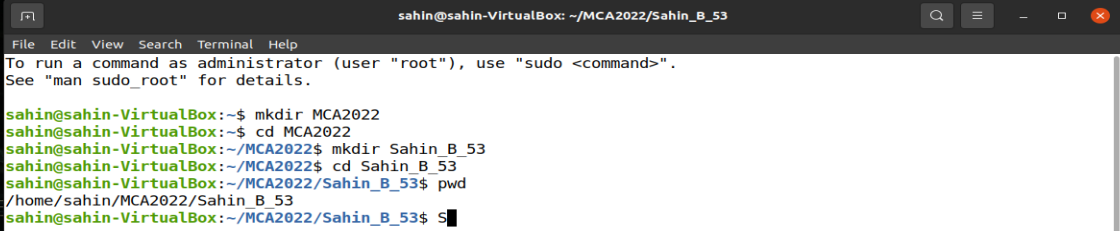
**7. Check and state the difference between man and whats command by checking man cp & whatis cp**



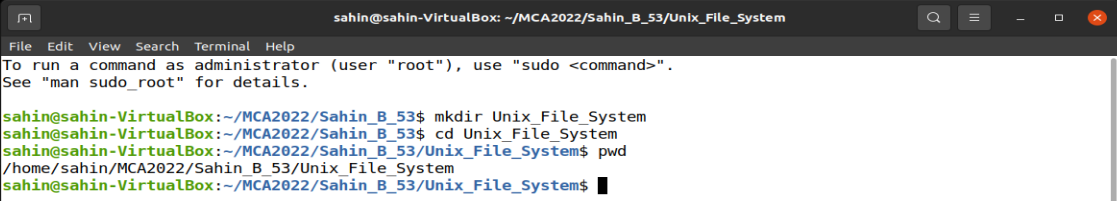
**8. What is primary difference between printf and echo command. Check and print.**



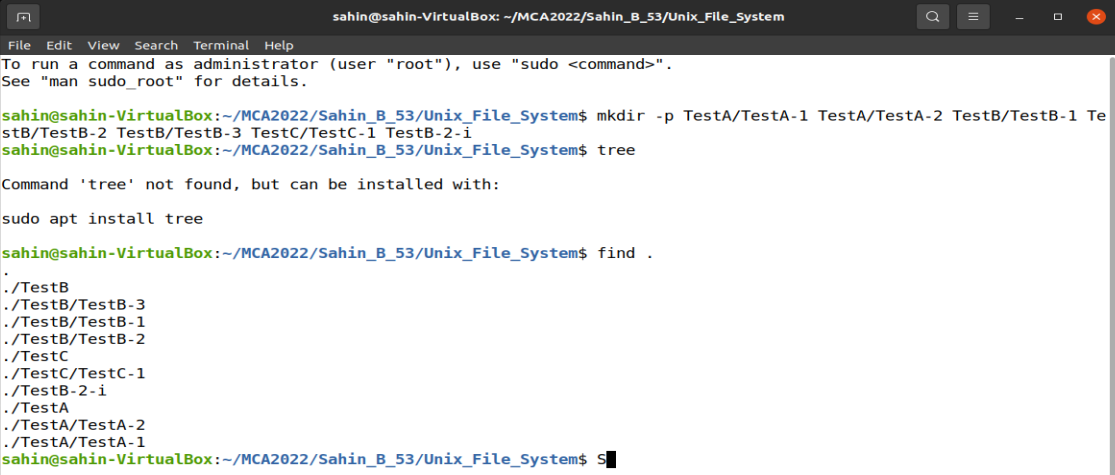
**9. In the home directory, create a directory MCA2022 inside the MCA2022, create another directory <FirstName\_Section\_ClassRoll> and get into the directory [-/MCA2022/Ankur\_A\_00$].**



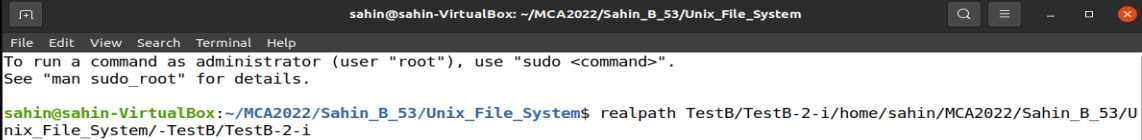
**10. Go to the subdirectory and create another subdirectory ‘Unix\_File\_System’ within it.**



**11. Create the subdirectories TestA,TestB,TestC and corresponding sub-directories TestA-1,TestA-2, TestB-1,TestB-2, TestC-1,TestB-2-i in a single command.**



**12. Show the absolute path of TestB-2-i.**

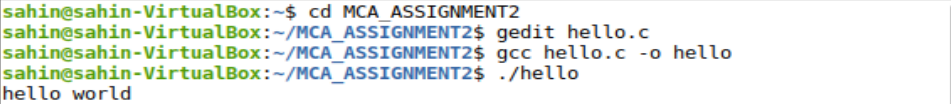


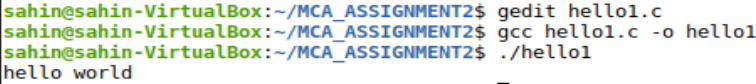
**ASSIGNMENT-2**

**1. Create two C files to print “Hello World!” in two different ways:**

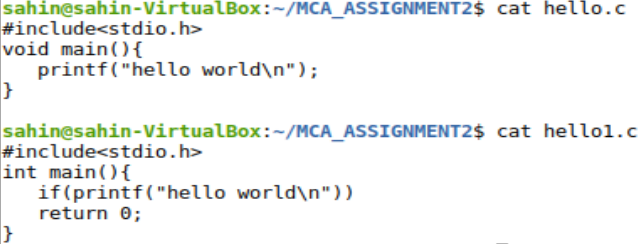
**a. Program containing normal statement terminator → HelloWorld1.c.**

**b. Program without any statement terminator → HelloWorld2.c.**

****

****

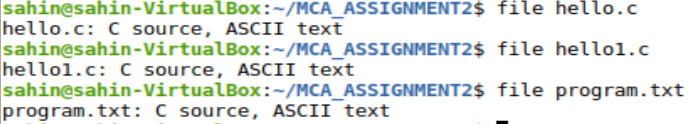
**2. Display the contents of the files.**

****

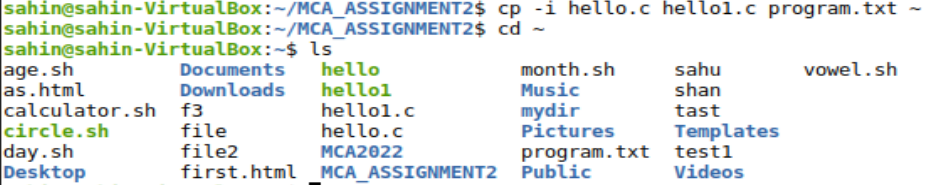
**3. Concatenate the two files to a third file.**

****

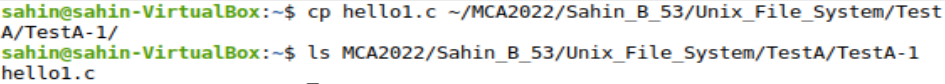
**4. Show the above file types.**

****

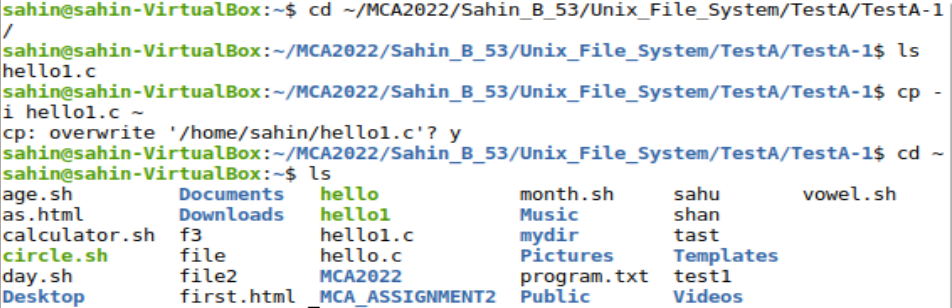
**5. Copy all the files to the home directory in an interactive manner.**

****

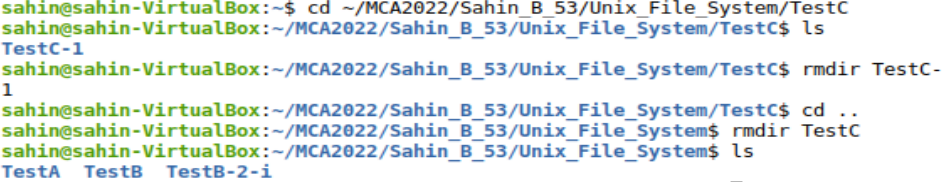
**6. Create a copy of the C file in TestA-1.**

****

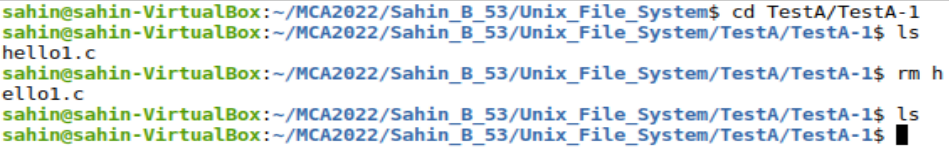
**7. Copy the file to the home directory in an interactive manner.**

****

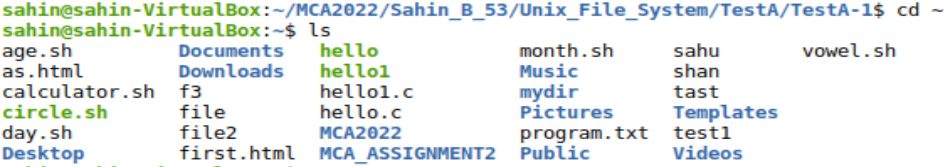
**8. Remove the directories TestC & TestC-1.**

****

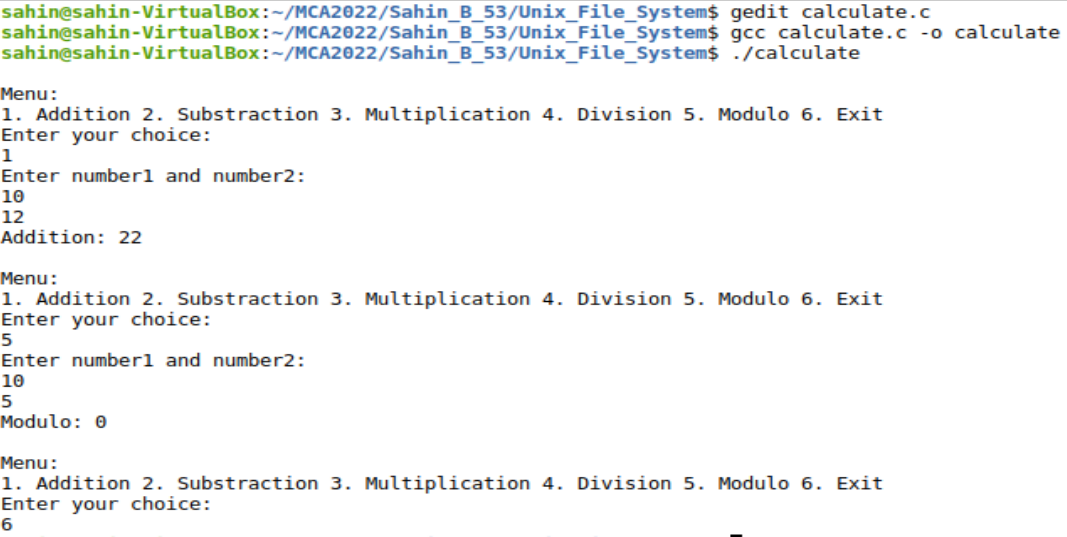
**9. Delete the file C file from TestA-1.**

****

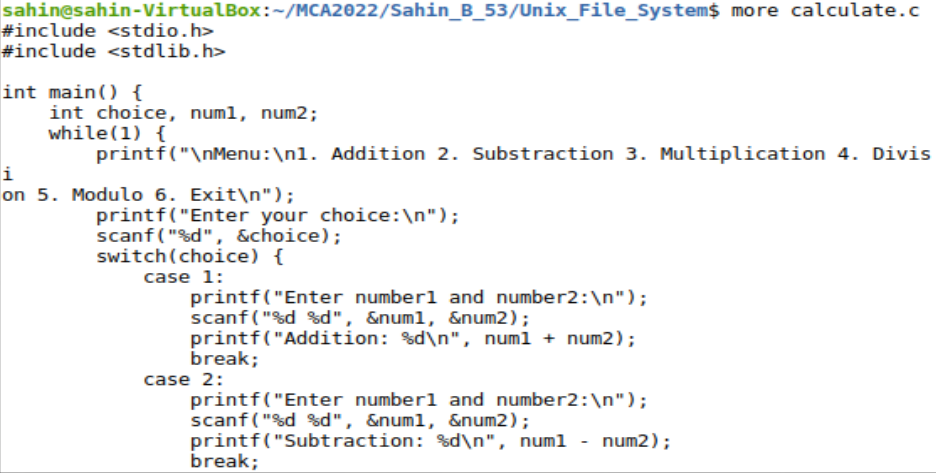
**10. Rename the text file in the home directory.**

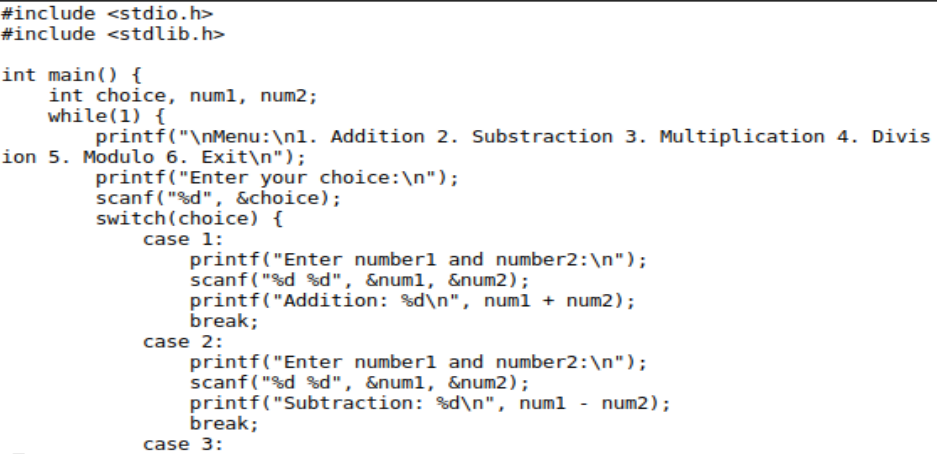
****

**11. Rename the text file in the home directory.**

****

**12. Show the C file in the paged manner using more and less commands.**

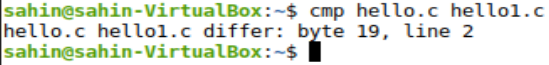
****

****

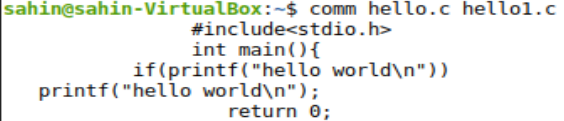
**13. Count the number of lines, words and characters separately.**



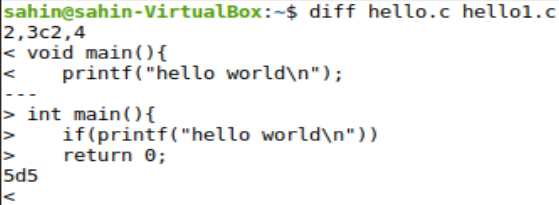
**14. Compare the two C files.**



**15. Find what is common in two C files.**

****

**16. Find the difference in two C files.**

****

**ASSIGNMENT-3**

**1. Write a C programme to simulate the following non-preemptive CPU scheduling algorithms to find the turnaround time and waiting time for the above problem. A. FCFS B. SJF C. Priority**

**A. FCFS cpu scheduling algorithm.. a. For the FCFS scheduling algorithm, read the number of processes/jobs in the system, and their CPU burst times.**  **b. The scheduling is performed based on the arrival time of the processes, irrespective of their other parameters. c. Each process will be executed according to its arrival time. . d. Each process will be executed according to its arrival time.**

**B. SJF cpu scheduling algorithm.. a. For the SJF scheduling algorithm, read the number of processes/jobs in the system, and their CPU burst times**.  **b. Arrange all the jobs in order with respect to their burst times. c. Two jobs may be in queue with the same execution time, and then the FCFS approach will be performed. d. Each process will be executed according to the length of its burst time.**

**e. Then calculate each process's waiting time and turnaround time accordingly.**

**C. PRIORITY cpu scheduling algorithm.. a. For the priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities.**  **b. Arrange all the jobs in order with respect to their priorities. c. There may be two jobs in queue with the same priority, and then FCFS approach will be performed.**  **d. Each process will be executed according to its priority. e. Calculate the waiting time and turnaround time of each of the processes accordingly.**

#include <stdio.h>

struct Process {

int id;

int burst\_time;

int arrival\_time;

int priority;

int waiting\_time;

int turnaround\_time; };

void sortByArrivalTime(struct Process proc[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (proc[j].arrival\_time > proc[j + 1].arrival\_time) {

struct Process temp = proc[j];

proc[j] = proc[j + 1];

proc[j + 1] = temp; }}}}

void sortByBurstTime(struct Process proc[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (proc[j].burst\_time > proc[j + 1].burst\_time) {

struct Process temp = proc[j];

proc[j] = proc[j + 1];

proc[j + 1] = temp;}}}}

void sortByPriority(struct Process proc[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (proc[j].priority > proc[j + 1].priority) {

struct Process temp = proc[j];

proc[j] = proc[j + 1];

proc[j + 1] = temp; }}}}

void calculateTimes(struct Process proc[], int n) {

int total\_waiting\_time = 0, total\_turnaround\_time = 0;

proc[0].waiting\_time = 0;

for (int i = 1; i < n; i++) {

proc[i].waiting\_time = proc[i - 1].waiting\_time + proc[i - 1].burst\_time; }

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

proc[i].turnaround\_time = proc[i].waiting\_time + proc[i].burst\_time;

total\_waiting\_time += proc[i].waiting\_time;

total\_turnaround\_time += proc[i].turnaround\_time;}

printf("Process ID | Burst Time | Arrival Time | Priority | Waiting Time | Turnaround Time\n");

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

printf(" %d | %d | %d | %d | %d | %d\n", proc[i].id, proc[i].burst\_time, proc[i].arrival\_time, proc[i].priority, proc[i].waiting\_time, proc[i].turnaround\_time);}

printf("\nAverage Waiting Time: %.2f\n", (float)total\_waiting\_time / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_turnaround\_time / n);}

int main() {

int n, choice;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process proc[n];

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

printf("Enter burst time for process %d: ", i + 1);

scanf("%d", &proc[i].burst\_time);

printf("Enter arrival time for process %d: ", i + 1);

scanf("%d", &proc[i].arrival\_time);

printf("Enter priority for process %d: ", i + 1);

scanf("%d", &proc[i].priority);

proc[i].id = i + 1;}

printf("\nChoose the scheduling algorithm:\n");

printf("1. First Come First Serve (FCFS)\n");

printf("2. Shortest Job First (SJF)\n");

printf("3. Priority\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

sortByArrivalTime(proc, n);

printf("\nFCFS Scheduling\n");

break;

case 2:

sortByBurstTime(proc, n);

printf("\nSJF Scheduling\n");

break;

case 3:

sortByPriority(proc, n);

printf("\nPriority Scheduling\n");

break;

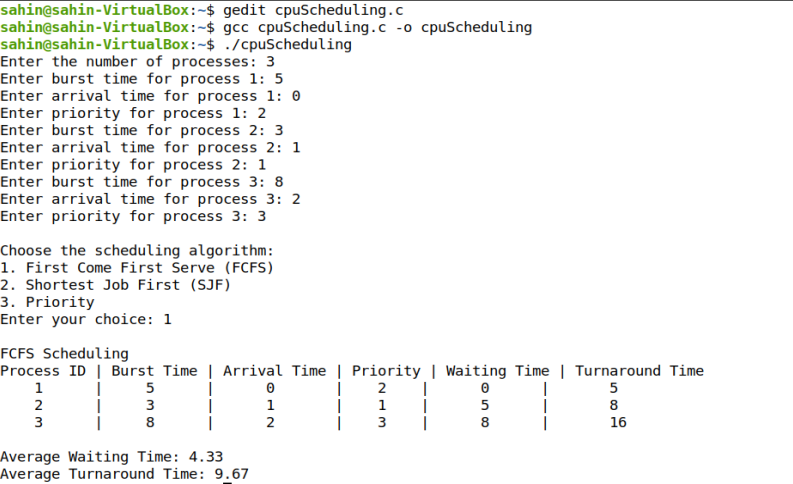
default:

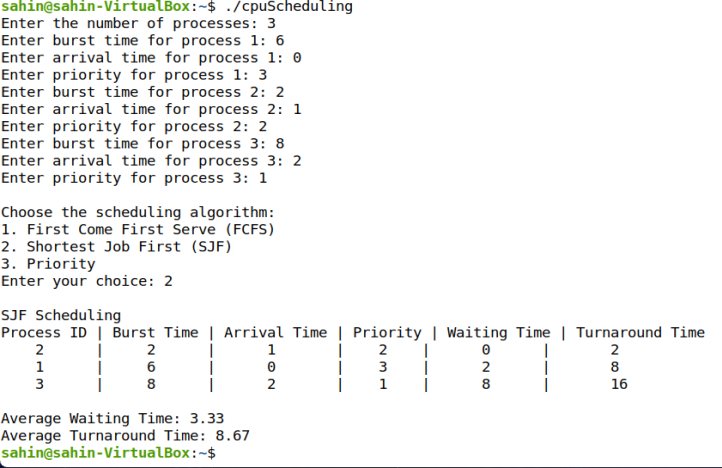
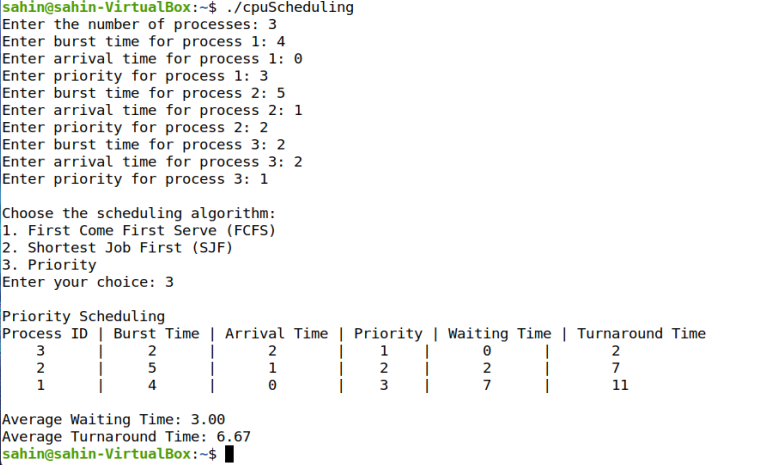
printf("Invalid choice\n");

return 1;}

calculateTimes(proc, n);

return 0;}

****

** **

**ASSIGNMENT-4**

**1. Write a C program to simulate a multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

typedef struct {

int id;

int arrival\_time;

int burst\_time;

} Process;

void sortByArrivalTime(Process queue[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (queue[j].arrival\_time > queue[j + 1].arrival\_time) {

Process temp = queue[j];

queue[j] = queue[j + 1];

queue[j + 1] = temp;

}}}}

void executeQueue(Process queue[], int n, const char\* queueName) {

printf("Executing %s queue (FCFS Scheduling):\n", queueName);

int time = 0;

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

if (time < queue[i].arrival\_time) {

time = queue[i].arrival\_time;}

printf("Process %d executed from time %d to %d\n", queue[i].id, time, time + queue[i].burst\_time);

time += queue[i].burst\_time;

}printf("\n");}

int main() {

Process systemQueue[MAX], userQueue[MAX];

int systemCount = 0, userCount = 0, n;

printf("Enter the number of processes: ");

scanf("%d", &n);

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

int type;

Process p;

printf("\nEnter details for Process %d\n", i + 1);

p.id = i + 1;

printf("Enter Arrival Time: ");

scanf("%d", &p.arrival\_time);

printf("Enter Burst Time: ");

scanf("%d", &p.burst\_time);

printf("Enter Type (0 for System, 1 for User): ");

scanf("%d", &type);

if (type == 0) {

systemQueue[systemCount++] = p;

}

else {

userQueue[userCount++] = p;}}

sortByArrivalTime(systemQueue, systemCount);

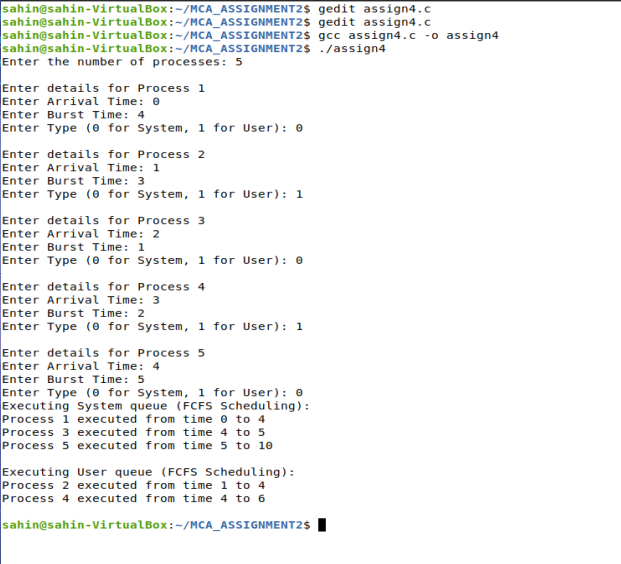
sortByArrivalTime(userQueue, userCount);

executeQueue(systemQueue, systemCount, "System");

executeQueue(userQueue, userCount, "User");

return 0;

}

****

**ASSIGNMENT-5**

**1. Write a C program to simulate the MVT and MFT memory management techniques.**

#include <stdio.h>

void mft() {

int total\_memory, block\_size, num\_blocks, num\_processes, i;

int internal\_fragmentation = 0, external\_fragmentation = 0;

int allocated\_blocks = 0;

printf("MFT MEMORY MANAGEMENT TECHNIQUE\n");

printf("Enter the total memory available (in Bytes): ");

scanf("%d", &total\_memory);

printf("Enter the block size (in Bytes): ");

scanf("%d", &block\_size);

num\_blocks = total\_memory / block\_size;

printf("Enter the number of processes: ");

scanf("%d", &num\_processes);

int memory\_required[num\_processes];

int allocated[num\_processes];

for (i = 0; i < num\_processes; i++) {

printf("Enter memory required for process %d (in Bytes): ", i + 1);

scanf("%d", &memory\_required[i]);

if (memory\_required[i] <= block\_size && allocated\_blocks < num\_blocks) {

allocated[i] = 1;

internal\_fragmentation += (block\_size - memory\_required[i]);

allocated\_blocks++;

} else { allocated[i] = 0; }}

printf("\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL FRAGMENTATION\n");

for (i = 0; i < num\_processes; i++) {

printf("%d\t%d\t\t", i + 1, memory\_required[i]);

if (allocated[i]) {

printf("YES\t\t%d\n", block\_size - memory\_required[i]);

} else {printf("NO\t\t--\n");}}

external\_fragmentation = total\_memory - (allocated\_blocks \* block\_size);

printf("\nMemory is full; the remaining processes cannot be accommodated.\n");

printf("The total internal fragmentation is %d.\n", internal\_fragmentation);

printf("Total External Fragmentation is %d\n", external\_fragmentation);

}

void mvt() {

int i, total\_memory, memory\_allocated = 0, memory\_required;

int process\_num = 0, choice;

printf("MVT MEMORY MANAGEMENT TECHNIQUE\n");

printf("Enter the total memory available (in Bytes): ");

scanf("%d", &total\_memory);

int allocated\_memory[100];

while (1) {

printf("Enter memory required for process %d (in Bytes): ", ++process\_num);

scanf("%d", &memory\_required);

if (memory\_allocated + memory\_required <= total\_memory) {

allocated\_memory[process\_num - 1] = memory\_required;

memory\_allocated += memory\_required;

printf("Memory is allocated for Process %d\n", process\_num);

} else {

printf("Memory is Full\n");

process\_num--;

break; }

printf("Do you want to continue(y=1/n=0): ");

scanf("%d", &choice);

if (choice == 0) break; }

printf("\nTotal Memory Available: %d\n", total\_memory);

printf("\nPROCESS\tMEMORY ALLOCATED\n");

for (i = 0; i < process\_num; i++) {

printf("%d\t%d\n", i + 1, allocated\_memory[i]); }

printf("\nTotal Memory Allocated is %d\n", memory\_allocated);

printf("Total External Fragmentation is %d\n", total\_memory - memory\_allocated); }

int main() {

int choice;

while (1) {

printf("\nChoose Memory Management Technique:\n");

printf("1. MFT\n2. MVT\n3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

mft();

break;

case 2:

mvt();

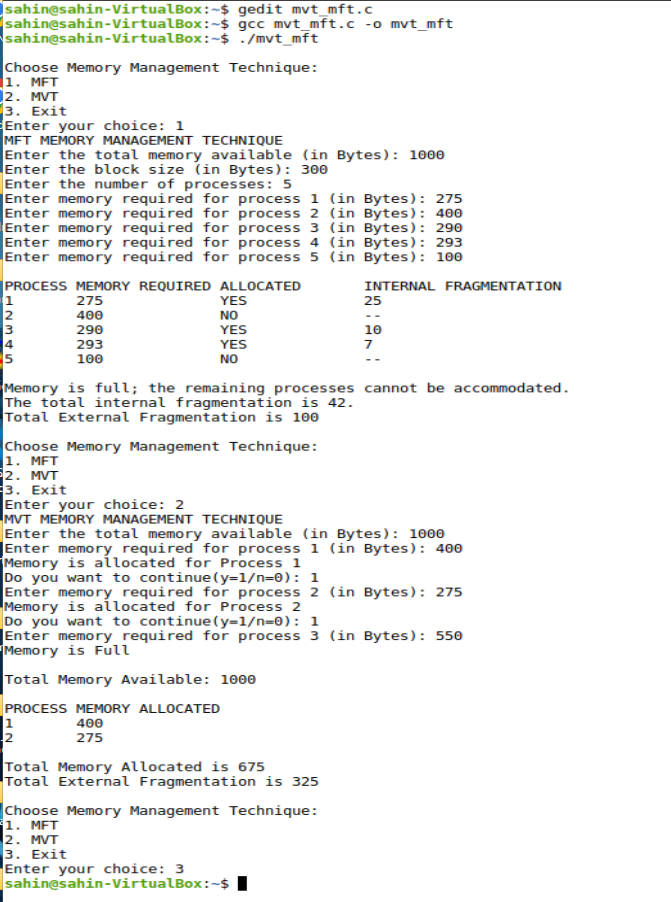
break;

case 3:

return 0;

default:

printf("Invalid choice! Please try again.\n"); } } return 0;}



**ASSIGNMENT-6**

**1. For deadlock avoidance, write a C program to simulate the Bankers algorithm. DESCRIPTION: In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, other waiting processes hold the resources a waiting process has requested, preventing it from changing state again. We refer to this situation as a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach necessitates providing the operating system with additional resources beforehand, as well as information about which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. The system considers the resources currently available, the resources allocated to each process, and the future requests and releases of each process to determine whether to satisfy the current request or delay it. Banker’s algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int isSafe(int processes[], int available[], int max[][MAX\_RESOURCES],

int allocation[][MAX\_RESOURCES], int need[][MAX\_RESOURCES], int n, int m) {

int work[MAX\_RESOURCES], finish[MAX\_PROCESSES], safeSeq[MAX\_PROCESSES];

int count = 0;

for (int i = 0; i < m; i++)

work[i] = available[i];

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

finish[i] = 0;

while (count < n) {

int found = 0;

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

if (finish[p] == 0) {

int canExecute = 1;

for (int r = 0; r < m; r++) {

if (need[p][r] > work[r]) {

canExecute = 0;

break; }}

if (canExecute) {

for (int r = 0; r < m; r++)

work[r] += allocation[p][r];

safeSeq[count++] = p;

finish[p] = 1;

found = 1; }}}

if (found == 0) {

printf("System is not in a safe state.\n");

return 0;

printf("System is in a safe state.\nSafe sequence is: ");

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

printf("%d ", safeSeq[i]);

printf("\n");

return 1;}

int main() {

int n, m; // Number of processes and resources

int processes[MAX\_PROCESSES], available[MAX\_RESOURCES];

int max[MAX\_PROCESSES][MAX\_RESOURCES], allocation[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter the number of resource types: ");

scanf("%d", &m);

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

processes[i] = i;}

printf("Enter the available instances of each resource:\n");

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

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

printf("Enter the maximum resource demand for each process:\n");

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

printf("For process %d:\n", i);

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

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

printf("Enter the allocated resources for each process:\n");

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

printf("For process %d:\n", i);

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

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

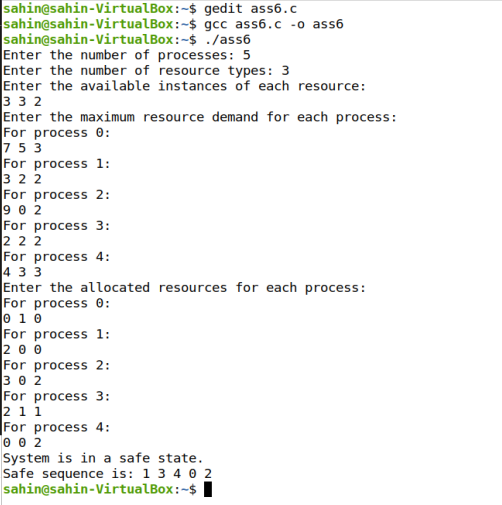
for (int i = 0; i < n; i++) {

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

need[i][j] = max[i][j] - allocation[i][j]; }}

isSafe(processes, available, max, allocation, need, n, m);

return 0; }

****