CSA0470 –

(OPERATING SYSTEMS with design principles)

K.Vaishnavi(192110431).

LABORATORY RECORD

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **SL.NO** | **DATE** | **PROGRAM NAME** | **PAGE NO** |
|  | **26-09-22** | **Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.** |  |
| **2.** | **26-09-22** | **Identify the system calls to copy the content of one file to another and illustrate the same using a C program.** |  |
| **3.** | **26-09-22** | **Design a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices.** |  |
| **4.** | **26-09-22** | **Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.** |  |
| **5.** | **26-09-22** | Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.. |  |
| **6.** | **27-09-22** | Construct a C program to simulate Round Robin scheduling algorithm with C. |  |
| **7.** | **27-09-22** | . Illustrate the concept of inter-process communication using shared memory with a C program |  |
| **8.** | **27-09-22** | Illustrate the concept of multithreading using a C program |  |
| **9.** | **27-09-22** | Design a C program to simulate the concept of Dining-Philosophers problem |  |
| **10.** | **27-09-22** | Construct a C program for implementation of memory allocation using first fit strategy |  |
| **11.** | **28-09-22** | Construct a C program to organize the file using single level directory |  |
| **12.** | **28-09-22** | Design a C program to organize the file using two level directory structure |  |
| **13.** | **28-09-22** | Develop a C program for implementing random access file for processing the employee details. |  |
| **14.** | **28-09-22** | **Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.** |  |
| **15.** | **28-09-22** | **Construct a C program to simulate producer-consumer problem using semaphores.** |  |
| **16.** | **29-09-22** | **Construct a program to simulate the First in First Out paging technique of memory management. When a page must be replaced, the oldest page is chosen.** |  |
| **17.** | **29-09-22** | **Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen** |  |
| **18.** | **29-09-22** | **Construct a program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future** |  |
| **19.** | **29-09-22** | **Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a program to simulate the file allocation strategy.** |  |
| **20.** | **29-09-22** | **Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy.** |  |
| **21.** | **29-09-22** | **With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy.** |  |
| **22.** | **29-09-22** | **Construct a C program to simulate the First Come First Served disk scheduling algorithm.** |  |
| **23.** | **29-09-22** | Design a C program to simulate SCAN disk scheduling algorithm |  |
| **24.** | **29-09-22** | Develop a C program to simulate C-SCAN disk scheduling algorithm. |  |
| **25.** | **29-09-22** | **Illustrate the various File Access Permission and different types users in Linux** |  |

|  |
| --- |
| EXPERIMENT 1:  Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program |

AIM:

Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(void) {

pid\_t pid = fork();

if(pid == 0) {

printf("Child => PPID: %d PID: %d\n", getppid(), getpid());

exit(EXIT\_SUCCESS);

}

else if(pid > 0) {

printf("Parent => PID: %d\n", getpid());

printf("Waiting for child process to finish.\n");

wait(NULL);

printf("Child process finished.\n");

}

else {

printf("Unable to create child process.\n");

}

return EXIT\_SUCCESS;

}

OUTPUT:

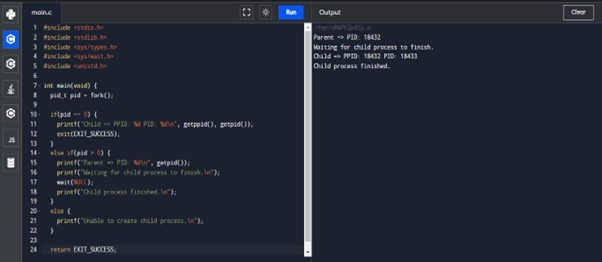
Parent => PID: 18432

Waiting for child process to finish.

Child => PPID: 18432 PID: 18433

Child process finished.

INPUT AND OUTPUT:



RESULT:

Thus the program has been successfully implemented using system call

|  |
| --- |
| EXPERIMENT 2:  Identify the system calls to copy the content of one file to another and illustrate the same using a C program. |

Aim:

To identify the system calls to copy the content of one file to another and illustrate the same using a C program.

Program:

#include <stdio.h>

#include <stdlib.h>

int main()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

printf("\nContents copied to %s", filename);

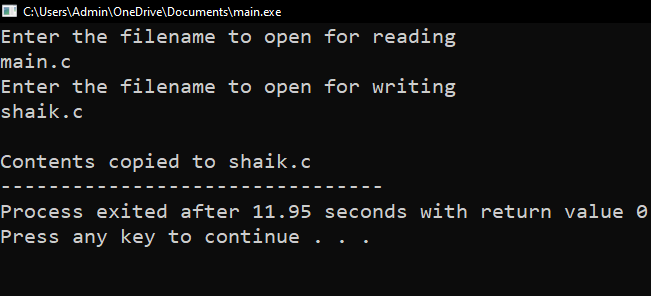
fclose(fptr1);

fclose(fptr2);

return 0;

}

INPUT AND OUTPUT:



Result: Thus the program copying one file to another has been successfully implemented using system calls.

|  |
| --- |
| EXPERIMENT 3:  Desig n a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices. |

AIM:

ToDesign a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices.

PROGRAM:

#include<stdio.h>

void main()

{

int n,bt[20],wt[20],tat[20],i,j; float avwt=0,avtat=0;printf("Enter total number of processes(maximum 20):");scanf("%d",&n);

printf("\nEnter Process Burst Time\n");for(i=0;i<n;i++)

{

printf("P[%d]:",i+1);

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

} wt[0]=0;

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

{ wt[i]=0;for(j=0;j<i;j++)

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)

{

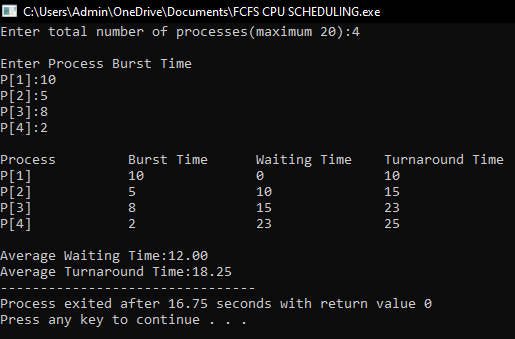
tat[i]=bt[i]+wt[i]; avwt+=wt[i]; avtat+=tat[i];printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

} avwt/=i; avtat/=i;printf("\n\nAverage Waiting Time:%.2f",avwt);

printf("\nAverage Turnaround Time:%.2f",avtat);

}

INPUT AND OUTPUT:



RESULT:

Thus the program of cpu scheduling by using “frist come first serve”technique is implemented successfully..

|  |
| --- |
| **EXPERIMENT 4:**  **Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.** |

AIM:

**Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.**

PROGRAM:

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp,floatavg\_wt,avg\_tat,avg\_wt;

printf("Enter number of process:");

scanf("%d",&n);

printf("\n Enter Burst Time:\n");for(i=0;i<n; i++)

{printf("p%d:",i+1);scanf("%d",&bt[i]);p[i]=i+1; }

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

{ pos=i;

for(j=i+1;j<n;j++)

{if(bt[j]<bt[pos]) pos=j; } temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp; temp=p[i]; p[i]=p[pos]; p[pos]=temp;

}wt[0]=0;for(i=1;i<n;i++)

{ wt[i]=0;for(j=0;j<i;j++) wt[i]+=bt[j]; total+=wt[i];

}avg\_wt=(float)total/n;

total=0;printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");for(i=0;i<n;i++)

{

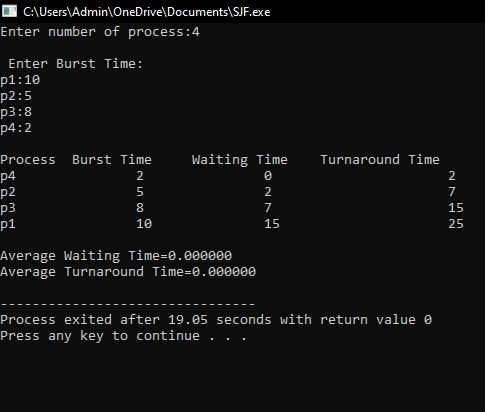
tat[i]=bt[i]+wt[i]; total+=tat[i];printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}avg\_tat=(float)total/n;printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

OUTPUT:



RESULT: Thus the c program for **scheduling program with C that selects the waiting process with the smallest execution time to execute next successfully implemented.**

|  |
| --- |
| EXPERIMENT 5: Construct a scheduling program with C that selects the waiting process with the highest priority to execute next. |

AIM:

# To Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

PROGRAM:

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:"); scanf("%d",&n); printf("\nEnter

Burst Time and Priority\n");

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

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

scanf("%d",&bt[i]); printf("Priority:"); scanf("%d",&pr[i]);

p[i]=i+1;

}

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

{

pos=i;

for(j=i+1;j<n;j++)

{

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n;

total=0;

printf("\nProcess\t Burst Time \tWaitingTime\tTurnaroundTime");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\nP[%d]\t\t %d\t\t

%d\t\t\t%d",p[i],bt[i

],wt[i],tat[i]);

}

avg\_tat=total/n;

printf("\n\nAverage Waiting

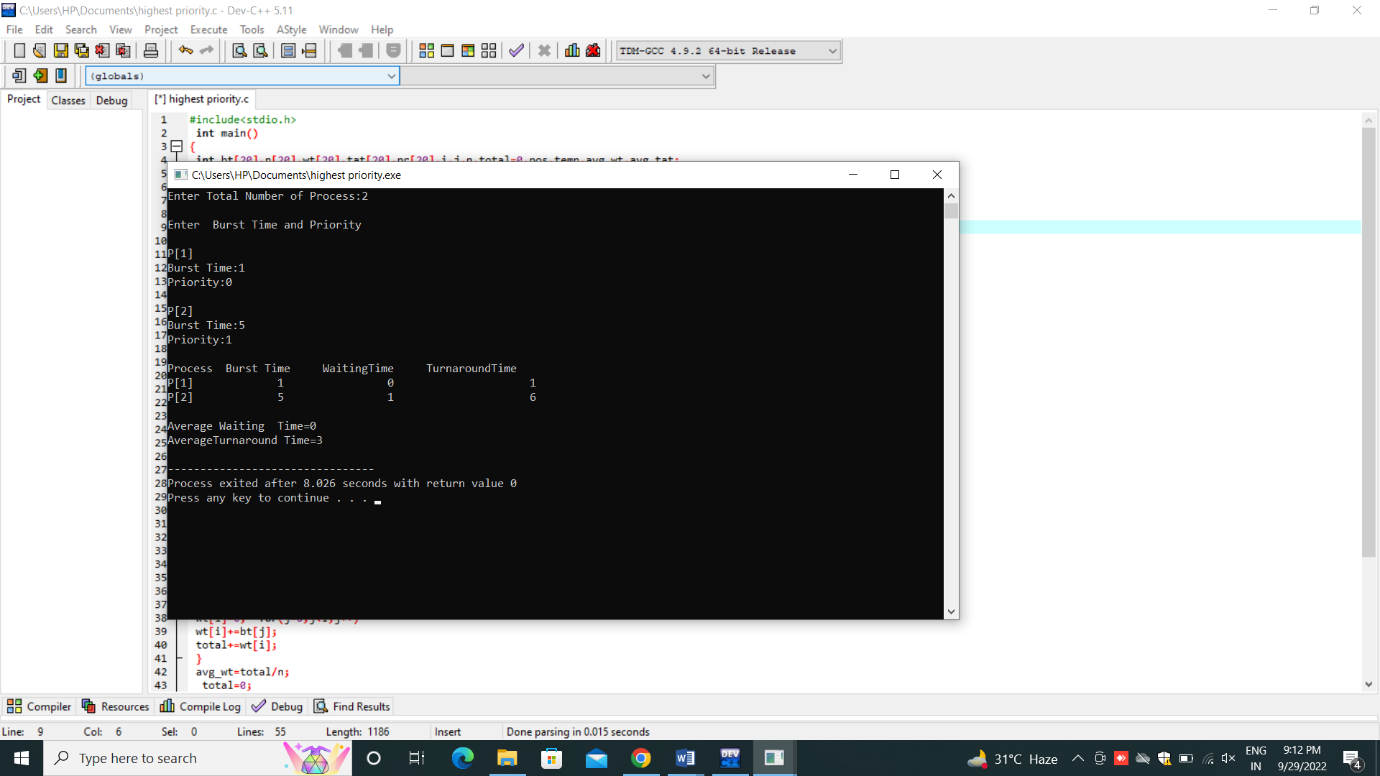
Time=%d",avg\_wt);

printf("\nAverageTurnaround Time=%d\n",avg\_tat);

return 0;

}

**OUTPUT:**



|  |
| --- |
| EXPERIMENT-6 Construct a C program to simulate Round Robin scheduling algorithm with C. |

AIM:

To Construct a C program to simulate Round Robin scheduling algorithm with C.

PROGRAM:

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(count=0;count<n;count++)

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

printf("Enter Time Quantum:\t");

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

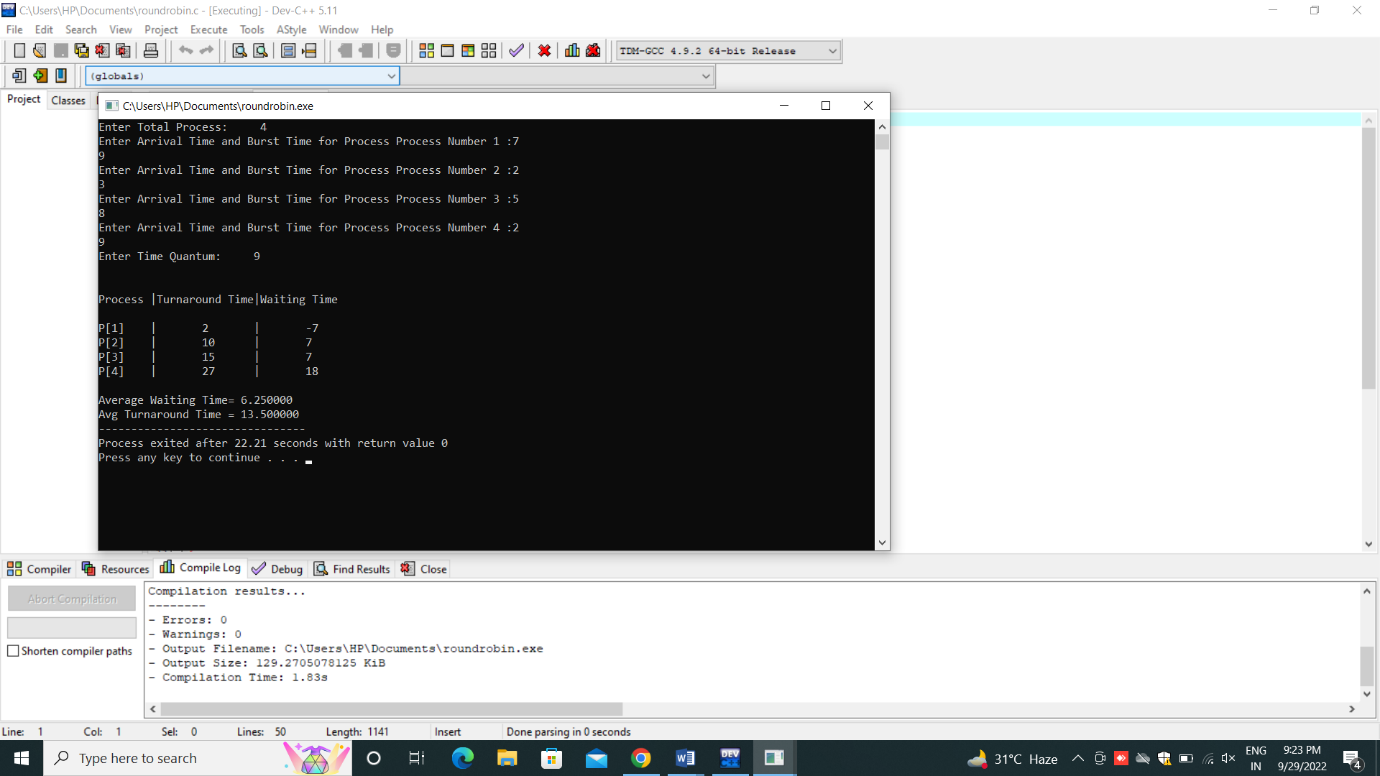
printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

**OUTPUT:**



|  |
| --- |
| EXPERIMENT-7 Illustrate the concept of inter-process communication using shared memory with a C program |

Aim:

To Illustrate the concept of inter-process communication using shared memory with a C program

PROGRAM:

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<unistd.h>**

**#include<sys/shm.h>**

**#include<string.h>**

**int main()**

**{**

**int i;**

**void \*shared\_memory;**

**char buff[100];**

**int shmid;**

**shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT);**

**printf("Key of shared memory is %d\n",shmid);**

**shared\_memory=shmat(shmid,NULL,0);**

**printf("Process attached at %p\n",shared\_memory);**

**printf("Enter some data to write to shared memory\n");**

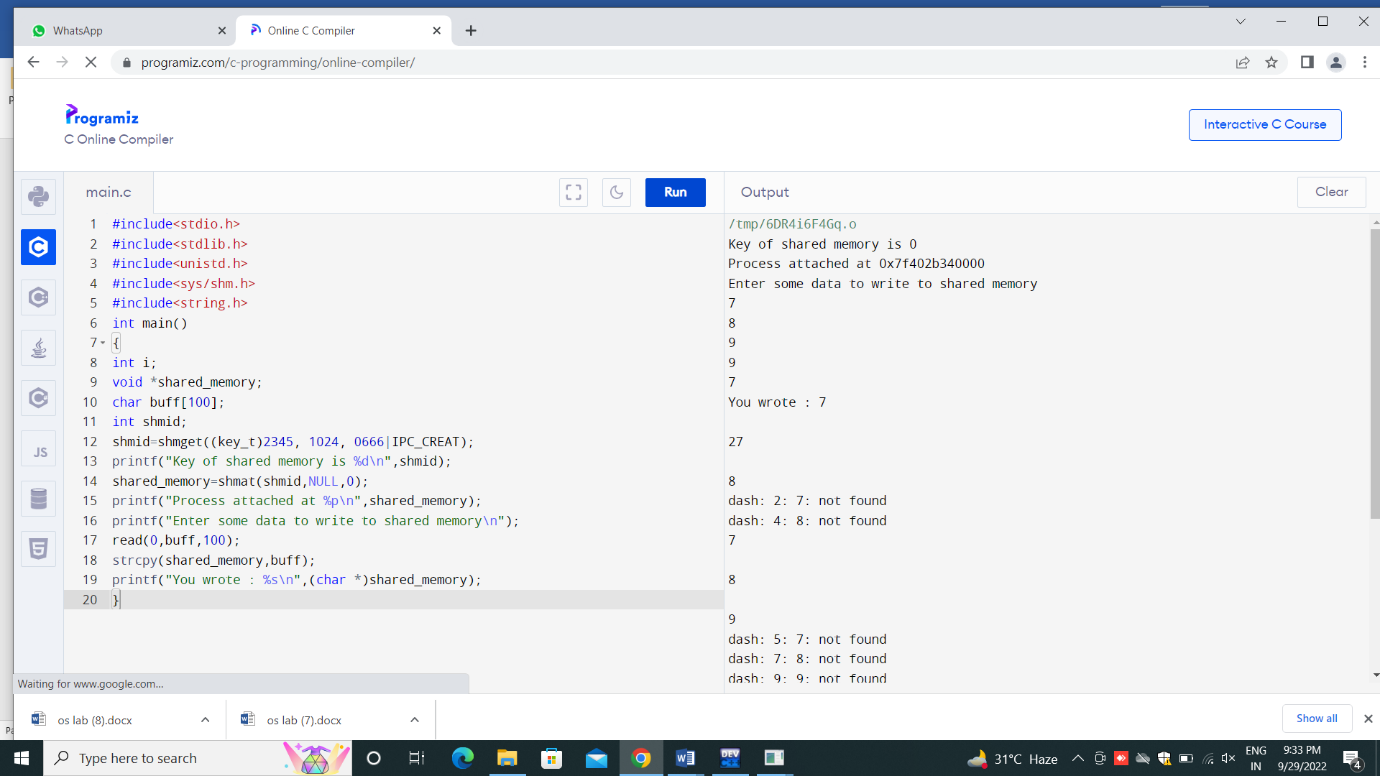
**read(0,buff,100);**

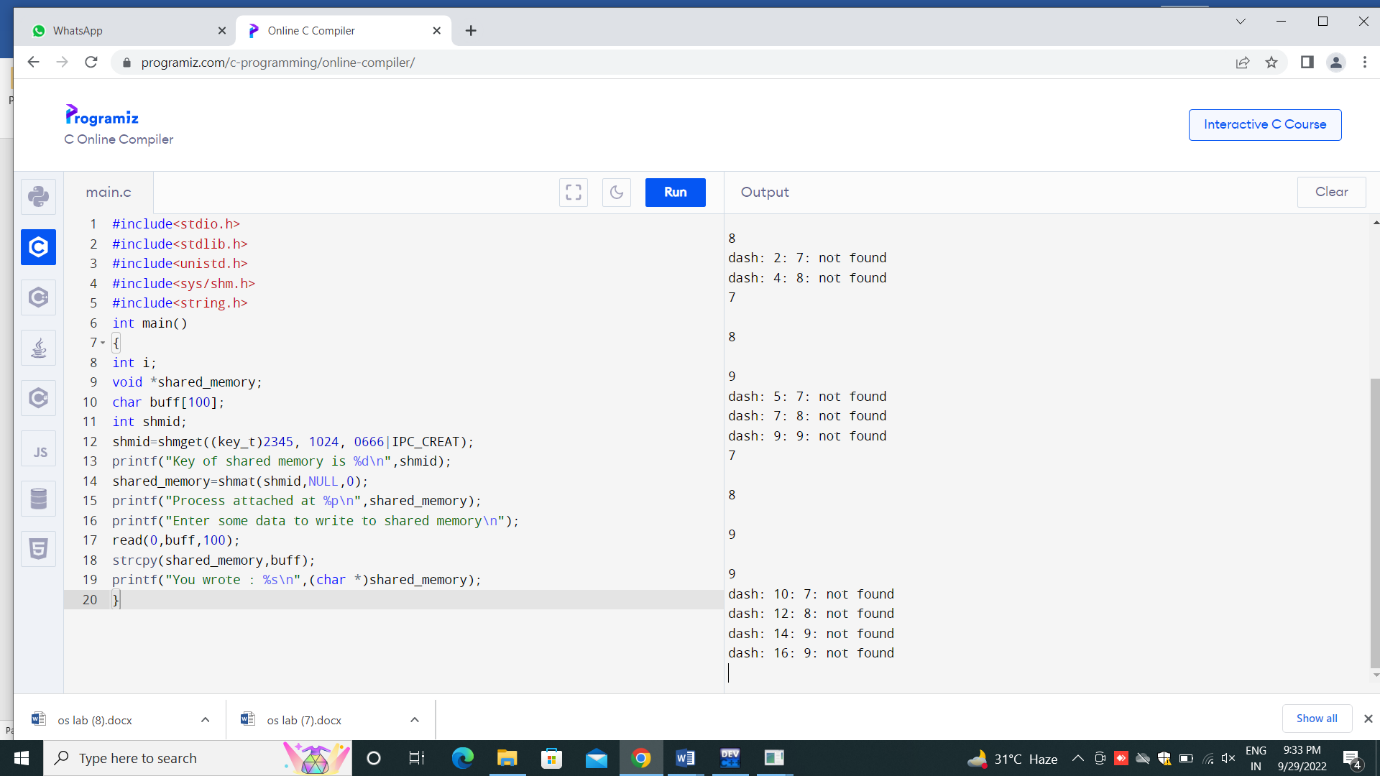
**strcpy(shared\_memory,buff);**

**printf("You wrote : %s\n",(char \*)shared\_memory);**

**}**

**OUTPUT:**

****

****

|  |
| --- |
| **EXPERIMENT 8:** Illustrate the concept of multithreading using a C program |

Aim:

# To Illustrate the concept of multithreading using a C program

PROGRAM:

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <pthread.h>**

**#include <semaphore.h>**

**#include <unistd.h>**

**int sum = 0;**

**sem\_t mutex;**

**void \*add(void \*arg){**

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

**while(\*ptr != -1){**

**sem\_wait(&mutex);**

**sum += \*ptr;**

**printf("value: %d sum %d\n", \*ptr,sum );**

**sem\_post(&mutex);**

**ptr++;**

**}**

**return NULL;**

**}**

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

**int A[4] = {1,2,3, -1};**

**int B[4] = {4,2,6, -1};**

**pthread\_t t\_a, t\_b;**

**sem\_init(&mutex, 0, 1);**

**pthread\_create(&t\_a , NULL, add, A);**

**pthread\_create(&t\_b, NULL, add, B);**

**pthread\_join(t\_a, NULL);**

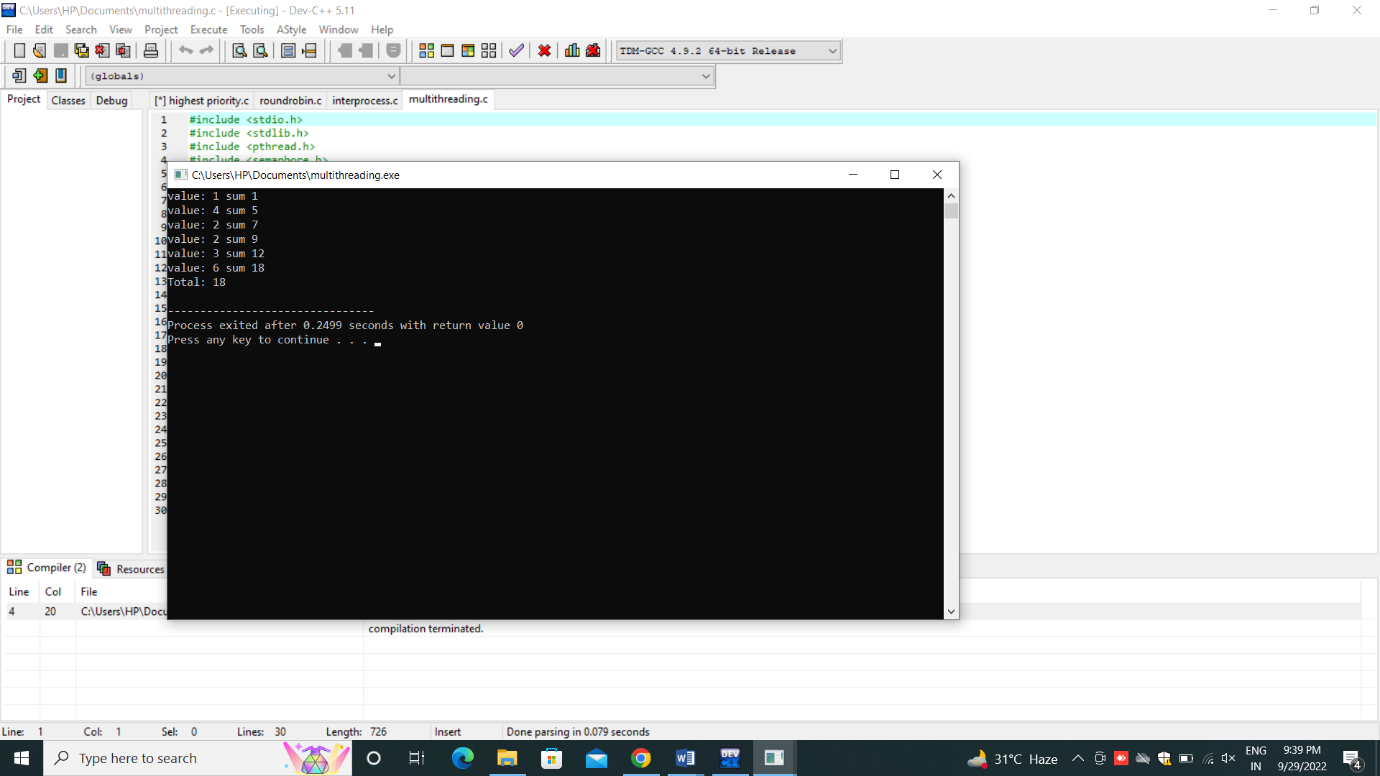
**pthread\_join(t\_b, NULL);**

**printf("Total: %d\n", sum);**

**return 0;**

**}**

**OUTPUT:**

****

|  |
| --- |
| **EXPERIMENT 9:** Design a C program to simulate the concept of Dining-Philosophers problem |

Aim:

# To Design a C program to simulate the concept of Dining-Philosophers problem

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void \* philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

for(i=0;i<5;i++)

sem\_init(&chopstick[i],0,1);

for(i=0;i<5;i++){

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);

}

for(i=0;i<5;i++)

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

}

void \* philosopher(void \* num)

{

int phil=\*(int \*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

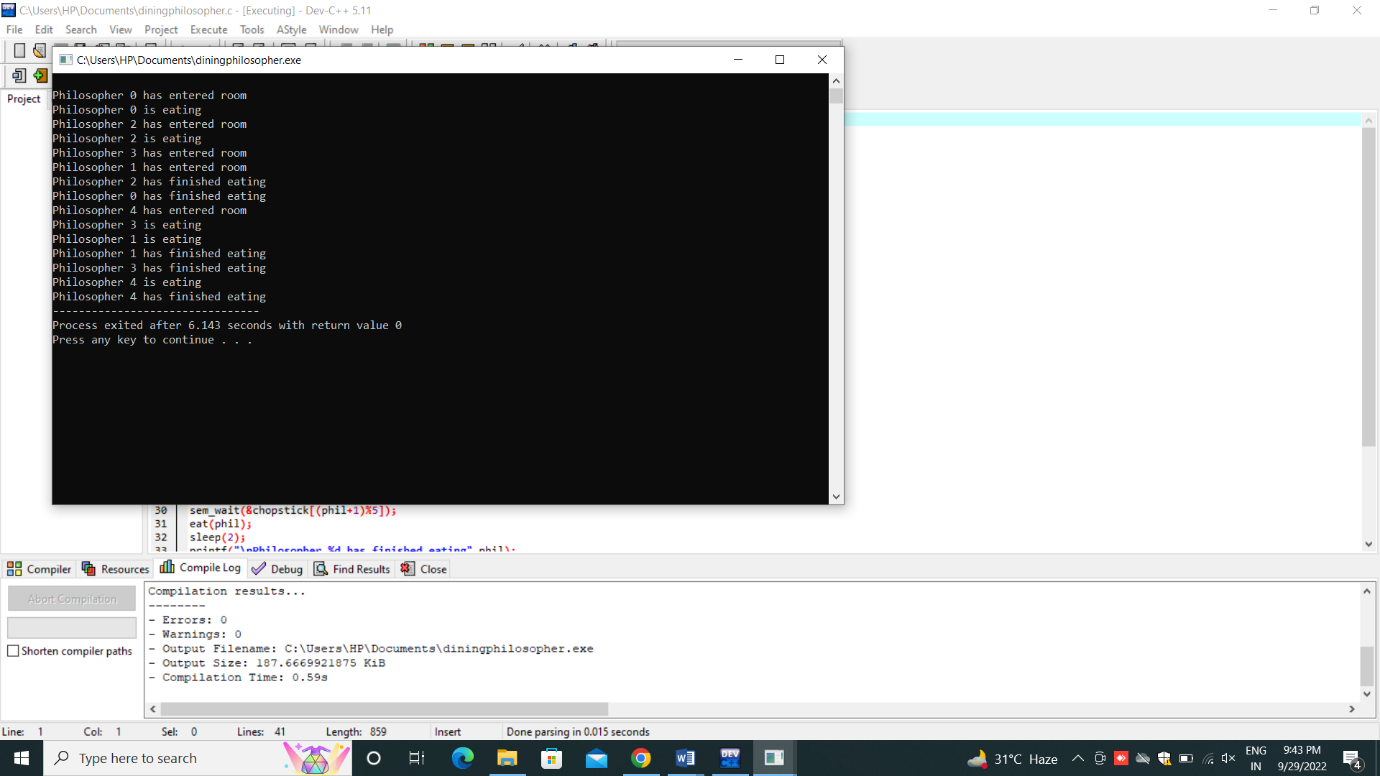
void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

**OUTPUT:**



|  |
| --- |
| **EXPERIMENT 10:** Construct a C program for implementation of memory allocation using first fit strategy |

Aim:

# Construct a C program for implementation of memory allocation using first fit strategy

PROGRAM:

#include<stdio.h>

void main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

for(i = 0; i < bno; i++)

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

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

for(i = 0; i < pno; i++)

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

for(i = 0; i < pno; i++)

for(j = 0; j < bno; j++)

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

for(i = 0; i < bno; i++)

{

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

if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

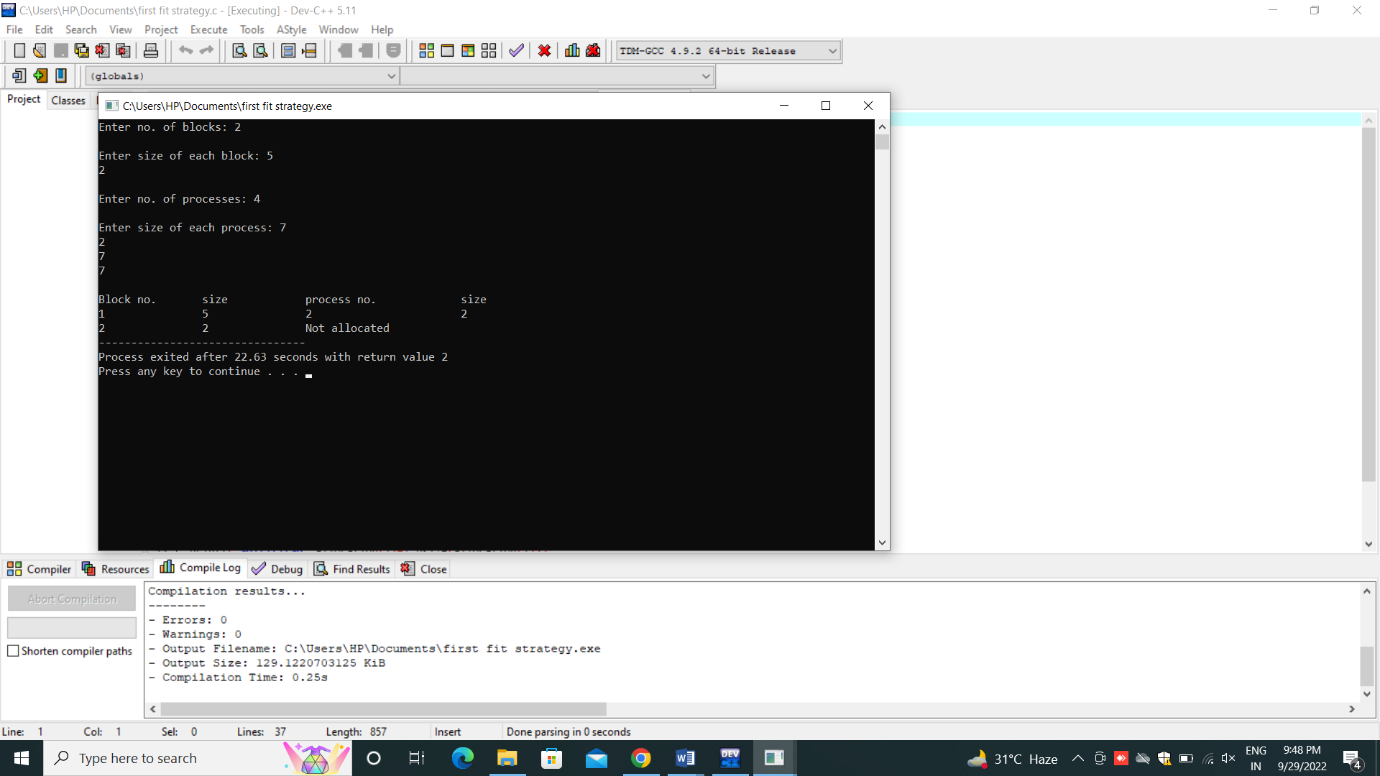
else

printf("Not allocated");

}

}

**OUTPUT:**



|  |
| --- |
| **EXPERIMENT 11:** Construct a C program to organize the file using single level directory |

Aim:

# Construct a C program to organize the file using single level directory

PROGRAM:

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

dir.fcnt = 0;

printf("\nEnter name of directory -- ");

scanf("%s", dir.dname);

while(1)

{

printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the name of the file -- ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\nEnter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is deleted ",f);

strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break; } }

if(i==dir.fcnt) printf("File %s not found",f);

else

dir.fcnt--;

break;

case 3: printf("\nEnter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is found ", f);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

break;

case 4: if(dir.fcnt==0)

printf("\nDirectory Empty");

else

{

printf("\nThe Files are -- ");

for(i=0;i<dir.fcnt;i++)

printf("\t%s",dir.fname[i]);

}

break;

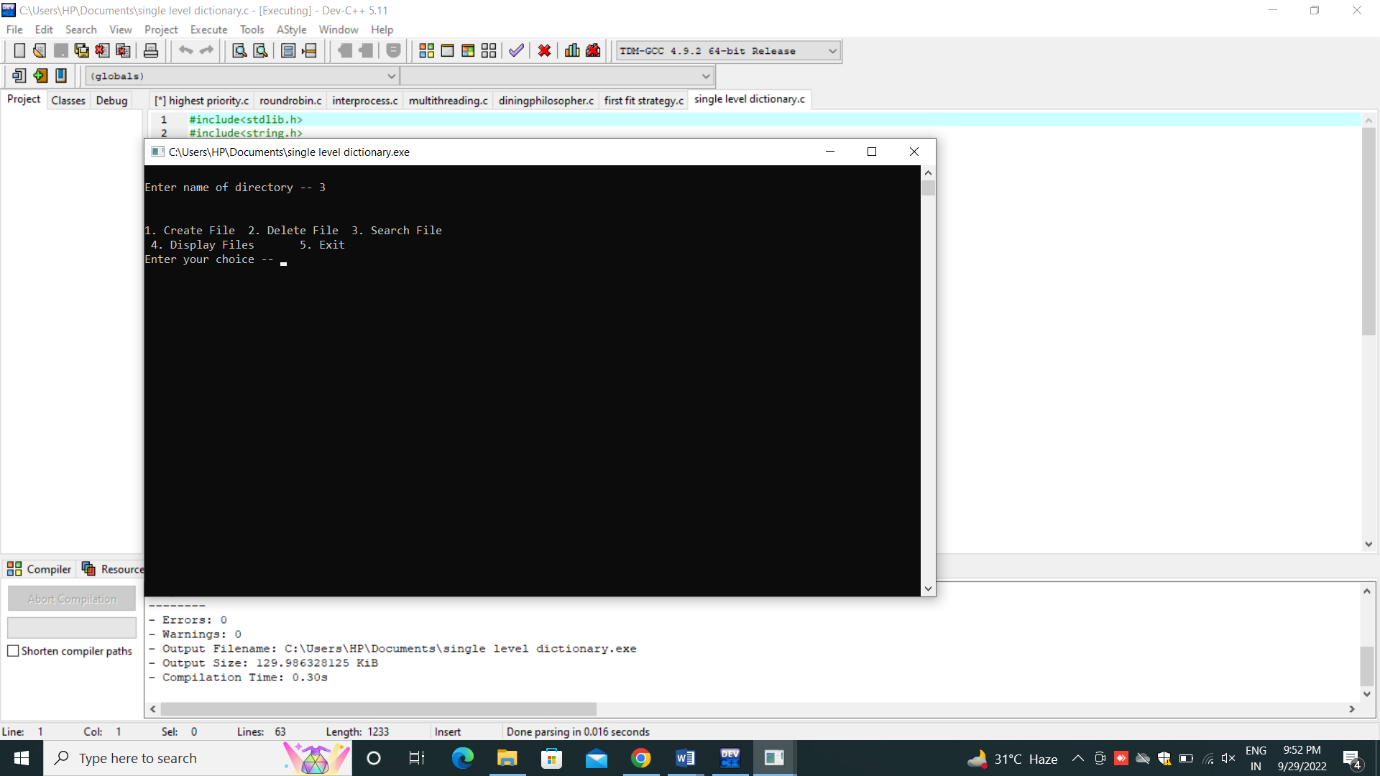
default: exit(0);

}

}

}

**OUTPUT:**



|  |
| --- |
| **EXPERIMENT 12:** Design a C program to organize the file using two level directory structure. |

Aim:

# To Design a C program to organize the file using two level directory structure

PROGRAM:

#include<string.h>

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

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir[10];

void main()

{ int i,ch,dcnt,k;

char f[30], d[30];

dcnt=0; while(1)

{

printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");

printf("\n4.Search File\t\t5. Display\t6. Exit\tEnter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter name of directory -- ");

scanf("%s", dir[dcnt].dname);

dir[dcnt].fcnt=0;

dcnt++;

printf("Directory created");

break;

case 2:

printf("\nEnter name of the directory -- ");

scanf("%s",d); for(i=0;i<dcnt;i++)

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",dir[i].fname[dir[i].fcnt]);

printf("File created");

break;

}

if(i==dcnt)

printf("Directory %s not found",d);

break;

case 3:

printf("\nEnter name of the directory -- ");

scanf("%s",d);

for(i=0;i<dcnt;i++)

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is deleted ",f); dir[i].fcnt--;

strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);

goto jmp;

}

}

printf("File %s not found",f);

goto jmp;

}

}

printf("Directory %s not found",d);

jmp :

break;

case 4: printf("\nEnter name of the directory -- ");

scanf("%s",d);

for(i=0;i<dcnt;i++)

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter the name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is found ",f);

goto jmp1;

}

}

printf("File %s not found",f);

goto jmp1;

}

}

printf("Directory %s not found",d);

jmp1: break;

case 5: if(dcnt==0)

printf("\nNo Directory's ");

else

{

printf("\nDirectory\tFiles");

for(i=0;i<dcnt;i++)

{

printf("\n%s\t\t",dir[i].dname);

for(k=0;k<dir[i].fcnt;k++)

printf("\t%s",dir[i].fname[k]);

}

}

break;

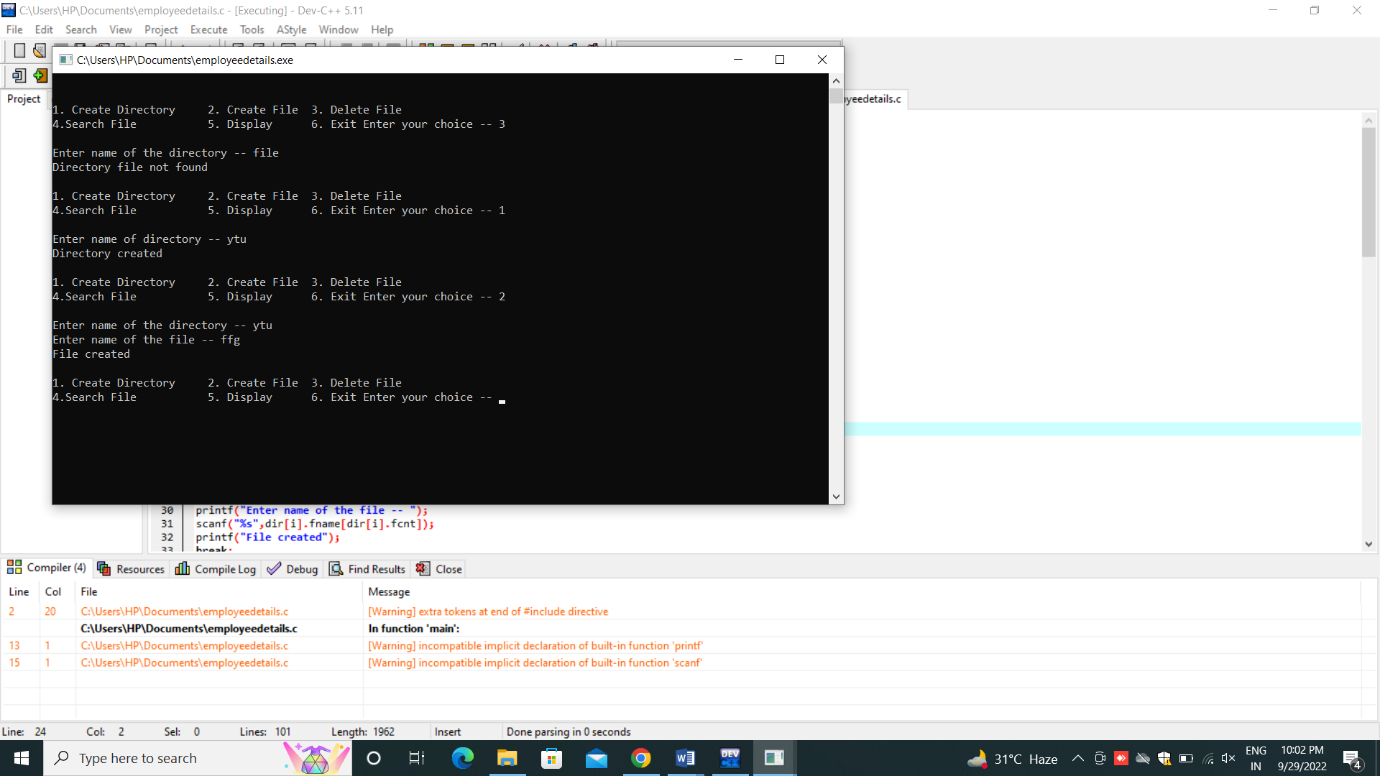
default:exit(0);

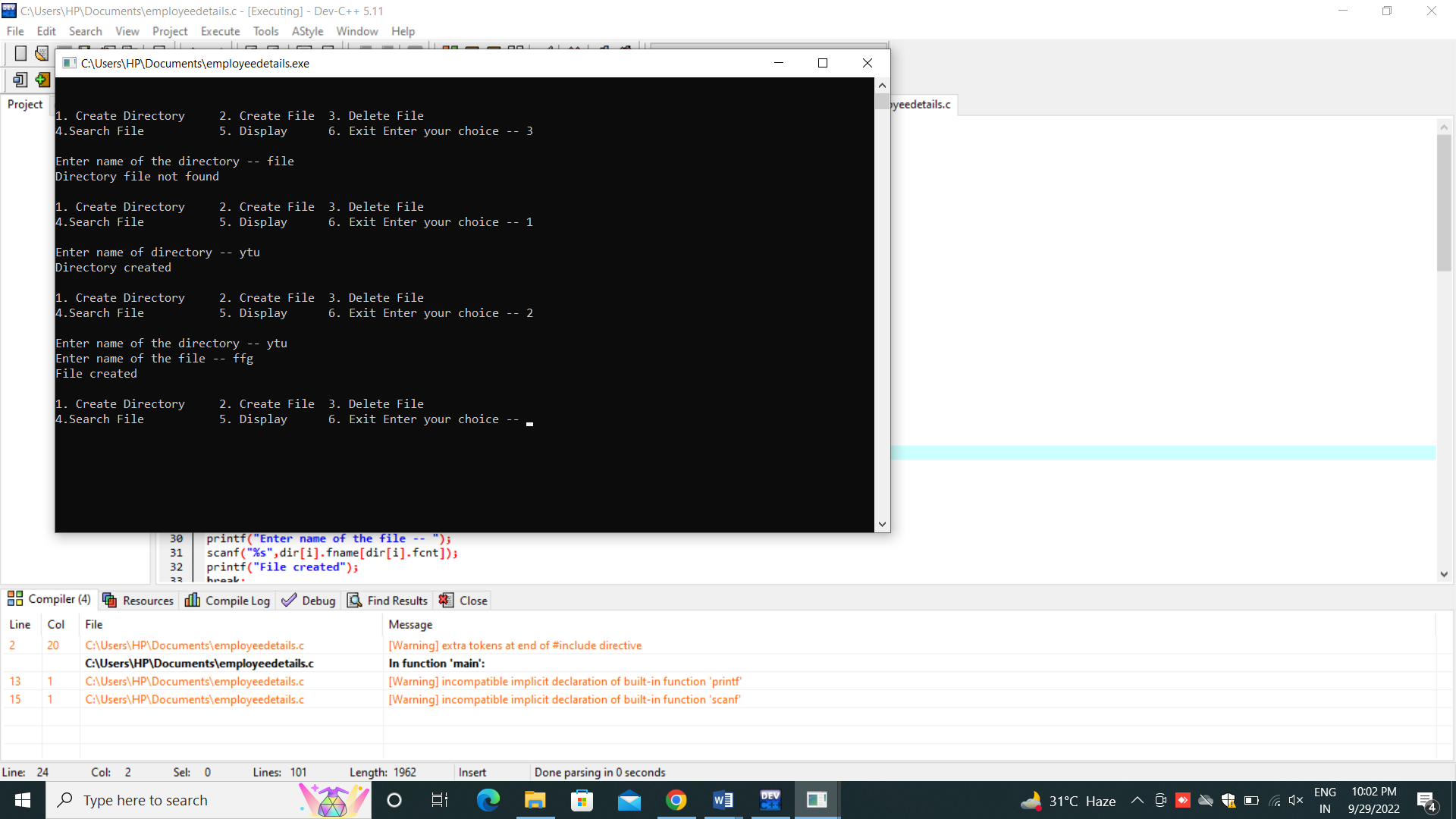
}

}

}

Output:

****



|  |
| --- |
| **EXPERIMENT 13:** Develop a C program for implementing random access file for processing the employee details. |

Aim:

# To Develop a C program for implementing random access file for processing the employee details

PROGRAM:

#include #include #include #include struct emp { char name[50]; float salary; int age; int id; }; struct emp e; long int size = sizeof(e); COORD cord = { 0, 0 }; void gotoxy(int x, int y) { cord.X = x; cord.Y = y; SetConsoleCursorPosition( GetStdHandle(STD\_OUTPUT\_HANDLE), cord); } FILE \*fp, \*ft; void addrecord() { system("cls"); fseek(fp, 0, SEEK\_END); char another = 'y'; while (another == 'y') { printf("\nEnter Name : "); scanf("%s", e.name); printf("\nEnter Age : "); scanf("%d", &e.age); printf("\nEnter Salary : "); scanf("%f", &e.salary); printf("\nEnter EMP-ID : "); scanf("%d", &e.id); fwrite(&e, size, 1, fp); printf("\nWant to add another" " record (Y/N) : "); fflush(stdin); scanf("%c", &another); } } void deleterecord() { system("cls"); char empname[50]; char another = 'y'; while (another == 'y') { printf("\nEnter employee " "name to delete : "); scanf("%s", empname); ft = fopen("temp.txt", "wb"); rewind(fp); while (fread(&e, size, 1, fp) == 1) { if (strcmp(e.name, empname) != 0) fwrite(&e, size, 1, ft); } fclose(fp); fclose(ft); remove("data.txt"); rename("temp.txt", "data.txt"); fp = fopen("data.txt", "rb+"); printf("\nWant to delete another" " record (Y/N) :"); fflush(stdin); another = getche(); } } void displayrecord() { system("cls"); rewind(fp); printf("\n=========================" "===========================" "======"); printf("\nNAME\t\tAGE\t\tSALARY\t\t" "\tID\n", e.name, e.age, e.salary, e.id); printf("===========================" "===========================" "====\n"); while (fread(&e, size, 1, fp) == 1) printf("\n%s\t\t%d\t\t%.2f\t%10d", e.name, e.age, e.salary, e.id); printf("\n\n\n\t"); system("pause"); } void modifyrecord() { system("cls"); char empname[50]; char another = 'y'; while (another == 'y') { printf("\nEnter employee name" " to modify : "); scanf("%s", empname); rewind(fp); while (fread(&e, size, 1, fp) == 1) { if (strcmp(e.name, empname) == 0) { printf("\nEnter new name:"); scanf("%s", e.name); printf("\nEnter new age :"); scanf("%d", &e.age); printf("\nEnter new salary :"); scanf("%f", &e.salary); printf("\nEnter new EMP-ID :"); scanf("%d", &e.id); fseek(fp, -size, SEEK\_CUR); fwrite(&e, size, 1, fp); break; } } printf("\nWant to modify another" " record (Y/N) :"); fflush(stdin); scanf("%c", &another); } } int main() { int choice; fp = fopen("data.txt", "rb+"); if (fp == NULL) { fp = fopen("data.txt", "wb+"); if (fp == NULL) { printf("\nCannot open file..."); exit(1); } } system("Color 3F"); printf("\n\n\n\n\t\t\t\t=============" "=============================" "==========="); printf("\n\t\t\t\t~~~~~~~~~~~~~~~~~~~" "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" "~~~~~"); printf("\n\t\t\t\t===================" "=============================" "====="); printf("\n\t\t\t\t[|:::>:::>:::>::> " "EMPLOYEE RECORD <::<:::

|  |
| --- |
| **EXPERIMENT 14:**  **Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.** |

AIM: To **illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.**

PROGRAM:

**#include<stdio.h>**

**int main()**

**{**

**int count = 0, m, n, process, temp, resource; int allocation\_table[5] = {0, 0, 0, 0, 0}; int available[5], current[5][5], maximum\_claim[5][5]; int maximum\_resources[5], running[5], safe\_state = 0; printf("\nEnter The Total Number Of Processes:\t"); scanf("%d", &process); for(m=0;m<process;m++)**

**{ running[m]=1; count++;**

**}**

**printf("\nEnter The Total Number Of Resources To Allocate:\t"); scanf("%d",&resource);**

**printf("\nEnter The Claim Vector:\t"); for(m=0;m<resource;m++)**

**{**

**scanf("%d",&maximum\_resources[m]);**

**}**

**printf("\nEnter Allocated Resource Table:\n"); for(m=0;m<process;m++)**

**{**

**for(n=0;n<resource;n++)**

**{**

**scanf("%d",&current[m][n]);**

**}**

**}**

**printf("\nEnter The Maximum Claim Table:\n");for(m=0;m<process;m++)**

**{**

**for(n=0;n<resource;n++)**

**{**

**scanf("%d",&maximum\_claim[m][n]);**

**} }**

**printf("\nThe Claim Vector \n");**

**for(m=0;m<resource;m++)**

**{printf("\t%d ",maximum\_resources[m]);**

**}**

**printf("\n The Allocated Resource Table\n"); for(m=0;m<process;m++)**

**{for(n=0;n<resource;n++)**

**{printf("\t%d",current[m][n]);**

**} printf("\n");**

**}printf("\nThe Maximum Claim Table \n"); for(m=0;m<process;m++)**

**{for(n=0;n<resource;n++)**

**{printf("\t%d",maximum\_claim[m][n]);**

**} printf("\n");**

**}for(m=0;m<process;m++)**

**{for(n=0;n<resource;n++)**

**{allocation\_table[n]=allocation\_table[n]+current[m][n];**

**}}**

**printf("\nAllocated Resources \n"); for(m=0;m<resource;m++)**

**{printf("\t%d",allocation\_table[m]);**

**}for(m=0;m<resource;m++)**

**{**

**available[m]=maximum\_resources[m]-allocation\_table[m];**

**}**

**printf("\nAvailable Resources:");**

**for(m=0;m<resource;m++)**

**{**

**printf("\t%d",available[m]);**

**} printf("\n"); while(count!=0)**

**{ safe\_state=0;**

**for(m=0;m<process;m++)**

**{**

**if(running[m])**

**{ temp=1;**

**for(n=0;n<resource;n++)**

**{**

**if(maximum\_claim[m][n]-current[m][n]>available[n])**

**{ temp=0; break;**

**}**

**} if(temp)**

**{**

**printf("\nProcess %d Is In Execution \n", m + 1); running[m]=0; count--; safe\_state=1;**

**for(n=0;n<resource;n++)**

**{available[n]=available[n]+current[m][n];**

**} break;**

**}}}if(!safe\_state)**

**{printf("\nThe Processes Are In An Unsafe State \n"); break; } else**

**{printf("\nThe Process Is In A Safe State \n"); printf("\nAvailable Vector\n");**

**for(m=0;m<resource;m++)**

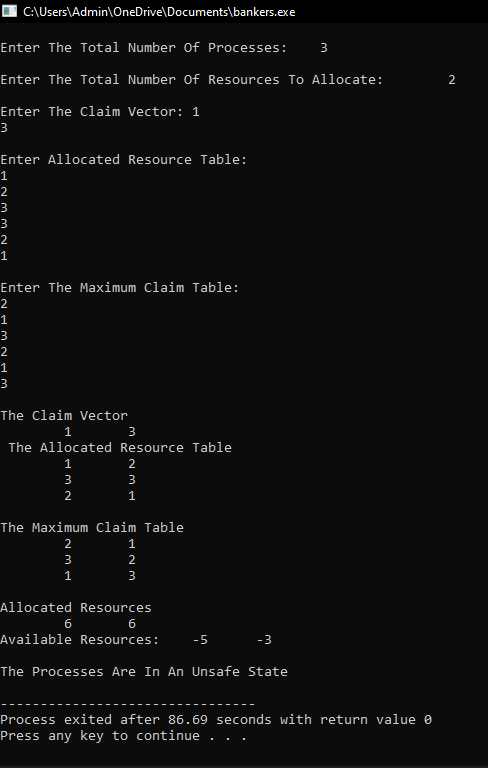
**{printf("\t%d",available[m]);**

**} printf("\n");**

**}}**

**}**

**Output:**



**RESULT:**

**Thus the program to illustrate the deadlock avoidance concept by simulating Banker’s algorithm is successfully implemented.**

|  |
| --- |
| **EXPERIMENT 15:**  **Construct a C program to simulate producer-consumer problem using semaphores.** |

**AIM:**

**To Construct a C program to simulate producer-consumer problem using semaphores**

**PROGRAM:**

**#include<stdio.h>**

**#include<stdlib.h>**

**int mutex=1,full=0,empty=3,x=0;**

**int main()**

**{**

**int n;**

**void producer();**

**void consumer();**

**int wait(int);**

**int signal(int);**

**printf("\n1.Producer\n2.Consumer\n3.Exit");**

**while(1)**

**{**

**printf("\nEnter your choice:");**

**scanf("%d",&n);**

**switch(n)**

**{**

**case 1: if((mutex==1)&&(empty!=0))**

**producer();**

**else**

**printf("Buffer is full!!");**

**break;**

**case 2: if((mutex==1)&&(full!=0))**

**consumer();**

**else**

**printf("Buffer is empty!!");**

**break;**

**case 3:**

**exit(0);**

**break;**

**}**

**}**

**return 0;**

**}**

**int wait(int s)**

**{**

**return (--s);**

**}**

**int signal(int s)**

**{ return(++s);**

**}void producer()**

**{ mutex=wait(mutex);**

**full=signal(full);**

**empty=wait(empty);**

**x++;**

**printf("\nProducer produces the item %d",x);**

**mutex=signal(mutex);**

**}void consumer()**

**{**

**mutex=wait(mutex);**

**full=wait(full);**

**empty=signal(empty);**

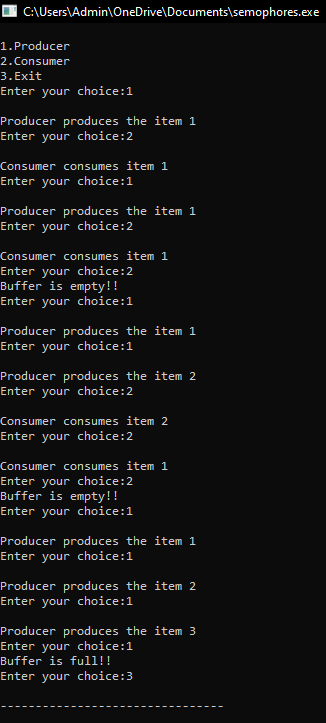
**printf("\nConsumer consumes item %d",x);**

**x--;**

**mutex=signal(mutex);**

**}**

**OUTPUT:**



**RESULT:**

**Thus the C program to simulate producer-consumer problem using semaphores is successfully implemented.**

|  |
| --- |
| **EXPERIMENT 16:**  **Construct a program to simulate the First in First Out paging technique of memory management. When a page must be replaced, the oldest page is chosen.** |

**AIM:  
 To construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen.**

PROGRAM:

#include<stdio.h>

Int main ()

{

int i,j,n,a[50],frame[10],nf,k,avail,count=0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\n ENTER THE PAGE

NUMBER:\n");for(i=1;i<=n;i++)scanf("%d",&a[i]);

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d",&nf);for(i=0;i<nf;i++)frame[i]= -1;j=0;

printf("\tref string\t page frames\n");for(i=1;i<=n;i++)

{

printf("%d\t\t",a[i]);avail=0;for(k=0;

k<nf;k++)

if(frame[k]==a[i])avail=1;

if (avail==0)

{

frame[j]=a[i];j=(j+1)%nf;c

ount++;for(k=0;k<nf;k++)

printf("%d\t",frame[k]);

}

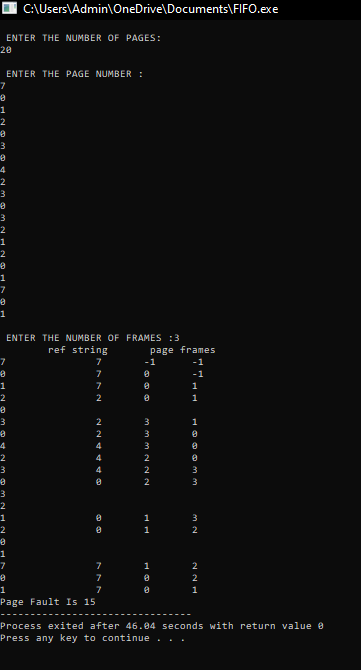
printf("\n");

}

printf("Page Fault Is %d",count);

}

OUTPUT:



RESULT:

Thus the c program to simulate the First in First Out paging technique of memory management. When a page must be replaced, the oldest page is chosen is successfully implemented.

|  |
| --- |
| **EXPERIMENT 17:**  **Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen** |

AIM:

To Construct a program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen

**PROGRAM:**

#include<stdio.h>

int findLRU(int time[], int n){int i, minimum = time[0], pos = 0;for(i = 1; i < n; ++i){if(time[i] < minimum){minimum = time[i];pos = i;

}}

return pos;

}

int main()

{int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,pos, faults = 0;printf("Enter number of frames: ");scanf("%d", &no\_of\_frames);printf("Enter number of pages: ");scanf("%d", &no\_of\_pages);printf("Enter 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){

flag1 = flag2 = 0;for(j = 0; j < no\_of\_frames; ++j){if(frames[j] == pages[i]){counter++;time[j] = counter;flag1 = flag2 = 1;

break;

}

}if(flag1 == 0){for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == -1){ counter++; faults++; frames[j] = pages[i]; time[j] = counter; flag2 = 1; break;

}}}if(flag2 == 0){ pos = findLRU(time, no\_of\_frames); counter++; faults++; frames[pos] = pages[i];

time[pos] = counter;

} printf("\n"); for(j = 0; j < no\_of\_frames; ++j){

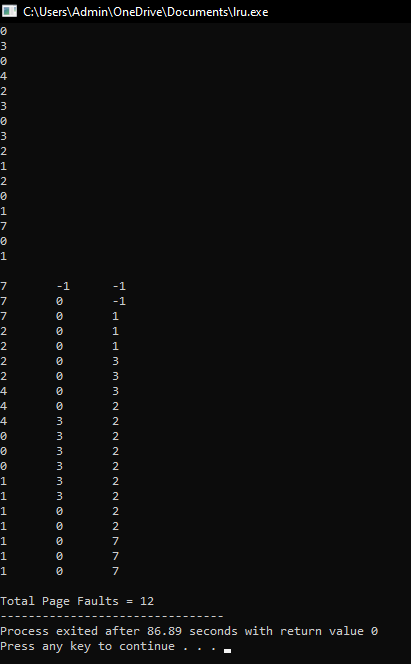
printf("%d\t", frames[j]);

}}

printf("\n\nTotal Page Faults = %d", faults); return 0;

}

OUTPUT:



RESULT: Thus the c program to simulate the Least Recently Used paging technique of memory management. When a page must be replaced, the oldest page is chosen is successfully implemented.

|  |
| --- |
| **EXPERIMENT 18:**  **Construct a program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future** |

AIM:

To Construct a program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future.

**PROGRAM:**

#include<stdio.h>

void main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k,pos, max, faults = 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){ flag1 = flag2 = 0; for(j = 0; j < no\_of\_frames; ++j){ if(frames[j] == pages[i]){ flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

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

{

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

frames[j] = pages[i];flag2 = 1;

break;

}

}

}

if(flag2 == 0){ flag3 =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; flag3 = 1; break;

}

} if(flag3 ==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++; } printf("\n");

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

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

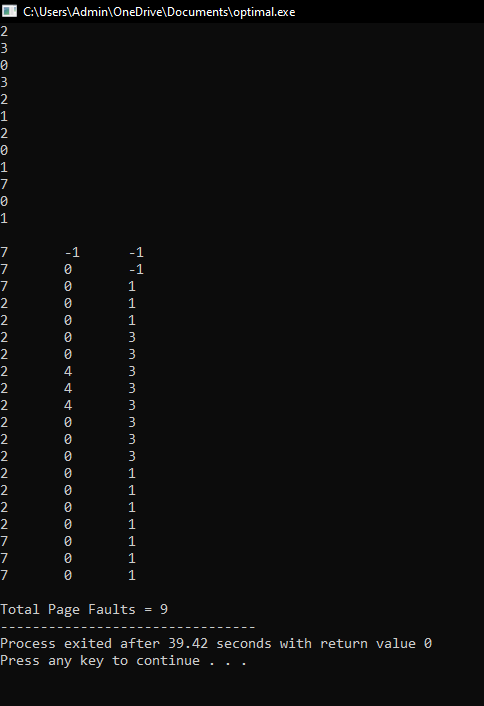
}

}

printf("\n\nTotal Page Faults = %d", faults);

}

OUTPUT:



RESULT:  
 Thus the c program to simulate the optimal paging technique of memory management. The operating system replaces the page that will not be used for the longest period of time in future is successfully implemented.

|  |
| --- |
| **EXPERIMENT 19:**  **Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a program to simulate the file allocation strategy.** |

AIM:

To Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a program to simulate the file allocation strategy.

PROGRAM:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void recurse(int files[]){

int flag = 0, startBlock, len, j, k, ch;

printf("Enter the starting block and the length of the files: ");

scanf("%d%d", &startBlock, &len);

for (j=startBlock; j<(startBlock+len); j++){

if (files[j] == 0)

flag++;

}

if(len == flag){

for (int k=startBlock; k<(startBlock+len); k++){

if (files[k] == 0){

files[k] = 1;

printf("%d\t%d\n", k, files[k]);

}

}

if (k != (startBlock+len-1))

printf("The file is allocated to the disk\n");

}

else

printf("The file is not allocated to the disk\n");

printf("Do you want to enter more files?\n");

printf("Press 1 for YES, 0 for NO: ");

scanf("%d", &ch);

if (ch == 1)

recurse(files);

else

exit(0);

return;

}

int main()

{

int files[50];

for(int i=0;i<50;i++)

files[i]=0;

printf("Files Allocated are :\n");

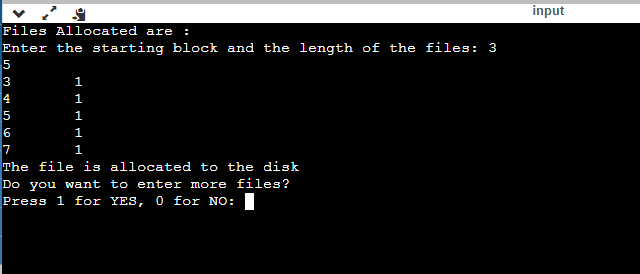
recurse(files);

getch();

return 0;

}

OUTPUT:



RESULT: Thus the c program to file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records.Design a program to simulate the file allocation strategy is successfully implemented.

|  |
| --- |
| **EXPERIMENT 20:**  **Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy.** |

AIM:

To Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy.

PROGRAM:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int files[50], indexBlock[50], indBlock, n;

void recurse1();

void recurse2();

void recurse1(){

printf("Enter the index block: ");

scanf("%d", &indBlock);

if (files[indBlock] != 1){

printf("Enter the number of blocks and the number of files needed for the index %d on the disk: ", indBlock);

scanf("%d", &n);

}

else{

printf("%d is already allocated\n", indBlock);

recurse1();

}

recurse2();

}

void recurse2(){

int ch;

int flag = 0;

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

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

if (files[indexBlock[i]] == 0)

flag++;

}

if (flag == n){

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

files[indexBlock[j]] = 1;

}

printf("Allocated\n");

printf("File Indexed\n");

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

{

printf("%d ------> %d : %d\n", indBlock, indexBlock[k], files[indexBlock[k]]);

}

}

else{

printf("File in the index is already allocated\n");

printf("Enter another indexed file\n");

recurse2();

}

printf("Do you want to enter more files?\n");

printf("Enter 1 for Yes, Enter 0 for No: ");

scanf("%d", &ch);

if (ch == 1)

recurse1();

else

exit(0);

return;

}

int main()

{

for(int i=0;i<50;i++)

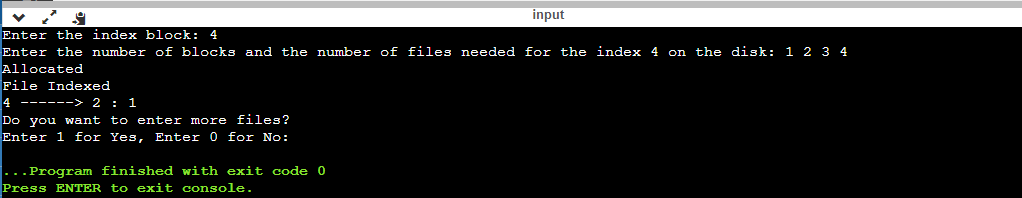
files[i]=0;

recurse1();

return 0;

}

OUTPUT:



RESULT:

Thus the c program to file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a program to simulate the file allocation strategy is successfully implemented.

|  |
| --- |
| **EXPERIMENT 21:**  **With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy.** |

AIM:

To do With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy.

PROGRAM:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void recursivePart(int pages[]){

int st, len, k, c, j;

printf("Enter the index of the starting block and its length: ");

scanf("%d%d", &st, &len);

k = len;

if (pages[st] == 0){

for (j = st; j < (st + k); j++){

if (pages[j] == 0){

pages[j] = 1;

printf("%d------>%d\n", j, pages[j]);

}

else {

printf("The block %d is already allocated \n", j);

k++;

}

}

}

else

printf("The block %d is already allocated \n", st);

printf("Do you want to enter more files? \n");

printf("Enter 1 for Yes, Enter 0 for No: ");

scanf("%d", &c);

if (c==1)

recursivePart(pages);

else

exit(0);

return;

}

int main(){

int pages[50], p, a;

for (int i = 0; i < 50; i++)

pages[i] = 0;

printf("Enter the number of blocks already allocated: ");

scanf("%d", &p);

printf("Enter the blocks already allocated: ");

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

scanf("%d", &a);

pages[a] = 1;

}

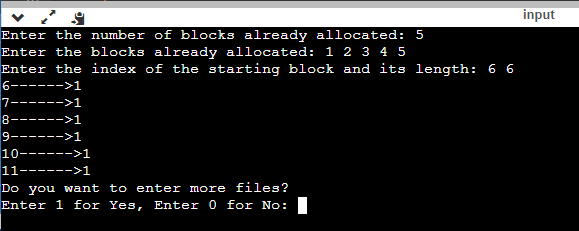
recursivePart(pages);

getch();

return 0;

}

OUTPUT:



RESULT:

Thus the c program to do With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a program to simulate the file allocation strategy is successfully implemented.

|  |
| --- |
| **EXPERIMENT 22:**  **Construct a C program to simulate the First Come First Served disk scheduling algorithm.** |

AIM:

To Construct a C program to simulate the First Come First Served disk scheduling algorithm.

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

// logic for FCFS disk scheduling

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

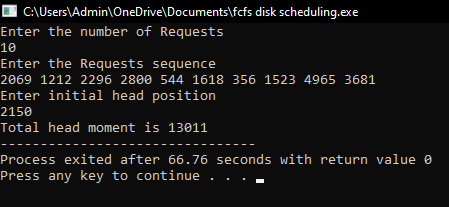
}

printf("Total head moment is %d",TotalHeadMoment);

return 0;

}

OUTPUT:



RESULT: Thus the C program to simulate the First Come First Served disk scheduling algorithm is successfully implemented.

|  |
| --- |
| **EXPERIMENT 23:** Design a C program to simulate SCAN disk scheduling algorithm |

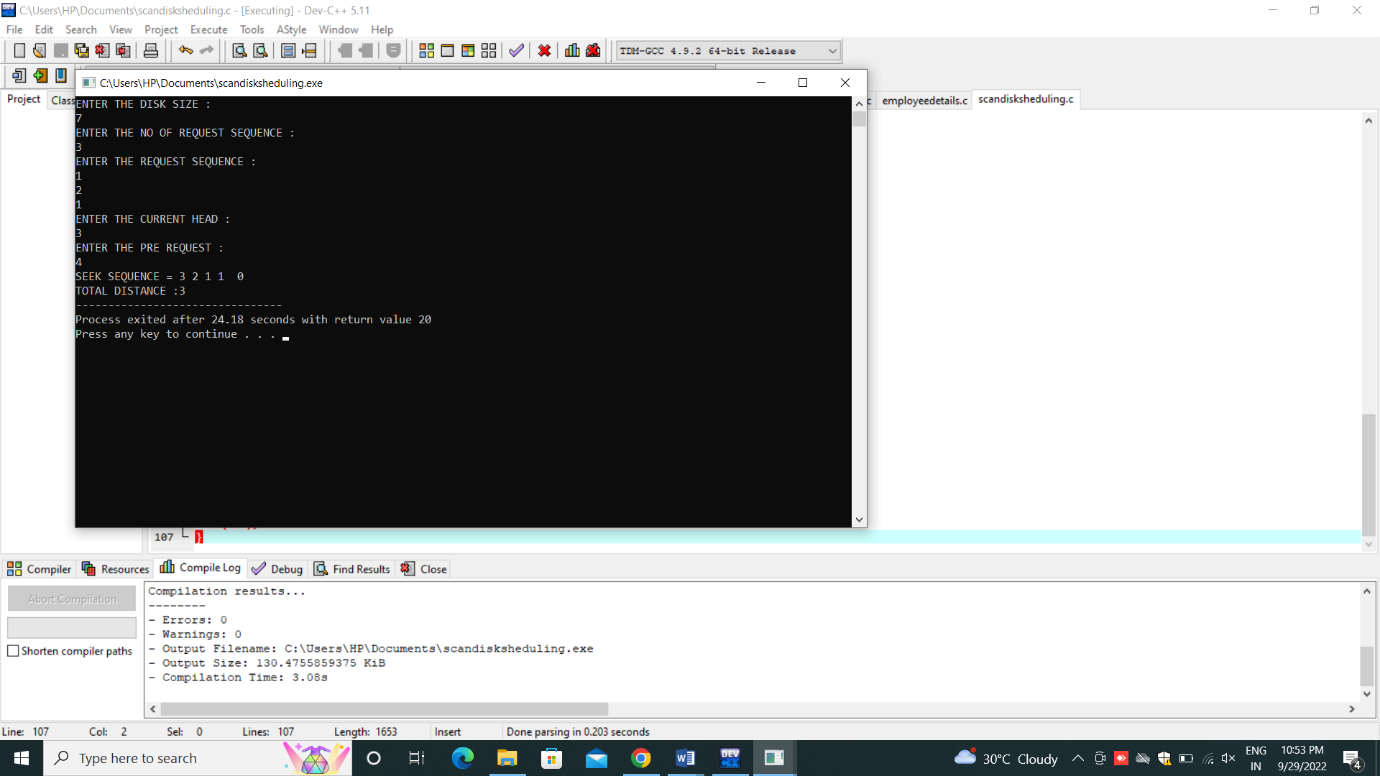
Aim:

# Design a C program to simulate SCAN disk scheduling algorithm

PROGRAM:

#include int request[50]; int SIZE; int pre; int head; int uptrack; int downtrack; struct max{ int up; int down; }kate[50]; int dist(int a,int b) { if(a>b) return a-b; return b-a; } void sort(int n) { int i,j; for (i = 0; i < n-1; i++) { for (j = 0; j < n-i-1; j++) { if (request[j] > request[j+1]) { int temp=request[j]; request[j]=request[j+1]; request[j+1]=temp; } } } j=0; i=0; while(request[i]!=head) { kate[j].down=request[i]; j++; i++; } downtrack=j; i++; j=0; while(i0;i--) { printf("%d ",head); seekcount=seekcount+dist(head,kate[i].down); head=kate[i].down; } } else { for(i=downtrack-1;i>=0;i--) { printf("%d ",head); seekcount=seekcount+dist(head,kate[i].down); head=kate[i].down; } for(i=0;i

OUTPUT:



|  |
| --- |
| **EXPERIMENT 24:** Develop a C program to simulate C-SCAN disk scheduling algorithm. |

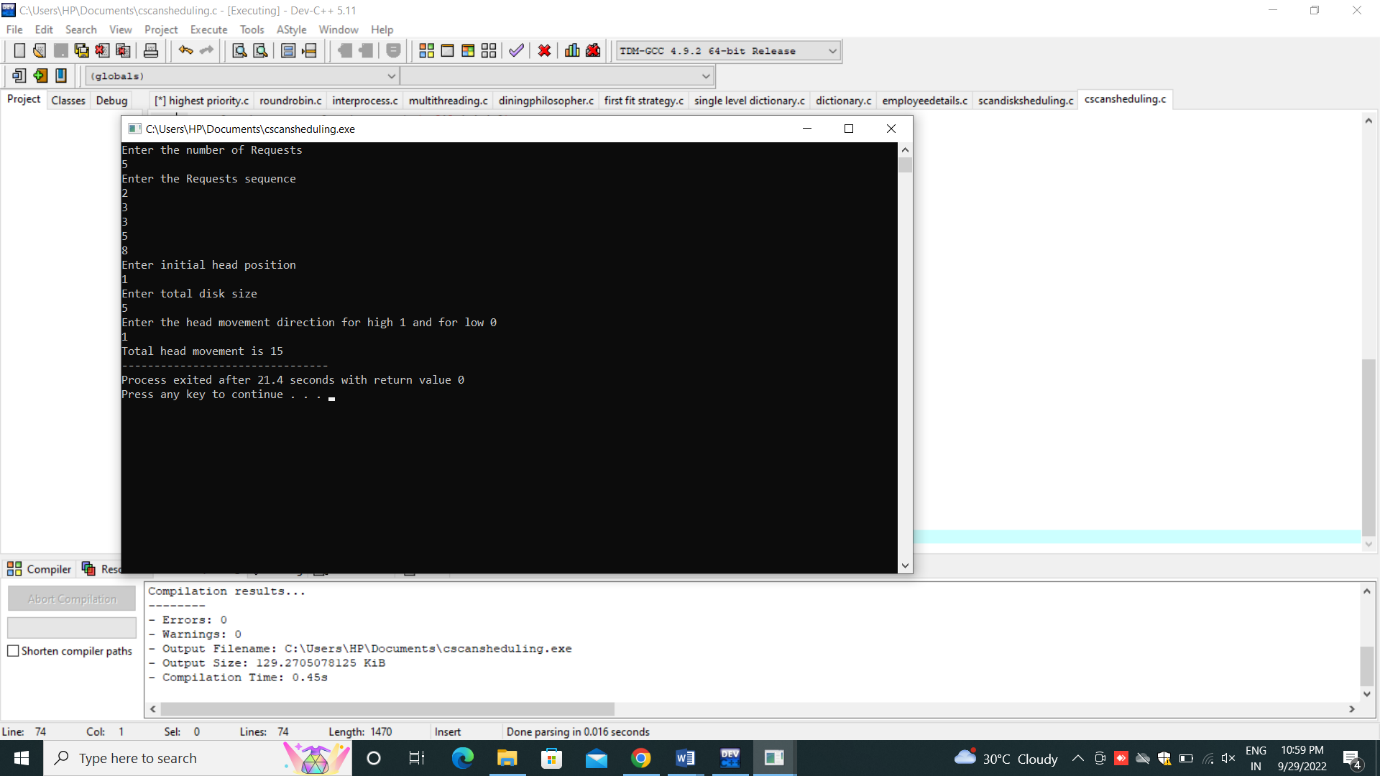
Aim:

# To Develop a C program to simulate C-SCAN disk scheduling algorithm

PROGRAM:

#include #include int main() { int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move; printf("Enter the number of Requests\n"); scanf("%d",&n); printf("Enter the Requests sequence\n"); for(i=0;iRQ[j+1]) { int temp; temp=RQ[j]; RQ[j]=RQ[j+1]; RQ[j+1]=temp; } } } int index; for(i=0;i=0;i--) { TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i]; } TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0); TotalHeadMoment=TotalHeadMoment+abs(size-1-0); initial =size-1; for(i=n-1;i>=index;i--) { TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i]; } } printf("Total head movement is %d",TotalHeadMoment); return 0; }

OUTPUT:



|  |
| --- |
| **EXPERIMENT 25:**  **Illustrate the various File Access Permission and different types users in Linux** |

AIM:

To illustrate the various File Access Permission and different types users in Linux.

1. C program to writing into a file.

AIM:

To write a simple c program to write contents into a file.

PROGRAM:

# include <stdio.h>

# include <string.h>

void main( )

{ FILE \*fp ;

char data[50];

printf( "Opening the file test.c in write mode" ) ;

fp = fopen("test.txt", "w") ;

if ( fp == NULL )

{

printf( "Could not open file test.txt" ) ;

} printf( "\n Enter some text from keyboard to write in the file test.txt" ) ; while ( strlen ( gets( data ) ) > 0 )

{ fputs(data, fp) ;

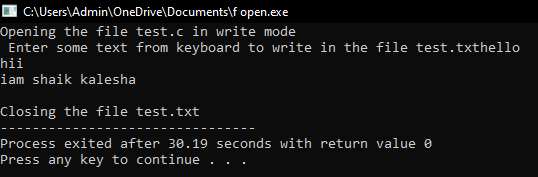
fputs("\n", fp) ;

} printf("Closing the file test.txt") ;

fclose(fp) ;

}

OUTPUT:



2. C program to read the file.

AIM:

To write a simple c program to read the contents of a file and display the same.

PROGRAM:

# include <stdio.h>

void main( )

{ FILE \*fp ;

char data[50] ;

printf( "Opening the file test.txt in read mode" ) ;

fp = fopen( "test.txt", "r" ) ;

if ( fp == NULL )

{ printf( "Could not open file test.txt" ) ;

} printf( "Reading the file test.txt" ) ;

while( fgets ( data, 50, fp ) != NULL )

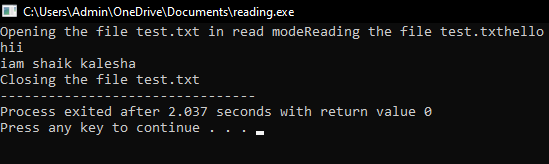
printf( "%s" , data ) ;

printf("Closing the file test.txt") ;

fclose(fp) ;

}

OUT PUT:



3. C program to reversing a file.

AIM:

To write a simple c program for reversing the file contents.

PROGRAM:

#include <stdio.h>

void main()

{

FILE \*fp;

char ch;

int i,pos;

fp=fopen("test.txt","r");

if(fp==NULL)

{

printf("file doesnot exist \n");

}

fseek(fp,0,SEEK\_END);

pos=ftell(fp);

i=0;

while(i<pos)

{

i++;

fseek(fp,-i,SEEK\_END);

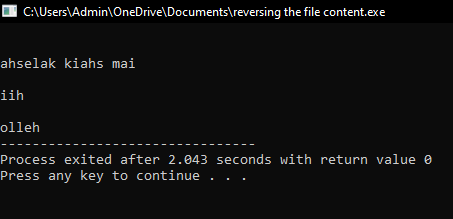
ch=fgetc(fp);

printf("%c",ch);

}

}

OUT PUT:



RESULT: Thus the illustration of the various file access permission and different types users in linux is successfully implemented.