**CSA0410 – OPERATING SYSTEM LAB PROGRAMS**

**DAY 2**

*NAME:S.NAVYA*

*REG.NO:192110511*

**9.Illustrate the concept of inter-process communication using shared memory with a C program.**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int pid;

int burst;

} Process;

void swap(Process \*a, Process \*b) {

Process temp = \*a;

\*a = \*b;

\*b = temp;

}

void sort(Process \*processes, int n) {

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

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

if (processes[j].burst > processes[j + 1].burst) {

swap(&processes[j], &processes[j + 1]);

}

}

}

}

void calculateTimes(Process \*processes, int n, int \*waitingTime, int \*turnaroundTime) {

waitingTime[0] = 0;

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

waitingTime[i] = waitingTime[i - 1] + processes[i - 1].burst;

}

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

turnaroundTime[i] = waitingTime[i] + processes[i].burst;

}

}

void calculateAverages(int \*waitingTime, int \*turnaroundTime, int n, float \*avgWaitingTime, float \*avgTurnaroundTime) {

int totalWaitingTime = 0, totalTurnaroundTime = 0;

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

totalWaitingTime += waitingTime[i];

totalTurnaroundTime += turnaroundTime[i];

}

\*avgWaitingTime = (float)totalWaitingTime / n;

\*avgTurnaroundTime = (float)totalTurnaroundTime / n;

}

void display(Process \*processes, int \*waitingTime, int \*turnaroundTime, int n, float avgWaitingTime, float avgTurnaroundTime) {

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burst, waitingTime[i], turnaroundTime[i]);

}

printf("Average Waiting Time: %.2f\n", avgWaitingTime);

printf("Average Turnaround Time: %.2f\n", avgTurnaroundTime);

}

int main() {

int n;

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

scanf("%d", &n);

Process \*processes = (Process \*)malloc(n \* sizeof(Process));

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

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

scanf("%d", &processes[i].burst);

processes[i].pid = i + 1;

}

sort(processes, n);

int \*waitingTime = (int \*)malloc(n \* sizeof(int));

int \*turnaroundTime = (int \*)malloc(n \* sizeof(int));

calculateTimes(processes, n, waitingTime, turnaroundTime);

float avgWaitingTime, avgTurnaroundTime;

calculateAverages(waitingTime, turnaroundTime, n, &avgWaitingTime, &avgTurnaroundTime);

display(processes, waitingTime, turnaroundTime, n, avgWaitingTime, avgTurnaroundTime);

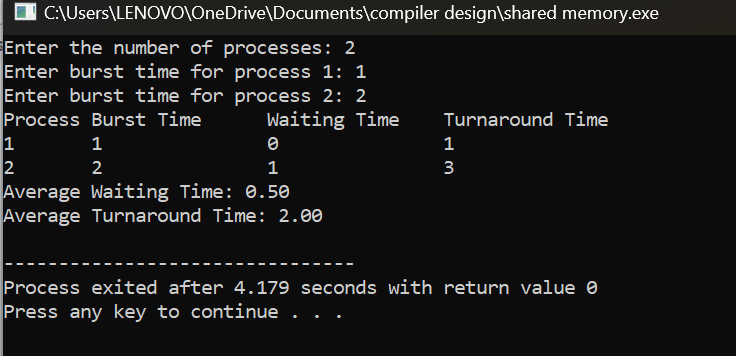
free(processes);

free(waitingTime);

free(turnaroundTime);

return 0;

}



**10. Illustrate the concept of inter-process communication using message queue with a C program.**

**11.** **Illustrate the concept of multithreading using a C program.**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NUM\_THREADS 2

#define NUM\_INCREMENTS 1000000

int counter = 0;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

void \*thread\_function(void \*arg) {

int i;

int thread\_id = \*((int \*)arg);

for (i = 0; i < NUM\_INCREMENTS; i++) {

pthread\_mutex\_lock(&mutex);

counter++;

pthread\_mutex\_unlock(&mutex);

}

printf("Thread %d finished\n", thread\_id);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

int i;

for (i = 0; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

if (pthread\_create(&threads[i], NULL, thread\_function, &thread\_ids[i]) != 0) {

perror("pthread\_create");

exit(1);

}

}

for (i = 0; i < NUM\_THREADS; i++) {

if (pthread\_join(threads[i], NULL) != 0) {

perror("pthread\_join");

exit(1);

}

}

printf("Counter value: %d\n", counter);

return 0;

}

**12. Design a C program to simulate** the **concept of Dining-Philosophers problem**

#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);

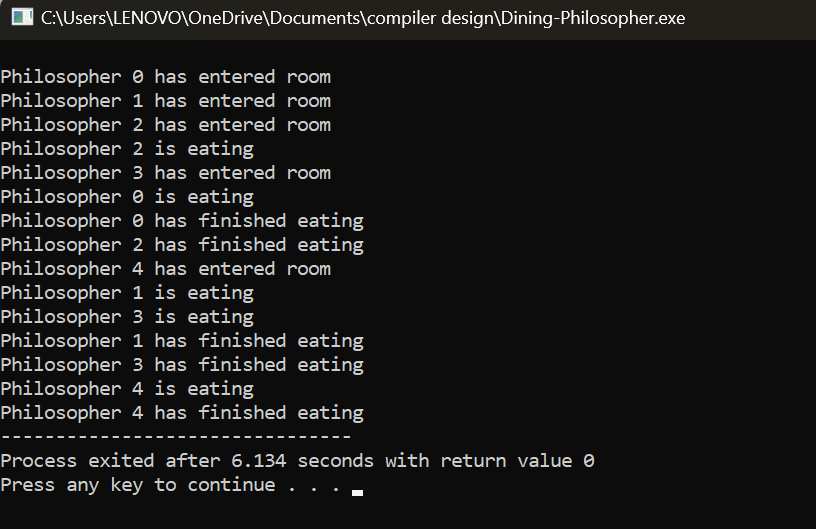
}

void eat(int phil)

{

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

}



**13. Construct a C program for implementation of memory allocation using first fit strategy**.

#include<stdio.h>

int 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)

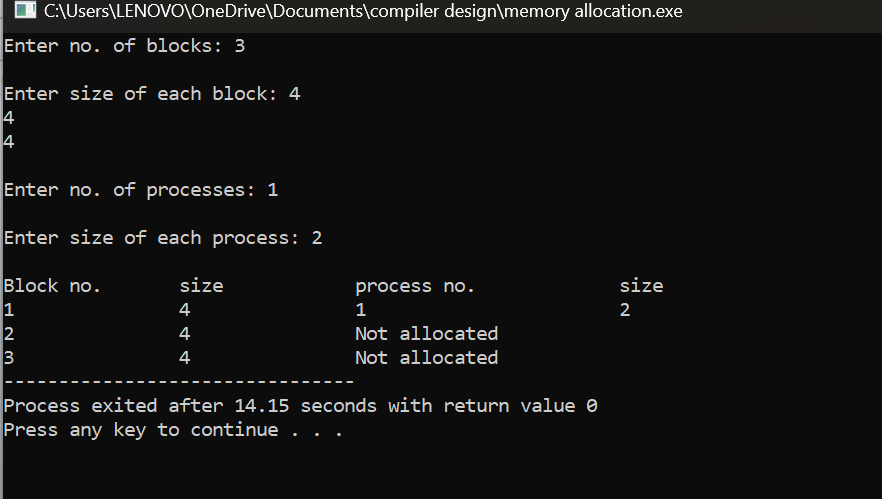
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

else

printf("Not allocated");

}

}



**14. Construct a C program to organize the file using single level directory.**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int nf=0,i=0,j=0,ch;

char mdname[10],fname[10][10],name[10];

printf("Enter the directory name:");

scanf("%s",mdname);

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

scanf("%d",&nf);

do

{

printf("Enter file name to be created:");

scanf("%s",name);

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

{

if(!strcmp(name,fname[i]))

break;

}

if(i==nf)

{

strcpy(fname[j++],name);

nf++;

}

else

printf("There is already %s\n",name);

printf("Do you want to enter another file(yes - 1 or no - 0):");

scanf("%d",&ch);

}

while(ch==1);

printf("Directory name is:%s\n",mdname);

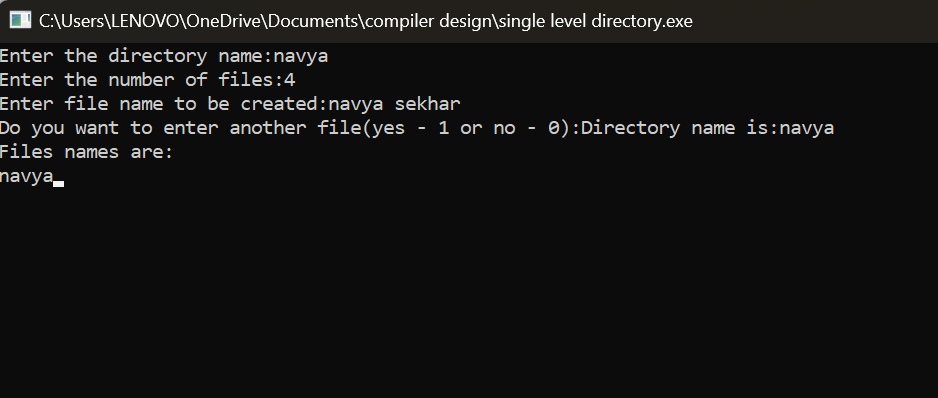
printf("Files names are:");

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

printf("\n%s",fname[i]);

getch();

}



**15. Design a C program to organize the file using two level directory structure**

#include<stdio.h>

#include<conio.h>

struct st

{

char dname[10];

char sdname[10][10];

char fname[10][10][10];

int ds,sds[10];

}dir[10];

int main()

{

int i,j,k,n;

printf("enter number of directories:");

scanf("%d",&n);

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

{

printf("enter directory %d names:",i+1);

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

printf("enter size of directories:");

scanf("%d",&dir[i].ds);

for(j=0;j<dir[i].ds;j++)

{

printf("enter subdirectory name and size:");

scanf("%s",&dir[i].sdname[j]);

scanf("%d",&dir[i].sds[j]);

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

{

printf("enter file name:");

scanf("%s",&dir[i].fname[j][k]);

}

}

}

printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");

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

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

{

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

for(j=0;j<dir[i].ds;j++)

{

printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);

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

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

printf("\n\t\t");

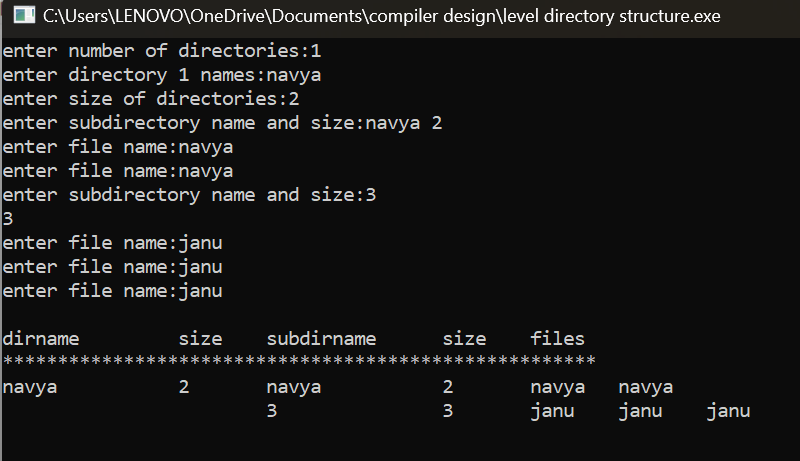
}

printf("\n");

}

getch();

}



**16. Develop a C program for implementing random access file for processing the employee details**.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NAME\_LENGTH 50

#define MAX\_EMPLOYEES 100

struct Employee {

int id;

char name[MAX\_NAME\_LENGTH];

float salary;

};

void addEmployee(FILE \*file);

void displayEmployee(FILE \*file);

void updateEmployee(FILE \*file);

void deleteEmployee(FILE \*file);

int main() {

FILE \*file;

struct Employee emp;

file = fopen("employees.dat", "rb+");

if (file == NULL) {

file = fopen("employees.dat", "wb+");

if (file == NULL) {

printf("Error creating file!\n");

return 1;

}

}

int choice;

do {

printf("\nEmployee Database\n");

printf("1. Add Employee\n");

printf("2. Display Employee\n");

printf("3. Update Employee\n");

printf("4. Delete Employee\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

addEmployee(file);

break;

case 2:

displayEmployee(file);

break;

case 3:

updateEmployee(file);

break;

case 4:

deleteEmployee(file);

break;

case 5:

printf("Exiting program...\n");

break;

default:

printf("Invalid choice!\n");

break;

}

} while (choice != 5);

fclose(file);

return 0;

}

void addEmployee(FILE \*file) {

struct Employee emp;

fseek(file, 0, SEEK\_END);

printf("Enter employee ID: ");

scanf("%d", &emp.id);

printf("Enter employee name: ");

scanf(" %[^\n]s", emp.name);

printf("Enter employee salary: ");

scanf("%f", &emp.salary);

fwrite(&emp, sizeof(struct Employee), 1, file);

printf("Employee added successfully.\n");

}

void displayEmployee(FILE \*file) {

struct Employee emp;

int id;

printf("Enter employee ID to display: ");

scanf("%d", &id);

fseek(file, (id - 1) \* sizeof(struct Employee), SEEK\_SET);

fread(&emp, sizeof(struct Employee), 1, file);

printf("Employee ID: %d\n", emp.id);

printf("Employee Name: %s\n", emp.name);

printf("Employee Salary: %.2f\n", emp.salary);

}

void updateEmployee(FILE \*file) {

struct Employee emp;

int id;

printf("Enter employee ID to update: ");

scanf("%d", &id);

fseek(file, (id - 1) \* sizeof(struct Employee), SEEK\_SET);

fread(&emp, sizeof(struct Employee), 1, file);

printf("Enter new employee name: ");

scanf(" %[^\n]s", emp.name);

printf("Enter new employee salary: ");

scanf("%f", &emp.salary);

fseek(file, (id - 1) \* sizeof(struct Employee), SEEK\_SET);

fwrite(&emp, sizeof(struct Employee), 1, file);

printf("Employee details updated successfully.\n");

}

void deleteEmployee(FILE \*file) {

int id;

printf("Enter employee ID to delete: ");

scanf("%d", &id);

struct Employee emp;

emp.id = -1;

fseek(file, (id - 1) \* sizeof(struct Employee), SEEK\_SET);

fwrite(&emp, sizeof(struct Employee), 1, file);

printf("Employee record deleted successfully.\n");

}

