Vigneshwar S

192465007

CSA0428 Operating Systems for UI Design

11. Illustrate the concept of multithreading using a C program

**Aim:**

To illustrate the concept of multithreading using a C program.

**Procedure:**

1. Include necessary header files.
2. Create a thread function to print a message.
3. Use pthread\_create() to create threads.
4. Use pthread\_join() to wait for thread completion.

**Code:**

#include <stdio.h>

#include <pthread.h>

void \*threadFunc(void \*arg) {

printf("Hello from thread!\n");

return NULL;

}

int main() {

pthread\_t thread;

pthread\_create(&thread, NULL, threadFunc, NULL);

pthread\_join(thread, NULL);

printf("Main thread exiting.\n");

return 0;

}

**Sample Input:** None

**Sample Output:**

Hello from thread!

Main thread exiting.

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Multithreading program executed successfully.

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

**Aim:**

To simulate the Dining Philosophers Problem using C and semaphores.

**Procedure:**

1. Initialize semaphores for forks.
2. Each philosopher thinks, picks two forks (semaphores), eats, and then releases them.
3. Use threads to represent philosophers.

**Code:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#define N 5

sem\_t forks[N];

void \*philosopher(void \*num) {

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

printf("Philosopher %d is thinking.\n", id);

sem\_wait(&forks[id]);

sem\_wait(&forks[(id + 1) % N]);

printf("Philosopher %d is eating.\n", id);

sem\_post(&forks[id]);

sem\_post(&forks[(id + 1) % N]);

printf("Philosopher %d finished eating.\n", id);

return NULL;

}

int main() {

pthread\_t tid[N];

int ids[N];

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

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

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

ids[i] = i;

pthread\_create(&tid[i], NULL, philosopher, &ids[i]);

}

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

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

return 0;

}

**Sample Input:** None

**Sample Output**

Philosopher 0 is thinking.

Philosopher 0 is eating.

Philosopher 0 finished eating.

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Dining Philosophers problem program executed successfully.

13. Construct a C program for implementation the various memory allocation strategies.

**Aim:**

To implement various memory allocation strategies (First Fit, Best Fit, Worst Fit).

**Procedure:**

1. Define block and process arrays.
2. For each strategy, traverse blocks and allocate if suitable.
3. Print allocation status.

**Code**

#include <stdio.h>

#define SIZE 5

void firstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

for (int i = 0; i < n; i++) allocation[i] = -1;

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

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

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

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

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

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[SIZE] = {100, 500, 200, 300, 600};

int processSize[4] = {212, 417, 112, 426};

firstFit(blockSize, SIZE, processSize, 4);

return 0;

}

**Sample Output:**

Process 1 -> Block 2

Process 2 -> Block 5

Process 3 -> Block 2

Process 4 -> Not Allocated

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Memory allocation strategies program executed successfully.

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

**Aim:**

To organize the file using a single-level directory.

**Procedure:**

1. Use an array of filenames.
2. Provide menu to create, delete, search files.

**Code:**

#include <stdio.h>

#include <string.h

struct { char fname[20]; } dir[10];

int main() {

int n = 0, ch; char name[20];

while (1) {

printf("\n1.Create 2.Delete 3.Search 4.Exit: ");

scanf("%d", &ch);

if (ch == 1) scanf("%s", dir[n++].fname);

else if (ch == 2) {

scanf("%s", name);

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

if (strcmp(name, dir[i].fname) == 0) strcpy(dir[i].fname, "deleted");

}

else if (ch == 3) {

scanf("%s", name);

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

if (strcmp(name, dir[i].fname) == 0) printf("Found\n");

}

else break;

}

}

**Sample Input:**

1

file1

1

file2

3

file1

2

file2

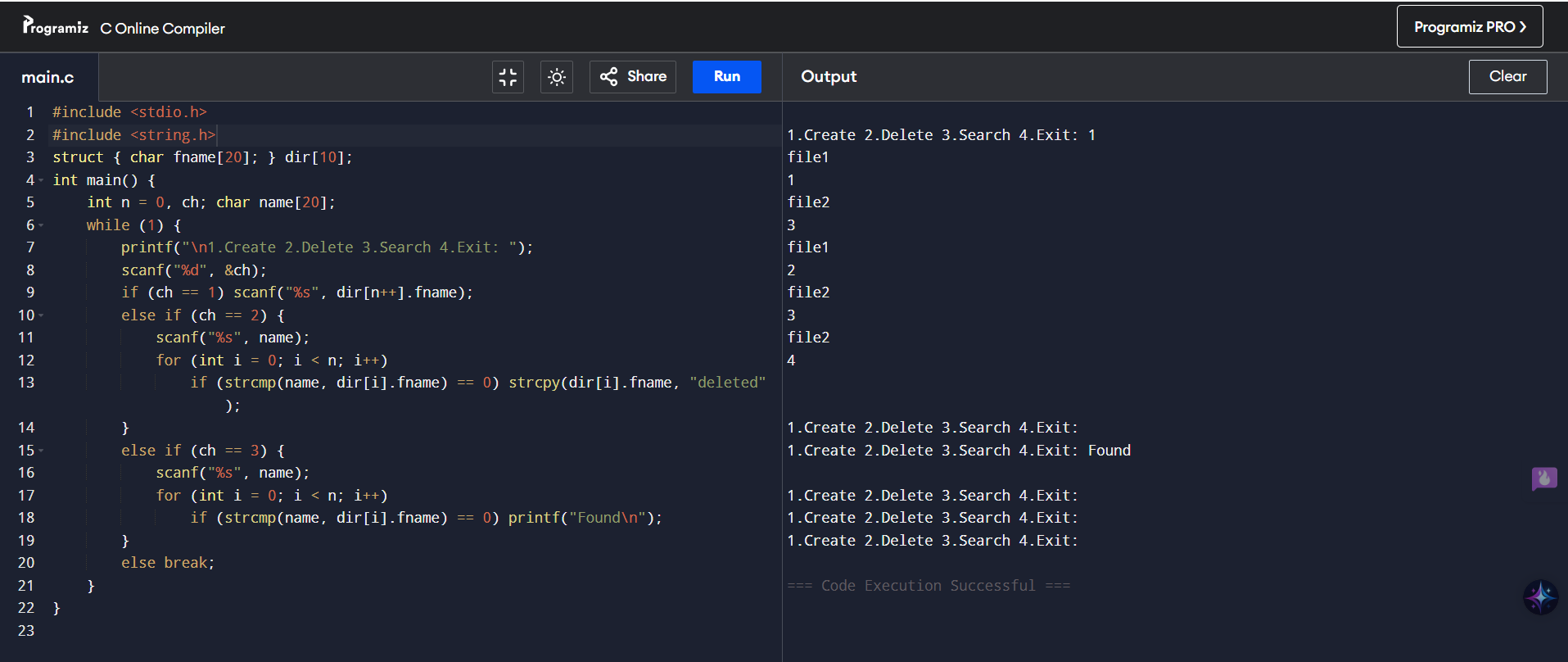
3

file2

4

**Sample Output:**

Found



**RESULT:**

Single-level directory structure program executed successfully.

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

**Aim:** To organize the file using a two-level directory structure.

**Procedure:**

1. Create multiple user directories.
2. Allow file creation inside user directories.

**Code:**

#include <stdio.h>

#include <string.h>

struct { char d[10], f[10][10]; int fc; } dir[10];

int main() {

int dc = 0, ch; char d[10];

while (1) {

printf("\n1.Dir 2.File 3.List 4.Exit: ");

scanf("%d", &ch);

if (ch == 1) { printf("Dir: "); scanf("%s", dir[dc].d); dir[dc++].fc = 0; }

else if (ch == 2) {

printf("Dir: "); scanf("%s", d);

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

if (!strcmp(d, dir[i].d))

scanf("%s", dir[i].f[dir[i].fc++]);

}

else if (ch == 3) {

printf("Dir: "); scanf("%s", d);

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

if (!strcmp(d, dir[i].d))

for (int j=0;j<dir[i].fc;j++) printf("%s\n", dir[i].f[j]);

}

else break;

}

}

**Sample Input:**

1

Dir: proj

2

Dir: proj

file1

2

Dir: proj

file2

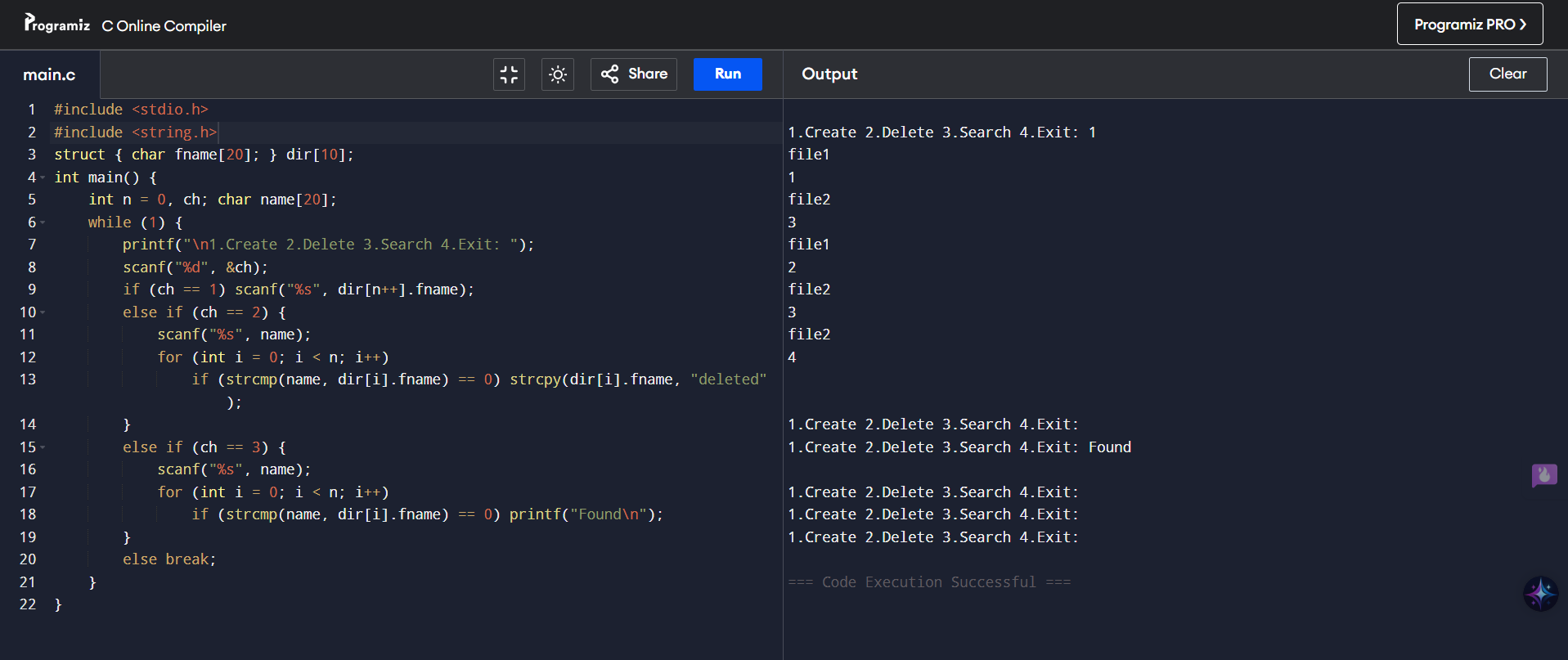
3

Dir: proj

**Sample Output:**

file1

file2



**RESULT:**

Two-level directory structure program executed successfully.

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

**Aim:**

To implement random access file processing employee details.

**Procedure:**

1. Open file in binary mode.
2. Write employee details into the file.
3. Use fseek() for random access.

**Code:**

#include <stdio.h>

struct Emp { int id; char name[20]; float sal; };

int main() {

struct Emp e;

FILE \*f = fopen("emp.dat", "wb+");

for (int i=0; i<2; i++) {

printf("ID Name Salary: ");

scanf("%d %s %f", &e.id, e.name, &e.sal);

fwrite(&e, sizeof(e), 1, f);

}

fseek(f, sizeof(e), SEEK\_SET);

fread(&e, sizeof(e), 1, f);

printf("Record 2 -> ID:%d Name:%s Salary:%.2f\n", e.id, e.name, e.sal);

fclose(f);

}

**Sample Input:**

ID Name Salary: 1 Ram 25000

ID Name Salary: 2 Ravi 30000

**Sample Output:**

Record 2 -> ID:2 Name:Ravi Salary:30000.00



**RESULT:**

Random access file program for employee records executed successfully.

17. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C

**Aim:**

To illustrate the deadlock avoidance concept by simulating Banker’s algorithm using C.

**Procedure:**

1. Input number of processes and resources.
2. Input Allocation, Maximum, and Available matrices.
3. Calculate Need matrix.
4. Find Safe Sequence using Banker's Algorithm.
5. Display result.

**Code:**

#include <stdio.h>

int main() {

int alloc[5][3] = {{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};

int max[5][3] = {{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};

int avail[3] = {3,3,2}, need[5][3], finish[5]={0}, safe[5], count=0;

for (int i=0;i<5;i++) for (int j=0;j<3;j++) need[i][j]=max[i][j]-alloc[i][j];

while (count<5) {

int found=0;

for (int i=0;i<5;i++) if (!finish[i]) {

int j; for (j=0;j<3;j++) if (need[i][j]>avail[j]) break;

if (j==3) { for (j=0;j<3;j++) avail[j]+=alloc[i][j]; safe[count++]=i; finish[i]=1; found=1; }

}

if (!found) return printf("Not Safe\n");

}

printf("Safe Sequence: "); for (int i=0;i<5;i++) printf("P%d ",safe[i]);

}

**Sample Output:**

Safe Sequence: P1 P3 P4 P0 P2

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Banker's algorithm for deadlock avoidance executed successfully.

18 Construct a C program to simulate producer-consumer problem using semaphores.

**Aim:**

To simulate producer-consumer problem using semaphores.

**Procedure:**

Initialize empty, full semaphores and mutex.

Producer inserts items into the buffer.

Consumer removes items from the buffer.

Use semaphores to ensure mutual exclusion and synchronization.

**Code:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

int buffer = 0;

sem\_t empty, full;

pthread\_mutex\_t mutex;

void \*producer() {

for (int i=1;i<=3;i++) {

sem\_wait(&empty); pthread\_mutex\_lock(&mutex);

buffer=i; printf("Produced: %d\n", buffer);

pthread\_mutex\_unlock(&mutex); sem\_post(&full);

}

}

void \*consumer() {

for (int i=1;i<=3;i++) {

sem\_wait(&full); pthread\_mutex\_lock(&mutex);

printf("Consumed: %d\n", buffer);

pthread\_mutex\_unlock(&mutex); sem\_post(&empty);

}

}

int main() {

pthread\_t p, c;

sem\_init(&empty,0,1); sem\_init(&full,0,0); pthread\_mutex\_init(&mutex,NULL);

pthread\_create(&p,NULL,producer,NULL); pthread\_create(&c,NULL,consumer,NULL);

pthread\_join(p,NULL); pthread\_join(c,NULL);

}

**Sample Output:**

Produced: 1

Consumed: 1

Produced: 2

Consumed: 2

Produced: 3

Consumed: 3

A screenshot of a computer

AI-generated content may be incorrect.

**RRESULT:**

Producer-consumer problem using semaphores executed successfully.

19. Design a C program to implement process synchronization using mutex locks.

**Aim:**

To implement process synchronization using mutex locks in C.

**Procedure:**

1. Create threads performing critical operations.
2. Protect critical section using pthread\_mutex\_lock() and pthread\_mutex\_unlock().
3. Execute critical section mutually exclusively.

**Code:**

#include <stdio.h>

#include <pthread.h>

int counter=0;

pthread\_mutex\_t lock;

void \*inc() {

for (int i=0;i<3;i++) {

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

}

}

int main() {

pthread\_t t1, t2;

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

pthread\_create(&t1,NULL,inc,NULL); pthread\_create(&t2,NULL,inc,NULL);

pthread\_join(t1,NULL); pthread\_join(t2,NULL);

}

**Sample Output**

Counter: 1

Counter: 2

Counter: 3

Counter: 4

Counter: 5

Counter: 6

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

Mutex-based synchronization program executed successfully.

20. Construct a C program to simulate Reader-Writer problem using Semaphores

**Aim:**

To simulate the Reader-Writer problem using semaphores.

**Procedure:**

1. Initialize semaphores for read/write synchronization.
2. Allow multiple readers but exclusive writers.
3. Ensure no simultaneous writer-reader conflict.

**Code:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t wrt; int readcount=0, data=0; pthread\_mutex\_t mutex;

void \*reader() {

pthread\_mutex\_lock(&mutex); readcount++; if (readcount==1) sem\_wait(&wrt);

pthread\_mutex\_unlock(&mutex);

printf("Reader read: %d\n", data);

pthread\_mutex\_lock(&mutex); readcount--; if (readcount==0) sem\_post(&wrt);

pthread\_mutex\_unlock(&mutex);

}

void \*writer() {

sem\_wait(&wrt); data++; printf("Writer wrote: %d\n", data); sem\_post(&wrt);

}

int main() {

pthread\_t r1,r2,w1;

sem\_init(&wrt,0,1); pthread\_mutex\_init(&mutex,NULL);

pthread\_create(&r1,NULL,reader,NULL);

pthread\_create(&w1,NULL,writer,NULL);

pthread\_create(&r2,NULL,reader,NULL);

pthread\_join(r1,NULL); pthread\_join(w1,NULL); pthread\_join(r2,NULL);

}

**Sample Output:**

Reader read: 0

Writer wrote: 1

Reader read: 1

A screenshot of a computer program

AI-generated content may be incorrect.

**RESULT:**

Reader-Writer problem using semaphores executed successfully.