11. Multithreding

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

void \*print\_message(void \*thread\_id) {

long tid = (long) thread\_id;

printf("Thread %ld is running.\n", tid);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[5];

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

pthread\_create(&threads[i], NULL, print\_message, (void \*)i);

}

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

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

}

return 0;

}

Output;-

Thread 0 is running.

Thread 1 is running.

...

12 **Dining Philosophers Problem**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

sem\_t forks[N];

void \*philosopher(void \*num) {

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

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]);

return NULL;

}

int main() {

pthread\_t phil[N];

int phil\_ids[N];

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

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

phil\_ids[i] = i;

}

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

pthread\_create(&phil[i], NULL, philosopher, &phil\_ids[i]);

}

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

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

}

return 0;

}

OUTPUT:-

Philosopher 0 is eating.

Philosopher 1 is eating.

...

13. **Memory Allocation Strategies**

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr, n;

printf("Enter number of elements: ");

scanf("%d", &n);

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

if (arr == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

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

arr[i] = i \* 10;

}

printf("Allocated memory: ");

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

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

}

free(arr);

return 0;}

OUTPUT:-

Allocated memory: 0 10 20 ...

14 **Single Level Directory**

#include <stdio.h>

#include <string.h>

struct File {

char name[20];

};

int main() {

struct File directory[10];

int count = 0;

strcpy(directory[count++].name, "file1.txt");

strcpy(directory[count++].name, "file2.txt");

printf("Files in directory:\n");

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

printf("%s\n", directory[i].name);

}

return 0;

}

OUTPUT:-

Files in directory:

file1.txt

file2.txt

15. **Two-Level Directory**

#include <stdio.h>

#include <string.h>

struct Directory {

char name[20];

struct File {

char name[20];

} files[10];

int file\_count;

};

int main() {

struct Directory user\_dir[5];

strcpy(user\_dir[0].name, "user1");

user\_dir[0].file\_count = 2;

strcpy(user\_dir[0].files[0].name, "file1.txt");

strcpy(user\_dir[0].files[1].name, "file2.txt");

printf("Directory: %s\n", user\_dir[0].name);

for (int i = 0; i < user\_dir[0].file\_count; i++) {

printf(" %s\n", user\_dir[0].files[i].name);

}

return 0;

}

OUTPUT:-

Directory: user1

file1.txt

file2.txt

16. **Random Access File for Employee Details**

#include <stdio.h>

#include <stdlib.h>

struct Employee {

int id;

char name[30];

float salary;

};

int main() {

struct Employee emp = {1, "John Doe", 50000};

FILE \*fp = fopen("employee.dat", "wb");

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

fclose(fp);

fp = fopen("employee.dat", "rb");

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

printf("ID: %d, Name: %s, Salary: %.2f\n", emp.id, emp.name, emp.salary);

fclose(fp);

return 0;

}

OUTPUT:-

ID: 1, Name: John Doe, Salary: 50000.00

17. **Banker's Algorithm (Deadlock Avoidance)**

#include <stdio.h>

int main() {

int n = 3, m = 3;

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

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

int avail[3] = {3, 3, 2};

int finish[3] = {0};

int safeSeq[3], index = 0;

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

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

int flag = 1;

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

if (max[i][j] - alloc[i][j] > avail[j]) {

flag = 0;

break;

}

}

if (flag && !finish[i]) {

safeSeq[index++] = i;

finish[i] = 1;

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

avail[j] += alloc[i][j];

}

}

}

}

printf("Safe Sequence: ");

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

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

}

return 0;

}

OUTPUT:-

Safe Sequence: 1 0 2

18 **Producer-Consumer Problem using Semaphores**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#define SIZE 5

int buffer[SIZE];

int count = 0;

sem\_t empty, full;

pthread\_mutex\_t mutex;

void \*producer(void \*arg) {

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

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[count++] = i;

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

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

return NULL;

}

void \*consumer(void \*arg) {

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

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[--count];

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

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

return NULL;

}

int main() {

pthread\_t prod, cons;

sem\_init(&empty, 0, SIZE);

sem\_init(&full, 0, 0);

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

pthread\_create(&prod, NULL, producer, NULL);

pthread\_create(&cons, NULL, consumer, NULL);

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

OUTPUT:-

Produced: 0

Produced: 1

...

Consumed: 0

Consumed: 1

...

19 **Process Synchronization using Mutex Locks**

#include <stdio.h>

#include <pthread.h>

pthread\_mutex\_t mutex;

int counter = 0;

void \*increment(void \*arg) {

pthread\_mutex\_lock(&mutex);

counter++;

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

pthread\_mutex\_unlock(&mutex);

return NULL;

}

int main() {

pthread\_t threads[5];

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

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

pthread\_create(&threads[i], NULL, increment, NULL);

}

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

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

}

pthread\_mutex\_destroy(&mutex);

return 0;

}

OUTPUT:-

Counter: 1

Counter: 2

20 **Reader-Writer Problem using Semaphores**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t rw\_mutex, mutex;

int read\_count = 0, data = 0;

void \*reader(void \*arg) {

sem\_wait(&mutex);

read\_count++;

if (read\_count == 1) sem\_wait(&rw\_mutex);

sem\_post(&mutex);

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

sem\_wait(&mutex);

read\_count--;

if (read\_count == 0) sem\_post(&rw\_mutex);

sem\_post(&mutex);

return NULL;

}

void \*writer(void \*arg) {

sem\_wait(&rw\_mutex);

data++;

printf("Writer writes data: %d\n", data);

sem\_post(&rw\_mutex);

return NULL;

}

int main() {

pthread\_t r1, r2, w1;

sem\_init(&rw\_mutex, 0, 1);

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

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);

sem\_destroy(&rw\_mutex);

sem\_destroy(&mutex);

return 0;

}

OUTPUT:-

Reader reads data: 0

Writer writes data: 1

Reader reads data: 1