



SIMATS
ENGINEERING



SIMATS
Savitribi Institute of Medical And Technical Sciences
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

Name:S.karnesh

Reg.No:192421200

Course code:CSA0429

Course Name: Operating System

1.

The screenshot shows a C program named `main.c` in a code editor. The program implements an employee management system using a Random Access File (RAFI). It includes headers for `stdio.h`, `stdlib.h`, and `string.h`. A structure `Employee` is defined with fields `id`, `name`, and `salary`. The `addEmployee()` function appends new employee data to the file `employee.dat` in binary mode. The `displayEmployees()` function reads the file and displays the list of employees. The main function provides a menu with options: 1. Add Employee, 2. Display Employees, 3. Update Employee, 4. Delete Employee, 5. Exit. The user has chosen option 2, and the program outputs the list of employees, which is currently empty.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define FILE_NAME "employee.dat"
6
7 typedef struct {
8     int id;
9     char name[50];
10    float salary;
11 } Employee;
12
13 // Function to add a new employee
14 void addEmployee() {
15     FILE *fp = fopen(FILE_NAME, "ab"); // Append in binary mode
16     if (!fp) {
17         printf("Unable to open file!\n");
18         return;
19     }
20
21     Employee emp;
22     printf("Enter Employee ID: ");
23     scanf("%d", &emp.id);
24     printf("Enter Name: ");
25     scanf("%s", emp.name); // Reads string with spaces
26     printf("Enter Salary: ");
27     scanf("%f", &emp.salary);
28
29     fwrite(&emp, sizeof(Employee), 1, fp);
30     fclose(fp);
31     printf("Employee added successfully!\n");
32 }
33
34 // Function to display all employees
35 void displayEmployees() {
36     FILE *fp = fopen(FILE_NAME, "rb");
37     if (!fp) {
38         printf("File not found!\n");
39         return;
40     }
41 }
```

Output:

```
--- Employee Management using Random Access File ---
1. Add Employee
2. Display Employees
3. Update Employee
4. Delete Employee
5. Exit
Enter your choice: 2
File not found!
```

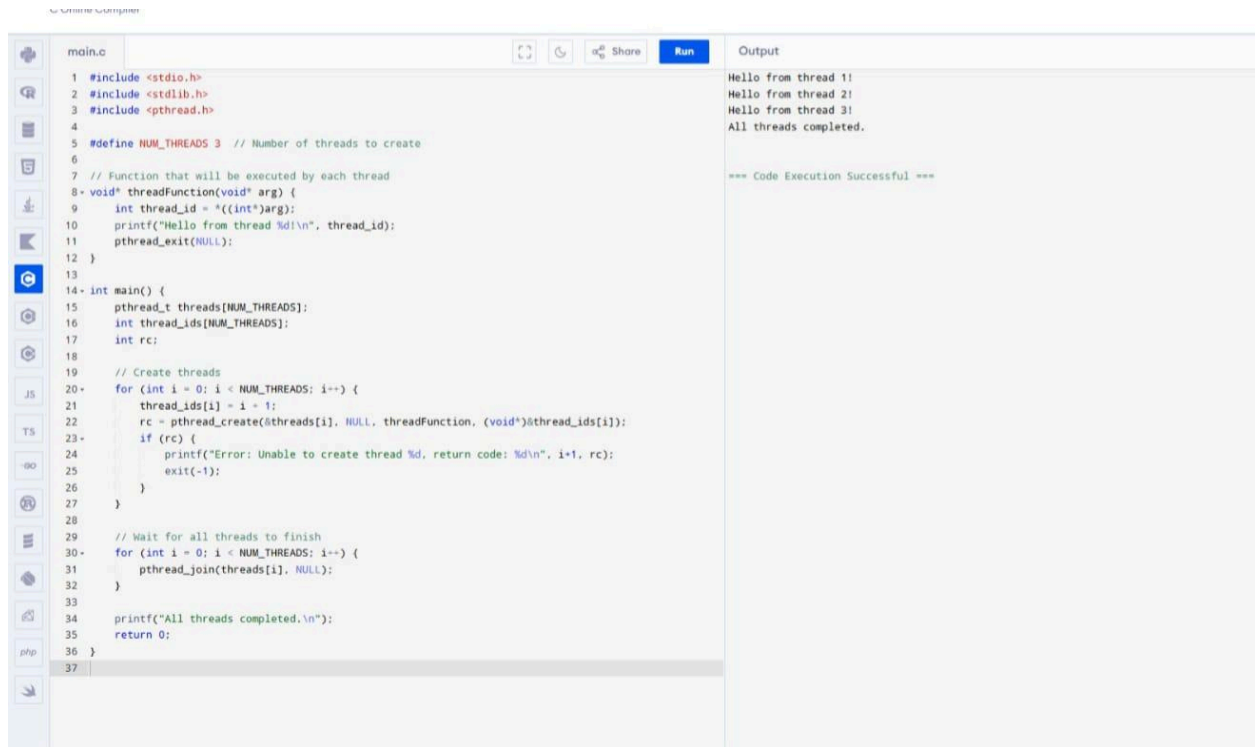
2.

The screenshot shows a C program named `main.c` in a code editor. The program demonstrates Message Queue IPC. It includes headers for `stdio.h`, `sys/ipc.h`, `sys/msg.h`, `sys/types.h`, `unistd.h`, and `string.h`. A message structure `msg_buffer` is defined with fields `msg_type` and `msg_text`. The `main` function generates a unique key, creates a message queue, and forks a child process. The child process receives a message from the parent and prints it. The parent process sends a message to the child and prints a confirmation message. The program outputs the following messages:

```
Parent Process: Message Sent
Child Process: Message Received = Hello from Parent using Message Queue IPC
Message Queue Deleted
```

=== Code Execution Successful ===

3.



The screenshot shows a C++ IDE with a file named `main.c`. The code defines three threads and prints their IDs. The output window shows the execution results.

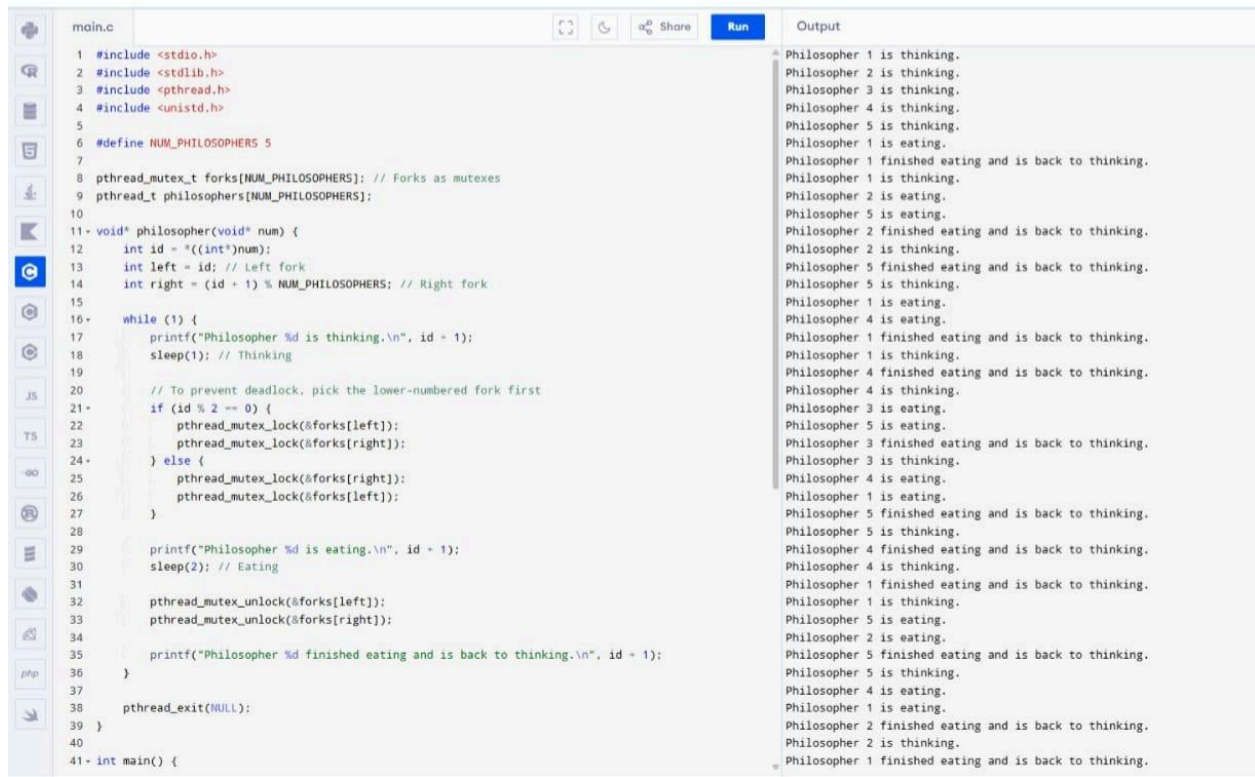
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4
5 #define NUM_THREADS 3 // Number of threads to create
6
7 // Function that will be executed by each thread
8 void* threadFunction(void* arg) {
9     int thread_id = *((int*)arg);
10    printf("Hello from thread %d!\n", thread_id);
11    pthread_exit(NULL);
12 }
13
14 int main() {
15     pthread_t threads[NUM_THREADS];
16     int thread_ids[NUM_THREADS];
17     int rc;
18
19     // Create threads
20     for (int i = 0; i < NUM_THREADS; i++) {
21         thread_ids[i] = i + 1;
22         rc = pthread_create(&threads[i], NULL, threadFunction, (void*)&thread_ids[i]);
23         if (rc) {
24             printf("Error: Unable to create thread %d, return code: %d\n", i+1, rc);
25             exit(-1);
26         }
27     }
28
29     // Wait for all threads to finish
30     for (int i = 0; i < NUM_THREADS; i++) {
31         pthread_join(threads[i], NULL);
32     }
33
34     printf("All threads completed.\n");
35     return 0;
36 }
37
```

Output:

```
Hello from thread 1!
Hello from thread 2!
Hello from thread 3!
All threads completed.

=== Code Execution Successful ===
```

4.



The screenshot shows a C program in a code editor with a sidebar on the left containing icons for various languages (C, JS, TS, PHP, etc.). The code is in a file named 'main.c' and implements the Dining Philosopher problem with 5 philosophers and 5 forks. The program uses pthreads for concurrency and mutexes to manage the forks. The output window on the right shows the execution log, where each philosopher's state (thinking, eating, finished eating) is printed. The sequence of events shows philosophers taking turns to eat, with some waiting for forks to become available. The program ends with all philosophers finished eating and back to thinking.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 #define NUM_PHILOSOPHERS 5
7
8 pthread_mutex_t forks[NUM_PHILOSOPHERS]; // Forks as mutexes
9 pthread_t philosophers[NUM_PHILOSOPHERS];
10
11 void* philosopher(void* num) {
12     int id = *((int*)num);
13     int left = id; // Left fork
14     int right = (id + 1) % NUM_PHILOSOPHERS; // Right fork
15
16     while (1) {
17         printf("Philosopher %d is thinking.\n", id + 1);
18         sleep(1); // Thinking
19
20         // To prevent deadlock, pick the lower-numbered fork first
21         if (id % 2 == 0) {
22             pthread_mutex_lock(&forks[left]);
23             pthread_mutex_lock(&forks[right]);
24         } else {
25             pthread_mutex_lock(&forks[right]);
26             pthread_mutex_lock(&forks[left]);
27         }
28
29         printf("Philosopher %d is eating.\n", id + 1);
30         sleep(2); // Eating
31
32         pthread_mutex_unlock(&forks[left]);
33         pthread_mutex_unlock(&forks[right]);
34
35         printf("Philosopher %d finished eating and is back to thinking.\n", id + 1);
36     }
37
38     pthread_exit(NULL);
39 }
40
41 int main() {
```

Output

```
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 5 is thinking.
Philosopher 1 is eating.
Philosopher 1 finished eating and is back to thinking.
Philosopher 1 is thinking.
Philosopher 2 is eating.
Philosopher 5 is eating.
Philosopher 2 finished eating and is back to thinking.
Philosopher 2 is thinking.
Philosopher 5 finished eating and is back to thinking.
Philosopher 5 is thinking.
Philosopher 1 is eating.
Philosopher 4 is eating.
Philosopher 1 finished eating and is back to thinking.
Philosopher 1 is thinking.
Philosopher 4 finished eating and is back to thinking.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 5 is eating.
Philosopher 3 finished eating and is back to thinking.
Philosopher 3 is thinking.
Philosopher 4 is eating.
Philosopher 1 is eating.
Philosopher 5 finished eating and is back to thinking.
Philosopher 5 is thinking.
Philosopher 4 finished eating and is back to thinking.
Philosopher 4 is thinking.
Philosopher 1 finished eating and is back to thinking.
Philosopher 1 is thinking.
Philosopher 5 is eating.
Philosopher 2 is eating.
Philosopher 5 finished eating and is back to thinking.
Philosopher 5 is thinking.
Philosopher 4 is eating.
Philosopher 1 is eating.
Philosopher 2 finished eating and is back to thinking.
Philosopher 2 is thinking.
Philosopher 1 finished eating and is back to thinking.
```

5.

C Online Compiler

main.c

1 #include <stdio.h>

2 #include <string.h>

3 #include <stdlib.h>

4

5 #define MAX_FILES 100

6 #define MAX_FILENAME 50

7

8 typedef struct {

9 char filename[MAX_FILENAME];

10 } File;

11

12 // Single-level directory

13 typedef struct {

14 File files[MAX_FILES];

15 int fileCount;

16 } Directory;

17

18 // Function to create a file

19 void createFile(Directory *dir, char *name) {

20 if (dir->fileCount >= MAX_FILES) {

21 printf("Directory full! Cannot create more files.\n");

22 return;

23 }

24

25 // Check if file already exists

26 for (int i = 0; i < dir->fileCount; i++) {

27 if (strcmp(dir->files[i].filename, name) == 0) {

28 printf("File '%s' already exists.\n", name);

29 return;

30 }

31 }

32

33 strcpy(dir->files[dir->fileCount].filename, name);

34 dir->fileCount++;

35 printf("File '%s' created successfully.\n", name);

36 }

37

38 // Function to display all files

39 void displayFiles(Directory *dir) {

40 if (dir->fileCount == 0) {

41 printf("Directory is empty.\n");

Share

Run

Output

Single Level Directory Operations:
1. Create File
2. Display Files
3. Search File
4. Delete File
5. Exit
Enter your choice: 1
Enter filename to create: FILE.TXT
File 'FILE.TXT' created successfully.

Single Level Directory Operations:
1. Create File
2. Display Files
3. Search File
4. Delete File
5. Exit
Enter your choice: