SIMATS
ENGINEERING

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

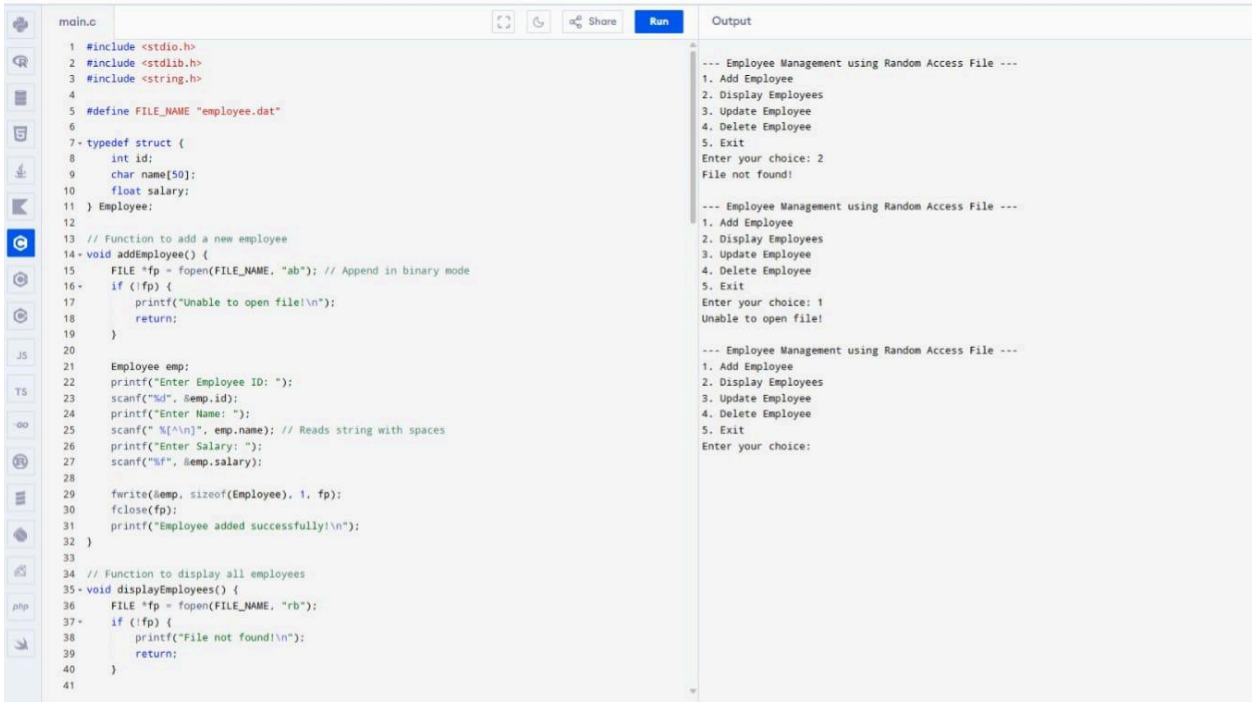
Name:S.karnesh

Reg.No:192421200

Course code:CSA0429

Course Name: Operating System

1.



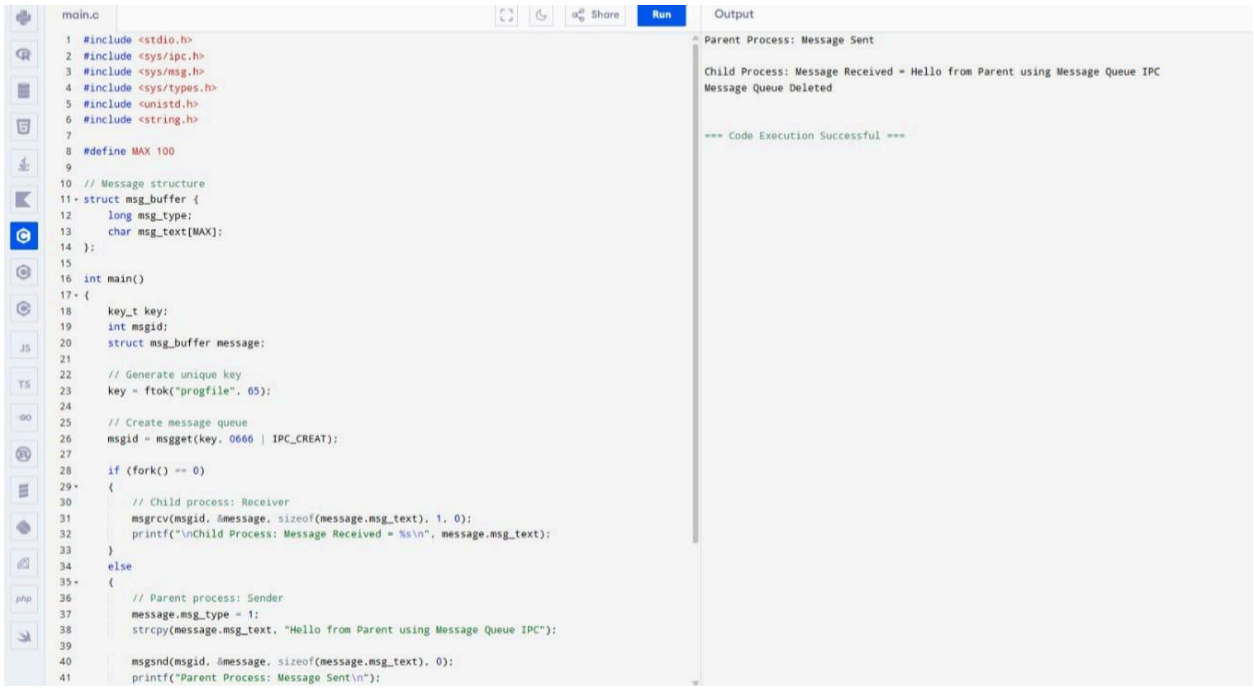
main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define FILE_NAME "employee.dat"

typedef struct {
    int id;
    char name[50];
    float salary;
} Employee;

// Function to add a new employee
void addEmployee() {
    FILE *fp = fopen(FILE_NAME, "ab"); // Append in binary mode
    if (!fp) {
        printf("Unable to open file!\n");
        return;
    }

    Employee emp;
    printf("Enter Employee ID: ");
    scanf("%d", &emp.id);
    printf("Enter Name: ");
    scanf(" %[^\n]", emp.name); // Reads string with spaces
    printf("Enter Salary: ");
    scanf("%f", &emp.salary);

    fwrite(&emp, sizeof(Employee), 1, fp);
    fclose(fp);
    printf("Employee added successfully!\n");
}

// Function to display all employees
void displayEmployees() {
    FILE *fp = fopen(FILE_NAME, "rb");
    if (!fp) {
        printf("File not found!\n");
        return;
    }

```

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!

--- Employee Management using Random Access File ---
1. Add Employee
2. Display Employees
3. Update Employee
4. Delete Employee
5. Exit
Enter your choice: 1
Unable to open file!

--- Employee Management using Random Access File ---
1. Add Employee
2. Display Employees
3. Update Employee
4. Delete Employee
5. Exit
Enter your choice:
```

2.



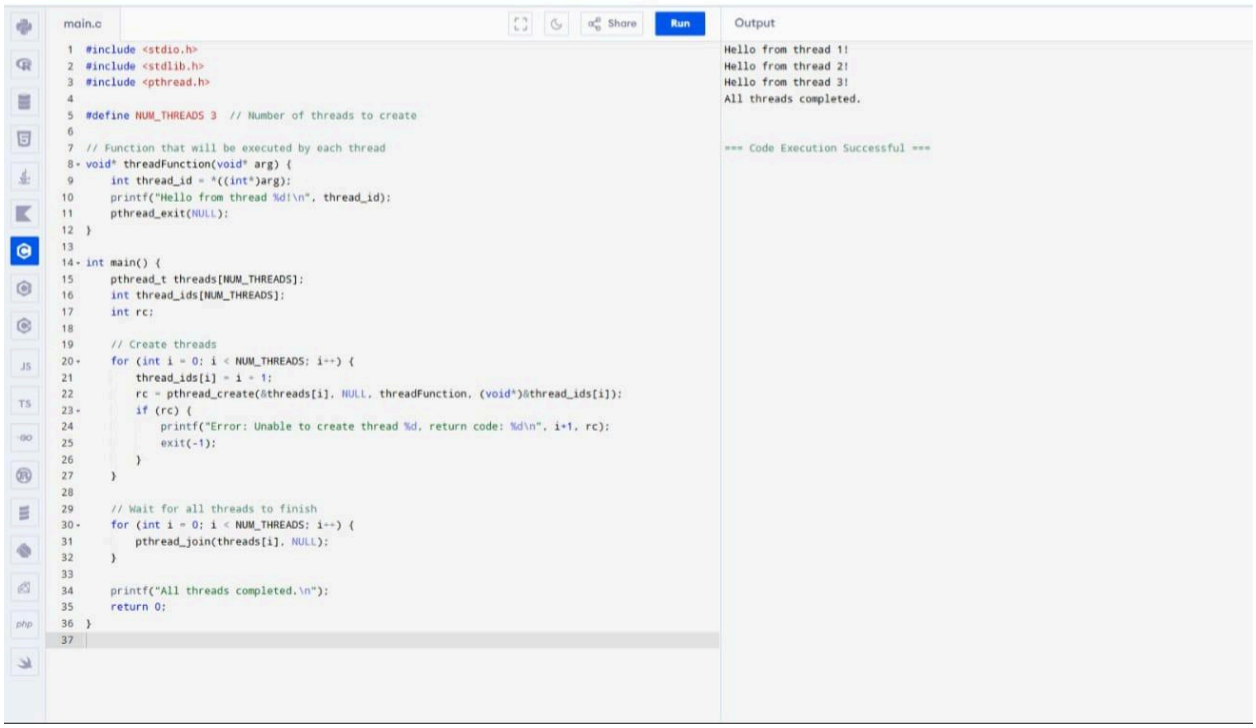
main.c

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>

#define MAX 100

// Message structure
struct msg_buffer {
    long msg_type;
    char msg_text[MAX];
};

int main()
{
    key_t key;
    int msgid;
    struct msg_buffer message;

    // Generate unique key
    key = ftok("progfile", 65);

    // Create message queue
    msgid = msgget(key, 0666 | IPC_CREAT);

    if (fork() == 0)
    {
        // Child process: Receiver
        msgrcv(msgid, &message, sizeof(message.msg_text), 1, 0);
        printf("\nChild Process: Message Received = %s\n", message.msg_text);
    }
    else
    {
        // Parent process: Sender
        message.msg_type = 1;
        strcpy(message.msg_text, "Hello from Parent using Message Queue IPC");

        msgsnd(msgid, &message, sizeof(message.msg_text), 0);
        printf("Parent Process: Message Sent\n");
```

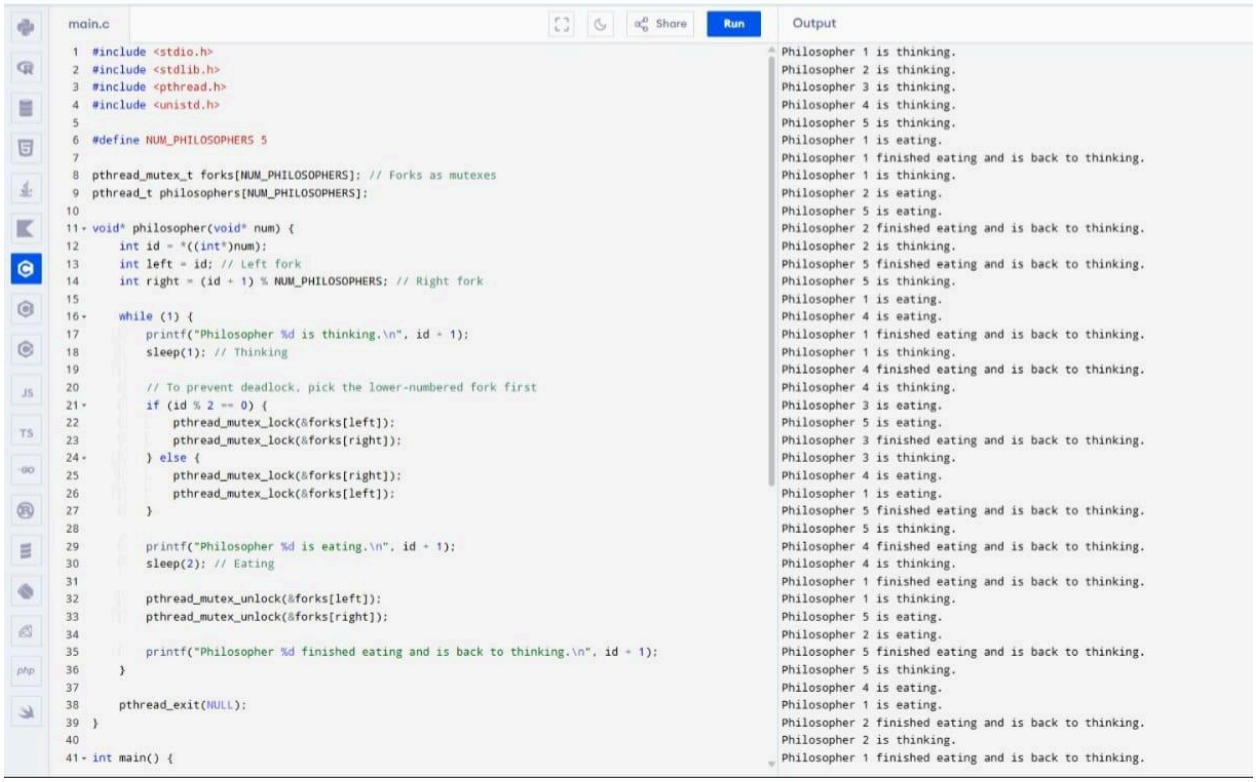
Output

```
Parent Process: Message Sent

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


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

3.


main.c
Share
Run
Output
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUM_THREADS 3 // Number of threads to create
// Function that will be executed by each thread
void* threadFunction(void* arg) {
int thread_id = *((int*)arg);
printf("Hello from thread %d!\n", thread_id);
pthread_exit(NULL);
}
int main() {
pthread_t threads[NUM_THREADS];
int thread_ids[NUM_THREADS];
int rc;
// Create threads
for (int i = 0; i < NUM_THREADS; i++) {
thread_ids[i] = i + 1;
rc = pthread_create(&threads[i], NULL, threadFunction, (void*)&thread_ids[i]);
if (rc) {
printf("Error: Unable to create thread %d, return code: %d\n", i+1, rc);
exit(-1);
}
}
// Wait for all threads to finish
for (int i = 0; i < NUM_THREADS; i++) {
pthread_join(threads[i], NULL);
}
printf("All threads completed.\n");
return 0;
}
Hello from thread 1!
Hello from thread 2!
Hello from thread 3!
All threads completed.
=== Code Execution Successful ===

4.



main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_PHILOSOPHERS 5

pthread_mutex_t forks[NUM_PHILOSOPHERS]; // Forks as mutexes
pthread_t philosophers[NUM_PHILOSOPHERS];

void* philosopher(void* num) {
    int id = *((int*)num);
    int left = id; // Left fork
    int right = (id + 1) % NUM_PHILOSOPHERS; // Right fork

    while (1) {
        printf("Philosopher %d is thinking.\n", id + 1);
        sleep(1); // Thinking

        // To prevent deadlock, pick the lower-numbered fork first
        if (id % 2 == 0) {
            pthread_mutex_lock(&forks[left]);
            pthread_mutex_lock(&forks[right]);
        } else {
            pthread_mutex_lock(&forks[right]);
            pthread_mutex_lock(&forks[left]);
        }

        printf("Philosopher %d is eating.\n", id + 1);
        sleep(2); // Eating

        pthread_mutex_unlock(&forks[left]);
        pthread_mutex_unlock(&forks[right]);

        printf("Philosopher %d finished eating and is back to thinking.\n", id + 1);
    }

    pthread_exit(NULL);
}

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

Share Run

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_FILES 100
#define MAX_FILENAME 50

typedef struct {
    char filename[MAX_FILENAME];
} File;

// Single-level directory
typedef struct {
    File files[MAX_FILES];
    int fileCount;
} Directory;

// Function to create a file
void createFile(Directory *dir, char *name) {
    if (dir->fileCount >= MAX_FILES) {
        printf("Directory full! Cannot create more files.\n");
        return;
    }

    // Check if file already exists
    for (int i = 0; i < dir->fileCount; i++) {
        if (strcmp(dir->files[i].filename, name) == 0) {
            printf("File '%s' already exists.\n", name);
            return;
        }
    }

    strcpy(dir->files[dir->fileCount].filename, name);
    dir->fileCount++;
    printf("File '%s' created successfully.\n", name);
}

// Function to display all files
void displayFiles(Directory *dir) {
    if (dir->fileCount == 0) {
        printf("Directory is empty.\n");
```

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:
```