

Name: K.SaiKrishna

Reg-No: 192311106

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

Aim:

To demonstrate the concept of multithreading in C by creating multiple threads that execute concurrently.

Algorithm:

1. **Start.**
2. Initialize the program and include the necessary libraries.
3. Define the functions that will be executed by the threads.
4. Create threads using the `pthread_create` function.
5. Execute the threads concurrently.
6. Use `pthread_join` to wait for threads to finish execution.
7. Print the results from each thread to demonstrate multithreading.
8. **End.**

Procedure:

1. Import `pthread.h` and `stdio.h` libraries.
2. Define the function for thread execution logic.
3. Use `pthread_create` to create multiple threads and pass the function as an argument.
4. Use `pthread_join` to ensure main program waits for all threads to finish.
5. Compile and run the program to observe concurrent thread execution.

Code:

```
#include <stdio.h>
```

```
#include <pthread.h>
```

```
#include <unistd.h>
```

```
void *print_message(void *thread_id) {
```

```
    int tid = *(int *)thread_id;
```

```
    printf("Thread %d is running\n", tid);
```

```
    sleep(1); // Simulate work
```

```
printf("Thread %d has finished\n", tid);

return NULL;

}

int main() {

pthread_t threads[3];

int thread_ids[3];

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

    thread_ids[i] = i + 1;

    pthread_create(&threads[i], NULL, print_message, &thread_ids[i]);

}

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

    pthread_join(threads[i], NULL);

}

printf("All threads have completed execution.\n");

return 0;

}
```

Output:

The screenshot displays a C++ IDE with a sidebar on the left containing navigation links: 'K Sai Krishna' (with a bell icon), 'New Project', 'Projects', 'Room' (with a 'new' tag), 'Programming', 'Pending Questions', 'Upgrade', and 'Logout' (with a dropdown arrow). The main editor area shows a C++ program with line numbers 33 to 44. The code defines two threads, thread1 and thread2, each with a function that prints its task name and completion status. The main function calls pthread_create to start these threads, uses pthread_join to wait for them to finish, and then prints a final message before returning 0. The output window at the bottom shows the execution results: 'Task 2: Executing thread 2.', 'Task 1: Executing thread 1.', 'Task 2: Thread 2 execution complete.', 'Task 1: Thread 1 execution complete.', and 'Main: All threads have completed.'. Below the output, it states '...Program finished with exit code 0' and 'Press ENTER to exit console.'.

```
33     perror("Failed to create thread 2");
34     exit(1);
35 }
36
37 // Wait for both threads to finish
38 pthread_join(thread1, NULL);
39 pthread_join(thread2, NULL);
40
41 printf("Main: All threads have completed.\n");
42 return 0;
43 }
44
```

Task 2: Executing thread 2.
Task 1: Executing thread 1.
Task 2: Thread 2 execution complete.
Task 1: Thread 1 execution complete.
Main: All threads have completed.

...Program finished with exit code 0
Press ENTER to exit console.

Result:

When executed, the program creates three threads. Each thread prints its start and end message, demonstrating concurrent execution