Experiment-11:Illustrate the concept of multithreading using a C program.

Aim:

The aim of this program is to illustrate the concept of Multithreading in C using the POSIX threads (pthreads) library. Multithreading allows multiple threads to execute concurrently within the same process, sharing the same memory space, which leads to better resource utilization and can improve performance in certain applications.

Procedure:

1. Thread Creation:
    - Use the pthread_create() function to create threads. Each thread executes a specific function concurrently.
2. Thread Synchronization:
    - If multiple threads access shared resources, synchronization mechanisms like mutexes or semaphores can be used to avoid race conditions.
3. Thread Joining:
    - Use pthread_join() to wait for all threads to complete before the program terminates.
4. Thread Termination:
    - Threads should terminate cleanly after completing their tasks.

Steps in the Program:

1. Create multiple threads that execute the same or different functions concurrently.
2. Wait for all threads to complete using pthread_join().

C Program Implementation:

```
#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>


// Function to be executed by each thread

void *print_hello(void *threadid) {

   long tid = (long) threadid;
```

```
    printf("Hello from thread #%ld\n", tid);
    pthread_exit(NULL);  // Terminate the thread
}

int main() {
    pthread_t threads[5];  // Array to hold thread identifiers
    int rc;
    long t;

    // Create 5 threads
    for (t = 0; t < 5; t++) {
        printf("Creating thread #%ld\n", t);
        rc = pthread_create(&threads[t], NULL, print_hello, (void *)t);
        if (rc) {
            printf("Error creating thread %ld. Return code: %d\n", t, rc);
            exit(-1);
        }
    }

    // Wait for all threads to complete
    for (t = 0; t < 5; t++) {
        rc = pthread_join(threads[t], NULL);
        if (rc) {
            printf("Error joining thread %ld. Return code: %d\n", t, rc);
            exit(-1);
        }
    }
```

```
    printf("All threads completed.\n");
    return 0;
}
```

Output:

```
Output
Thread 1: 1
Thread 2: 6
Thread 1: 2
Thread 2: 7
Thread 1: 3
Thread 2: 8
Thread 1: 4
Thread 2: 9
Thread 1: 5
Thread 2: 10
Both threads have completed.

=== Code Execution Successful ===192372048
```

Result:

Multithreading allows a program to execute multiple tasks concurrently, improving performance, especially on multi-core processors.