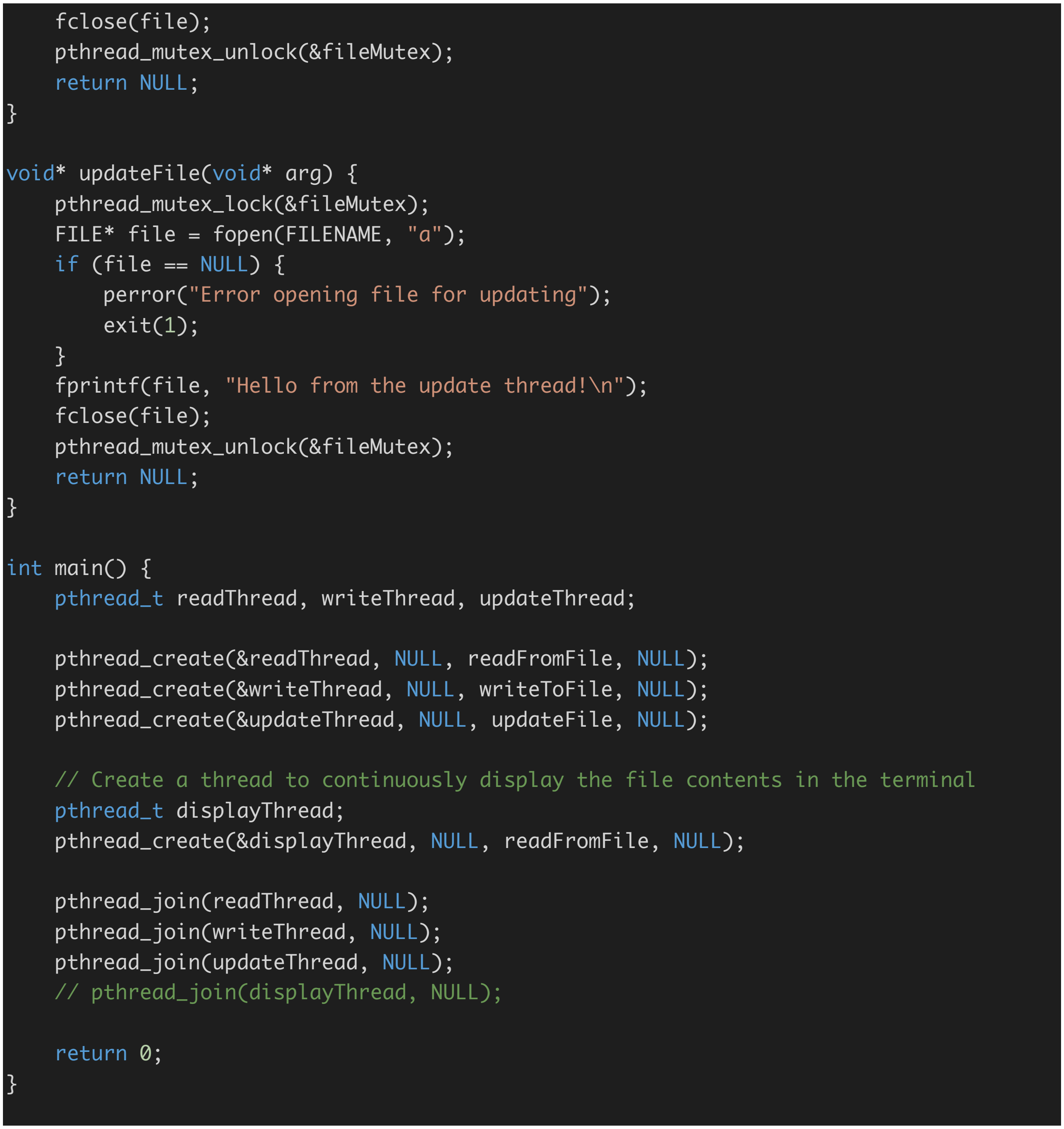
Name-Sugandh Mishra
Reg-20204211
Sec-CSE C

Motilal Nehru National Institute of Technology Allahabad Prayagraj
Distributed System (CS17201)
B.Tech (CSE) – VII Sem

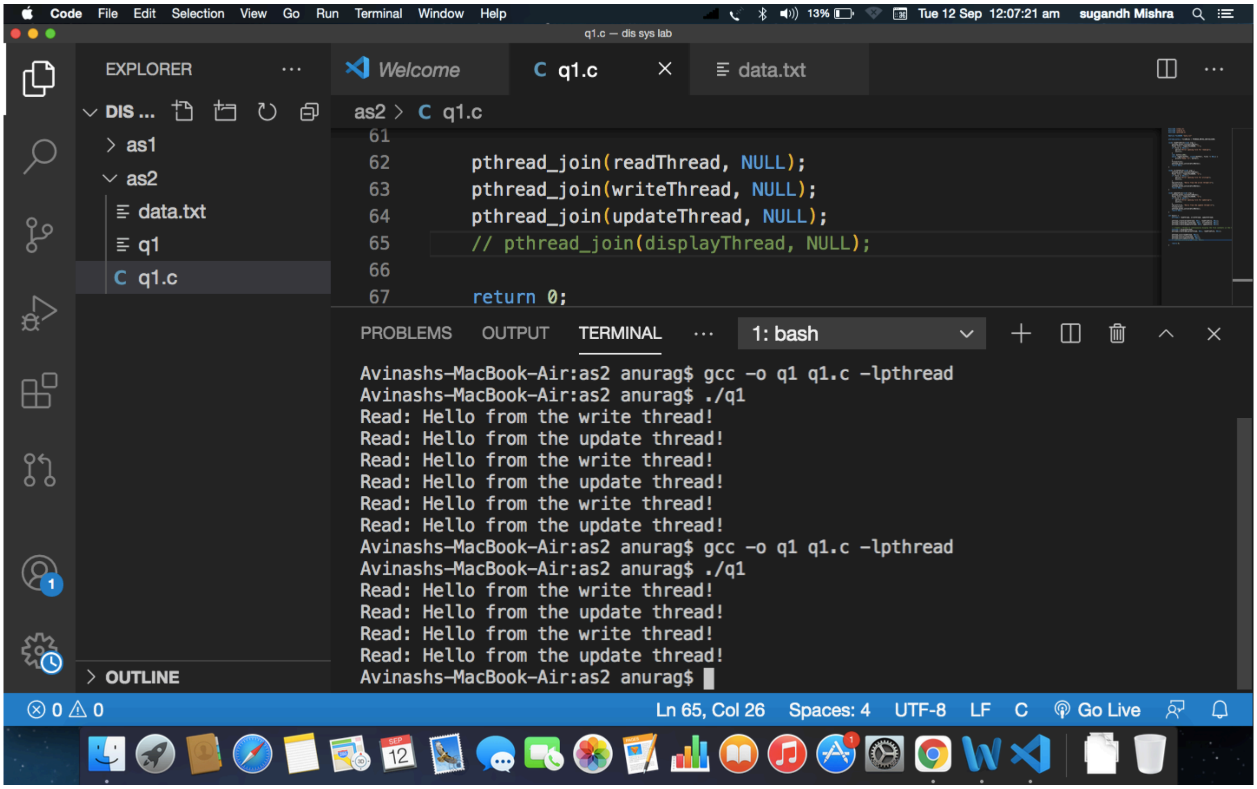
Lab 2

1. Suppose there exists a file and you have to read, write and update the file concurrently. Write a multithreaded program such that , there should be different threads for all different tasks and each thread access the file synchronously. Note: Use Mutex

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

#define FILENAME "data.txt"

pthread_mutex_t fileMutex = PTHREAD_MUTEX_INITIALIZER;

void* readFromFile(void* arg) {
    pthread_mutex_lock(&fileMutex);
    FILE* file = fopen(FILENAME, "r");
    if (file == NULL) {
        perror("Error opening file for reading");
        exit(1);
    }
    char buffer[100];
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("Read: %s", buffer);
    }
    fclose(file);
    pthread_mutex_unlock(&fileMutex);
    return NULL;
}

void* writeToFile(void* arg) {
    pthread_mutex_lock(&fileMutex);
    FILE* file = fopen(FILENAME, "a");
    if (file == NULL) {
        perror("Error opening file for writing");
        exit(1);
    }
    fprintf(file, "Hello from the write thread!\n");
```



```
    fclose(file);
    pthread_mutex_unlock(&fileMutex);
    return NULL;
}

void* updateFile(void* arg) {
    pthread_mutex_lock(&fileMutex);
    FILE* file = fopen(FILENAME, "a");
    if (file == NULL) {
        perror("Error opening file for updating");
        exit(1);
    }
    fprintf(file, "Hello from the update thread!\n");
    fclose(file);
    pthread_mutex_unlock(&fileMutex);
    return NULL;
}

int main() {
    pthread_t readThread, writeThread, updateThread;

    pthread_create(&readThread, NULL, readFromFile, NULL);
    pthread_create(&writeThread, NULL, writeToFile, NULL);
    pthread_create(&updateThread, NULL, updateFile, NULL);

    // Create a thread to continuously display the file contents in the terminal
    pthread_t displayThread;
    pthread_create(&displayThread, NULL, readFromFile, NULL);

    pthread_join(readThread, NULL);
    pthread_join(writeThread, NULL);
    pthread_join(updateThread, NULL);
    // pthread_join(displayThread, NULL);

    return 0;
}
```

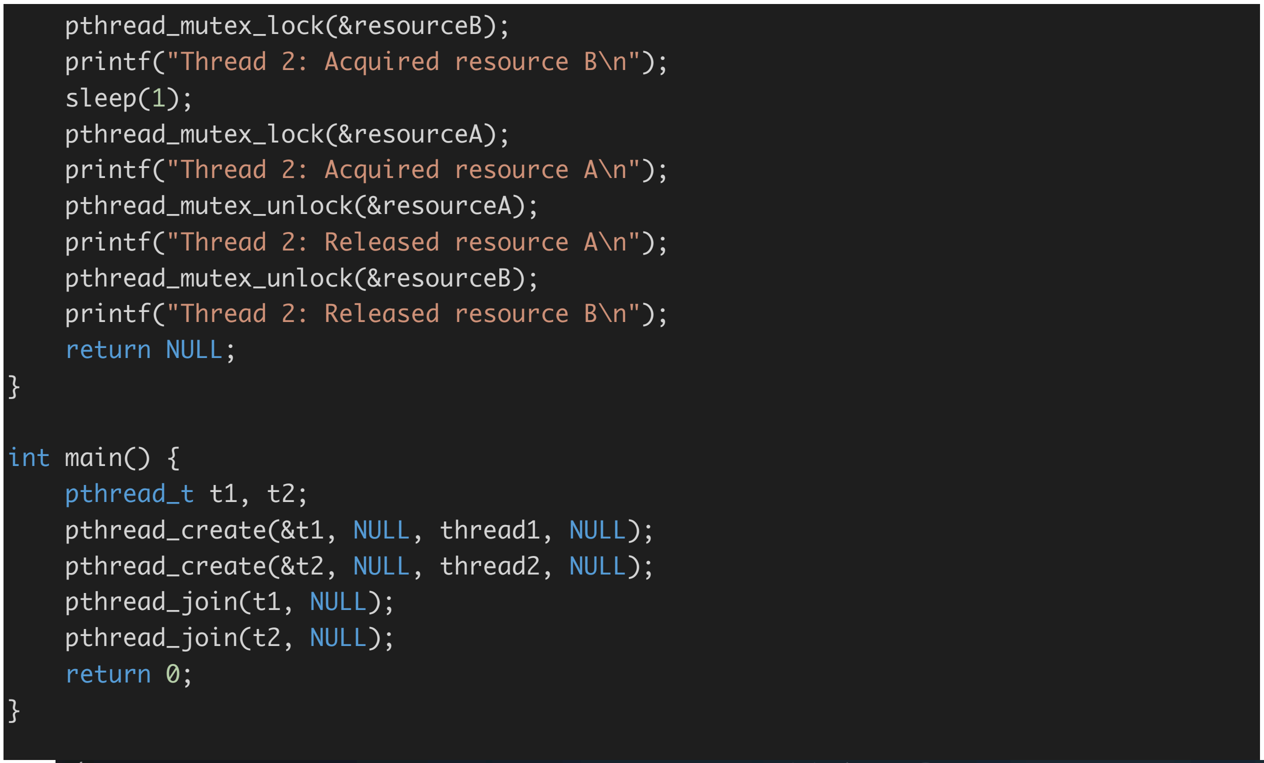

Code File Edit Selection View Go Run Terminal Window Help
Tue 12 Sep 12:07:21 am sugandh Mishra
q1.c — dis sys lab
EXPLORER
DIS ...
as1
as2
data.txt
q1
q1.c
OUTLINE
Welcome
q1.c
data.txt
as2 > q1.c
PROBLEMS OUTPUT TERMINAL
1: bash
0 0
Ln 65, Col 26 Spaces: 4 UTF-8 LF C Go Live

```
    pthread_join(readThread, NULL);
    pthread_join(writeThread, NULL);
    pthread_join(updateThread, NULL);
    // pthread_join(displayThread, NULL);

    return 0;
```

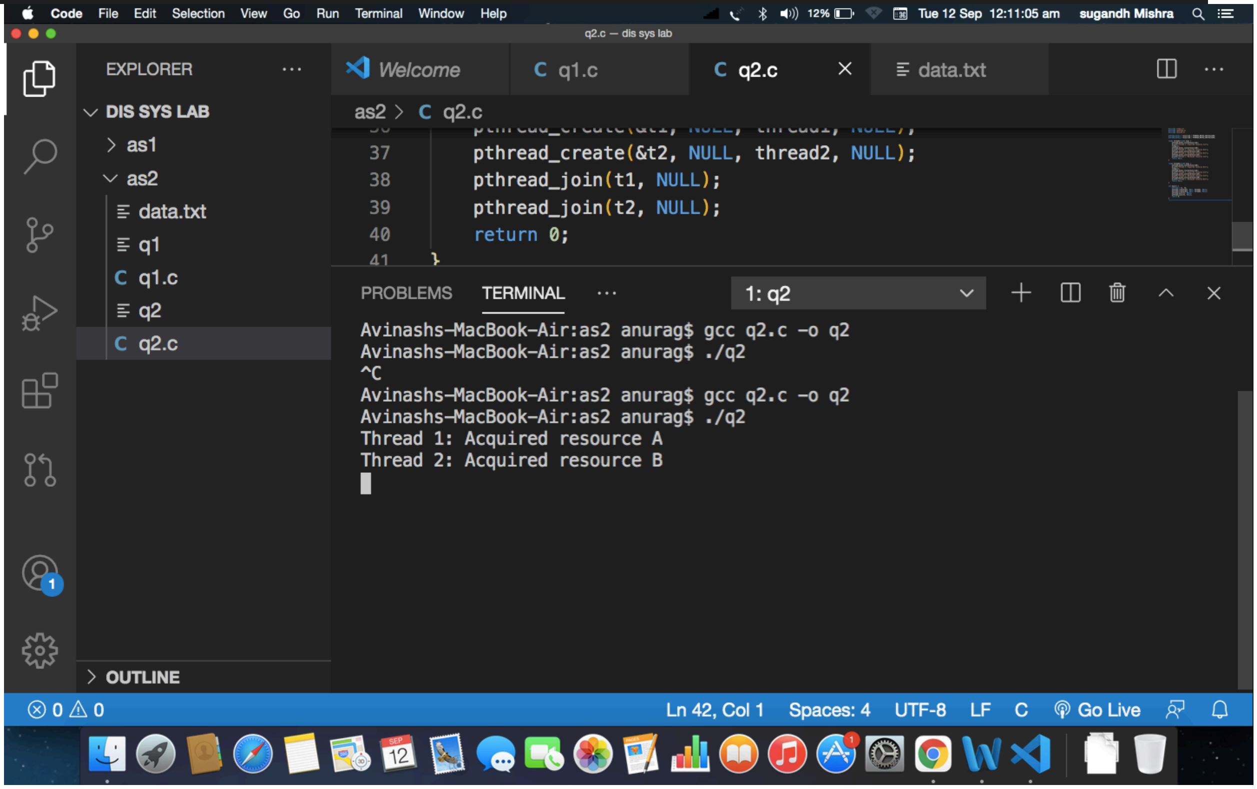
```
Avinashs-MacBook-Air:as2 anurag$ gcc -o q1 q1.c -lpthread
Avinashs-MacBook-Air:as2 anurag$ ./q1
Read: Hello from the write thread!
Read: Hello from the update thread!
Read: Hello from the write thread!
Read: Hello from the update thread!
Read: Hello from the write thread!
Read: Hello from the update thread!
Avinashs-MacBook-Air:as2 anurag$ gcc -o q1 q1.c -lpthread
Avinashs-MacBook-Air:as2 anurag$ ./q1
Read: Hello from the write thread!
Read: Hello from the update thread!
Read: Hello from the write thread!
Read: Hello from the update thread!
Avinashs-MacBook-Air:as2 anurag$ 
```

2. Write a program to implement a deadlock scenario, in which two threads are accessing two resources concurrently.

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

pthread_mutex_t resourceA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t resourceB = PTHREAD_MUTEX_INITIALIZER;

void* thread1(void* arg) {
    pthread_mutex_lock(&resourceA);
    printf("Thread 1: Acquired resource A\n");
    sleep(1);
    pthread_mutex_lock(&resourceB);
    printf("Thread 1: Acquired resource B\n");
    pthread_mutex_unlock(&resourceB);
    printf("Thread 1: Released resource B\n");
    pthread_mutex_unlock(&resourceA);
    printf("Thread 1: Released resource A\n");
    return NULL;
}

void* thread2(void* arg) {
```



```
    pthread_mutex_lock(&resourceB);
    printf("Thread 2: Acquired resource B\n");
    sleep(1);
    pthread_mutex_lock(&resourceA);
    printf("Thread 2: Acquired resource A\n");
    pthread_mutex_unlock(&resourceA);
    printf("Thread 2: Released resource A\n");
    pthread_mutex_unlock(&resourceB);
    printf("Thread 2: Released resource B\n");
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

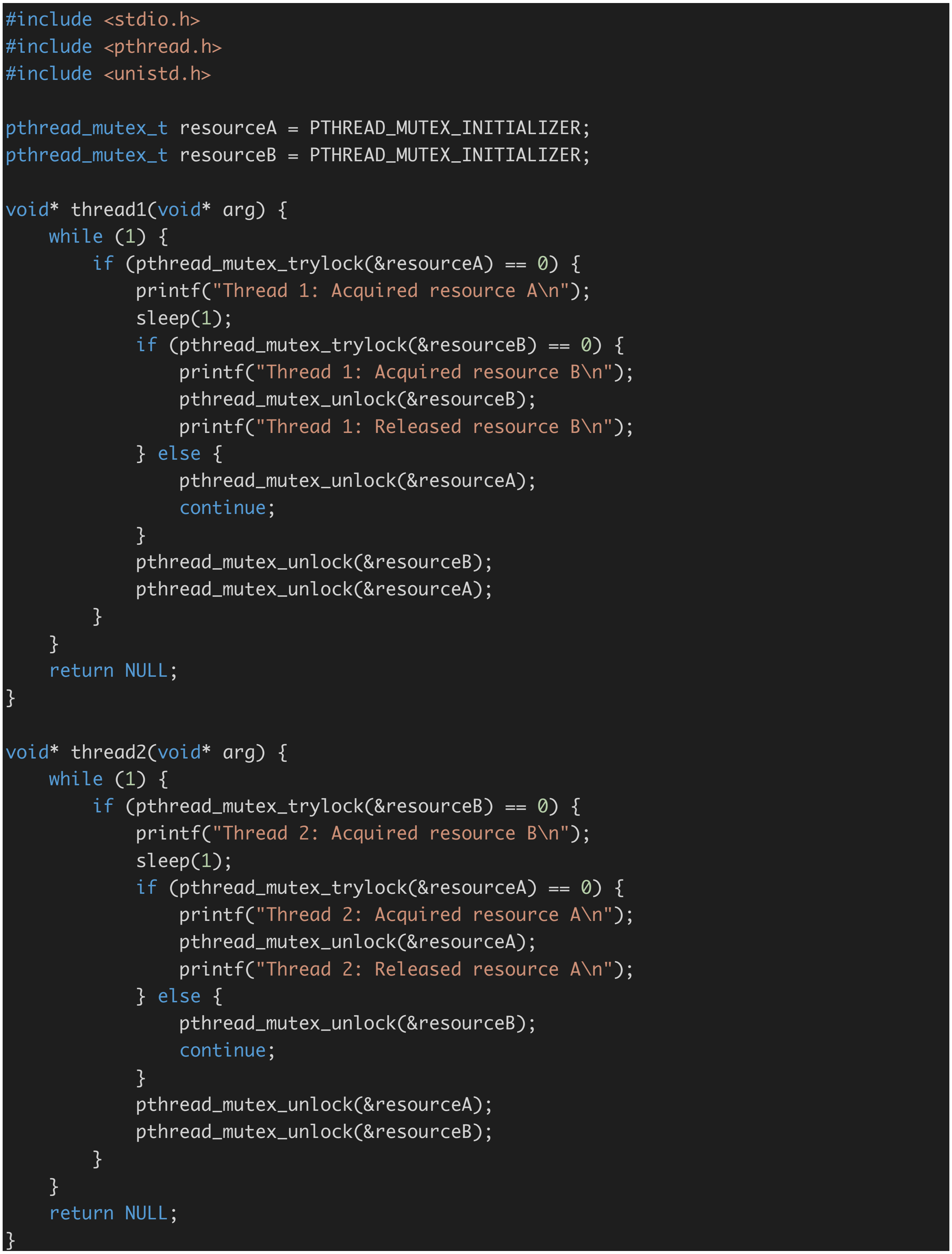


Code File Edit Selection View Go Run Terminal Window Help
Tue 12 Sep 12:11:05 am sugandh Mishra
q2.c — dis sys lab
EXPLORER
DIS SYS LAB
as1
as2
data.txt
q1
q1.c
q2
q2.c
OUTLINE
Welcome
q1.c
q2.c
data.txt
as2 > q2.c
PROBLEMS
TERMINAL
1: q2
Ln 42, Col 1 Spaces: 4 UTF-8 LF C Go Live

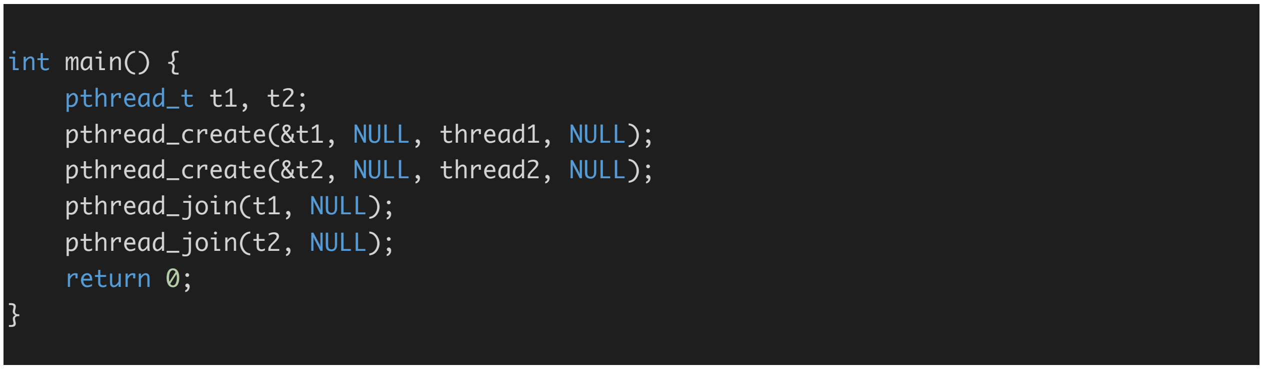
```
    pthread_create(&t2, NULL, thread2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

```
Avinashs-MacBook-Air:as2 anurag$ gcc q2.c -o q2
Avinashs-MacBook-Air:as2 anurag$ ./q2
^C
Avinashs-MacBook-Air:as2 anurag$ gcc q2.c -o q2
Avinashs-MacBook-Air:as2 anurag$ ./q2
Thread 1: Acquired resource A
Thread 2: Acquired resource B
```

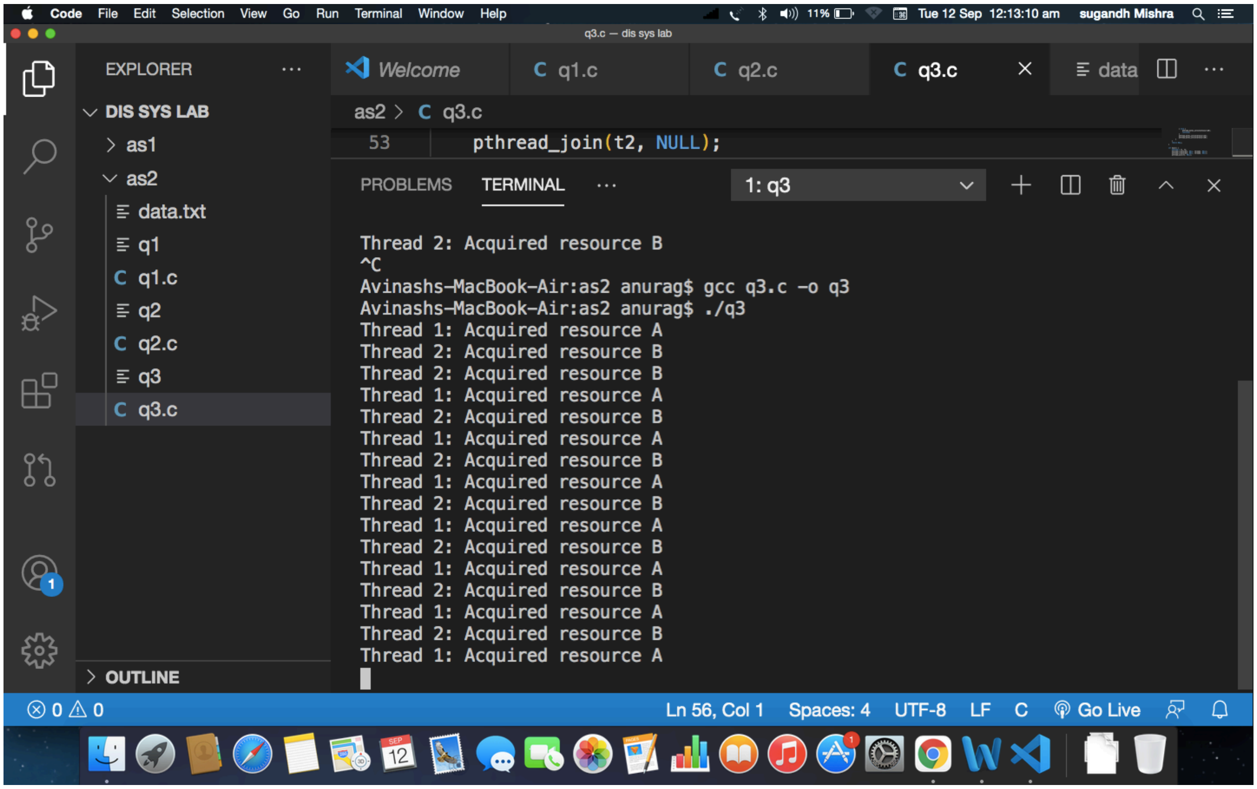
3. Write a program to implement deadlock avoidance using conditional locking in which two threads are accessing two resources concurrently. Note: user pthread_mutex_trylock() functional locking



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

pthread_mutex_t resourceA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t resourceB = PTHREAD_MUTEX_INITIALIZER;

void* thread1(void* arg) {
    while (1) {
        if (pthread_mutex_trylock(&resourceA) == 0) {
            printf("Thread 1: Acquired resource A\n");
            sleep(1);
            if (pthread_mutex_trylock(&resourceB) == 0) {
                printf("Thread 1: Acquired resource B\n");
                pthread_mutex_unlock(&resourceB);
                printf("Thread 1: Released resource B\n");
            } else {
                pthread_mutex_unlock(&resourceA);
                continue;
            }
            pthread_mutex_unlock(&resourceB);
            pthread_mutex_unlock(&resourceA);
        }
    }
    return NULL;
}

void* thread2(void* arg) {
    while (1) {
        if (pthread_mutex_trylock(&resourceB) == 0) {
            printf("Thread 2: Acquired resource B\n");
            sleep(1);
            if (pthread_mutex_trylock(&resourceA) == 0) {
                printf("Thread 2: Acquired resource A\n");
                pthread_mutex_unlock(&resourceA);
                printf("Thread 2: Released resource A\n");
            } else {
                pthread_mutex_unlock(&resourceB);
                continue;
            }
            pthread_mutex_unlock(&resourceA);
            pthread_mutex_unlock(&resourceB);
        }
    }
    return NULL;
}
```



```c
int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```



```c
pthread_join(t2, NULL);
```

```
Thread 2: Acquired resource B
^C
Avinashs-MacBook-Air:as2 anurag$ gcc q3.c -o q3
Avinashs-MacBook-Air:as2 anurag$ ./q3
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 2: Acquired resource B
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 1: Acquired resource A
Thread 2: Acquired resource B
Thread 1: Acquired resource A
```