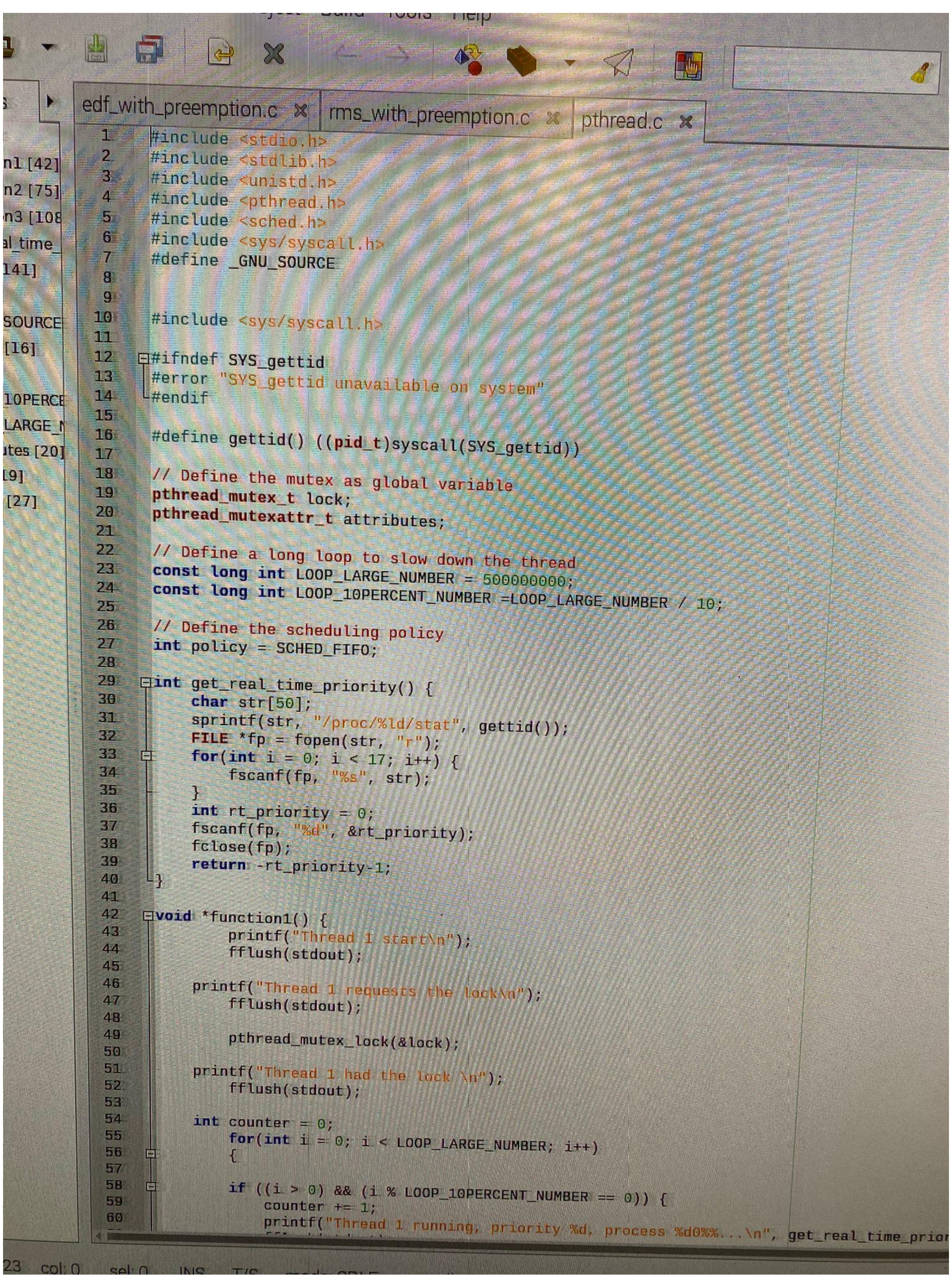
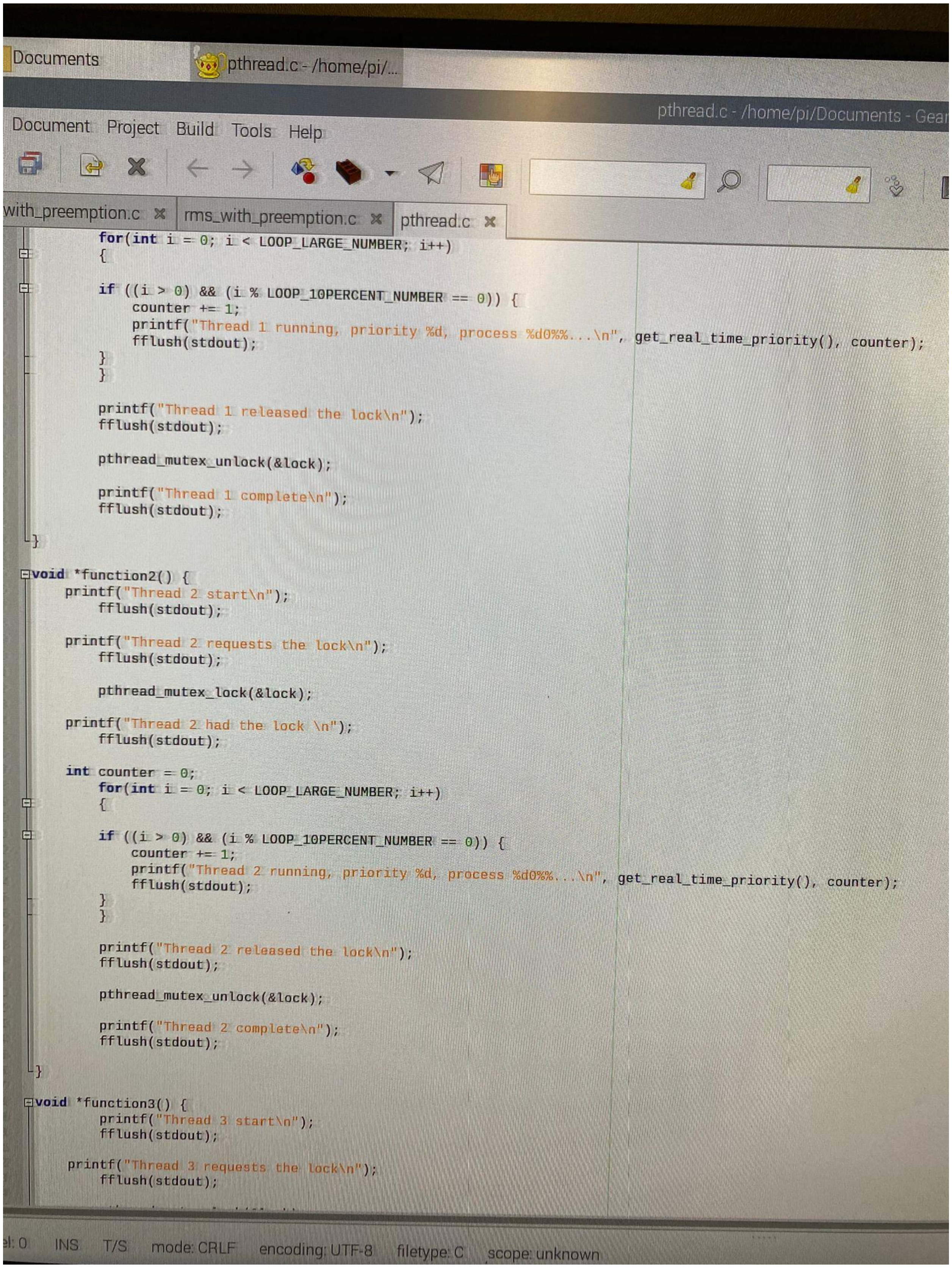
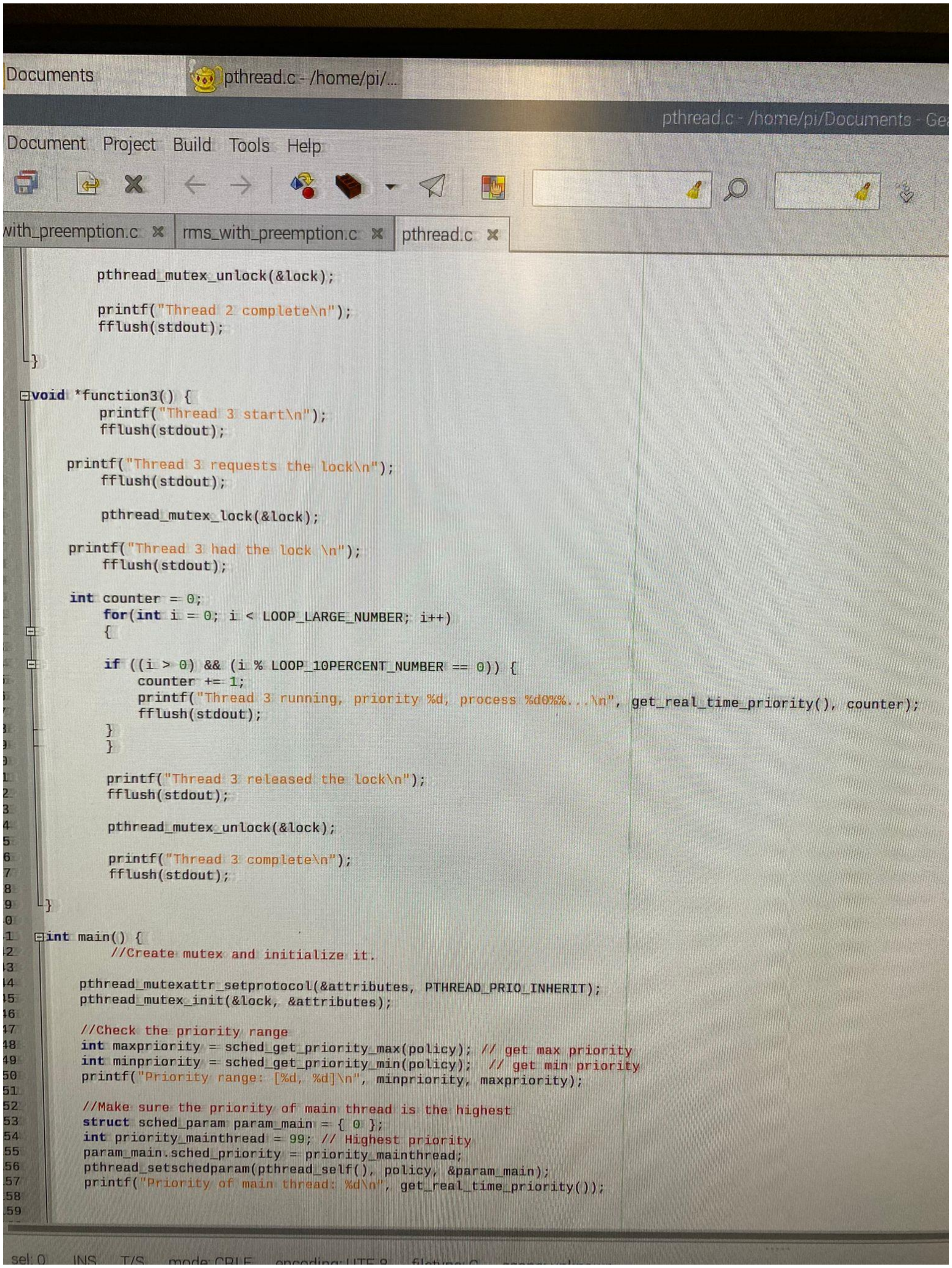


```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#include <sys/syscall.h>
#define _GNU_SOURCE


#include <sys/syscall.h>

#ifndef SYS_gettid
#error "SYS_gettid unavailable on system"
#endif

#define gettid() ((pid_t)syscall(SYS_gettid))

// Define the mutex as global variable
pthread_mutex_t lock;
pthread_mutexattr_t attributes;

// Define a long loop to slow down the thread
const long int LOOP_LARGE_NUMBER = 500000000;
const long int LOOP_10PERCENT_NUMBER =LOOP_LARGE_NUMBER / 10;

// Define the scheduling policy
int policy = SCHED_FIFO;

int get_real_time_priority() {
    char str[50];
    sprintf(str, "/proc/%ld/stat", gettid());
    FILE *fp = fopen(str, "r");
    for(int i = 0; i < 17; i++) {
        fscanf(fp, "%s", str);
    }
    int rt_priority = 0;
    fscanf(fp, "%d", &rt_priority);
    fclose(fp);
    return -rt_priority-1;
}

void *function1() {
        printf("Thread 1 start\n");
        fflush(stdout);

    printf("Thread 1 requests the lock\n");
        fflush(stdout);

        pthread_mutex_lock(&lock);

    printf("Thread 1 had the lock \n");
        fflush(stdout);

    int counter = 0;
        for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
        {

        if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
            counter += 1;
            printf("Thread 1 running, priority %d, process %d0%%...\n", get_real_time_prior
```

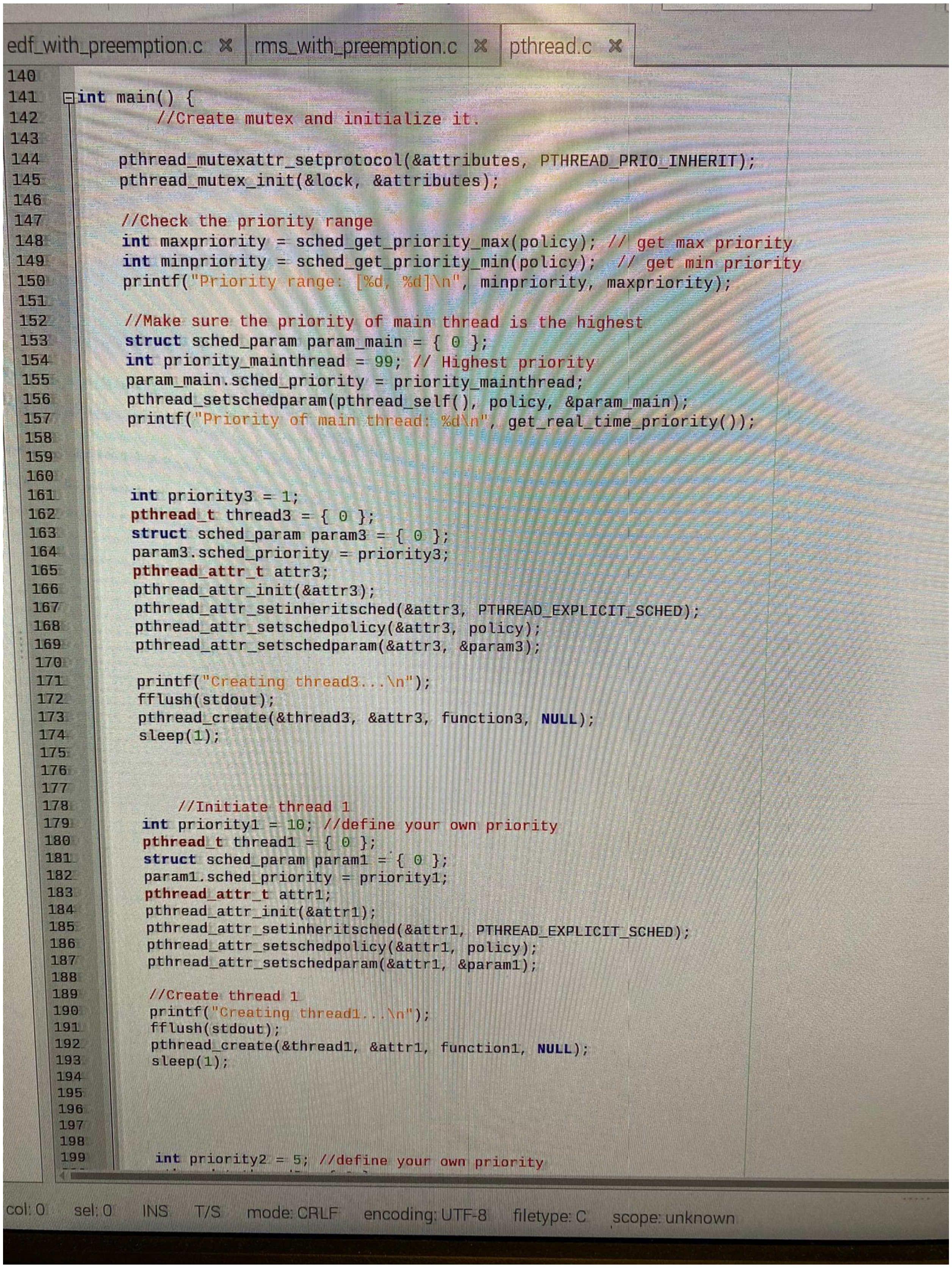


```
        for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
        {

        if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
            counter += 1;
            printf("Thread 1 running, priority %d, process %d0%%...\n", get_real_time_priority(), counter);
            fflush(stdout);
        }
        }

        printf("Thread 1 released the lock\n");
        fflush(stdout);

        pthread_mutex_unlock(&lock);

        printf("Thread 1 complete\n");
        fflush(stdout);

}

void *function2() {
    printf("Thread 2 start\n");
        fflush(stdout);

    printf("Thread 2 requests the lock\n");
        fflush(stdout);

        pthread_mutex_lock(&lock);

    printf("Thread 2 had the lock \n");
        fflush(stdout);

    int counter = 0;
        for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
        {

        if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
            counter += 1;
            printf("Thread 2 running, priority %d, process %d0%%...\n", get_real_time_priority(), counter);
            fflush(stdout);
        }
        }

        printf("Thread 2 released the lock\n");
        fflush(stdout);

        pthread_mutex_unlock(&lock);

        printf("Thread 2 complete\n");
        fflush(stdout);

}

void *function3() {
        printf("Thread 3 start\n");
        fflush(stdout);

    printf("Thread 3 requests the lock\n");
        fflush(stdout);
```

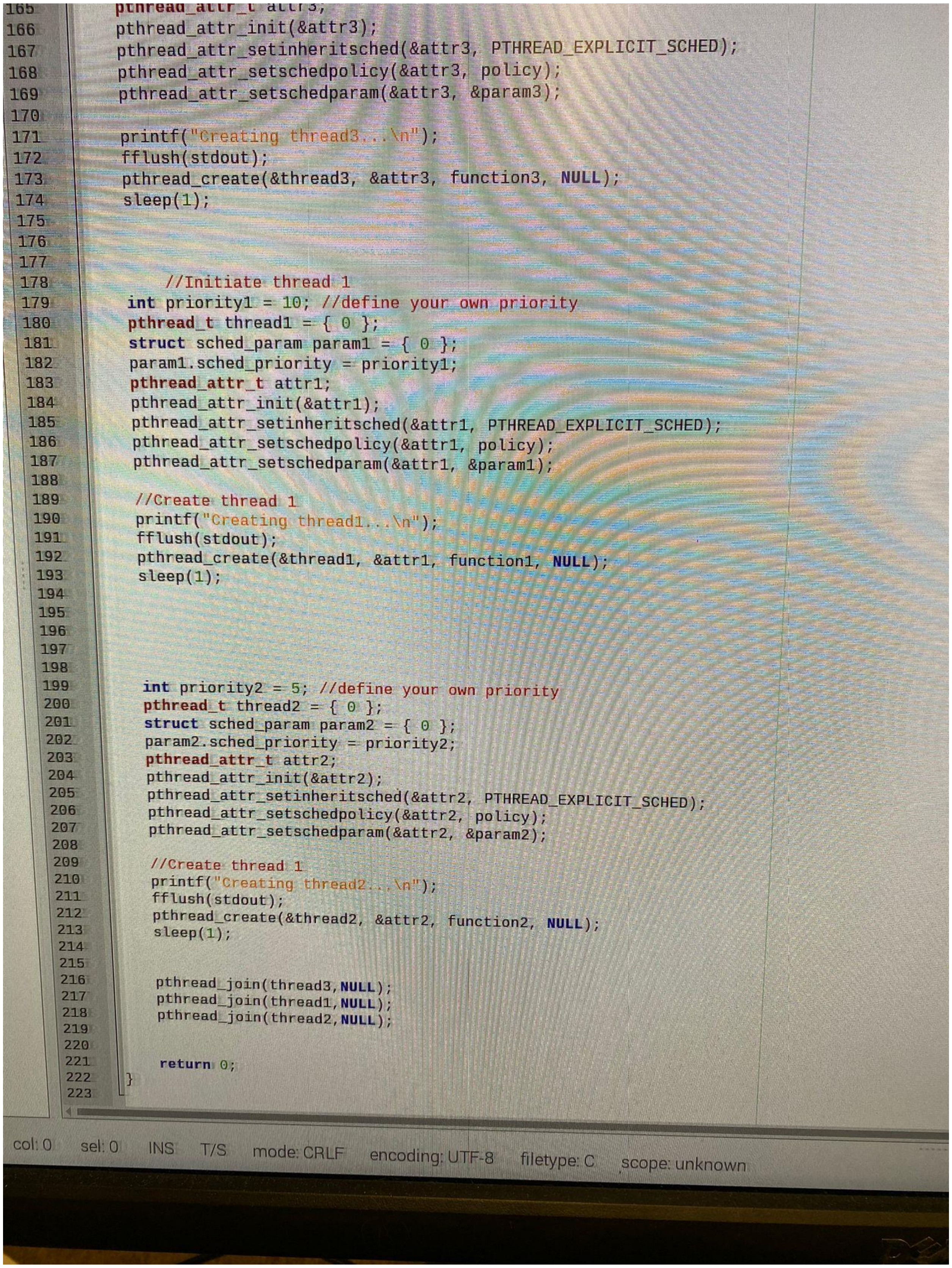
el: 0 INS T/S mode: CRLF encoding: UTF-8 filetype: C scope: unknown



```c
        pthread_mutex_unlock(&lock);

        printf("Thread 2 complete\n");
        fflush(stdout);

}

void *function3() {
        printf("Thread 3 start\n");
        fflush(stdout);

    printf("Thread 3 requests the lock\n");
        fflush(stdout);

        pthread_mutex_lock(&lock);

    printf("Thread 3 had the lock \n");
        fflush(stdout);

    int counter = 0;
        for(int i = 0; i < LOOP_LARGE_NUMBER; i++)
        {

        if ((i > 0) && (i % LOOP_10PERCENT_NUMBER == 0)) {
            counter += 1;
            printf("Thread 3 running, priority %d, process %d0%%...\n", get_real_time_priority(), counter);
            fflush(stdout);
        }
        }

        printf("Thread 3 released the lock\n");
        fflush(stdout);

        pthread_mutex_unlock(&lock);

        printf("Thread 3 complete\n");
        fflush(stdout);
}

int main() {
        //Create mutex and initialize it.

    pthread_mutexattr_setprotocol(&attributes, PTHREAD_PRIO_INHERIT);
    pthread_mutex_init(&lock, &attributes);

    //Check the priority range
    int maxpriority = sched_get_priority_max(policy); // get max priority
    int minpriority = sched_get_priority_min(policy);  // get min priority
    printf("Priority range: [%d, %d]\n", minpriority, maxpriority);

    //Make sure the priority of main thread is the highest
    struct sched_param param_main = { 0 };
    int priority_mainthread = 99; // Highest priority
    param_main.sched_priority = priority_mainthread;
    pthread_setschedparam(pthread_self(), policy, &param_main);
    printf("Priority of main thread: %d\n", get_real_time_priority());
```



```c
int main() {
        //Create mutex and initialize it.

    pthread_mutexattr_setprotocol(&attributes, PTHREAD_PRIO_INHERIT);
    pthread_mutex_init(&lock, &attributes);

    //Check the priority range
    int maxpriority = sched_get_priority_max(policy); // get max priority
    int minpriority = sched_get_priority_min(policy);  // get min priority
    printf("Priority range: [%d, %d]\n", minpriority, maxpriority);

    //Make sure the priority of main thread is the highest
    struct sched_param param_main = { 0 };
    int priority_mainthread = 99; // Highest priority
    param_main.sched_priority = priority_mainthread;
    pthread_setschedparam(pthread_self(), policy, &param_main);
    printf("Priority of main thread: %d\n", get_real_time_priority());



    int priority3 = 1;
    pthread_t thread3 = { 0 };
    struct sched_param param3 = { 0 };
    param3.sched_priority = priority3;
    pthread_attr_t attr3;
    pthread_attr_init(&attr3);
    pthread_attr_setinheritsched(&attr3, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr3, policy);
    pthread_attr_setschedparam(&attr3, &param3);

    printf("Creating thread3...\n");
    fflush(stdout);
    pthread_create(&thread3, &attr3, function3, NULL);
    sleep(1);



        //Initiate thread 1
    int priority1 = 10; //define your own priority
    pthread_t thread1 = { 0 };
    struct sched_param param1 = { 0 };
    param1.sched_priority = priority1;
    pthread_attr_t attr1;
    pthread_attr_init(&attr1);
    pthread_attr_setinheritsched(&attr1, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr1, policy);
    pthread_attr_setschedparam(&attr1, &param1);

    //Create thread 1
    printf("Creating thread1...\n");
    fflush(stdout);
    pthread_create(&thread1, &attr1, function1, NULL);
    sleep(1);





    int priority2 = 5; //define your own priority
```



```c
    pthread_attr_t attr3;
    pthread_attr_init(&attr3);
    pthread_attr_setinheritsched(&attr3, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr3, policy);
    pthread_attr_setschedparam(&attr3, &param3);

    printf("Creating thread3...\n");
    fflush(stdout);
    pthread_create(&thread3, &attr3, function3, NULL);
    sleep(1);



        //Initiate thread 1
    int priority1 = 10; //define your own priority
    pthread_t thread1 = { 0 };
    struct sched_param param1 = { 0 };
    param1.sched_priority = priority1;
    pthread_attr_t attr1;
    pthread_attr_init(&attr1);
    pthread_attr_setinheritsched(&attr1, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr1, policy);
    pthread_attr_setschedparam(&attr1, &param1);

    //Create thread 1
    printf("Creating thread1...\n");
    fflush(stdout);
    pthread_create(&thread1, &attr1, function1, NULL);
    sleep(1);





    int priority2 = 5; //define your own priority
    pthread_t thread2 = { 0 };
    struct sched_param param2 = { 0 };
    param2.sched_priority = priority2;
    pthread_attr_t attr2;
    pthread_attr_init(&attr2);
    pthread_attr_setinheritsched(&attr2, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr2, policy);
    pthread_attr_setschedparam(&attr2, &param2);

    //Create thread 1
    printf("Creating thread2...\n");
    fflush(stdout);
    pthread_create(&thread2, &attr2, function2, NULL);
    sleep(1);


    pthread_join(thread3,NULL);
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);


    return 0;
}
```

col: 0 sel: 0 INS T/S mode: CRLF encoding: UTF-8 filetype: C scope: unknown