Operating System Lab

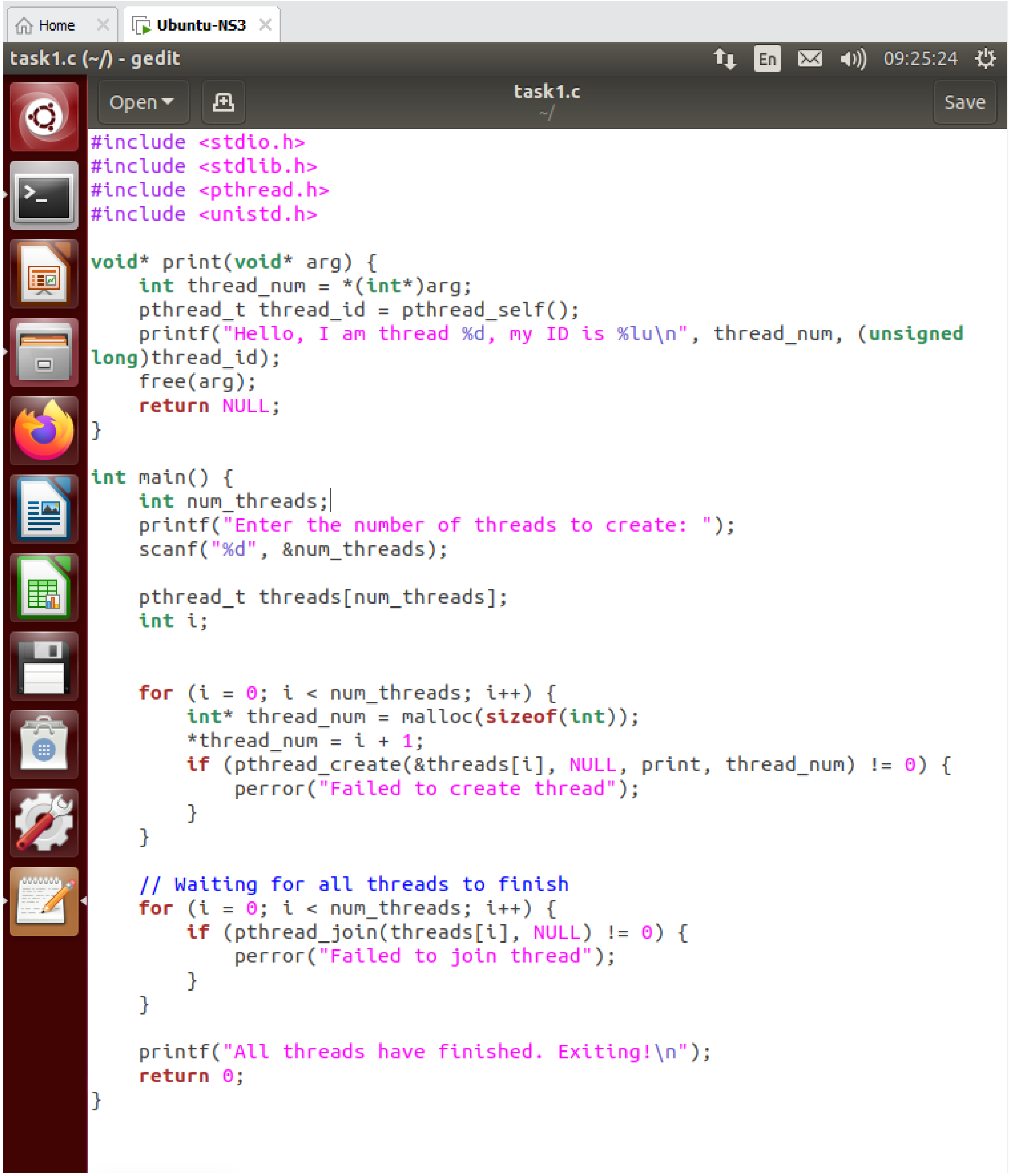
22F-3168

Mariam Fatima

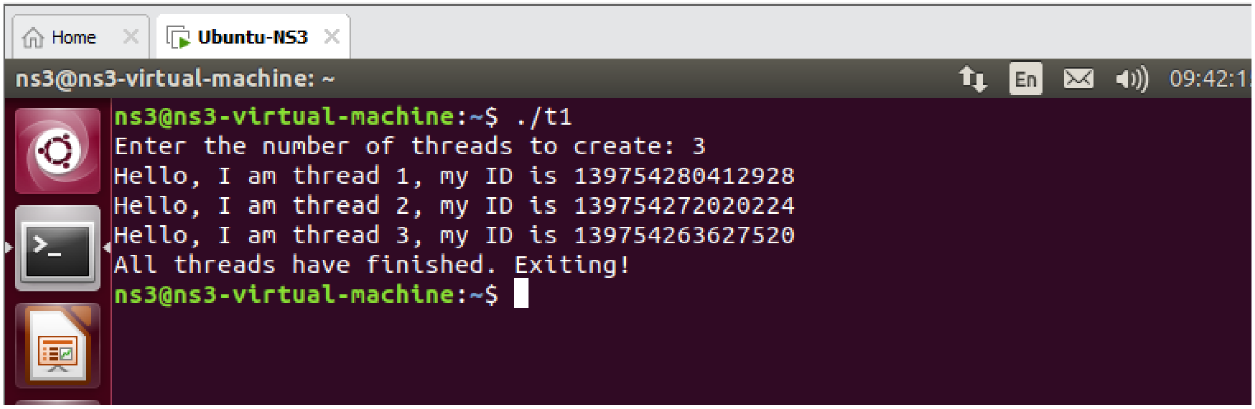
Lab 8

Task 1:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>


void* print(void* arg) {

    int thread_num = *(int*)arg;

    pthread_t thread_id = pthread_self();

    printf("Hello, I am thread %d, my ID is %lu\n", thread_num, (unsigned long)thread_id);

    free(arg);

    return NULL;

}


int main() {

    int num_threads;

    printf("Enter the number of threads to create: ");

    scanf("%d", &num_threads);


    pthread_t threads[num_threads];

    int i;



    for (i = 0; i < num_threads; i++) {
        int* thread_num = malloc(sizeof(int));
        *thread_num = i + 1;
        if (pthread_create(&threads[i], NULL, print, thread_num) != 0) {
            perror("Failed to create thread");
        }
    }

    // Waiting for all threads to finish
    for (i = 0; i < num_threads; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("Failed to join thread");
        }
    }

    printf("All threads have finished. Exiting!\n");
    return 0;
}
```



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

void* print(void* arg) {
    int thread_num = *(int*)arg;
    pthread_t thread_id = pthread_self();
    printf("Hello, I am thread %d, my ID is %lu\n", thread_num, (unsigned
long)thread_id);
    free(arg);
    return NULL;
}

int main() {
    int num_threads;
    printf("Enter the number of threads to create: ");
    scanf("%d", &num_threads);

    pthread_t threads[num_threads];
    int i;


    for (i = 0; i < num_threads; i++) {
        int* thread_num = malloc(sizeof(int));
        *thread_num = i + 1;
        if (pthread_create(&threads[i], NULL, print, thread_num) != 0) {
            perror("Failed to create thread");
        }
    }

    // Waiting for all threads to finish
    for (i = 0; i < num_threads; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("Failed to join thread");
        }
    }

    printf("All threads have finished. Exiting!\n");
    return 0;
}
```

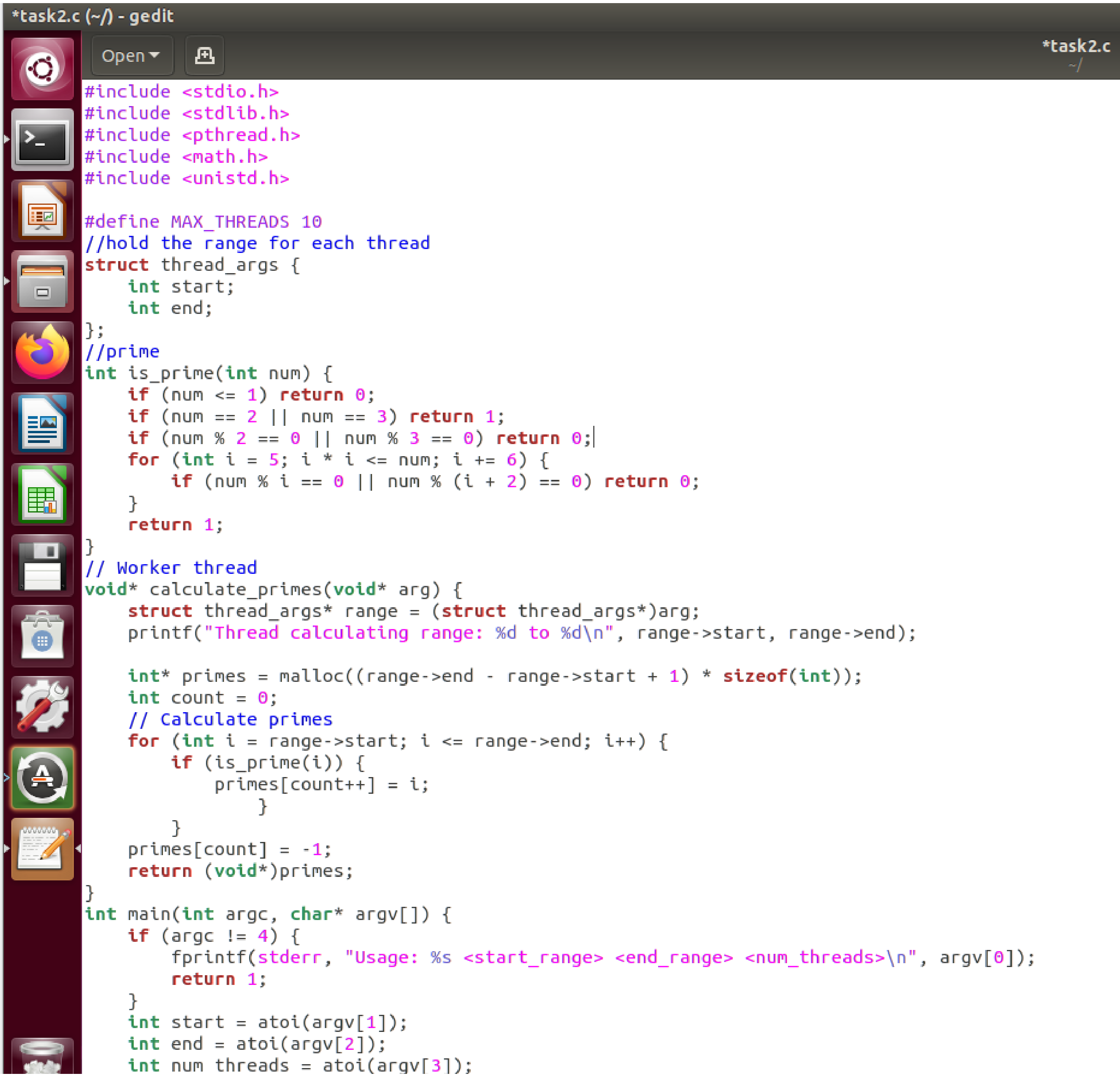


```
ns3@ns3-virtual-machine:~$ ./t1
Enter the number of threads to create: 3
Hello, I am thread 1, my ID is 139754280412928
Hello, I am thread 2, my ID is 139754272020224
Hello, I am thread 3, my ID is 139754263627520
All threads have finished. Exiting!
ns3@ns3-virtual-machine:~$
```

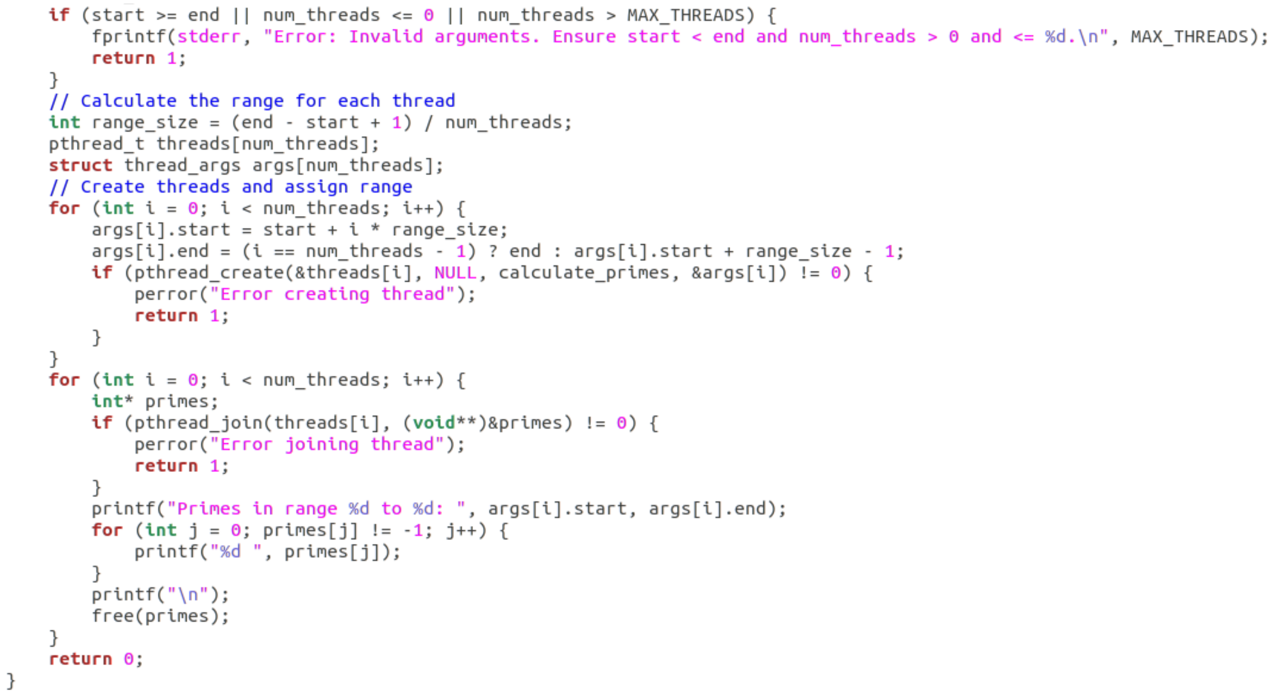
Task 2:

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

#define MAX_THREADS 10

//hold the range for each thread
struct thread_args {
   int start;
   int end;
};

//prime
int is_prime(int num) {
   if (num <= 1) return 0;
   if (num == 2 || num == 3) return 1;
   if (num % 2 == 0 || num % 3 == 0) return 0;
```

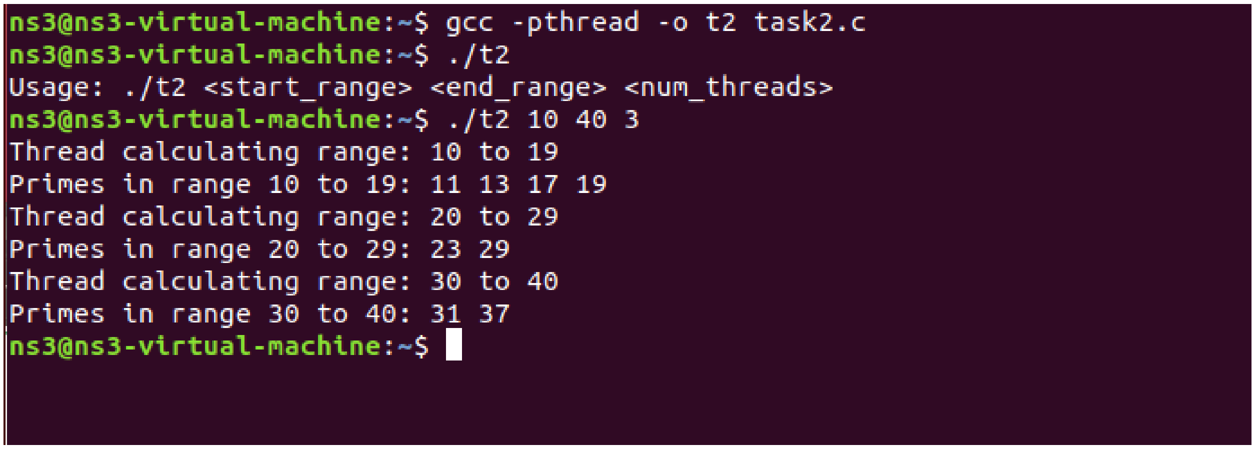
```
    for (int i = 5; i * i <= num; i += 6) {
        if (num % i == 0 || num % (i + 2) == 0) return 0;
    }
    return 1;
}

// Worker thread
void* calculate_primes(void* arg) {
    struct thread_args* range = (struct thread_args*)arg;
    printf("Thread calculating range: %d to %d\n", range->start, range->end);

    int* primes = malloc((range->end - range->start + 1) * sizeof(int));
    int count = 0;

    // Calculate primes
    for (int i = range->start; i <= range->end; i++) {
        if (is_prime(i)) {
            primes[count++] = i;
        }
    }

    primes[count] = -1;
    return (void*)primes;
}

int main(int argc, char* argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <start_range> <end_range> <num_threads>\n", argv[0]);
        return 1;
    }

    int start = atoi(argv[1]);
    int end = atoi(argv[2]);
    int num_threads = atoi(argv[3]);

    if (start >= end || num_threads <= 0 || num_threads > MAX_THREADS) {
        fprintf(stderr, "Error: Invalid arguments. Ensure start < end and num_threads > 0 and <= %d.\n",
MAX_THREADS);
        return 1;
    }

    // Calculate the range each thread will handle
    int range_size = (end - start + 1) / num_threads;
    pthread_t threads[num_threads];
    struct thread_args args[num_threads];

    // Create threads and assign range
    for (int i = 0; i < num_threads; i++) {
        args[i].start = start + i * range_size;
        args[i].end = (i == num_threads - 1) ? end : args[i].start + range_size - 1;

        if (pthread_create(&threads[i], NULL, calculate_primes, &args[i]) != 0) {
            perror("Error creating thread");
            return 1;
        }
    }
    for (int i = 0; i < num_threads; i++) {
        int* primes;
        if (pthread_join(threads[i], (void**)&primes) != 0) {
            perror("Error joining thread");
            return 1;
        }

        printf("Primes in range %d to %d: ", args[i].start, args[i].end);
        for (int j = 0; primes[j] != -1; j++) {
            printf("%d ", primes[j]);
        }
        printf("\n");

        free(primes);
    }

    return 0;
}
```



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

#define MAX_THREADS 10
//hold the range for each thread
struct thread_args {
    int start;
    int end;
};
//prime
int is_prime(int num) {
    if (num <= 1) return 0;
    if (num == 2 || num == 3) return 1;
    if (num % 2 == 0 || num % 3 == 0) return 0;
    for (int i = 5; i * i <= num; i += 6) {
        if (num % i == 0 || num % (i + 2) == 0) return 0;
    }
    return 1;
}
// Worker thread
void* calculate_primes(void* arg) {
    struct thread_args* range = (struct thread_args*)arg;
    printf("Thread calculating range: %d to %d\n", range->start, range->end);

    int* primes = malloc((range->end - range->start + 1) * sizeof(int));
    int count = 0;
    // Calculate primes
    for (int i = range->start; i <= range->end; i++) {
        if (is_prime(i)) {
            primes[count++] = i;
                }
        }
    primes[count] = -1;
    return (void*)primes;
}
int main(int argc, char* argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <start_range> <end_range> <num_threads>\n", argv[0]);
        return 1;
    }
    int start = atoi(argv[1]);
    int end = atoi(argv[2]);
    int num_threads = atoi(argv[3]);
```



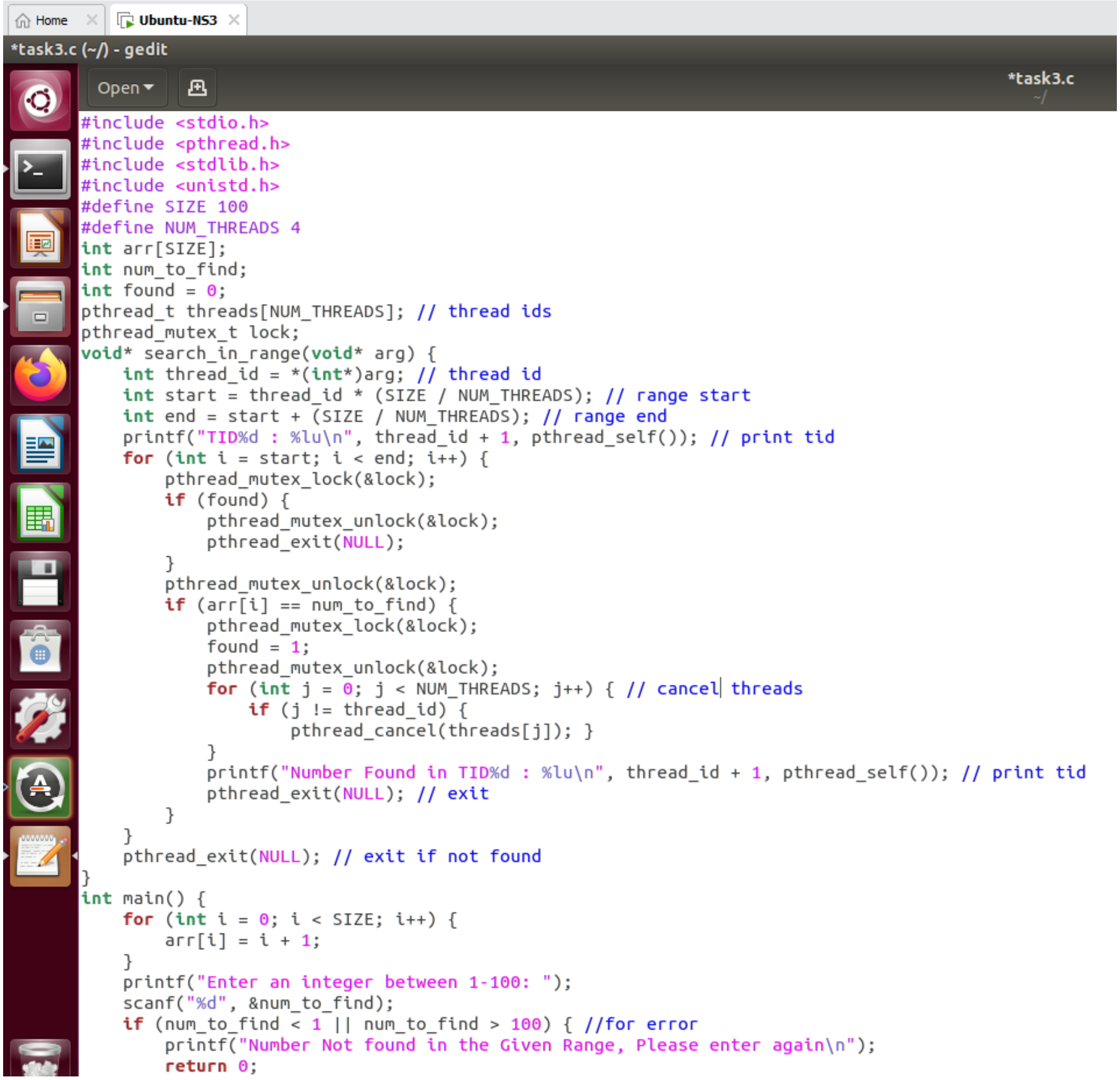
```
    if (start >= end || num_threads <= 0 || num_threads > MAX_THREADS) {
        fprintf(stderr, "Error: Invalid arguments. Ensure start < end and num_threads > 0 and <= %d.\n", MAX_THREADS);
        return 1;
    }
    // Calculate the range for each thread
    int range_size = (end - start + 1) / num_threads;
    pthread_t threads[num_threads];
    struct thread_args args[num_threads];
    // Create threads and assign range
    for (int i = 0; i < num_threads; i++) {
        args[i].start = start + i * range_size;
        args[i].end = (i == num_threads - 1) ? end : args[i].start + range_size - 1;
        if (pthread_create(&threads[i], NULL, calculate_primes, &args[i]) != 0) {
            perror("Error creating thread");
            return 1;
        }
    }
    for (int i = 0; i < num_threads; i++) {
        int* primes;
        if (pthread_join(threads[i], (void**)&primes) != 0) {
            perror("Error joining thread");
            return 1;
        }
        printf("Primes in range %d to %d: ", args[i].start, args[i].end);
        for (int j = 0; primes[j] != -1; j++) {
            printf("%d ", primes[j]);
        }
        printf("\n");
        free(primes);
    }
    return 0;
}
```



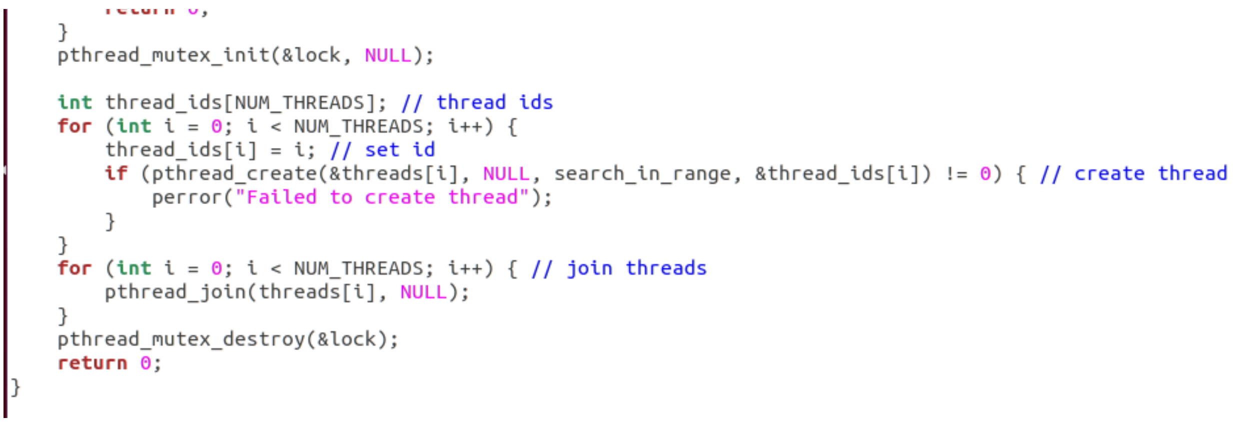
```
ns3@ns3-virtual-machine:~$ gcc -pthread -o t2 task2.c
ns3@ns3-virtual-machine:~$ ./t2
Usage: ./t2 <start_range> <end_range> <num_threads>
ns3@ns3-virtual-machine:~$ ./t2 10 40 3
Thread calculating range: 10 to 19
Primes in range 10 to 19: 11 13 17 19
Thread calculating range: 20 to 29
Primes in range 20 to 29: 23 29
Thread calculating range: 30 to 40
Primes in range 30 to 40: 31 37
ns3@ns3-virtual-machine:~$
```

Task 3:

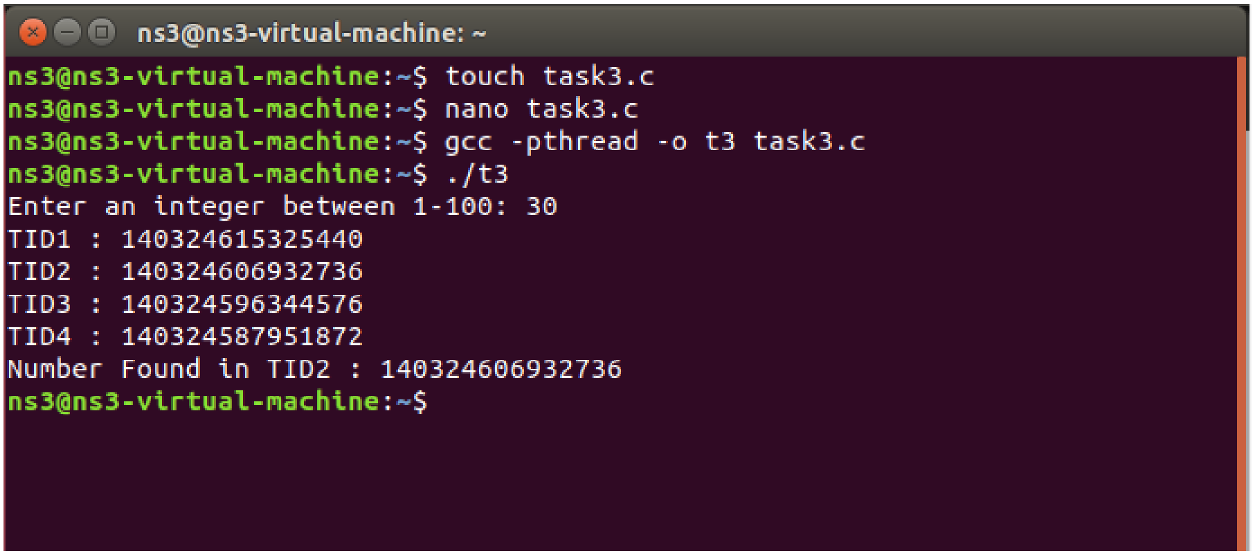
```
#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

#define SIZE 100

#define NUM_THREADS 4

int arr[SIZE];

int num_to_find;
int found = 0;
pthread_t threads[NUM_THREADS]; // thread ids
pthread_mutex_t lock;
void* search_in_range(void* arg) {
    int thread_id = *(int*)arg; // thread id
    int start = thread_id * (SIZE / NUM_THREADS); // range start
    int end = start + (SIZE / NUM_THREADS); // range end
    printf("TID%d : %lu\n", thread_id + 1, pthread_self()); // print tid
    for (int i = start; i < end; i++) {
        pthread_mutex_lock(&lock);
        if (found) {
            pthread_mutex_unlock(&lock);
            pthread_exit(NULL);
        }
        pthread_mutex_unlock(&lock);
        if (arr[i] == num_to_find) {
            pthread_mutex_lock(&lock);
            found = 1;
            pthread_mutex_unlock(&lock);
            for (int j = 0; j < NUM_THREADS; j++) { // cancel other threads
                if (j != thread_id) {
                    pthread_cancel(threads[j]); // cancel thread
                }
            }
            printf("Number Found in TID%d : %lu\n", thread_id + 1, pthread_self()); // print tid
            pthread_exit(NULL); // exit
        }
    }
    pthread_exit(NULL); // exit if not found
}
int main() {
    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }
    printf("Enter an integer between 1-100: ");
    scanf("%d", &num_to_find);
    if (num_to_find < 1 || num_to_find > 100) { //for error
        printf("Number Not found in the Given Range, Please enter again\n");
        return 0;
    }
    pthread_mutex_init(&lock, NULL);

    int thread_ids[NUM_THREADS]; // thread ids
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i; // set id
        if (pthread_create(&threads[i], NULL, search_in_range, &thread_ids[i]) != 0) { // create thread
            perror("Failed to create thread");
        }
    }
    for (int i = 0; i < NUM_THREADS; i++) { // join threads
        pthread_join(threads[i], NULL);
    }
    pthread_mutex_destroy(&lock);
    return 0;
}
```



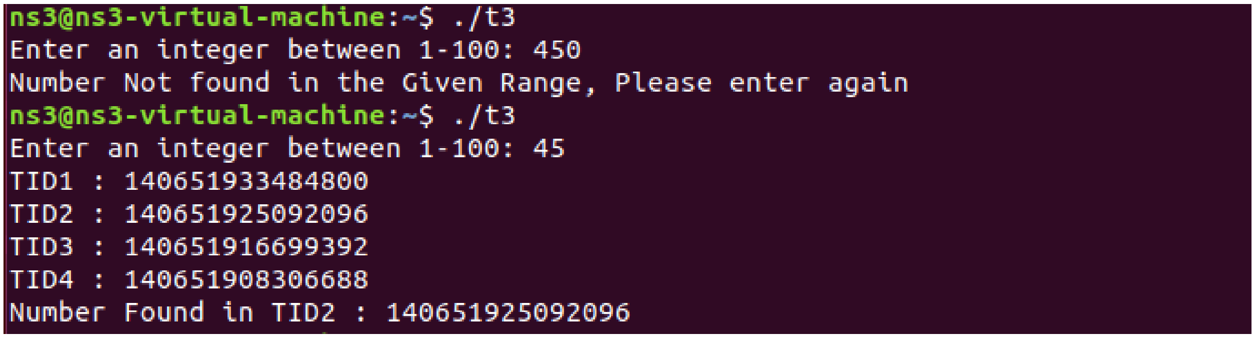
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#define SIZE 100
#define NUM_THREADS 4
int arr[SIZE];
int num_to_find;
int found = 0;
pthread_t threads[NUM_THREADS]; // thread ids
pthread_mutex_t lock;
void* search_in_range(void* arg) {
    int thread_id = *(int*)arg; // thread id
    int start = thread_id * (SIZE / NUM_THREADS); // range start
    int end = start + (SIZE / NUM_THREADS); // range end
    printf("TID%d : %lu\n", thread_id + 1, pthread_self()); // print tid
    for (int i = start; i < end; i++) {
        pthread_mutex_lock(&lock);
        if (found) {
            pthread_mutex_unlock(&lock);
            pthread_exit(NULL);
        }
        pthread_mutex_unlock(&lock);
        if (arr[i] == num_to_find) {
            pthread_mutex_lock(&lock);
            found = 1;
            pthread_mutex_unlock(&lock);
            for (int j = 0; j < NUM_THREADS; j++) { // cancel threads
                if (j != thread_id) {
                    pthread_cancel(threads[j]); }
            }
            printf("Number Found in TID%d : %lu\n", thread_id + 1, pthread_self()); // print tid
            pthread_exit(NULL); // exit
        }
    }
    pthread_exit(NULL); // exit if not found
}
int main() {
    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }
    printf("Enter an integer between 1-100: ");
    scanf("%d", &num_to_find);
    if (num_to_find < 1 || num_to_find > 100) { //for error
        printf("Number Not found in the Given Range, Please enter again\n");
        return 0;
```



```c
        return 0;
    }
    pthread_mutex_init(&lock, NULL);

    int thread_ids[NUM_THREADS]; // thread ids
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i; // set id
        if (pthread_create(&threads[i], NULL, search_in_range, &thread_ids[i]) != 0) { // create thread
            perror("Failed to create thread");
        }
    }
    for (int i = 0; i < NUM_THREADS; i++) { // join threads
        pthread_join(threads[i], NULL);
    }
    pthread_mutex_destroy(&lock);
    return 0;
}
```



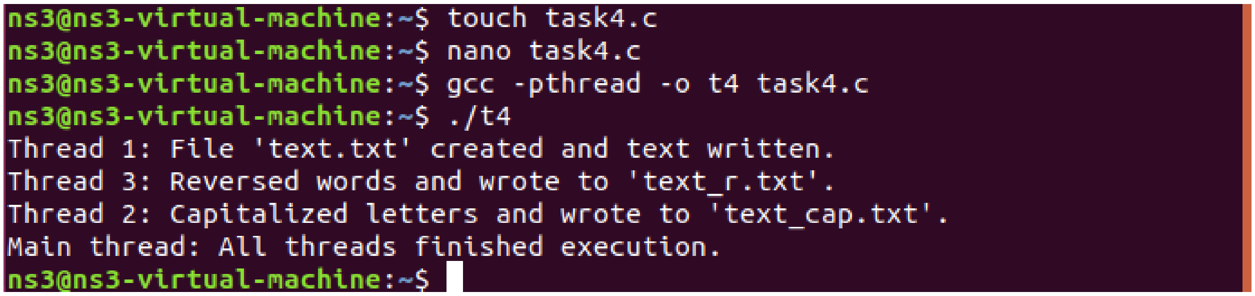
```
ns3@ns3-virtual-machine:~$ touch task3.c
ns3@ns3-virtual-machine:~$ nano task3.c
ns3@ns3-virtual-machine:~$ gcc -pthread -o t3 task3.c
ns3@ns3-virtual-machine:~$ ./t3
Enter an integer between 1-100: 30
TID1 : 140324615325440
TID2 : 140324606932736
TID3 : 140324596344576
TID4 : 140324587951872
Number Found in TID2 : 140324606932736
ns3@ns3-virtual-machine:~$
```



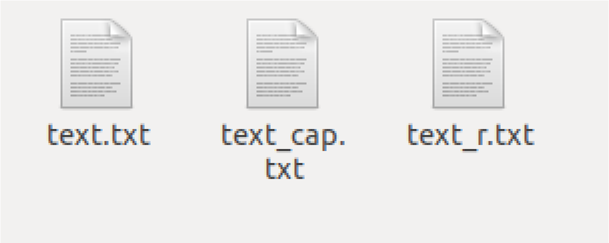
```
ns3@ns3-virtual-machine:~$ ./t3
Enter an integer between 1-100: 450
Number Not found in the Given Range, Please enter again
ns3@ns3-virtual-machine:~$ ./t3
Enter an integer between 1-100: 45
TID1 : 140651933484800
TID2 : 140651925092096
TID3 : 140651916699392
TID4 : 140651908306688
Number Found in TID2 : 140651925092096
```

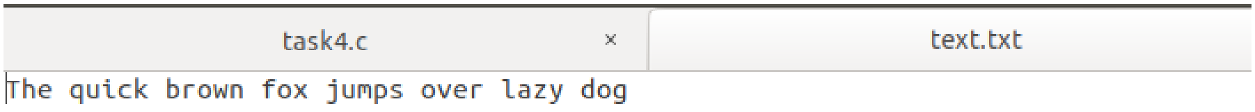
Task 4:

```
#include <stdio.h>

#include <pthread.h>

#include <ctype.h>

#include <string.h>

#include <stdlib.h>

#define MAX_LINE_LENGTH 256

// Thread 1

void* thread_create_file(void* arg) {

    FILE *file = fopen("text.txt", "w");

    if (file == NULL) {

        perror("Failed to create file");

        pthread_exit(NULL);
    }
    // Write text
    fprintf(file, "The quick brown fox jumps over lazy dog");
    fclose(file);
    printf("Thread 1: File 'text.txt' created and text written.\n");
    pthread_exit(NULL);
}
void capitalize_first_last(char *str) {
    int length = strlen(str);
    for (int i = 0; i < length; i++) {
        if (isalpha(str[i])) {
            int start = i;
            while (i < length && isalpha(str[i])) {
                i++;
            }
            int end = i - 1;
            // Capitalize
            str[start] = toupper(str[start]);
            str[end] = toupper(str[end]);
        }
    }
}
// Thread 2
void* thread_capitalize(void* arg) {
    FILE *input = fopen("text.txt", "r");
    FILE *output = fopen("text_cap.txt", "w");
    if (input == NULL || output == NULL) {
        perror("Failed to open file");
        pthread_exit(NULL);
    }
    char line[MAX_LINE_LENGTH];
    while (fgets(line, sizeof(line), input) != NULL) {
        capitalize_first_last(line);
        fprintf(output, "%s", line);
    }
    fclose(input);
    fclose(output);
    printf("Thread 2: Capitalized letters and wrote to 'text_cap.txt'.\n");
    pthread_exit(NULL);
}
// Reverse a word
void reverse_word(char* word, int start, int end) {
    while (start < end) {
        char temp = word[start];
        word[start] = word[end];
        word[end] = temp;
        start++;
        end--;
    }
}
// Thread 3
void* thread_reverse(void* arg) {
    FILE *input = fopen("text.txt", "r");
    FILE *output = fopen("text_r.txt", "w");
    if (input == NULL || output == NULL) {
        perror("Failed to open file");
        pthread_exit(NULL);
    }
    char line[MAX_LINE_LENGTH];
    while (fgets(line, sizeof(line), input) != NULL) {
        int length = strlen(line);
        for (int i = 0; i < length; i++) {
            if (isalpha(line[i])) {
                int start = i;
                while (i < length && isalpha(line[i])) {
                    i++;
                }
                reverse_word(line, start, i - 1);
            }
        }
        fprintf(output, "%s", line);
    }
    fclose(input);
    fclose(output);
    printf("Thread 3: Reversed words and wrote to 'text_r.txt'.\n");
    pthread_exit(NULL);
}
int main() {
    pthread_t tid1, tid2, tid3;
    //T1 create file and write text
    pthread_create(&tid1, NULL, thread_create_file, NULL);
    pthread_join(tid1, NULL); // Wait for thread 1 to complete

    //T2 capitalize first and last letter of each word
    pthread_create(&tid2, NULL, thread_capitalize, NULL);

    //T3 reverse each word
    pthread_create(&tid3, NULL, thread_reverse, NULL);

    pthread_join(tid2, NULL);

    pthread_join(tid3, NULL);

    printf("Main thread: All threads finished execution.\n");

    return 0;

}
```

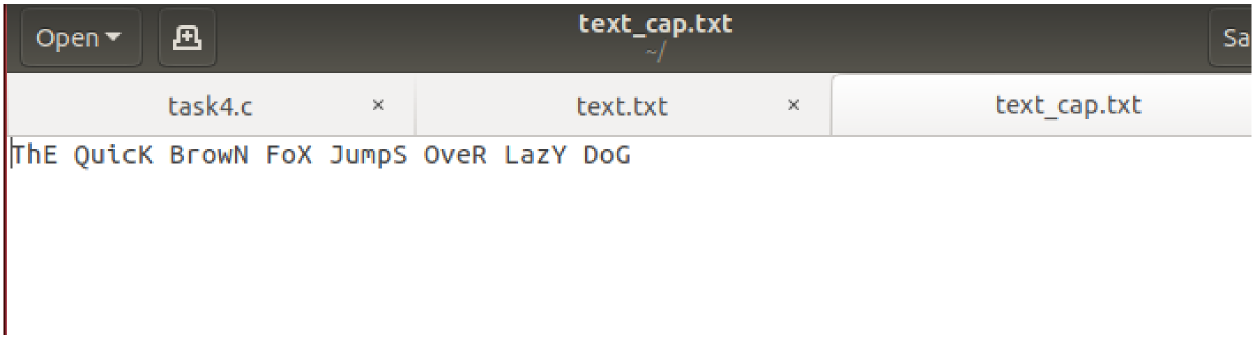


```
ns3@ns3-virtual-machine:~$ touch task4.c
ns3@ns3-virtual-machine:~$ nano task4.c
ns3@ns3-virtual-machine:~$ gcc -pthread -o t4 task4.c
ns3@ns3-virtual-machine:~$ ./t4
Thread 1: File 'text.txt' created and text written.
Thread 3: Reversed words and wrote to 'text_r.txt'.
Thread 2: Capitalized letters and wrote to 'text_cap.txt'.
Main thread: All threads finished execution.
ns3@ns3-virtual-machine:~$ 
```

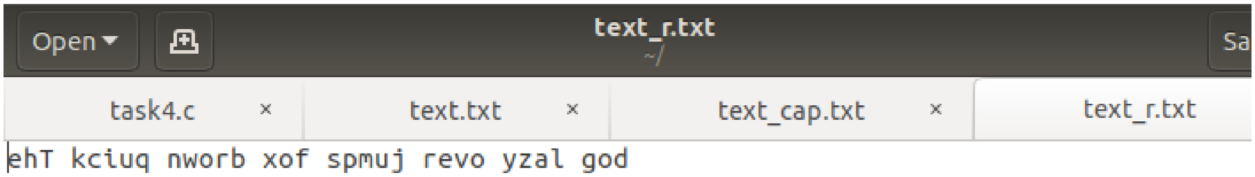




```
The quick brown fox jumps over lazy dog
```



```
ThE QuicK BrowN FoX JumpS OveR LazY DoG
```



```
ehT kciuq nworb xof spmuj revo yzal god
```