

РГПУ имени А.И. Герцена

Институт Компьютерных Наук и Технологического Образования

Информатика и вычислительная техника

Работу выполнил Цирулик И.А.

### **Лабораторная работа №3.**

#### **Практическое знакомство с потоками и синхронизацией потоков ОС UNIX**

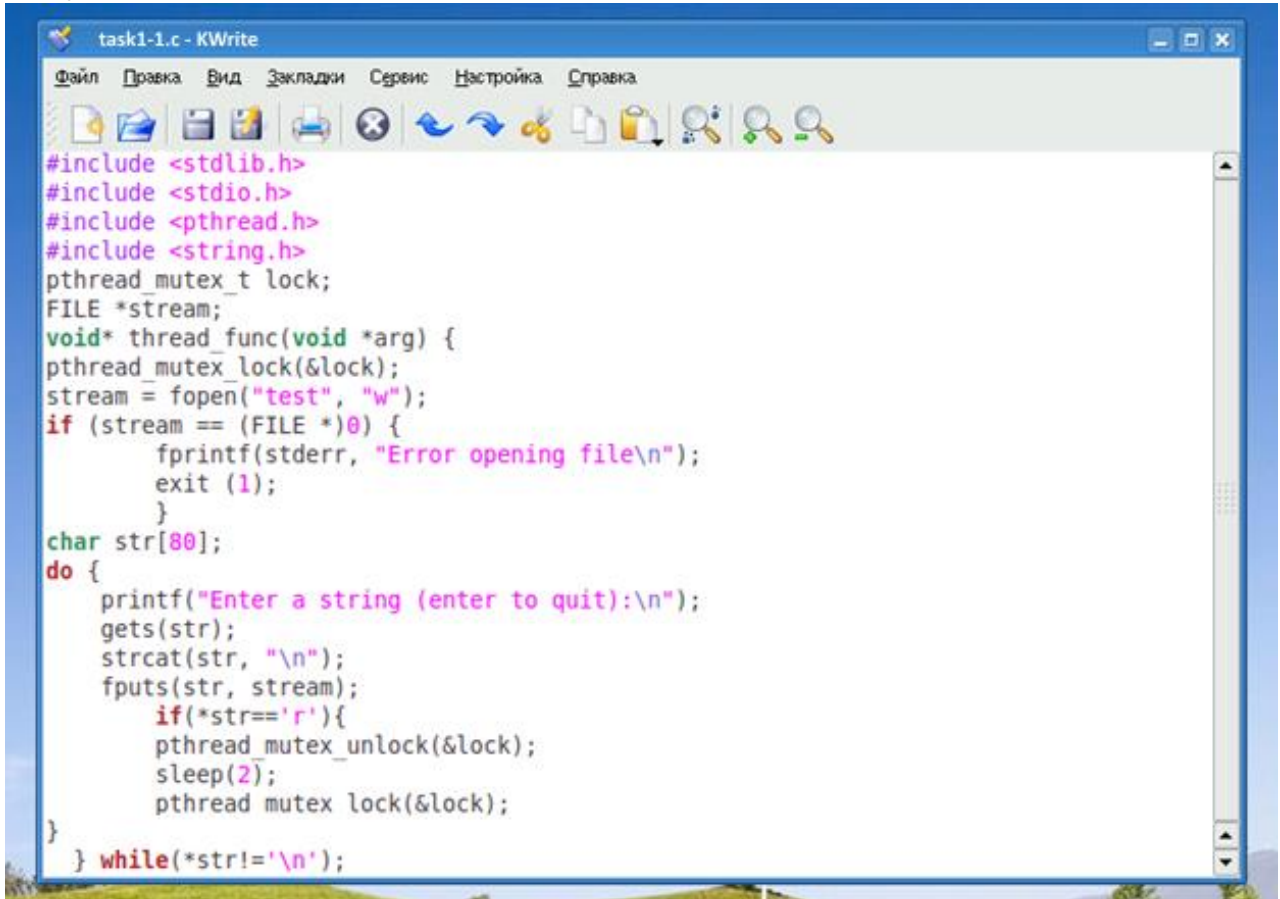
**Цель работы:** Ознакомиться с подсистемой управления потоками в операционной системе Unix и основными программными средствами для создания, управления и удаления потоков.

**Задание:** Изучить основные программные средства управления потоками ОС Unix, а также способы синхронизации потоков. Разработать приложения для многопоточных вычислений с использованием синхронизации посредством мьютексов, семафоров и условных переменных.

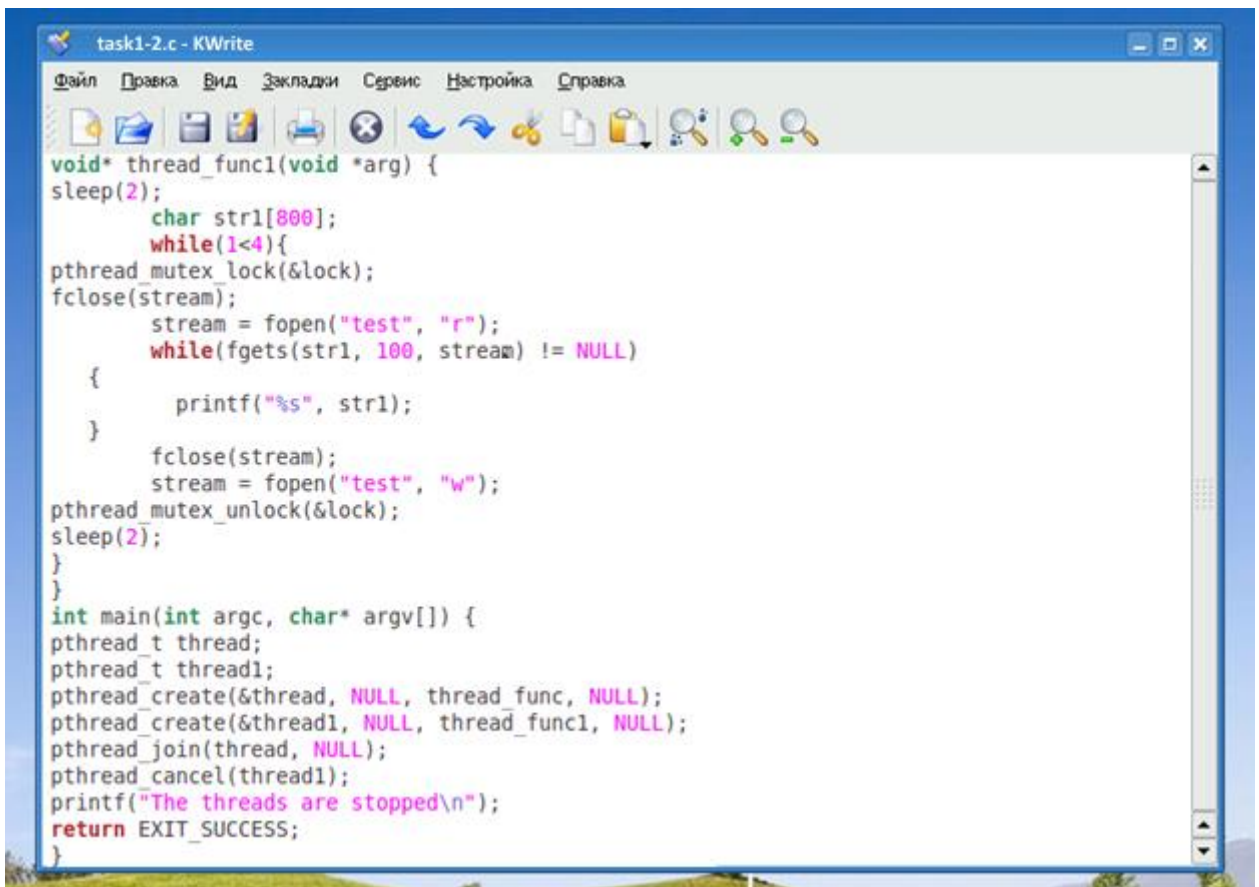
#### **Задание 1.**

В программе имеются два потока, один поток получает введенную в консоль информацию и записывает ее в текстовый документ, второй поток, при запросе от пользователя, выводит содержимое того же

документа на экран. Синхронизация с помощью мьютекса



```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>
pthread_mutex_t lock;
FILE *stream;
void* thread_func(void *arg) {
    pthread_mutex_lock(&lock);
    stream = fopen("test", "w");
    if (stream == (FILE *)0) {
        fprintf(stderr, "Error opening file\n");
        exit (1);
    }
    char str[80];
    do {
        printf("Enter a string (enter to quit):\n");
        gets(str);
        strcat(str, "\n");
        fputs(str, stream);
        if(*str=='r'){
            pthread_mutex_unlock(&lock);
            sleep(2);
            pthread_mutex_lock(&lock);
        }
    } while(*str!='\n');
}
```



```
void* thread_func1(void *arg) {
    sleep(2);
    char str1[800];
    while(1<4){
        pthread_mutex_lock(&lock);
        fclose(stream);
        stream = fopen("test", "r");
        while(fgets(str1, 100, stream) != NULL)
        {
            printf("%s", str1);
        }
        fclose(stream);
        stream = fopen("test", "w");
        pthread_mutex_unlock(&lock);
        sleep(2);
    }
}

int main(int argc, char* argv[]) {
    pthread_t thread;
    pthread_t thread1;
    pthread_create(&thread, NULL, thread_func, NULL);
    pthread_create(&thread1, NULL, thread_func1, NULL);
    pthread_join(thread, NULL);
    pthread_cancel(thread1);
    printf("The threads are stopped\n");
    return EXIT_SUCCESS;
}
```

Enter a string (enter to quit):

```
test1
Enter a string (enter to quit):
test2
Enter a string (enter to quit):
test3
Enter a string (enter to quit):
r
test1
test2
test3
r
Enter a string (enter to quit):
The threads are stopped
```

## **Задание 2.**

Во второй программе первый поток добавляет к общей переменной 10 а второй поток отнимает 5, после завершения одного цикла потока поток приостанавливается и начинает действие другой поток. Синхронизация с использованием семафора.

```
task2.c - KWrite
Файл  Правка  Вид  Закладки  Сервис  Настройка  Справка
[Icons]

}
void* thread_func1(void *arg) {
while(s<50){
sem_wait(&sem);
s+=10;
printf("%i",s);
printf("\n");
sem_post(&sem);
sleep(1);
}}
int main(int argc, char* argv[]) {
sem_init(&sem,0,0);
sem_post(&sem);
pthread_t thread;
pthread_t thread1;
pthread_create(&thread, NULL, thread_func, NULL);
pthread_create(&thread1, NULL, thread_func1, NULL);
pthread_join(thread, NULL);
printf("The threads are stopped\n");
return EXIT_SUCCESS;
}
```

```
task2.c - KWrite
Файл  Правка  Вид  Закладки  Сервис  Настройка  Справка
[Icons]

#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <semaphore.h>
sem_t sem;
int s;
void* thread_func(void *arg) {
while(s<50){
sem_wait(&sem);
s-=5;
printf("%i",s);
printf("\n");
sem_post(&sem);
sleep(1);
}
}
void* thread_func1(void *arg) {
while(s<50){
sem_wait(&sem);
s+=10;
```

-5

5

0

10

5

15

25

20

30

25

35

30

40

35

45

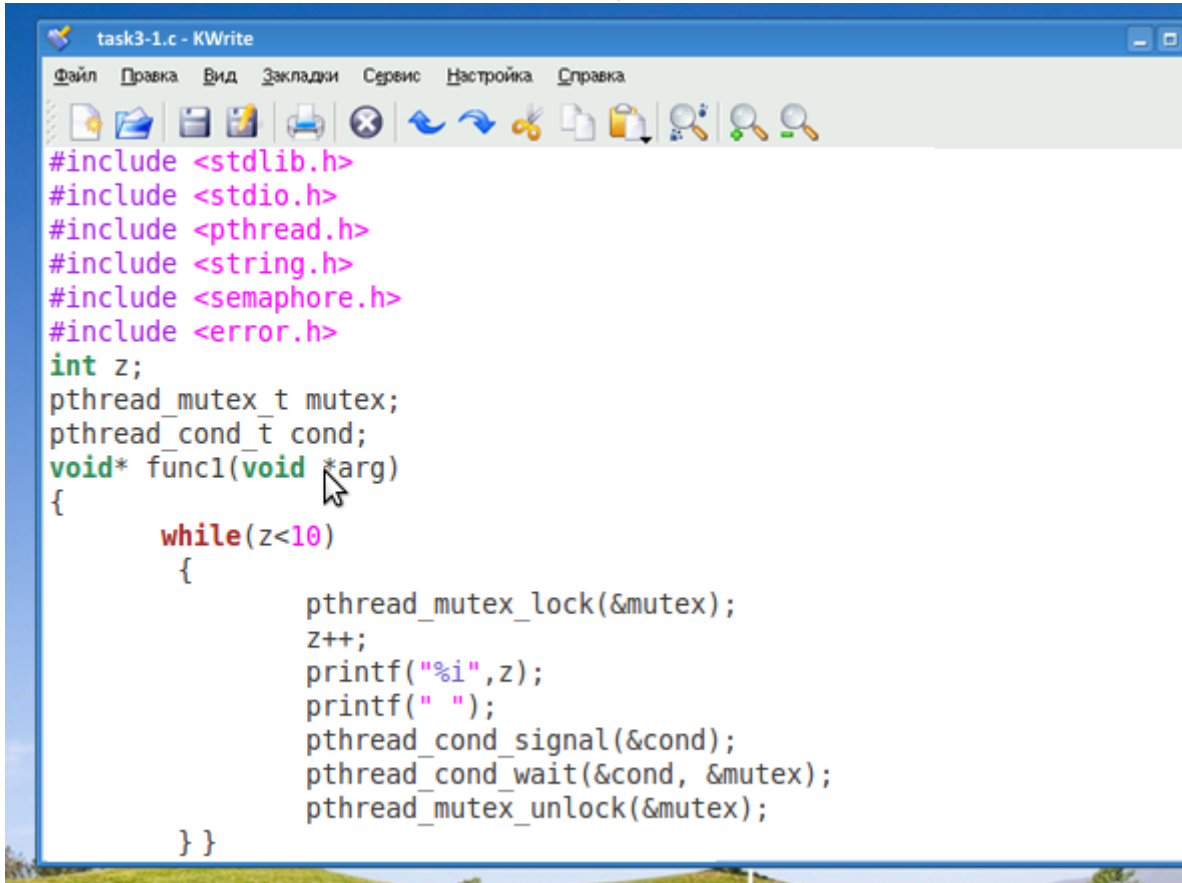
40

50

The threads are stopped

### Задание 3.

В данной программе два потока, с разной частотой, изменяют значение общей переменной, один поток прибавляет 1, другой отнимает 1, первый поток имеет в 2 раза большую частоту. Синхронизация с использованием условных переменных.

A screenshot of a KWrite text editor window titled "task3-1.c - KWrite". The window has a menu bar with "Файл", "Правка", "Вид", "Закладки", "Сервис", "Настройка", and "Справка". Below the menu is a toolbar with various icons for file operations and editing. The main text area contains C code for a multi-threaded program. The code includes headers for `stdlib.h`, `stdio.h`, `pthread.h`, `string.h`, `semaphore.h`, and `error.h`. It declares an integer `z`, a `pthread_mutex_t` `mutex`, and a `pthread_cond_t` `cond`. A function `func1` is defined, which takes a `void*` argument. Inside `func1`, there is a `while(z < 10)` loop. The loop body contains: `pthread_mutex_lock(&mutex);`, `z++;`, `printf("%i", z);`, `printf(" ");`, `pthread_cond_signal(&cond);`, `pthread_cond_wait(&cond, &mutex);`, and `pthread_mutex_unlock(&mutex);`. The function ends with `}}`.

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <semaphore.h>
#include <error.h>

int z;
pthread_mutex_t mutex;
pthread_cond_t cond;
void* func1(void *arg)
{
    while(z<10)
    {
        pthread_mutex_lock(&mutex);
        z++;
        printf("%i",z);
        printf(" ");
        pthread_cond_signal(&cond);
        pthread_cond_wait(&cond, &mutex);
        pthread_mutex_unlock(&mutex);
    }
}
```

```
task3-2.c - KWrite
Файл Правка Вид Закладки Сервис Настройка Справка
void* func2(void *arg)
{
    while (1)
    {
        pthread_mutex_lock(&mutex);
        pthread_cond_signal(&cond);
        pthread_cond_wait(&cond, &mutex);
        z--;
        printf("%i",z);
        printf(" ");
        pthread_cond_signal(&cond);
        pthread_cond_wait(&cond, &mutex);
        pthread_mutex_unlock(&mutex);
    }
}
```

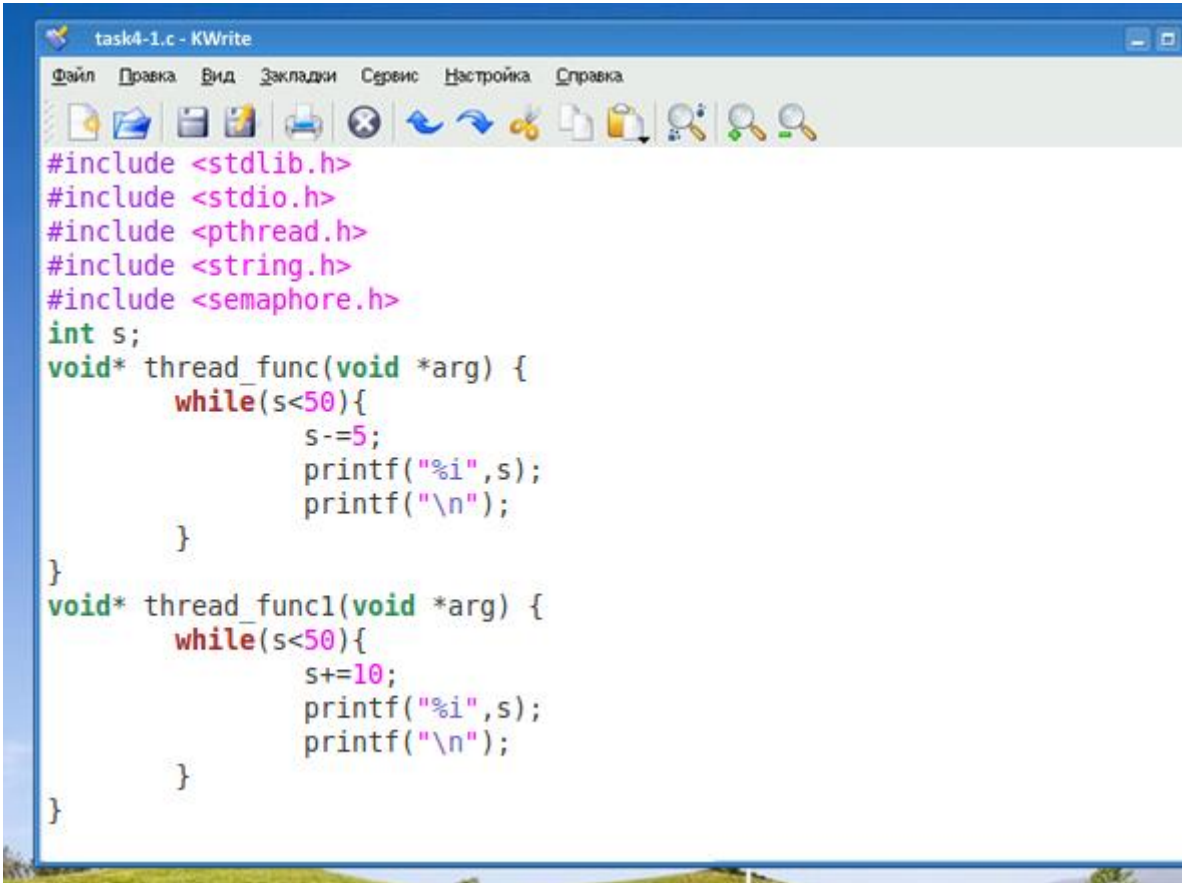
```
task3-3.c - KWrite
Файл Правка Вид Закладки Сервис Настройка Справка
int main()
{
    pthread_mutex_init(&mutex,0);
    pthread_cond_init(&cond,0);
    pthread_t t1;
    pthread_t t2;
    pthread_create(&t1,NULL,func1,NULL);
    pthread_create(&t2,NULL,func2,NULL);
    pthread_join(t1, NULL);
    pthread_cancel(t2);
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);
}
```

1 2 1 2 3 2 3 4 3 4 5 4 5 6 5 6 7 6 7 8 7 8 9 8 9 10 9 10

#### Задание 4.

Убедиться в результативности применения средств синхронизации потоков, сравнив результаты работы программ с использованием и без использования средств синхронизации

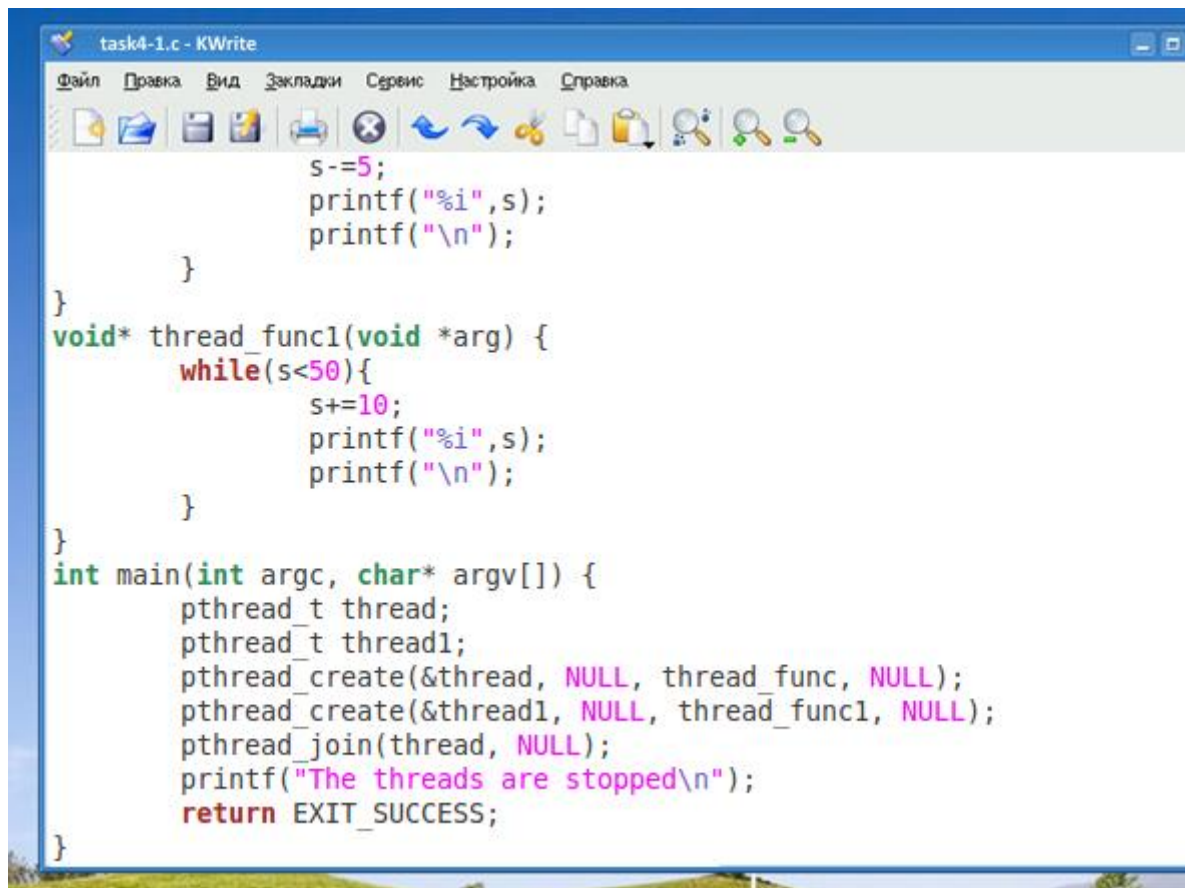
Во второй программе были убраны семафоры, следовательно, потоки получились асинхронные, потоки работают в произвольном порядке, в результате чего мы не можем гарантировать правильность работы приложения, в некоторых случаях это приводит к ошибке приложения или зацикливанию потока



```
task4-1.c - KWrite
Файл  Правка  Вид  Закладки  Сервис  Настройка  Справка

#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <semaphore.h>
int s;
void* thread_func(void *arg) {
    while(s<50){
        s-=5;
        printf("%i",s);
        printf("\n");
    }
}
void* thread_func1(void *arg) {
    while(s<50){
        s+=10;
        printf("%i",s);
        printf("\n");
    }
}
```





```
task4-1.c - KWrite
Файл  Правка  Вид  Закладки  Сервис  Настройка  Справка

s-=5;
printf("%i",s);
printf("\n");
}
}
void* thread_func1(void *arg) {
    while(s<50){
        s+=10;
        printf("%i",s);
        printf("\n");
    }
}
int main(int argc, char* argv[]) {
    pthread_t thread;
    pthread_t thread1;
    pthread_create(&thread, NULL, thread_func, NULL);
    pthread_create(&thread1, NULL, thread_func1, NULL);
    pthread_join(thread, NULL);
    printf("The threads are stopped\n");
    return EXIT_SUCCESS;
}
```

-5  
0  
-5  
-10  
-15  
-20  
-255  
-15  
-5  
5  
15  
25  
35  
45  
55

The threads are stopped

## **Выводы**

В ходе данной лабораторной работы были получены навыки работы с потоками которые помогают оптимизировать работу приложения, и в некоторых случаях повысить скорость работы приложения. Так же изучены основные методы для синхронизации потоков, которые помогают избежать ошибочной работы многопоточных программ.

2018