

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
Факультет информационных технологий и прикладной математики  
Кафедра вычислительной математики и программирования

**Лабораторная работа № 3 по курсу**  
**«Операционные системы»**

Студент: Тулин Иван Денисович  
Группа: М8О-201Б-21  
Вариант: 18  
Преподаватель Миронов Евгений Сергеевич  
Оценка: \_\_\_\_\_  
Дата: \_\_\_\_\_  
Подпись: \_\_\_\_\_

Москва, 2022

## **Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

## Репозиторий

<https://github.com/YusayuSharingan/Opsys-labs/tree/main/lab3>

## Постановка задачи

### Цель работы

Научиться создавать потоки, и взаимодействовать с ними.

### Задание

17) Найти в большом целочисленном массиве минимальный элемент

## Общие сведения о программе

Программа компилируется из файла lab3.cpp. Скомпилированная программа при запуске принимает аргумент — количество потоков. В программе используются следующие системные вызовы:

1. pthread\_mutex\_init() - создает mutex
2. pthread\_mutex\_lock() – блокирует все остальные потоки, до вызова pthread\_mutex\_unlock()
3. pthread\_mutex\_unlock() – говорит о конце блокировки после pthread\_mutex\_lock()
4. pthread\_mutex\_destroy() -- уничтожает mutex
5. pthread\_create() – создает поток.
6. pthread\_join() – закрывает поток.

## Общий метод и алгоритм решения

Входной массив параллельно считывается разными потоками, но переменная с которой сравнивается и в которую записывается минимальный элемент одна общая для всех потоков.

Таким образом массив обрабатывается разными потоками.

## Исходный код

### lab3.cpp

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

```
pthread_mutex_t mutex;
const int STOP_SIG = -1;
```

```

void* ThreadFunc(void* arg){
    int num, *minV=(int*)arg;
    while(scanf("%d", &num)!=STOP_SIG){
        pthread_mutex_lock(&mutex);
        if(num < *minV){
            *minV = num;
        }
        pthread_mutex_unlock(&mutex);

    }
    return NULL;
}

int main_routine(int threadNum){
    int minValue = INT_MAX;
    pthread_t *ths = (pthread_t*)malloc(sizeof(pthread_t)*threadNum);
    if (ths == NULL){
        printf("ERROR: couldn't allocate memmmory\n");
        return EXIT_FAILURE;
    }
    pthread_mutex_init(&mutex, NULL);
    for (int i=0; i<threadNum; i++){
        if(pthread_create(ths + i, NULL, ThreadFunc, &minValue)){
            printf("ERROR: couldn't create thread\n");
            return EXIT_FAILURE;
        }
    }
    for(int i=0; i<threadNum; i++){
        pthread_join(ths[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    return minValue;
}

```

### **Демонстрация работы программы**

```
yorokobeshounen@YS:~/Рабочий стол/OpSys/lab3/build$ cat input.txt
```

```
8991 993 34 345609 43185 4 5643132165 1
```

```
yorokobeshounen@YS:~/Рабочий стол/OpSys/lab3/build$ ./lab3 5 < input.txt
```

```
Min value is 1
```

```
Time of executing: 0
```

### **Выводы**

Чтобы проверить, реально ли ускоряют потоки программы я специально сделал два одинаковых input, отличающихся только количеством потоков. Из фотографий результата видно, что программа, работающая на 5 потоках значительно быстрее работает, чем программа работающая на 1 потоке... Также хочется отметить, что с потоками намного проще и интереснее работать, тк для их взаимодействий не нужно создавать pipe. Да и в целом потоки очень полезная, ускоряющая программу, штука.