Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Петрухин Дмитрий Олегович

Группа: М80 – 201Б-18

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2019

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Тестирование
6. Демонстрация работы программы
7. Вывод

**Постановка задачи**.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

**Задание согласно 13 варианту:**

1. Есть К массивов одинаковой длины. Необходимо сложить эти массивы. Необходимо предусмотреть стратегию, адаптирующуюся под количество массивов и их длину (по количеству операций).

**Общие сведения о программе**

Программа компилируется из трех файлов queue.h, queue.c, main.c с помощью утилиты make. В main.c используются заголовочные файлы pthread.h, stdio.h, stdlib.h. В программе используются следующие системные вызовы:

1. **pthread\_create()** – предназначен для создания новой нити исполнения внутри текущего процесса. При удачном завершении функция возвращает значение 0 и помещает идентификатор новой нити исполнения по адресу, на который указывает параметр thread. В случае ошибки возвращается положительное значение.
2. **pthread\_join()** – Функция *pthread\_join* блокирует работу вызвавшей ее нити исполнения до завершения thread'а с идентификатором thread. Аргумент функции, после разблокирования в указатель, расположенный по адресу status\_addr, заносится адрес, который вернул завершившийся thread либо при выходе из ассоциированной с ним функции, либо при выполнении функции *pthread\_exit()* . Если нас не интересует, что вернула нам нить исполнения, в качестве этого параметра можно использовать значение NULL.
3. **pthread\_mutex\_lock()/pthread\_mutex\_unlock()** – системный вызов захвата и освобождения мьютекса, или приостановления выполнения вызывающего потока, пока другой поток «владелец» мьютекса, не освободит его вызовом **pthread\_mutex\_unlock()**.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Реализовать очередь массивов.
2. Создать заданное количество потоков в цикле с помощью системного вызвова **pthread\_create()**.
3. В каждом потоке складывать определенное количество массивов, брав их из очереди, а конечный результат сложение ложить обратно в очередь.
4. Если длина массива превышает 100, то сложение массивов распоточится на определенное количество потоков с заданными параметрами.
5. После вызова потоков для сложения массивов, ожидаем их завершения.
6. Выполнять пункт 3-5, до тех пор пока в очереди не останется один элемент.

**Основные файлы программы**.

**Файл queue.h**

#ifndef OS\_LAB\_3\_QUEUE\_H

#define OS\_LAB\_3\_QUEUE\_H

#include <stdbool.h>

#include <stdlib.h>

typedef struct QueueItem queue\_item;

struct QueueItem {

struct QueueItem\* next;

struct QueueItem\* prev;

int \*value;

};

typedef struct Queue queue;

struct Queue {

queue\_item\* head;

queue\_item\* tail;

size\_t size;

};

void q\_init(queue\* q);

int \*q\_top(queue\* q);

int \*q\_pop(queue\* q);

size\_t q\_size(queue\* q);

void q\_push(queue\* q, int \*arrray);

bool q\_empty(queue\* q);

void q\_destroy(queue\* q);

#endif //OS\_LAB\_3\_QUEUE\_H

**Файл queue.c**

#include "queue.h"

void q\_init(queue\* q) {

q->head = NULL;

q->tail = NULL;

q->size = 0;

}

int \*q\_top(queue\* q) {

return q->head->value;

}

int \*q\_pop(queue\* q) {

int \*temp = q\_top(q);

queue\_item\* ptr\_to\_free = q->head;

q->head = q->head->next;

if (q->head == NULL) {

q->tail = NULL;

}

free(ptr\_to\_free);

q->size--;

return temp;

}

void q\_push(queue\* q, int \*array) {

queue\_item\* new\_elem = malloc(sizeof(queue\_item));

new\_elem->value = array;

new\_elem->next = 0;

if (q->head == NULL) {

q->head = new\_elem;

q->tail = new\_elem;

new\_elem->prev = 0;

} else {

q->tail->next = new\_elem;

new\_elem->prev = q->tail;

q->tail = new\_elem;

}

q->size++;

}

bool q\_empty(queue\* q) {

return q->head == NULL;

}

size\_t q\_size(queue\* q) {

return q->size;

}

void q\_destroy(queue\* q) {

queue\_item\* start = q->head;

while (start != NULL) {

queue\_item\* next = start->next;

free(start);

start = next;

}

q->head = NULL;

q->tail = NULL;

q->size = 0;

}

**Файл main.c**

#include "queue.h"

#include <pthread.h>

#include <stdio.h>

#include <math.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

typedef struct ThreadParams thread\_params;

struct ThreadParams {

int number\_arrays;

int length;

int number\_length\_threads;

queue \*q;

int move;

};

typedef struct ThreadParams2 thread\_params2;

struct ThreadParams2 {

int \*a;

int \*ressumm;

int start;

int move;

int length;

};

void \* asplitt(void \*arg){

thread\_params2 targ2 = \*(thread\_params2 \*) arg;

int start = targ2.start;

int move = targ2.move;

int length = targ2.length;

int finish = start + move;

if(length < finish) finish = length;

for(int i=start; i<finish; i++) targ2.ressumm[i] += targ2.a[i];

}

void \* sumfunc(void \* arg){

thread\_params targ = \*(thread\_params \*) arg;

int length = targ.length;

int n = targ.number\_arrays;

int lt = targ.number\_length\_threads;

int move = targ.move;

int \*a, \*ressumm;

ressumm = (int\*)malloc(length \* sizeof(int));

for(int i=0;i<length;i++) ressumm[i] = 0;

for(int i=0; i<n; i++){

if (!q\_empty(targ.q)){

pthread\_mutex\_lock(&mutex);

a = q\_pop(targ.q);

pthread\_mutex\_unlock(&mutex);

if(lt){

pthread\_t\* threads2 = malloc(sizeof(pthread\_t) \* lt); //lt количество тредов

thread\_params2\* params2 = malloc(sizeof(thread\_params) \* lt);

for (int i = 0; i < lt; ++i) {

params2[i].a = a;

params2[i].ressumm = ressumm;

params2[i].move = move;

params2[i].start = i\*move;

params2[i].length = length;

if( pthread\_create(&threads2[i], NULL,

&asplitt, &params2[i]) != 0) {

printf("Error create 2\n");

exit(-1);

}

}

for (int i = 0; i < lt; ++i) {

pthread\_join(threads2[i], NULL);

}

} else {

for(int j=0; j<length; j++) ressumm[j] += a[j];

}

} else break;

}

pthread\_mutex\_lock(&mutex);

q\_push(targ.q, ressumm);

pthread\_mutex\_unlock(&mutex);

return NULL;

}

int main(){

int i, m, number\_threads, number\_length\_threads, move;

int \*a;

int n = 0;

int length = 0;

queue q;

q\_init(&q);

while(n==0 && length==0){

printf("Enter the number and length of arrays\n");

scanf("%d", &n);

scanf("%d", &length);

}

printf("Enter number threads for addition arrays from 1 to %d\n", n/2);

scanf("%d", &number\_threads);

m = n / number\_threads;

if(length > 100){

printf("Enter number threads for program from 1 to %d\n", length/25);

scanf("%d", &number\_length\_threads);

move = ceil(length / number\_length\_threads);

} else number\_length\_threads = 0;

pthread\_t\* threads = malloc(sizeof(pthread\_t) \* number\_threads);

thread\_params params;

params.q = &q;

params.length = length; // длина массива

params.number\_arrays = m;

params.number\_length\_threads = number\_length\_threads; // количество потоков для "разбиения" массива

params.move = move; // по сколько элементов массива складываются в одном потоке

for(i=0; i<n; i++){

a = (int\*)malloc(length\*sizeof(int));

for (int j = 0; j<length; j++) a[j] = 1;

q\_push(&q,a);

}

while(q\_size(&q) != 1){

//printf("I entered while\n");

for(i=0; i<number\_threads; ++i){

if(pthread\_create(&threads[i], NULL,

&sumfunc, &params) != 0){

printf("Error create\n");

return 1;

}

}

for(i=0; i<number\_threads; ++i){

if(pthread\_join(threads[i], NULL) != 0) {

printf("Error join\n");

return 1;

}

}

}

a = q\_pop(&q);

printf("Result array\n");

for(i=0; i<length; i++) printf("%d ", a[i]);

printf("\n");

pthread\_mutex\_destroy(&mutex);

q\_destroy(&q);

return 0;

}

**Makefile**

CC = gcc

LD = gcc

CCFLAGS = -g

LDFLAGS = -g

l3: main.o queue.o

$(LD) $(LDFLAGS) -pthread -o l3 main.o queue.o

maim.o: main.c queue.h

$(CC) $(CCFLAGS) -c main.c -o main.o

queue.o: queue.c queue.h

$(CC) $(CCFLAGS) -c queue.c -o queue.o

clear:

rm -f \*.o l3

**Тестирование.**

Числа вводятя по одному через enter. Вводится n массивов и их длины k, затем вводится желаемое количество потоков для сложения n массивов. Если длина массива k>100, то вводится количество потоков для разбиения сложения массива, адаптирующееся под длину

**test1:**

10 13

4

**test2:**

15 150

4

3

**test2:**

1000

12

500

**Демонстрация работы программы.**

==> make

gcc -c -o main.o main.c

gcc -g -c queue.c -o queue.o

gcc -g -pthread -o l3 main.o queue.o

==> ./l3

Enter the number and length of arrays

10 13

Enter number threads for addition arrays from 1 to 5

4

Result array

10 10 10 10 10 10 10 10 10 10 10 10 10

==> ./l3

Enter the number and length of arrays

15

150

Enter number threads for addition arrays from 1 to 7

4

Enter number threads for program from 1 to 6

3

Result array

15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15

Enter the number and length of arrays

1000

12

Enter number threads for addition arrays from 1 to 500

500

Result array

1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000

**Вывод.**

Управление потоками в OC очень непросто. Правильное создание потоков может сильно ускорить программу, но неправильное использование мьютекса и ожидание завершения thread’a могут только замедлить программу, нежели чем реализовывать последовательный алгоритм выполнения программы. Помимо этого стоит отметить, что многопоточность стоит реализовывать при трудоемких операциях или когда выполнение процесса имеет гораздо большие затраты, чем затраты на создание потока. Также можно столкнуться с проблемой взаимных блокировок, при реализации сложного взаимодействия потоков друг с другом. Но в случае почти идеальной реализации работы с потоками мы можем получить увеличение скорости работы программы.