Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Фролов М.А.

Группа: М8О–206Б–20

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Оценка: 5

Дата: 30.10.21

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

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

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 13:

Есть набор 128 битных чисел, записанных в шестнадцатеричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом"

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, ctype.h, stdbool.h, pthread.h, stdint.h, inttypes.h. В программе используются следующие системные вызовы:

1. **pthread\_create ––** (является оберткой над системным вызовом clone) создает новый поток в вызывающем процессе. В качестве аргументов принимает указатель на структуру-идентификатор потока pthread\_t, атрибуты потока, функцию, которая будет запускаться в потоке, список аргументов для функции в виде указателя на void. В случае успеха возвращает 0, иначе возвращает номер ошибки.
2. **pthread\_join ––** используется для ожидания завершения потока. Данная функция блокирует вызывающий поток, пока указанный поток не завершится. В качестве аргументов принимает структуру pthread\_t потока и указатель на переменную, в которую будет записан результат, возвращаемый потоком. В случае успеха возвращает 0, иначе возвращает номер ошибки.
3. **pthread\_exit ––** завершает вызываемый поток. В качестве аргумента принимает значение, которое вернется при завершении потока. Функция всегда завершается успехом.
4. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
5. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с потоками в C.
2. Написать вспомогательные функции для работы с матрицами.
3. С помощью принципа разбиения задачи на маленькие подзадачи, реализовать функцию, которая будет проводить частичную обработку данных, для ее запуска в многопоточном режиме.
4. Реализовать функцию, которая будет создавать потоки, запускать потоковую функцию, управлять потоками.
5. Реализовать обработку системных ошибок согласно заданию

**Основные файлы программы**

**main.c:**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <pthread.h>

#include <fcntl.h>

#include "unistd.h"

#include <stdint.h>

#include <inttypes.h>

#include "string.h"

#include <math.h>

void read\_filename(string\* str, int fd){

char cur = 'a';

while(read(fd, &cur, sizeof(char)) > 0){

if(cur == '\n'){

break;

}

str\_push(str, cur);

}

}

typedef enum{

reading\_suc,

reading\_wrong\_value,

reading\_eof,

spare\_eol,

eof,

} read\_rvl\_stat;

read\_rvl\_stat reading\_128\_int(int fd, \_\_uint128\_t\* cur){

char c;

\*cur = 0;

int k = read(fd, &c, sizeof(char));

bool fnd\_smth = false;

int alr\_read = 0;

while(k > 0){

if(c == '\n'){

if(fnd\_smth) return reading\_eof;

else return spare\_eol;

}

if(c == ' '){

if(fnd\_smth) break;

else return spare\_eol;

}

if((c < 'a' || c > 'f') && (c < '0' || c > '9')){

return reading\_wrong\_value;

}

fnd\_smth = true;

alr\_read++;

if(alr\_read > 32){

return reading\_wrong\_value;

}

if(c >= 'a' && c <= 'f')

\*cur = \*cur \* 16 + (c - 'a' + 10);

if(c >= '0' && c <= '9')

\*cur = \*cur \* 16 + c - '0';

k = read(fd, &c, sizeof(char));

}

if(k == 0){

if (fnd\_smth) return reading\_eof;

else return eof;

}

return reading\_suc;

}

typedef struct{

\_\_uint128\_t sum;

bool res;

int cnt;

int fd;

} result\_value;

pthread\_mutex\_t mutex;

bool end = false;

void\* func(void\* thread\_data){

result\_value\* res = (result\_value\*) thread\_data;

\_\_uint128\_t sum = 0;

\_\_uint128\_t cur;

while(true){

pthread\_mutex\_lock(&mutex);

if(end){

res->res = true;

break;

}

read\_rvl\_stat last\_read = reading\_128\_int(res->fd, &cur);

if(last\_read == reading\_wrong\_value){

res->res = false;

end = true;

break;

} else if (last\_read == eof){

res->res = true;

end = true;

break;

} else if (last\_read == reading\_eof){

res->res = true;

sum += cur;

res->cnt++;

end = true;

break;

} else if (last\_read == spare\_eol){

res->res = true;

} else {

res->res = true;

sum += cur;

res->cnt++;

}

pthread\_mutex\_unlock(&mutex);

}

pthread\_mutex\_unlock(&mutex);

res->sum = sum;

pthread\_exit(0);

}

int main(int argc, char\* argv[]){

if(argc > 3){

printf("Wrong argument number\n");

printf("a.out <Memory size> <b for bytes/kb for kilobytes/mb for megabytes>\n");

return 0;

}

long int N;

if(argc == 3){

long int sum\_mem = atol(argv[1]);

if(sum\_mem <= 0){

printf("Wrong thread number\n");

return 0;

}

int status = -1;

if(!strcmp(argv[2], "b")) status = 0;

if(!strcmp(argv[2], "kb")) status = 1;

if(!strcmp(argv[2], "mb")) status = 2;

if(status == -1){

printf("Wrong argument number\n");

printf("a.out <Memory size> <b for bytes/kb for kilobytes/mb for megabytes>\n");

return 0;

}

for(int i = 0; i < status; i++) sum\_mem \*= 1024;

N = (sum\_mem - 2652) / 8196;

if(N <= 0){

printf("%ld - Wrong thread number\n", N);

return 0;

}

}

if(argc == 1){

N = 1;

}

if(argc == 2){

long int sum\_mem = atol(argv[1]);

if(sum\_mem <= 0){

printf("Wrong thread number\n");

return 0;

}

N = (sum\_mem \* 1024 - 2652) / 8196;

if(N <= 0){

printf("%ld - Wrong thread number\n", N);

return 0;

}

}

string file\_name;

str\_create(&file\_name);

read\_filename(&file\_name, 0); // 0 = STDIN

str\_push(&file\_name, '\0');

int file = open(str\_get\_all(&file\_name), O\_RDONLY);

if(file == -1){

perror("File can't be opened");

str\_destroy(&file\_name);

return -5;

}

str\_destroy(&file\_name);

pthread\_mutex\_init(&mutex, NULL);

pthread\_t threads[N];

result\_value threads\_rvl[N];

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

threads\_rvl[i].sum = 0;

threads\_rvl[i].res = false;

threads\_rvl[i].cnt = 0;

threads\_rvl[i].fd = file;

}

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

pthread\_create(&threads[i], NULL, func, &threads\_rvl[i]);

}

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

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

}

\_\_uint128\_t res\_sum = 0;

int res\_cnt = 0;

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

if(!threads\_rvl[i].res){

printf("Wrong value in file!\n");

pthread\_mutex\_destroy(&mutex);

return 0;

}

res\_sum += threads\_rvl[i].sum;

res\_cnt += threads\_rvl[i].cnt;

}

res\_sum = res\_sum / res\_cnt;

\_\_uint128\_t high = (res\_sum >> 64);

if(high){

printf("%" PRIx64, (uint64\_t)high);

}

printf("%llx\n", (unsigned long long)(res\_sum & 0xFFFFFFFFFFFFFFFF));

pthread\_mutex\_destroy(&mutex);

return 0;

}

**Пример работы**

michael@michael-VirtualBox:~/Desktop/os/os\_laba\_3$ ls

a.out file.txt prog.c string.c string.h write\_log.txt

michael@michael-VirtualBox:~/Desktop/os/os\_laba\_3$ cat > file.txt

43 345 7658 3

michael@michael-VirtualBox:~/Desktop/os/os\_laba\_3$ ./a.out

file.txt

1e78

**Вывод**

В ходе выполнения данной лабораторной работы я познакомился с тем, как производится работа с потоками в языке Си. Основная идея применения потоков состоит в разбиение большой задачи на множество более мелких задач, которые выполняются параллельно. Но данный подход требует особенной внимательности программиста к совместно используемым данным и тщательного продумывания алгоритма.