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: Дегтярев Денис Андреевич

Группа: М8О-207Б-21

Вариант: 15

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/CHISH08/OCI

**Постановка задачи**

**Цель работы**

Научиться создавать потоки, и взаимодействовать с ними.

**Задание**

Перемножение полиномов. На вход подается N-полиномов, необходимо их перемножить.

**Общие сведения о программе**

Программа компилируется из файла 3lab.cpp. В нее подается файл input3.txt, в котором содержится количество потоков, количество полиномов, максимальная степень входящих полиномов, а затем сами полиномы.В программе используются следующие системные вызовы:

1. pthread\_mutex\_lock() – блокирует все остальные потоки, до вызова pthread\_mutex\_unlock()
2. pthread\_mutex\_unlock() – говорит о конце блокировки после pthread\_mutex\_lock()
3. dup2() – копирует old\_file\_descriptor в new\_file\_descriptor.
4. std::chrono::steady\_clock::now() – делает замеры по времени(для вычисления скорости работы программы.
5. pthread\_create() – создает поток.
6. pthread\_join() – закрыввает поток.

**Общий метод и алгоритм решения**

Распределяем полиномы на N / 2 или K(если K < N / 2) пар. Каждая пара полиномов перемножается внутри потоков. Это повторяется, пока N != 1, а затем печатается ответ.

**Исходный код**

**#include <bits/stdc++.h>**

**#include <sys/wait.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <signal.h>**

**#include <pthread.h>**

**#include <chrono>**

**#include <ctime>**

**using namespace std;**

**pthread\_mutex\_t mtx;**

**typedef struct {**

**int N;**

**vector<vector<long long>> pol\_mtx;**

**} dat;**

**dat dt;**

**void\* mult(void \*arg)**

**{**

**pthread\_mutex\_lock(&mtx);**

**vector<long long> m1, m2;**

**int N = dt.pol\_mtx.size();**

**m1 = dt.pol\_mtx[N - 1];**

**m2 = dt.pol\_mtx[N - 2];**

**dt.pol\_mtx.pop\_back();**

**dt.pol\_mtx.pop\_back();**

**pthread\_mutex\_unlock(&mtx);**

**int K1 = m1.size(), K2 = m2.size();**

**vector<long long> m3(K1 + K2 - 1, 0);**

**for (int i = 0; i < K1; ++i) {**

**for (int j = 0; j < K2; ++j) {**

**m3[i + j] += m1[i] \* m2[j];**

**}**

**}**

**pthread\_mutex\_lock(&mtx) == 0;**

**dt.pol\_mtx.push\_back(m3);**

**--dt.N;**

**pthread\_mutex\_unlock(&mtx);**

**return NULL;**

**}**

**int main()**

**{**

**pthread\_mutex\_init(&mtx, NULL);**

**int file1 = open("./input3.txt", O\_RDONLY);**

**dup2(file1, STDIN\_FILENO);**

**int K, N, th\_c;**

**//cout << "Введите количество потоков: ";**

**cin >> th\_c;**

**pthread\_t threads[th\_c];**

**//cout << "\nВведите количество полиномов: ";**

**cin >> N;**

**//cout << "\nВведите максимальную степень вводимых полиномов: ";**

**cin >> K;**

**dt.N = N;**

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

**vector<long long> pol(K + 1, 0);**

**for (int j = K; j >= 0; --j) {**

**// cout << "\nВведите полином " << j << " степени: ";**

**cin >> pol[j];**

**}**

**(dt.pol\_mtx).push\_back(pol);**

**}**

**auto start\_time = std::chrono::steady\_clock::now();**

**delete new int(1);**

**while(dt.N > 1) {**

**int j = 0;**

**for (int i = 0; i < th\_c && i < dt.N / 2 && dt.N > 1 && dt.pol\_mtx.size() >= 2; ++i) {**

**++j;**

**if (pthread\_create(&threads[i], NULL, &mult, NULL) != 0) {**

**perror("Failed to creating thread!");**

**}**

**}**

**for(int i = 0; i < j; ++i) {**

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

**perror("Failed to joining thread!");**

**}**

**}**

**}**

**auto end\_time = std::chrono::steady\_clock::now();**

**auto elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_time - start\_time);**

**std::cout << elapsed\_ns.count() << " ns\n";**

**pthread\_mutex\_destroy(&mtx);**

**for (int j = 0; j < dt.pol\_mtx.size(); ++j) {**

**for (int i = dt.pol\_mtx[j].size() - 1; i >= 0; --i) {**

**cout << dt.pol\_mtx[j][i] << "x^" << i;**

**if (i != 0) {**

**cout << " + ";**

**} else {**

**cout << '\n';**

**}**

**}**

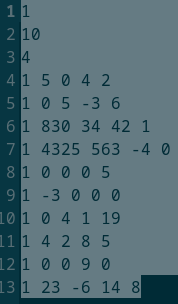
**}**

**return 0;**

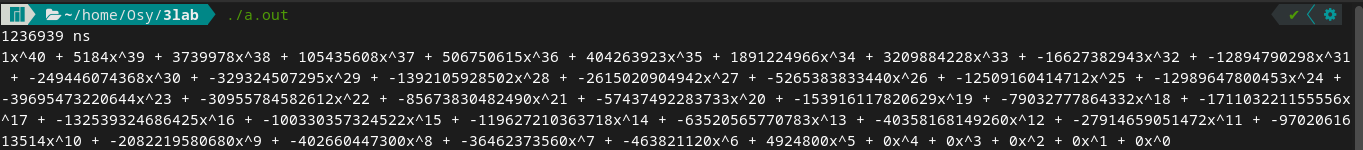
**}**

**Демонстрация работы программы**

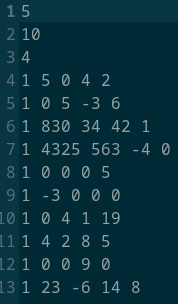
input1:



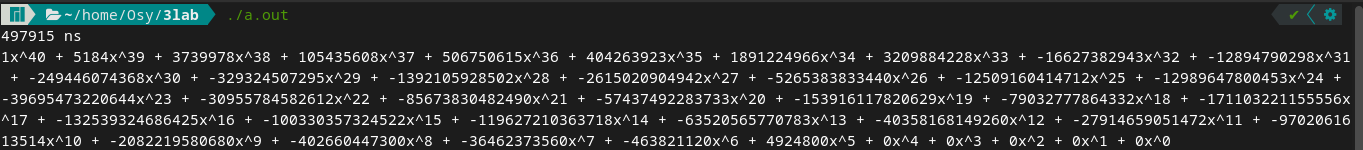
output1:



Input2:



output2:



Результат работы strace:

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11581]}, 88) = 11581

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11582]}, 88) = 11582

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11583]}, 88) = 11583

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11584]}, 88) = 11584

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11585]}, 88) = 11585

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11586]}, 88) = 11586

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[11587]}, 88) = 11587

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[0]}, 88) = 11588

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f7f09da0990, parent\_tid=0x7f7f09da0990, exit\_signal=0, stack=0x7f7f095a0000, stack\_size=0x7fff80, tls=0x7f7f09da06c0} => {parent\_tid=[0]}, 88) = 11589

+++ exited with 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 интреснее работать, тк для их взаимодействий не нужно создавать pipe. Да и в целом потоки очень полезная, ускоряющая программу, штука.