Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

ПОТОКИ

Студент: Харьков Павел Александрович
Группа: М8О–206Б–19
Вариант: 9
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021.

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

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.

Вариант 9: Рассчитать детерминант матрицы.

Общие сведения о программе.

Программа компилируется из файлов laba_.c, get_int.c . Также используется заголовочные файлы: unistd.h, stdlib.h, stdio.h, pthread.h, time.h. В программе используются следующие системные вызовы:

- read – читает в память из файлового дескриптора определенное количество байт.
- write – пишет в файловый дескриптор из памяти определенное количество байт.
- pthread_create – создает новый поток программы.
- pthread_join – ожидание выполнения потока.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы read, write, pthread_create, pthread_join.
2. Написать считывание матрицы в память.
3. Написать функцию расчёта детерминанта матрицы с использованием потоков.
4. Отладить программу и протестировать на тестах.

Основные файлы программы.

laba_2.c

```
// Посчитать детерминант матрицы
```

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
#include "get_int/get_int.h"

#define ll long long

struct counts
{
   int n;
   int* matr;
   int firstI;
   int lastI;
   int j;
   ll* res;
};

ll Minor(int n, int* matr, int i, int j)
{
   if(n == 1)
      return 1;

   ll res = 0;

   int* new_matr = (int*)malloc(sizeof(int)* (n-1) *(n-1));
   if(new_matr == NULL)
      exit(-1);
   if(!new_matr)
   {
      perror("Make matrix");
      exit(-1);
   }
   int k = 0;
   for(int a = 0; a < n; ++a)
   {
      if(a == i)
         continue;
      for(int b = 1; b < n; ++b)
      {
         new_matr[k] = matr[a*n + b];
         ++k;
      }
   }
   for(int a = 0; a < n-1; ++a)
   {
      if(a % 2 == 0)
         res += new_matr[a*(n-1)] * Minor(n-1, new_matr, a, j);
      if(a % 2 == 1)
```

```
            res -= new_matr[a*(n-1)] * Minor(n-1, new_matr, a, j);
    }
    free(new_matr);
    return res;

}

void* CountThradeMinors(void* args)
{
    struct counts* arg = (struct counts*) args;

    for(int i = arg->firstI; i < arg->lastI; ++i)
    {
        if(i % 2 == 0)
            *(arg->res) += arg->matr[i * arg->n] * Minor(arg->n, arg->matr, i, arg->j);
        if(i % 2 == 1)
            *(arg->res) -= arg->matr[i * arg->n] * Minor(arg->n, arg->matr, i, arg->j);
    }

    pthread_exit(0);
}

ll CountMinors(int n, int* matr, int firstI, int lastI, int j)
{
    ll res = 0;
    for(int i = firstI; i < lastI; ++i)
    {
        if(i % 2 == 0)
            res += matr[i * n] * Minor(n, matr, i, j);
        if(i % 2 == 1)
            res -= matr[i * n] * Minor(n, matr, i, j);
    }

    return res;
}

int main(int argc, char* argv[])
{
    int thread = 1;
    if(argc == 2)
        thread = (int)(*argv[1] - '0');

    int n;
    if(get_int(STDIN_FILENO, &n) != 0)
        exit(-1);

    int* matr = (int*)malloc(sizeof(int) * (n*n));
    if(!matr){
        exit(-1);
    }
```

```
for(int i = 0; i < n*n; ++i)
{
   if(get_int(STDIN_FILENO, &matr[i]) != 0)
      exit(-1);
}

pthread_t tids[thread];
for(int i = 0; i < thread; ++i){
   tids[i] = 0;
}

ll results[thread];
struct counts data[thread];
int k = 0;

struct timespec begin, end;
clock_gettime(CLOCK_REALTIME, &begin);
for(int t = 0; t < thread; ++t)
{
   data[t].n = n;
   data[t].matr = matr;
   data[t].firstI = k;

   k = k + n/(thread+1);

   data[t].lastI = k;
   data[t].j = 0;
   results[t] = 0;
   data[t].res = &results[t];

   if(data[t].lastI - data[t].firstI > 0)
   {
      //printf("thread: From %d to %d \n", data[t].firstI, data[t].lastI);
      if (pthread_create(&tids[t], NULL, CountThradeMinors, (void*) &data[t]) != 0)
      {
         perror("Creating the thread");
         exit(-1);
      }
   }
}


//printf("main: From %d to %d \n", k, n);
ll res = CountMinors(n, matr, k, n, 0);
for(int t = 0; t < thread; ++t)
{
   int stat;
   if (tids[t] != 0 && pthread_join(tids[t], (void**)&stat) != 0)
   {
      perror("Joining the thread");
```

```
            exit(-1);
        }
        if(stat != 0){
            exit(-1);
        }
        res += results[t];
    }

    clock_gettime(CLOCK_REALTIME, &end);
    double time_spent = ( end.tv_sec - begin.tv_sec ) + (double)( end.tv_nsec - begin.tv_nsec )/ (d
ouble)1000000000L;
    printf("res: %lld, time: %.4lf\n", res, time_spent);

    free(matr);

    return 0;
}
```

get_int.c

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

#include "get_int.h"

int get_char(int fd)
{
    char c;
    if (read(fd, &c, 1) == 1)
        return (unsigned char)c;
    return EOF;
}

int get_int(int fd, int* res)
{
    int c;
    int sign = 1;
    int i = 0;
    *res = 0;
    while((c = get_char(fd)) != EOF && c != ' ' && c != '\n' && c!= '\r')
    {
        if(i == 0 && c == '-'){
            sign = -1;
        }
        else if(c >= '0' && c <= '9'){
            *res *= 10;
            *res += (int)(c - '0');
        }else{
            return -1;
```

```
        }

        i++;
    }
    *res *= sign;

    return 0;
}
```

Пример работы программы.

```
pablo$ make
gcc -pthread   -c -o src/laba_3.o src/laba_3.c
gcc -pthread   -c -o src/get_int/get_int.o src/get_int/get_int.c
gcc -pthread -o laba_3 src/laba_3.o src/get_int/get_int.o
pablo$ cat tests/test2
11
34 37 28 16 44 36 37 43 50 22 13
28 41 10 14 27 41 27 23 37 12 19
18 30 33 31 13 24 18 36 30 3 23
9 20 18 44 7 12 43 30 24 22 20
35 38 49 25 16 21 14 27 42 31 7
24 13 21 47 32 6 26 35 28 37 6
47 30 14 8 25 46 33 46 15 18 35
15 44 1 38 9 27 29 39 35 4 2
5 50 33 11 27 19 40 13 27 37 45
40 46 21 35 29 18 2 48 3 18 43
3 7 2 31 37 42 16 40 45 20 41
pablo$ ./laba_3 1  < tests/test2
res: 572523893989395712, time: 1.0836
pablo$ ./laba_3 2 < tests/test2
res: 572523893989395712, time: 0.7142

pablo$ ./laba_3 1
2
1 2
3 4
res: -2, time: 0.0003
```

Выводы.

Выполнив данную лабораторную работу, я научился делать программы, использующие несколько потоков для ускорения вычислений. Если посмотреть примеры работы программы, то можно увидеть, что с 1 потоком программа вычислила детерминант матрицы ранга 11 за 1.08 секунды. С 2 же потоками программа отработала за 0.71 секунду. Программа с 2 потока не отработала в 2 раза быстрее, чем с 1 потоком, так как мой алгоритм расчета детерминанта не является самым оптимальным.