Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М80-207Б-20

Студент: Мерц С.П.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 04.12.21

Москва, 2021.

Содержание

Постановка задачи

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 задания: 2. Необходимо отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки

Общие сведения о программе

Программа состоит из одного файла task.c.

Программа использует следующие системные вызовы:

1. **pthread_create – для создания потока**.
2. **pthread_join** – для остановки родительского потока и ожидания результата от под потока.
3. **clock_gettime –** для получения реального времени, которое используется для подсчёта времени работы алгоритма.
4. **sleep –** для перевода процесса в ожидание, чтобы его можно было заметить.

Общий метод и алгоритм решения

Алгоритм быстрой сортировки не нуждается в описании, для распараллеливания создаю новый поток, если возможно, для половины рассматриваемой части массива, а вторую половину обрабатываю в родительском процессе.

Код программы

parent.c:

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

```
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>

#define ERROR_CREATE_THREAD -11
#define ERROR_JOIN_THREAD   -12
#define SUCCESS               0

unsigned current_number, number;
pthread_t* threads;

typedef struct
{
    unsigned left, right;
    int *array;
} thread_data;

void* native_quick_sort(void* arguments)
{
      //sleep(1);
      thread_data* data = (thread_data*) arguments;
      int pivot; // разрешающий элемент
      unsigned left = data→left;
      unsigned right = data→right;
      pivot = data→array[left];
      while (left < right)
      {
      while ((data→array[right] >= pivot) && (left < right))
                  right--;
      if (left != right)
      {
                  data→array[left] = data→array[right];
                  left++;
```

```
        }
        while ((data→array[left] <= pivot) && (left < right))
                left++;
        if (left != right)
        {
                data→array[right] = data→array[left];
                right--;
        }
        }
        data→array[left] = pivot; // ставим разрешающий элемент на место
        pivot = left;
        left = data→left;
        right = data→right;


        thread_data a = {left, pivot - 1, data→array}, b = {pivot + 1,
right, data→array};
        if ((left < pivot) && (right > pivot))
       {
          if (current_number)
          {
                  current_number--;
                  int id_thread = number - current_number - 1;
                  if ((pivot - left) >= (right - pivot))
                  {
                        int status = pthread_create(&threads[id_thread],
NULL, native_quick_sort, &a);
                        if (status != SUCCESS)
                        {
                              printf("Ошибка при создании процесса\n");
                     printf("Код ошибки %d\n", status);
                     exit(ERROR_CREATE_THREAD);
                        }
                        native_quick_sort(&b);
                  }
                  else
```

```
                    {
                        int status = pthread_create(&threads[id_thread],
NULL, native_quick_sort, &b);
                        if (status != SUCCESS)
                        {
                            printf("Ошибка при создании процесса\n");
                        printf("Код ошибки %d\n", status);
                        exit(ERROR_CREATE_THREAD);
                        }
                        native_quick_sort(&a));
                    }
                    int status_arrt = pthread_join(threads[id_thread], NULL);
                    if (status_arrt != SUCCESS)
                    {
                        printf("Ошибка при присоединении процесса\n");
                    printf("Код ошибки %d\n", status_arrt);
                    exit(ERROR_JOIN_THREAD);
                    }
            }
            else
            {
                native_quick_sort(&a);
                native_quick_sort(&b);
            }
        }
        else
        {
            if (left < pivot)
               native_quick_sort(&a);
               if (right > pivot)
               native_quick_sort(&b);
        }
          return SUCCESS;
    }
```

```
int main(int argc, char* argv[])
{
     if ((argc != 3) || (atoi(argv[1]) < 0) || (atoi(argv[2]) < 1))
    {
        printf("Аргументы вызова программы: (не отрицательное) число потоков (позитивное) размер массива\n");
        exit(1);
    }
    number = strtol(argv[1], NULL, 10);
    unsigned size = strtol(argv[2], NULL, 10);
    if (number > size)
    {
        printf("Количество потоков больше размера массива. Количество потоков приравнено к размеру массива\n");
        number = size;
    }

     pid_t pid = getpid();
    printf("PID: %d\n", pid);

    current_number = number;
    int *array = (int*) malloc(size * sizeof(int));
    for (int i = 0; i < size; i++)
        array[i] = rand();

     struct timespec mt1, mt2;

    threads = (pthread_t*)malloc(number * sizeof(pthread_t));
    thread_data a = {0, size - 1, array};
     clock_gettime (CLOCK_REALTIME, &mt1);
    native_quick_sort(&a);
     clock_gettime (CLOCK_REALTIME, &mt2);

    printf("Массив отсортирован:\n")
```

```
            long int tt=1000000000*(mt2.tv_sec - mt1.tv_sec)+(mt2.tv_nsec -
 mt1.tv_nsec);
            printf ("Затрачено время: %ld нс\n",tt);


          free(array);
          free(threads);
          return 0;
      }
```

STRACE

sawa@DESKTOP-3KLEPQB:/mnt/c/Users/Савелий/Documents/GitHub/OS/lab3$ strace ./a.out 5 100

execve("./a.out", ["./a.out", "5", "100"], 0x7ffff42b24e0 /* 19 vars */) = 0

brk(NULL)   = 0x7ffff1e77000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffffa6242d0) = -1 EINVAL (Invalid argument)

access("/etc/ld.so.preload", R_OK)   = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=41063, ...}) = 0

mmap(NULL, 41063, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5558325000

close(3)   = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\345Ga\367\265T\320\374\301V)Yf]\223\337"..., 68, 824) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5558360000

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\345Ga\367\265T\320\374\301V)Yf]\223\337"..., 68, 824) = 68

mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5558302000

mmap(0x7f5558309000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7f5558309000

mmap(0x7f555831a000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18000) = 0x7f555831a000

mmap(0x7f555831f000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c000) = 0x7f555831f000

mmap(0x7f5558321000, 13432, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5558321000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"...,
68, 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"...,
68, 880) = 68

mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5558110000

mprotect(0x7f5558135000, 1847296, PROT_NONE) = 0

mmap(0x7f5558135000, 1540096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f5558135000

mmap(0x7f55582ad000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x19d000) = 0x7f55582ad000

mmap(0x7f55582f8000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f55582f8000

mmap(0x7f55582fe000, 13528, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f55582fe000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f5558100000

arch_prctl(ARCH_SET_FS, 0x7f5558100740) = 0

mprotect(0x7f55582f8000, 12288, PROT_READ) = 0

mprotect(0x7f555831f000, 4096, PROT_READ) = 0

mprotect(0x7f555836b000, 4096, PROT_READ) = 0

mprotect(0x7f555835d000, 4096, PROT_READ) = 0

```
munmap(0x7f5558325000, 41063)           = 0

set_tid_address(0x7f5558100a10)         = 119

set_robust_list(0x7f5558100a20, 24)     = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x7f5558309bf0, sa_mask=[],
sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f55583173c0}, NULL, 8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f5558309c90, sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f55583173c0}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=8192*1024}) = 0

getpid()                                = 119

fstat(1, {st_mode=S_IFCHR|0660, st_rdev=makedev(0x4, 0x2), ...}) = 0

ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0

brk(NULL)                               = 0x7ffff1e77000

brk(0x7ffff1e98000)                     = 0x7ffff1e98000

write(1, "PID: 119\n", 9PID: 119

)             = 9

clock_gettime(CLOCK_REALTIME, {tv_sec=1638737842, tv_nsec=859911900}) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f55578f0000

mprotect(0x7f55578f1000, 8388608, PROT_READ|PROT_WRITE) = 0

clone(child_stack=0x7f55580effb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[120],
tls=0x7f55580f0700, child_tidptr=0x7f55580f09d0) = 120

futex(0x7f55580f09d0, FUTEX_WAIT, 120, NULL) = -1 EAGAIN (Resource temporarily unavailable)

munmap(0x7f55560c0000, 8392704)         = 0

clock_gettime(CLOCK_REALTIME, {tv_sec=1638737842, tv_nsec=861732300}) = 0

write(1, "\320\234\320\260\321\201\321\201\320\270\320\262
\320\276\321\202\321\201\320\276\321\200\321\202\320\270\321\200\320\276\320"..., 39Массив
отсортирован:

) = 39

write(1, "\320\227\320\260\321\202\321\200\320\260\321\207\320\265\320\275\320\276
\320\262\321\200\320\265\320\274\321\217: 1"..., 44Затрачено время: 1820400 нс

) = 44
```

exit_group(0)   = ?

+++ exited with 0 +++

Исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков

Метрики параллельных вычислений - это система показателей, позволяющая оценивать преимущества, получаемые при параллельном решении задачи на n процессорах, по сравнению с последовательным решением той же задачи на единственном процессоре. С другой стороны, они позволяют судить об обоснованности применения данного числа процессоров для решения конкретной задачи.

Базисом для определения метрик являются следующие характеристики вычислений:

Tp – время выполнения на p различных потоках/вычислительных ядрах

Sp – ускорение. Sp = T1/Tp, Sp < p

Xp – эффективность/загруженность. Xp = Sp/p, Xp<1

Приведём таблицу с результатами вычисления метрик для размера массива 10000:

p	T1 (nsec)	Tp (nsec)	Ускорение (Sp=T1/Tp)	Эффективность (Xp=Sp/p)
4	923800	1057800	0.87	0.2175
6	923800	914100	1.01	0.168
8	923800	851700	1.08	0.135
10	923800	867900	1.06	0.106

Приведём таблицу с результатами вычисления для 1000000:

p	T1 (nsec)	Tp (nsec)	Ускорение (Sp=T1/Tp)	Эффективность (Xp=Sp/p)
4	110992000	100345700	1,106	0,2765
6	110992000	95165300	1,166	0,19
8	110992000	95497600	1,162	0,14
10	110992000	99146200	1,119	0,1119

И для 100000000:

p	T1 (nsec)	Tp (nsec)	Ускорение (Sp=T1/Tp)	Эффективность (Xp=Sp/p)
4	14424255800	12442504300	1,159	0.289
6	14424255800	12410890400	1,162	0.193
8	14424255800	12417981200	1,161	0.145
10	14424255800	12419270200	1,161	0.116

Таким образом, алгоритм хорошо параллелится при больших количествах входных данных и небольшого числа потоков. Это связано с тем, что в работе решета потоки могут выполнять лишнюю работу и при этом тратится дополнительное время на создание и ожидание этих потоков. Чем больше потоков, тем больше лишней работы будет выполняться потоками и чем больше входное число, тем больше времени уйдёт на работу потоков, занятых лишней работой. Максимальное ускорение дают первые дополнительные потоки, потому что вызываются для большого куска массива. Последующий обрабатывают в 2 раза меньше предыдущего уже частично отсортированного куска массива.

Вывод

Использование потоков может пригодится в любой системе: в однопроцессорной, в которой достаточная часть времени уходит на ожидание ввода-вывода и в многопроцессорных, где задачи могут выполняться параллельно на разных процессорах, что даёт рост производительности программы. К сожалению, не любой алгоритм хорошо выполняется как многопоточная программа. Однако для некоторых из них есть параллельные реализации. Я думал алгоритм

быстрой сортировки будет гораздо более эффективен по времени в параллельной реализации.