Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные сети»
Тема: Управление потоками в ОС

Студент: Д. С. Ляшун
Преподаватель: Е. С. Миронов
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2020

1 Постановка задачи

Цель работы: приобретение практических навыков в управлении потоками в ОС, а также обеспечении 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. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

Вариант №19. Необходимо реализовать проверку числа на простоту при помощи алгоритма «решето Эратосфена».

2 Алгоритм решения

«Решето Эратосфена» устроено следующим образом: создается массив, в котором хранится информация о простоте чисел (от 1 до n, где n - проверяемое на простоту число), по-началу все числа считаются простыми. Далее осуществляется линейный проход по массиву, начиная с 2. Если рассматриваемое число x является простым, то происходит дополнительный проход по массиву, в котором отмечаются все числа, большие числа x и кратные ему, как составные; если же число составное, то происходит переход к следующему числу. Алгоритм работает за $O(nlog_2log_2n)$.

Для решения данной задачи с использованием потоков можно поступить так: при нахождении простого числа будет вызываться поток, который самостоятельно отмечает все числа, кратные данному простому числу в массиве. Но стоит учитывать тот факт, что поток не всегда успеет отработать простое число до встречи в основном потоке с числом, кратным этому простому числу. В таком случае основному потоку необходимо приостановить работу и дождаться завершения выполнения потока обработки соответствующего простого числа.

Для работы с потоками будут использоваться следующие процедуры и функции:

1. int pthread_create(pthread_t* thread, const pthread_attr_t* attr, void *(*routine)(void*), void* arg) – функция, создающая поток с атрибутами, заданными attr (если attr=NULL используются атрибуты по умолчанию). Идентификатор потока сохраняется в thread. В самом потоке выполняется функция routine с передаваемым аргументом arg. Функция pthread_create() возвращает ноль в случае успеха, иначе - номер возникшей ошибки.

2. int pthread_join(pthread_t thread, void value_ptr) – функция, приостанавливающая выполнение вызывающего потока до тех пор, пока не завершится указанный поток thread. Значение, которое передавалось в pthread_exit() при выходе из потока thread станет доступно по указателю value_ptr. В случае успеха функция возвращает 0, иначе - номер возникшей ошибки.

3 Листинг программы

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <mcheck.h>
#include <time.h>
typedef struct
{
    char* matrix;
    unsigned long long prime_num;
    unsigned long long max_num;
    unsigned long long* count_threads;
} pthrData;
void* threadFunc(void* thread_data)
{
    pthrData *data = (pthrData*) thread_data;
    for (unsigned long long i = data->prime_num*data->prime_num; i <= data->max_num &&
        i > data->prime_num; i += data->prime_num)
    {
        data->matrix[i] = 1;
    }
    *data->count_threads = *data->count_threads - 1;
    return NULL;
}
const int MAXIMUM_PROCESS = 1000;
int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        perror("Error! The number of threads to create is not specified!\n");
        return -1;
    }
    unsigned long long count = atoi(argv[1]);
    if (count > MAXIMUM_PROCESS)
    {
        perror("Error! Big count of threads!\n");
        return -1;
    }
    printf("Input numbrer to check: ");
    unsigned long long number;
    scanf("%llu", &number);
    char* matrix = NULL;
    matrix = (char*) malloc((number+1)*sizeof(char));
    if (matrix == NULL)
    {
        perror("Memory allocation error!\n");
        return -2;
    }
```

```
47      for (int i = 0; i <= number; ++i)
48      {
49          matrix[i] = 0;
50      }
51      pthread_t* threads = NULL;
52      pthrData* threadsData = NULL;
53      threads = (pthread_t*) malloc(count*sizeof(pthread_t));
54      threadsData = (pthrData*) malloc(count*sizeof(pthrData));
55      if (threads == NULL || threadsData == NULL)
56      {
57          perror("Memory allocation error!\n");
58          return -2;
59      }
60      unsigned long long count_threads = 0;
61      long double start, finish;
62      start = clock();
63      for (unsigned long long i = 2; i*i <= number && i*i > i; ++i)
64      {
65          if (matrix[i] == 0)
66          {
67            if (count_threads < count)
68            {
69                matrix[i*i] = 2+count_threads;
70                threadsData[count_threads].matrix = matrix;
71                threadsData[count_threads].prime_num = i;
72                threadsData[count_threads].max_num = number;
73                threadsData[count_threads].count_threads = &count_threads;
74                if (pthread_create(&threads[count_threads], NULL, threadFunc, &threadsData
                      [count_threads]))
75                {
76                    perror("Error creating thread!\n");
77                    return -4;
78                }
79                ++count_threads;
80            }
81            else
82            {
83              for (unsigned long long j = i*i; j <= number && j > i; j += i)
84              {
85                  matrix[j] = 1;
86              }
87            }
88           }
89           else if (matrix[i] >= 2)
90           {
91              if (pthread_join(threads[matrix[i]-2], NULL))
92              {
93                  perror("Error executing thread!\n");
94                  return -6;
```

```
95              }
96          }
97      }
98      finish = clock();
99      printf("Time of execution %Lf ms.\n", (finish - start)/1000.0);
100     if (matrix[number])
101     {
102         printf("%llu is composite number.\n", number);
103     }
104     else
105     {
106         printf("%llu is prime number.\n", number);
107     }
108     free(threads);
109     free(threadsData);
110     free(matrix);
111     return 0;
112 }
```

4 Тесты и протокол работы

```
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ ls
a1.txt  prog  prog.c
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ gcc prog.c -o prog -lpthread
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ ./prog 1000
Input numbrer to check: 9999971
Time of execution 331.258000 ms.
9999971 is prime number.
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ ./prog 5000
Error! Big count of threads!
: Success
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ ./prog 500
Input numbrer to check: 5000000
Time of execution 205.383000 ms.
5000000 is composite number.
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ ./prog 300
Input numbrer to check: 12
Time of execution 0.214000 ms.
12 is composite number.
dmitry@dmitry-VirtualBox:~/OS_labs/Lab3$ ./prog 300
Input numbrer to check: 123459
Time of execution 7.907000 ms.
123459 is composite number.
```

5 Strace

```
dmitry@dmitry-VirtualBox:~/Work_place/OS_labs/Lab3$ strace -f ./prog 2
execve("./prog",["./prog","2"],0x7ffc5899c3d0 /* 49 vars */) = 0
brk(NULL)                                  = 0x55729e03d000
arch_prctl(0x3001 /* ARCH_??? */,0x7ffc7119f980) = -1 EINVAL (Недопустимый
аргумент)
access("/etc/ld.so.preload",R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD,"/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
fstat(3,st_mode=S_IFREG|0644,st_size=67205,...) = 0
mmap(NULL,67205,PROT_READ,MAP_PRIVATE,3,0) = 0x7ff3d8929000
close(3)                                   = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libpthread.so.0",O_RDONLY|O_CLOEXEC)
= 3
read(3,"77ELF>2001"...,832) = 832
```

```
pread64(3,"4GNUO0574364B216442406@61327o"...,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) =
0x7ff3d8927000
pread64(3,"4GNUO0574364B216442406@61327o"...,68,824) = 68
mmap(NULL,140408,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff3d8904000
mmap(0x7ff3d890b000,69632,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x7000) = 0x7ff3d890b000
mmap(0x7ff3d891c000,20480,PROT_READ,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x18000)
= 0x7ff3d891c000
mmap(0x7ff3d8921000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x1c000) = 0x7ff3d8921000
mmap(0x7ff3d8923000,13432,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS,-1,0) = 0x7ff3d8923000
close(3)                                = 0
openat(AT_FDCWD,"/lib/x86_64-linux-gnu/libc.so.6",O_RDONLY|O_CLOEXEC) = 3
read(3,"77ELF>60q"...,832) = 832
pread64(3,"@@@"...,784,64) = 784
pread64(3,"0GNU00",32,848) = 32
pread64(3,"4GNU6377?320070704d45n55Y77~34"...,68,880) = 68
fstat(3,st_mode=S_IFREG|0755,st_size=2029224,...) = 0
pread64(3,"@@@"...,784,64) = 784
pread64(3,"0GNU00",32,848) = 32
pread64(3,"4GNU6377?320070704d45n55Y77~34"...,68,880) = 68
mmap(NULL,2036952,PROT_READ,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff3d8712000
mprotect(0x7ff3d8737000,1847296,PROT_NONE) = 0
mmap(0x7ff3d8737000,1540096,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x25000) = 0x7ff3d8737000
mmap(0x7ff3d88af000,303104,PROT_READ,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x19d000) = 0x7ff3d88af000
mmap(0x7ff3d88fa000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE,3,0x1e7000) = 0x7ff3d88fa000
mmap(0x7ff3d8900000,13528,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS,-1,0) = 0x7ff3d8900000
close(3)                                = 0
mmap(NULL,12288,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0)
= 0x7ff3d870f000
arch_prctl(ARCH_SET_FS,0x7ff3d870f740) = 0
mprotect(0x7ff3d88fa000,12288,PROT_READ) = 0
mprotect(0x7ff3d8921000,4096,PROT_READ) = 0
mprotect(0x55729dbe9000,4096,PROT_READ) = 0
```

```
mprotect(0x7ff3d8967000,4096,PROT_READ) = 0
munmap(0x7ff3d8929000,67205)            = 0
set_tid_address(0x7ff3d870fa10)          = 3113
set_robust_list(0x7ff3d870fa20,24)      = 0
rt_sigaction(SIGRTMIN,sa_handler=0x7ff3d890bbf0,sa_mask=[],sa_flags=SA_RESTORER|
SA_SIGINFO,sa_restorer=0x7ff3d89193c0,NULL,8) = 0
rt_sigaction(SIGRT_1,sa_handler=0x7ff3d890bc90,sa_mask=[],sa_flags=SA_RESTORER|
SA_RESTART|SA_SIGINFO,sa_restorer=0x7ff3d89193c0,NULL,8) = 0
rt_sigprocmask(SIG_UNBLOCK,[RTMIN RT_1],NULL,8) = 0
prlimit64(0,RLIMIT_STACK,NULL,rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY)
= 0
fstat(1,st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...) = 0
brk(NULL)                                = 0x55729e03d000
brk(0x55729e05e000)                      = 0x55729e05e000
fstat(0,st_mode=S_IFCHR|0620,st_rdev=makedev(0x88,0),...) = 0
write(1,"Input numbrer to check: ",24Input numbrer to check: ) = 24
read(0,100
"100",1024)                   = 4
clock_gettime(CLOCK_PROCESS_CPUTIME_ID,tv_sec=0,tv_nsec=2831771) = 0
mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0)
= 0x7ff3d7f0e000
mprotect(0x7ff3d7f0f000,8388608,PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7ff3d870dfb0,flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|
CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID,parent_tid=[3115],tls=0x7ff3d870e700,
child_tidptr=0x7ff3d870e9d0) = 3115
mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0)
= 0x7ff3d770d000
mprotect(0x7ff3d770e000,8388608,PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7ff3d7f0cfb0,flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|
CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID,parent_tid=[3116],tls=0x7ff3d7f0d700,
child_tidptr=0x7ff3d7f0d9d0) = 3116
futex(0x7ff3d870e9d0,FUTEX_WAIT,3115,NULLstrace: Process 3116 attached
<unfinished ...>
[pid  3116] set_robust_list(0x7ff3d7f0d9e0,24) = 0
[pid  3116] madvise(0x7ff3d770d000,8368128,MADV_DONTNEEDstrace: Process 3115
attached
<unfinished ...>
[pid  3115] set_robust_list(0x7ff3d870e9e0,24) = 0
[pid  3115] madvise(0x7ff3d7f0e000,8368128,MADV_DONTNEED) = 0
```

```
[pid  3116] <... madvise resumed>)      = 0
[pid  3116] exit(0 <unfinished ...>
[pid  3115] exit(0 <unfinished ...>
[pid  3116] <... exit resumed>)         = ?
[pid  3115] <... exit resumed>)         = ?
[pid  3116] +++ exited with 0 +++
[pid  3115] +++ exited with 0 +++
<... futex resumed>)                    = 0
clone(child_stack=0x7ff3d870dfb0,flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|
CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID,parent_tid=[3117],tls=0x7ff3d870e700,
child_tidptr=0x7ff3d870e9d0) = 3117
mmap(NULL,8392704,PROT_NONE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,-1,0)
= 0x7ff3d6f0c000
mprotect(0x7ff3d6f0d000,8388608,PROT_READ|PROT_WRITE) = 0
clone(child_stack=0x7ff3d770bfb0,flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|
CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID,parent_tid=[3118],tls=0x7ff3d770c700,
child_tidptr=0x7ff3d770c9d0) = 3118
futex(0x7ff3d870e9d0,FUTEX_WAIT,3117,NULLstrace: Process 3118 attached
<unfinished ...>
[pid  3118] set_robust_list(0x7ff3d770c9e0,24) = 0
[pid  3118] madvise(0x7ff3d6f0c000,8368128,MADV_DONTNEED) = 0
[pid  3118] exit(0)                     = ?
[pid  3118] +++ exited with 0 +++
strace: Process 3117 attached
[pid  3117] set_robust_list(0x7ff3d870e9e0,24) = 0
[pid  3117] madvise(0x7ff3d7f0e000,8368128,MADV_DONTNEED) = 0
[pid  3117] exit(0)                     = ?
[pid  3113] <... futex resumed>)        = 0
[pid  3113] clock_gettime(CLOCK_PROCESS_CPUTIME_ID,tv_sec=0,tv_nsec=3889737)
= 0
[pid  3113] write(1,"Time of execution 1.058000 ms.",31Time of execution 1.058000
ms.
) = 31
[pid  3113] write(1,"100 is composite number.",25100 is composite number.
) = 25
[pid  3113] lseek(0,-1,SEEK_CUR)      = -1 ESPIPE (Недопустимая операция смещения)
[pid  3113] exit_group(0)               = ?
[pid  3117] +++ exited with 0 +++
+++ exited with 0 +++
```

Как видно, сперва вызывается функция execv, которая запускает файл программы на исполнение. В её аргументах указывается число потоков, которое было введено в консоли. Печать и вывод в консоль с использованием функций scanf и printf реализуется соответственно с помощью системных вызовов read и write, где в аргументах указывается буфер, откуда читаются или записываются данные, их размер в байтах, а также дескриптор файла, откуда читаются или записываются данные (0 - стандартный поток ввода, 1 - стандартный поток вывода).

Для проведения замеров времени работы алгоритма вызывается функция clock, которая делает системный вызов функции clock_gettime, записывающей текущие показания часов выполнения данного процесса в наносекундах, разность полученных значений и будет являться временем работы.

В ходе запуска потоков видно, что это производится путем вызова системных функций: mmap, где указывается, что под каждый поток выделяется примерно 8 Мбайт памяти; mprotect, контролирующий доступ к области памяти, в нём указывается, что в выделенной памяти разрешается чтение и запись данных для создаваемого потока; и clone, который производит клонирование потоков с указанным образом выполнения (адреса исполняемой функции), возвращая id потока. Ожидания выполнения того или иного потока выполняется с помощью вызова pthread_join, в котором происходит системный вызов futex, использующийся для ожидания основным потоком изменения значения адреса указанной памяти и также пробуждения ожидающих выполнения потоков на указанном адресе.

6 Ускорение и эффективность работы

K	Время исполнения	Ускорение	Эффективность
1	3367 ms	1.00	1.00
2	1859 ms	1.81	0.90
3	1638 ms	2.05	0.68
4	1503 ms	2.24	0.56
5	1523 ms	2.21	0.42
7	1523 ms	2.21	0.30
8	1530 ms	2.20	0.27
9	1522 ms	2.21	0.24
10	1530 ms	2.20	0.22

Тестирование программы проводилось на числах размера 10^8. Как видно, после запуска 5 потока ускорение начало оставаться примерно одинаковым, из-за чего эффективность стала уменьшаться. Это объясняется тем, что на моем компьютере всего 2 ядра и 4 потока. Поэтому удалось достичь максимального ускорения только при таком количестве потоков.

7 Выводы

В результате выполнения лабораторной работы я познакомился с использованием и управлением потоков в ОС Linux, а также обеспечении синхронизации между ними. В ходе проведения замеров ускорения и эффективности программы в зависимости от числа потоков я выяснил, что их количество не прямо пропорционально влияет на время работы, все во многом зависит от характеристик используемого процессора.

Основную трудность для меня составила разработка программы с учетом многопоточности – я не совсем понимал, как это можно сделать, не нарушив синхронизацию работы алгоритма решения рассматриваемой задачи. Также в ходе выполнения лабораторной я понял, что при использовании многопоточности написание исходного кода гораздо усложняется ввиду создания структур для передачи данных в потоки, написания их функций работы, а также необходимости определения мест, где тот или иной поток должен уже завершиться.

При работе с потоками мне показалось неудобным то, что они вызываются в произвольном порядке, а не по порядку их создания, но, несмотря на это, я считаю, что подход многопоточности в написании программ является правильным архитектурным решением, поскольку позволяет разбить их на самостоятельные части, которые как кирпичики будут связываться между собой для выстраивания общей логики работы, и по эффективности такие программы могут в разы быть быстрее обычных.

Список литературы

[1] Таненбаум Э., Босх Х. *Современные операционные системы. – 4-е изд.* – СПб.: Издательский дом «Питер», 2018. – С. 123-146

[2] *Linux Man Pages*
URL: `http://ru.manpages.org` (дата обращения: 18.11.2020).