Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные системы»

Студентка: А. Довженко
Преподаватель: Е. С. Миронов
Группа: 08-207
Вариант: 19
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №3

Задача: Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы. При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемых программой. При возможности необходимо использовать максимальное количество возможных потоков.

Вариант 19: Есть набор 128 битных чисел, записанных в шестнадцатиричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом".

1 Описание

Алгоритм моей программы не сильно отличает от классического, который описан на первых страницах любого учебника по математике средней школы. Единственное отличие – это многопоточность, но и ее можно избежать, если использовать один поток. Для хранения 128битных чисел я использую редкоиспользуемый тип int128. У меня есть две структуры – в первой хранятся пользовательские данные: количество потоков и размер оперативной памяти, во второй – параметры одного потока. Файл с числами у меня перегенерируется при каждом запуске программы, это действие изолировано в функции generate(), в случае чего этого можно избежать. После того, как я узнаю, сколько чисел записано в файле, я инициализирую каждый поток начальными данными в функции init(), там же я распределяю числа по потокам. Далее создаю потоки, в которые зачитываю числа этого потока, и там считаю локальные средние арифметические. Далее объединяю потоки и складываю локальные суммы. Остается только напечатать 128битное число в десятичном представлении.

Из библиотеки pthreads использовались следующие функции:

pthreadcreate() – создание нового потока.
pthreadjoin() – ожидает завершение потока, переданного в аргументах.

Системные вызовы:

void exit(int status) – выходит из процесса с заданным статусом.
int close(int fd) – закрывает файловый дескриптор.
int open(const char *path, int oflag, ...) – открытие файлового дескриптора: первый аргумент – путь до файла, второй – флаги открытия.
int write(int fd, void *buffer, int nbyte) – записывает количество байтов в 3 аргументе из буфера в файл с дискриптором fd.
offt lseek(int fd, offt offset, int whence) – устанавливает смещение для файлового дескриптора в значение аргумента offset в соответствии с директивой whence.

2 Исходный код

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <inttypes.h>
#include <limits.h>
#include <string.h>

#define DEC_SIZE 40
#define NUMBER_SIZE 32
#define FILE_SIZE 1

typedef unsigned __int128 int128_t;

void *thread_function(void *);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;

typedef struct _params {
    long long nc;
    int128_t localsum;
    unsigned int counter;
    off_t start_pos;
    long long cc; //thread count controller
} Params;

typedef struct _command {
    int threads_num;
    int memory_set;
} Command;

const char *file_name = "test.dat";
void init(Params *ptr, Command *command, long long num_count);
void generate();
void parse_commdand_line(int argc, char **argv, Command *command);
void print_int128(int128_t u128);
int is_num(char *s);
int hex_to_dec(char *s);


int main(int argc, char **argv)
{
    Command command;
```

```
48      parse_commdand_line(argc, argv, &command);
49
50      if (command.threads_num * sizeof(Params) + command.threads_num * sizeof(pthread_t)
            > command.memory_set) {
51          fprintf(stderr, "%s\n", "Too much threads for this amount of memory");
52          exit(EXIT_FAILURE);
53      }
54      pthread_t *thread_id = (pthread_t *) malloc(command.threads_num * sizeof(pthread_t)
            );
55
56      Params *params = (Params *) malloc(command.threads_num * sizeof(Params));
57
58      int i, j;
59
60      generate();
61      int fd = open(file_name, O_RDWR| O_CREAT, 0666);
62      long long size = lseek(fd, 0, SEEK_END);
63      close(fd);
64      long long num_count = size / (NUMBER_SIZE + 1); //+1 for /n
65      init(params, &command, num_count);
66      for (i = 0; i < command.threads_num; ++i) {
67          pthread_create(&thread_id[i], NULL, thread_function, (void *) &params[i]); //
                last null is param
68      }
69
70      for (j = 0; j < command.threads_num; ++j) {
71          pthread_join(thread_id[j], NULL);
72      }
73
74      int128_t sum = 0;
75      for (int i = 0; i < command.threads_num; ++i) {
76          sum += params[i].localsum;
77      }
78      print_int128(sum);
79      return 0;
80  }
81
82
83  void init(Params *ptr, Command *command, long long num_count)
84  {
85      ptr[0].nc = num_count;
86      ptr[0].localsum = 0;
87      ptr[0].counter = 0;
88      ptr[0].start_pos = 0;
89      ptr[0].cc = (num_count / command->threads_num);
90      for (int i = 1; i < command->threads_num; ++i) {
91          ptr[i].nc = num_count;
92          ptr[i].localsum = 0;
93          ptr[i].counter = 0;
```

```
94          ptr[i].start_pos = i * (ptr[i - 1].cc * (NUMBER_SIZE + 1));
95          ptr[i].cc = ptr[i-1].cc;
96      }
97      ptr[command->threads_num - 1].cc += num_count % command->threads_num;
98  }
99
100 void generate()
101 {
102     int fd = open(file_name, O_RDWR| O_CREAT, 0666);
103     char buf[NUMBER_SIZE];
104     srand(time(NULL));
105     for (int i = 0; i < FILE_SIZE; ++i) {
106         for (int i = 0; i < NUMBER_SIZE; ++i) {
107             if (((int) rand()) % 2 == 0) {
108                 buf[i] = '0' + (((int) rand()) % 10);
109             } else {
110                 buf[i] = 'A' + (((int) rand()) % 6);
111             }
112         }
113         write(fd, &buf, NUMBER_SIZE);
114         write(fd, "\n", 1);
115     }
116     close(fd);
117 }
118
119 void print_int128(int128_t u128)
120 {
121     char buf[DEC_SIZE + 1];
122     int i;
123     for (i = 0; i < DEC_SIZE; ++i) {
124         buf[i] = '0';
125     }
126     buf[DEC_SIZE] = '\0';
127     for (i = DEC_SIZE - 1; u128 > 0; --i) {
128         buf[i] = (int) (u128 % 10) + '0';
129         u128 /= 10;
130     }
131     if (i == DEC_SIZE - 1) {
132         printf("%d\n", 0);
133     } else {
134         printf("%s\n", &buf[i + 1]);
135     }
136 }
137
138 int is_num(char *s)
139 {
140     return (*s >= '0' && *s <= '9');
141 }
142
```

```
143 int hex_to_dec(char *s)
144 {
145     if (*s == 'A')
146         return 10;
147     if (*s == 'B')
148         return 11;
149     if (*s == 'C')
150         return 12;
151     if (*s == 'D')
152         return 13;
153     if (*s == 'E')
154         return 14;
155     if (*s == 'F')
156         return 15;
157     return 0;
158 }
159 
160 int128_t atobigint(char *str)
161 {
162     int128_t res = 0;
163     while (*str) {
164         if(is_num(str))
165             res = res * 16 + (*str - '0');
166         else {
167             int kek = hex_to_dec(str);
168             res = res * 16 + kek;
169         }
170         ++str;
171     }
172     return res;
173 }
174 
175 
176 void *thread_function(void *dummyPtr)
177 {
178     Params *ptr = (Params *)dummyPtr;
179     char buf[NUMBER_SIZE + 1];
180     char c;
181     int fd = open(file_name, O_RDWR| O_CREAT, 0666);
182 
183     lseek(fd, ptr->start_pos, SEEK_SET);
184     for (int i = 0; i < ptr->cc; ++i)
185     {
186         read(fd,buf,NUMBER_SIZE);
187         buf[NUMBER_SIZE] = '\0'; //important
188         int128_t s;
189         s = atobigint(buf);
190         s /= ptr->nc;
191         ptr->localsum += s;
```

```
192         read(fd, &c, 1);
193         if (c != '\n' && c != '\0') {
194             fprintf(stderr, "%s %d\n","c = ", c);
195             fprintf(stderr, "%s %ld\n","start pos = ", ptr->start_pos);
196             fprintf(stderr, "%s %lld\n","cc = ", ptr->cc);
197             fprintf(stderr, "%s\n", "format error");
198             exit(EXIT_FAILURE);
199         }
200     }
201     close(fd);
202     return 0; //0 == NULL
203 }
204
205
206 void parse_commdand_line(int argc, char **argv, Command *command)
207 {
208     if (argc != 3) {
209         fprintf(stderr, "%s\n","Usage: ThreadsNumber RAM");
210         exit(EXIT_FAILURE);
211     }
212     command->threads_num = atoi(argv[1]);
213     command->memory_set = atoi(argv[2]);
214 }
```

3 Тестирование

- Некорректные входные данные:

  Запуск без ключей

```
karma@karma:~/mai_study/OS/lab3$ ./run
Usage: ./run ThreadsNumber RAM
karma@karma:~/mai_study/OS/lab3$ ./run 2
Usage: ./run ThreadsNumber RAM
```

  Отрицательные ключи (число потоков и размер оперативной памяти)

```
karma@karma:~/mai_study/OS/lab3$ ./run -10 200
ThreadsNumber and RAM must be >0
karma@karma:~/mai_study/OS/lab3$ ./run 10 -200
ThreadsNumber and RAM must be >0
```

  Слишком большое количество потоков

```
karma@karma:~/mai_study/OS/lab3$ ./run 10 200
Too much threads for this amount of memory
karma@karma:~/mai_study/OS/lab3$ ./run 123 987
Too much threads for this amount of memory
```

- В файле записано 1 число:
  Утилитой bc проверяем корректность выполнения программы (должно выводиться то же число, что записано в файле)

  Для одного потока

```
karma@karma:~/mai_study/OS/lab3$ cat test.dat
6A05EFCFDA39AE8054A777334EBFBDFB
karma@karma:~/mai_study/OS/lab3$ bc
bc 1.06.95
Copyright 1991-1994,1997,1998,2000,2004,2006 Free Software Foundation,Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
ibase = 16
6A05EFCFDA39AE8054A777334EBFBDFB
140928993001156134839717901008612867579
karma@karma:~/mai_study/OS/lab3$ ./run 1 1000
DEC: 140928993001156134839717901008612867579
```

Для одного потока с другим количеством оперативной памяти

```
karma@karma:~/mai_study/OS/lab3$ ./run 1 150
DEC: 14092899300115613483971790100861286757 9
```

Для нескольких потоков с тем же количеством оперативной памяти, что и для одного потока

```
karma@karma:~/mai_study/OS/lab3$ ./run 2 150
DEC: 14092899300115613483971790100861286757 9
```

Для нескольких потоков с другим количеством оперативной памяти

```
karma@karma:~/mai_study/OS/lab3$ ./run 2 5000
DEC: 14092899300115613483971790100861286757 9
```

- В файле записано несколько чисел (1000):
  Для одного потока с минимальным количеством оперативной памяти

```
karma@karma:~/mai_study/OS/lab3$ ./run 1 72
DEC: 20082230063523945697073558962738845544 2
```

Для одного потока с большим количеством оперативной памяти

```
karma@karma:~/mai_study/OS/lab3$ ./run 1 10000000000
DEC: 20588046796281412218031815091700680356 5
```

Для нескольких потоков с минимальным количеством оперативной памяти

```
karma@karma:~/mai_study/OS/lab3$ ./run 10 720
DEC: 18088380036902914553853284044550452854 5
```

Для нескольких потоков с большим количеством оперативной памяти

```
karma@karma:~/mai_study/OS/lab3$ ./run 10 10000000
DEC: 20742626439096994468786568388589725936 4
```

Для большого количества потоков с минимальным количеством оперативной памяти. Тут программа падает.

```
karma@karma:~/mai_study/OS/lab3$ ./run 100000 7200000
DEC: 0
```

Для большого количества потоков с большим количеством оперативной памяти. И тут падает.

```
karma@karma:~/mai_study/OS/lab3$ ./run 100000 10000000
DEC: 0
```

4 Выводы

Совршенно очевидно, что многопоточность – это великолепная идея. Чтобы осознать это, достаточно взглянуть на большинство серверных приложений: в них потоки позволяют "изолировать"одинаковые участки программы для разных данных. Например, игровой сервер может выделять каждому игроку отдельный поток, стартующий при подключении игрока и завершающийся при его отключении. Код, реализующий аналогичную функциональность без многопоточности, очень быстро становится совершенно захламленным, а ошибки в нем находятся все труднее и труднее. Можно сказать, что потоки состоят со своими управляющими функциями в тех же отношениях, что объекты состоят с классами. Это значит, что мы получаем те же преимущества.

Неоспоримым недостатком является проблема синхронизации: в описанном выше игровом приложении нельзя допустить, чтобы несколько игроков одновременно читали и изменяли базу данных, поскольку в таком случае они могли бы повредить ее содержимое.

В итоге, я потренировалась в работе с POSIX threads, что очень полезно, так как потоки используются повсеместно в промышленном программировании. В который раз мне повезло чуть меньше, чем моим коллегам. 32битные шестнадцатиричные числа редко где встречаются, первое, что приходит в голову, – md5 суммы. Сложно представить себе задачу, в которой нужно вычислять среднее арифметическое таких сумм.