Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные системы»

Управление потоками в ОС. Обеспечение синхронизации между потоками

Студент: П. А. Мохляков
Преподаватель: Е. С. Миронов
Группа: М8О-208Б-19
Вариант: 1
Дата:
Оценка:
Подпись:

Москва, 2021

1 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.

Отсортировать массив целых чисел при помощи битонической сортировки.

2 Сведения о программе

Программа написанна на Си в Unix подобной операционной системе на базе ядра Linux. Для компиляции требуется ключ -lpthread, для запуска программы нелбходимо указать в качестве аргумента количество потоков, которые максимально могут быть использованы.

Программа сортирует массив с помощью битонной сортировки. Сначала пользователь должен ввести колличество элементов массива. Далее должен передать все элементы.

Существует три глобальных переменных, которые являются мьютексом и 2 переменнами колличества потоков, используемые и максимальные. Их использование между потоками регулируется мьютексом. Сортируемый массив передается как указатель. И для него не нужен мьютекс, так как каждый поток работает на своем участке, не перекрываемом другими.

3 Общий метод и алгоритм решения

Парамментром запуска программы мы указваем максимальное количество используемых потоков.

Далее мы указываем длину массива, и создаем массив дополив длину до ближайшего числа 2^n. Так как это необходимо для работы алгоритма сортировки. Далее мы считываем данные, а оставшийся участок массива заполняем максимально возможными элементами, чтобы после сортировки они оказалить в конце массива, и мы могли также легко их удалить.

Далее вызывается функция сортировки. Сначала мы рекурсивно разбиваем массив

по полам, задавая целевое направление сначала вниз, а потом вверх, чтобы получить битонную последовательность. При этом пока у нас есть свободные потоки мы выдиляем под вторую половину отдельный поток, а когда они закончатся начинаем работать в однопотоке, на выделенном участке.

Когда мы достигаем массива из одного элемента его можно считать отсортированным, как по возрастанию так и по убыванию. Поэтому два соседних элемента можно считать битонной последовательностью, которую можно собрать в одну возрастающую или убывающую.

Начинаем объядинять битонные последовательности. Находим расстояние между двумя элиментами, которые будем сравнивать, оно равно половине участка массива. Далее мы проходим по половине участка и сравниваем каждый элемент с элементом через заданное расстояние. При необходимости меняем их местами. Далее разбиваем данный участок на 2 и повторяем слияние, и так пока его длина не станет равна 1.

Таким образом мы получаем битоническую последовательность большего размера, к которой можно опять применить слияние. Повторяем эту процедуру до оканчания сортироки. При этом когда мы будем делать слияние для 2 участков, созданных разными потоками, слияние мы опять разбиваем на несколько потоков, пока это возможно.

При всем этом если один поток подготовил последовательность, а второй еще нет, то первый его ждет, из чего следует вывод, что увеличение производительности будет только при увеличении количества потоков до следующей степени двойки.

4 Листинг программы

main.c

```
1  #include "stdio.h"
2  #include "stdlib.h"
3  #include "bitonic.h"
4  #include <sys/time.h>
5
6  #define MAXINT 2147483647
7
8  int SizeStep(int Num){
9      int i = 1;
10     while(i < Num)
11         i *= 2;
12     return i;
13 }
14
15
16 int main(int argc, char *argv[]){
17     int threads = 1;
```

```
18
19      if(argc == 2){
20          threads = atoi(argv[1]);
21      }
22
23      int input_size;
24      scanf("%d",&input_size);
25
26      int size_array = SizeStep(input_size);
27      int *array = malloc(sizeof(int)*size_array);
28
29      for(int i = 0; i < input_size; ++i)
30          scanf("%d",array+i);
31      for(int i = input_size; i < size_array; ++i)
32          array[i] = MAXINT;
33
34      #ifdef time
35      struct timeval startwtime, endwtime;
36      gettimeofday(&startwtime, NULL);
37      #endif
38
39      bitonicsort(array, size_array, threads);
40
41      #ifdef time
42      gettimeofday(&endwtime, NULL);
43      double time = (double)((endwtime.tv_usec - startwtime.tv_usec)/1.0e6 + endwtime.
            tv_sec - startwtime.tv_sec);
44      printf("%f\n", time);
45      #endif
46
47
48      #ifndef time
49      for(int i=0;i<input_size;++i){
50          printf("%d\n",array[i]);
51      }
52      #endif
53
54      free(array);
55      return 0;
56  }
```

bitonuc.h

```
1   #pragma once
2
3   #include "pthread.h"
4
5   #define UP 1
6   #define DOWN 0
7
```

```c
typedef struct ArgsBitonic{
    int *array;
    int size;
    int start;
    int dir;
}ArgsBitonic;

void InitArgs(ArgsBitonic *args, int *array, int size, int start, int dir);
void Comparator(int *array, int i, int j, int dir);
void BitonicMergeSinglThread(ArgsBitonic *args);
void BitonicSortSinglThread(ArgsBitonic *args);
void BitonicMergeMultiThreads(ArgsBitonic *args);
void BitonicSortMultiThreads(ArgsBitonic *args);
void bitonicsort(int *array, int size, int threads);
```

bitonuc.c

```c
#include "pthread.h"
#include "bitonic.h"
#include "stdio.h"

#define UP 1
#define DOWN 0

pthread_mutex_t lock;
size_t max_threads = 1;
size_t use_threads = 1;

void InitArgs(ArgsBitonic *args, int *array, int size, int start, int dir){
    args->array = array;
    args->size = size;
    args->start = start;
    args->dir = dir;
}


void Comparator(int *array, int i, int j, int dir){
    if(dir == (array[i] > array[j])){
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}

void BitonicMergeSinglThread(ArgsBitonic *args){
    if(args->size > 1){
        int nextsize = args->size / 2;
        for(int i = args->start; i < nextsize + args->start; ++i){
               Comparator(args->array, i, i + nextsize, args->dir);
        }
```

```
34
35          ArgsBitonic args1;
36          ArgsBitonic args2;
37          InitArgs(&args1, args->array, nextsize, args->start, args->dir);
38          InitArgs(&args2, args->array, nextsize, args->start + nextsize, args->dir);
39
40          BitonicMergeSinglThread(&args1);
41          BitonicMergeSinglThread(&args2);
42      }
43  }
44
45  void BitonicSortSinglThread(ArgsBitonic *args){
46      if(args->size > 1){
47          int nextsize = args->size / 2;
48
49          ArgsBitonic args1;
50          ArgsBitonic args2;
51          InitArgs(&args1, args->array, nextsize, args->start, DOWN);
52          InitArgs(&args2, args->array, nextsize, args->start + nextsize, UP);
53
54          BitonicSortSinglThread(&args1);
55          BitonicSortSinglThread(&args2);
56          BitonicMergeSinglThread(args);
57      }
58  }
59
60  void BitonicMergeMultiThreads(ArgsBitonic *args){
61      if(args->size > 1){
62          int nextsize = args->size / 2;
63          int isParal = 0;
64          pthread_t tid;
65
66          for(int i = args->start; i < nextsize + args->start; ++i){
67                  Comparator(args->array, i, i + nextsize, args->dir);
68          }
69
70          ArgsBitonic args1;
71          ArgsBitonic args2;
72          InitArgs(&args1, args->array, nextsize, args->start, args->dir);
73          InitArgs(&args2, args->array, nextsize, args->start + nextsize, args->dir);
74
75          pthread_mutex_lock(&lock);
76          if(use_threads < max_threads){
77              ++use_threads;
78              pthread_mutex_unlock(&lock);
79              isParal = 1;
80              pthread_create(&tid, NULL,(void*) &BitonicMergeMultiThreads, &args1);
81              BitonicMergeMultiThreads(&args2);
82          } else {
```

```
83              pthread_mutex_unlock(&lock);
84              BitonicMergeSinglThread(&args1);
85              BitonicMergeSinglThread(&args2);
86          }
87
88          if(isParal){
89              pthread_join(tid, NULL);
90              pthread_mutex_lock(&lock);
91              --use_threads;
92              pthread_mutex_unlock(&lock);
93          }
94      }
95  }
96
97  void BitonicSortMultiThreads(ArgsBitonic *args){
98      if(args->size > 1 ){
99          int nextsize = args->size / 2;
100         int isParal = 0;
101         pthread_t tid;
102
103         ArgsBitonic args1;
104         ArgsBitonic args2;
105         InitArgs(&args1, args->array, nextsize, args->start, DOWN);
106         InitArgs(&args2, args->array, nextsize, args->start + nextsize, UP);
107
108         pthread_mutex_lock(&lock);
109         if(use_threads < max_threads){
110             ++use_threads;
111             pthread_mutex_unlock(&lock);
112             isParal = 1;
113             pthread_create(&tid, NULL,(void*) &BitonicSortMultiThreads, &args1);
114             BitonicSortMultiThreads(&args2);
115         } else {
116             pthread_mutex_unlock(&lock);
117             BitonicSortSinglThread(&args1);
118             BitonicSortSinglThread(&args2);
119         }
120
121         if(isParal){
122             pthread_join(tid, NULL);
123             pthread_mutex_lock(&lock);
124             --use_threads;
125             pthread_mutex_unlock(&lock);
126         }
127         BitonicMergeMultiThreads(args);
128     }
129 }
130
131 void bitonicsort(int *array, int size, int threads){
```

```
132     pthread_mutex_init(&lock, NULL);
133 
134     ArgsBitonic args;
135     InitArgs(&args,array,size,0,UP);
136 
137     if(threads > 1)
138         max_threads = threads;
139 
140     BitonicSortMultiThreads(&args);
141 
142     pthread_mutex_destroy(&lock);
143 }
```

5 Демонстрация работы программы

```
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ make clean
rm -r *.o main
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ make
gcc -c -Wall src/main.c
gcc -c -Wall src/bitonic.c
gcc main.o bitonic.o -pthread -o main
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ ./main 1
5
5 4 3 2 1
1
2
3
4
5
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ ./main 2
8
23 54 2 -12 6 2 13423354 -1231
-1231
-12
2
2
6
23
54
13423354
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ ./main 4
```

```
4
2 5 0 -10
-10
0
2
5
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 1
28.916599
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 2
15.838372
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 3
15.663935
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 4
9.641949
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 5
9.792694
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 6
10.508794
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 7
10.540718
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat test | ./main 8
11.463141
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ strace -f -e
trace="%process,write" -o log.txt ./main 4
8
23 55 -36 7 0 254 -454 3
-454
-36
0
3
7
23
55
254
pavel@DESKTOP-K5KMLPV:~/Project/mai/2_course/OS/LB3$ cat log.txt
1273  execve("./main",["./main","4"],0x7ffda4153150 /* 29 vars */) = 0
1273  arch_prctl(0x3001 /* ARCH_??? */,0x7ffc25e05a30) = -1 EINVAL
(Invalid argument)
1273  arch_prctl(ARCH_SET_FS,0x7f2ea1d04740) = 0
1273  clone(child_stack=0x7f2ea1d02fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS
```

```
|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,parent_tid=[1274],
tls=0x7f2ea1d03700,child_tidptr=0x7f2ea1d039d0) = 1274
1273  clone(child_stack=0x7f2ea1501fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,parent_tid=[1275],
tls=0x7f2ea1502700,child_tidptr=0x7f2ea15029d0) = 1275
1275  exit(0)                           = ?
1274  clone(child_stack=0x7f2ea0d00fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
1275  +++ exited with 0 +++
1273  clone(child_stack=0x7f2ea1501fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
1274  <... clone resumed>,parent_tid=[1276],tls=0x7f2ea0d01700,
child_tidptr=0x7f2ea0d019d0) = 1276
1273  <... clone resumed>,parent_tid=[1277],tls=0x7f2ea1502700,
child_tidptr=0x7f2ea15029d0) = 1277
1277  exit(0)                           = ?
1277  +++ exited with 0 +++
1276  exit(0)                           = ?
1276  +++ exited with 0 +++
1274  clone(child_stack=0x7f2ea0d00fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,parent_tid=[1278],
tls=0x7f2ea0d01700,child_tidptr=0x7f2ea0d019d0) = 1278
1274  clone(child_stack=0x7f2ea1501fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
1278  exit(0 <unfinished ...>
1274  <... clone resumed>,parent_tid=[1279],tls=0x7f2ea1502700,
child_tidptr=0x7f2ea15029d0) = 1279
1278  <... exit resumed>)               = ?
1278  +++ exited with 0 +++
1279  exit(0)                           = ?
1279  +++ exited with 0 +++
1274  exit(0)                           = ?
1274  +++ exited with 0 +++
1273  clone(child_stack=0x7f2ea1d02fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,parent_tid=[1280],
```

```
tls=0x7f2ea1d03700,child_tidptr=0x7f2ea1d039d0) = 1280
1273  clone(child_stack=0x7f2ea0d00fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
1280  clone(child_stack=0x7f2ea1501fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
1273  <... clone resumed>,parent_tid=[1281],tls=0x7f2ea0d01700,
child_tidptr=0x7f2ea0d019d0) = 1281
1280  <... clone resumed>,parent_tid=[1282],tls=0x7f2ea1502700,
child_tidptr=0x7f2ea15029d0) = 1282
1281  exit(0)                           = ?
1281  +++ exited with 0 +++
1282  clone(child_stack=0x7f2ea0d00fb0,flags=CLONE_VM|CLONE_FS|
CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,parent_tid=[1283],
tls=0x7f2ea0d01700,child_tidptr=0x7f2ea0d019d0) = 1283
1283  exit(0)                           = ?
1283  +++ exited with 0 +++
1282  exit(0)                           = ?
1282  +++ exited with 0 +++
1280  exit(0)                           = ?
1280  +++ exited with 0 +++
1273  write(1,"-454\n",5)            = 5
1273  write(1,"-36\n",4)             = 4
1273  write(1,"0\n",2)               = 2
1273  write(1,"3\n",2)               = 2
1273  write(1,"7\n",2)               = 2
1273  write(1,"23\n",3)              = 3
1273  write(1,"55\n",3)              = 3
1273  write(1,"254\n",4)             = 4
1273  exit_group(0)                     = ?
1273  +++ exited with 0 +++
```

6 Исследование ускорения и эффективности

Для исследования производительности будем использовать несколько тестов: на 100000, 1000000, 10000000

Таблица 1. Исследование на 100000 элементах.

Количество потоков(n)	Время работы	Ускорение ($S_n = T_1/T_n$)	Эффективность ($X_n = S_n/n$)
1	0.137688	-	-
2	0.070220	1.96	0.98
3	0.074930	1.83	0.61
4	0.054423	2.53	0.63
5	0.043279	3.18	0.63
6	0.051713	2.66	0.44
7	0.045668	3.01	0.43
8	0.043089	3.19	0.39

Таблица 2. Исследование на 1000000 элементах.

Количество потоков(n)	Время работы	Ускорение ($S_n = T_1/T_n$)	Эффективность ($X_n = S_n/n$)
1	1.516532	-	-
2	0.950804	1.59	0.79
3	0.804578	1.88	0.63
4	0.562645	2.69	0.67
5	0.519941	2.92	0.58
6	0.615402	2.46	0.41
7	0.623988	2.43	0.35
8	0.645662	2.35	0.29

Таблица 3. Исследование на 10000000 элементах.

Количество потоков(n)	Время работы	Ускорение ($S_n = T_1/T_n$)	Эффективность ($X_n = S_n/n$)
1	32.312667	-	-
2	17.034210	1.89	0.95
3	17.034277	1.89	0.63
4	10.538664	3.07	0.76
5	11.227089	2.88	0.57
6	11.185609	2.89	0.48
7	11.993463	2.69	0.38
8	10.383236	3.11	0.39

7 Вывод

Многие языки программирования позволяют пользователю работать с потоками. Создание потоков происходит быстрее, чем создание процессов, за счет того, что при создании потока не копируется область памяти, а они все работают с одной областью памяти. Поэтому многопоточность используют для ускарения не зависящих друг от друга, однотипнях задач, которые будут работать параллельно.

Язык Си предоставляет данный функционал пользователям Unix-подобных операционных систем с помощью библиотеки pthread.h. Средствами языка Си можно совершать системные запросы на создание и ожидания завершения потока, а также использовать различные примитивы синхронизации.

В данной лабораторной работе бый реализован и исследован алгоритм битонной сортировки. Установив при этом, что используя 4 потока можно молучить выигрыш по времени в 2,5-3 раза. При дальнейшем увеличении потоков прирост почти не увеличивается, а даже может уменьшаться из-за того, что на управление и переключение потоков уходит больше времени, чем они выигрывают.