Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные системы»

Студент: Т.А.Габдуллин
Преподаватель: Е. С. Миронов
Группа: 80-206Б

Вариант: 1
Дата:
Оценка:
Подпись:

Москва, 2017-2018

Лабораторная работа №3

Вариант: 1. Отсортировать массив строк при помощи битонической сортировки.

Алгоритм основан на сортировке битонных последовательностей. Такой последовательностью называется последовательность, которая сначала монотонно не убывает, а затем монотонно не возрастает, либо приводится к такому виду в результате циклического сдвига.

Любая последовательнось, входящая в битонную, любая последовательность состоящая из одного или двух элементов, а также любая монотонная последовательность также является битонной. Например, последовательности {3,5,10,4,1}, {1,5}, {10,14,5,-1,-4} являются битонными, а {4,6,1,9,2} не является. Каждое множество неотсортированных элементов можно считать множеством битонных последовательностей, состоящих из двух элементов.

Процесс битонного слияния преобразует битонную последовательность в полностью отсортированную последовательность. Алгоритм битонной сортировки состоит из применения битонных преобразований до тех пор, пока множество не будет полностью отсортировано

2 Исходный код

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>

#define ASCENDING 1
#define DESCENDING 0

// data array size
int num_threads, cnt_threads;

// data array to be sorted:
char* a[]=
    {
        "witchers",
        "are",
        "mutants,",
        "men",
        "subjected",
        "to",
        "gruelling",
        "training",
        "and",
        "flesh-altering",
        "experiments",
        "that",
        "prepare",
        "them",
        "for",
        "one",
        "purpose:",
        "to",
        "kill",
        "monsters.",
    };

int N = sizeof(a)/sizeof(char*);

void print()
{
    for (int i = 0; i < N; i++)
        printf("%s  ", a[i]);
    printf("\n");
}

void exchange(int i, int j)
{
```

```
  char* tmp;
  tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}
void compare(int i, int j, int dir)
{
   if ((strcmp(a[i],a[j])>0)==dir)
      exchange(i,j);
}

void bitonicMerge(int low, int cnt, int dir)
{
   if (cnt>1)
   {
      int k=cnt/2;
      for (int i=low; i<low+k; ++i)
         compare(i, i+k, dir);
      bitonicMerge(low, k, dir);
      bitonicMerge(low+k, k, dir);
   }
}

void bitonicSort(int low, int cnt, int dir)
{
   if (cnt>1)
   {
      int k=cnt/2;
      bitonicSort(low, k, ASCENDING);
      bitonicSort(low+k, k, DESCENDING);
      bitonicMerge(low, cnt, dir);
   }
}

void sort()
{
   bitonicSort(0, N, ASCENDING);
}

struct sarg
{
   int low;
   int cnt;
   int dir;
   int level;
};

void* thread_bitonicMerge(void* thread_arg)
{
   struct sarg* p = (struct sarg*)thread_arg;
   int low = p->low;
   int cnt = p->cnt;
```

```
    int dir = p->dir;
    int level = p->level;

    if(cnt>1)
    {
        int k=cnt/2;
        for(int i=low; i<low+k; ++i)
            compare(i,i+k,dir);
        if(level<=0)
        {
            bitonicMerge(low, k, dir);
            bitonicMerge(low+k, k, dir);
            return NULL;
        }
        struct sarg thread_arg1;
        pthread_t thread1;
        thread_arg1.low = low;
        thread_arg1.cnt = k;
        thread_arg1.dir = dir;
        thread_arg1.level = level-1;

        struct sarg thread_arg2;
        pthread_t thread2;
        thread_arg2.low = low+k;
        thread_arg2.cnt = k;
        thread_arg2.dir = dir;
        thread_arg2.level = level-1;

        pthread_create(&thread1, NULL, thread_bitonicMerge, &thread_arg1);
        cnt_threads++;
        pthread_create(&thread2, NULL, thread_bitonicMerge, &thread_arg2);
        cnt_threads++;

        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
    }
    return NULL;
}

void* thread_bitonicSort(void* thread_arg)
{
    struct sarg* p = (struct sarg*)thread_arg;
    int low = p->low;
    int cnt = p->cnt;
    int dir = p->dir;
    int level = p->level;

    if (cnt>1)
    {
        int k = cnt/2;
        if(level >= num_threads)
        {
```

```
            bitonicSort(low, k,ASCENDING);
            bitonicSort(low+k,k,DESCENDING);
        }
        else
        {
            struct sarg thread_arg1;
            pthread_t thread1;
            thread_arg1.low = low;
            thread_arg1.cnt = k;
            thread_arg1.dir = ASCENDING;
            thread_arg1.level = level+1;
            pthread_create(&thread1, NULL, thread_bitonicSort, &thread_arg1);
            cnt_threads++;

            struct sarg thread_arg2;
            pthread_t thread2;
            thread_arg2.low = low+k;
            thread_arg2.cnt = k;
            thread_arg2.dir = DESCENDING;
            thread_arg2.level = level+1;
            pthread_create(&thread2, NULL, thread_bitonicSort, &thread_arg2);
            cnt_threads++;

            pthread_join(thread1, NULL);
            pthread_join(thread2, NULL);
        }

        struct sarg thread_arg3;
        thread_arg3.low = low;
        thread_arg3.cnt = cnt;
        thread_arg3.dir = dir;
        thread_arg3.level = num_threads-level;
        thread_bitonicMerge(&thread_arg3);
    }
    return NULL;
}

void thread_sort()
{
    struct sarg thread_arg;
    thread_arg.low = 0;
    thread_arg.cnt = N;
    thread_arg.dir = ASCENDING;
    thread_arg.level = 0;

    thread_bitonicSort(&thread_arg);
}

int main(int argc, char **argv)
{
    if (argc!=2 || atoi(argv[1])>256)
    {t_threads = 0;
```

```
    num_threads = atoi(argv[1]);

    printf("\nArray (%d elements):\n", N);
    print();

    thread_sort();
    printf( "Bitonic parallel recursive with %d threads\n", cnt_threads);
    printf("\nSorted array:\n");
    print();
    printf("\n");
    return 0;
}
        printf( "Usage: t is the number of threads, <=256, to use.\n");
        exit(1);
    }
    cnt_threads = 0;
    num_threads = atoi(argv[1]);

    printf("\nArray (%d elements):\n", N);
    print();

    thread_sort();
    printf( "Bitonic parallel recursive with %d threads\n", cnt_threads);
    printf("\nSorted array:\n");
    print();
    printf("\n");
    return 0;
}
```

3 Консоль

```
timxag@KEKNOTE:~/Документы$ gcc -Wall -pthread -o o3 lab3.c
timxag@KEKNOTE:~/Документы$ ./o3 2

Array (20 elements):
witchers are mutants, men subjected to gruelling training and flesh-altering experiments that prepare them for one purpose: to kill monsters.
Bitonic parallel recursive with 16 threads

Sorted array:
and are experiments gruelling flesh-altering kill men mutants, one for prepare purpose: that them monsters. to to training witchers subjected
```

4 Вывод

Мной была изучена битоническая сортировка и написан код, осуществляющий эту сортировку в многопоточном режиме. При обработке использовались стандартные средства создания потоков Unix.