Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Операционные системы»
III Семестр

Задание 3
Вариант 1

Студент:	Алексеева М.А.
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

Москва 2019

1. Описание задания

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

Вариант 1:

Отсортировать массив строк при помощи битонической сортировки.

2. Код программы

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

#define MAX 4700

int THREAD_MAX;
int k;

struct Args {
    char **arr;
    int l;
    int r;
    int dir;
};
typedef struct Args Args;

int max(int a, int b) {
    return (a > b) ? a : b;
}

int min(int a, int b) {
    return (a < b) ? a : b;
}
```

```
void swap(char *o1, char *o2) {
   char *tmp1 = (char*)malloc(strlen(o1) * sizeof(char));
   char *tmp2 = (char*)malloc(strlen(o2) * sizeof(char));
   strcpy(tmp1, o1);
   strcpy(tmp2, o2);
   strcpy(o2, tmp1);
   strcpy(o1, tmp2);
}

void compAndSwap(char** a, int i, int j, int dir)
{
   if (dir==(strcmp(a[i],a[j]) > 0))
      swap(a[i],a[j]);
}


void bitonicMerge(char** a, int low, int cnt, int dir)
{
   if (cnt>1 )
   {
      int m = cnt/2;
      for (int i=low; i<low+m; i++)
         compAndSwap(a, i, m+i, dir);
      bitonicMerge(a, low, m, dir);
      bitonicMerge(a, m+low , m, dir);
   }

}



void* bitonicSort( void* arg)
{
   Args buf = *((Args*)arg);
   char** a = buf.arr;
   int low = buf.l;
   int cnt = buf.r;
   int dir = buf.dir;

   if (cnt>1)
   {
     int  m = cnt/2 ;

      Args forLeft;
      forLeft.arr = a;
```

```
        forLeft.l = low;
        forLeft.r = m;
        forLeft.dir = 1;

        Args forRight;
        forRight.arr = a;
        forRight.l = low + m;
        forRight.r = m;
        forRight.dir = 0;

        if(THREAD_MAX != 0){
            pthread_t left_thread;
            THREAD_MAX --;
            k++;
            pthread_create(&left_thread, NULL, &bitonicSort, &forLeft);
            bitonicSort(&forRight);

            pthread_join(left_thread, NULL);
        }
        else{
            bitonicSort(&forLeft);
            bitonicSort(&forRight);

        }
        bitonicMerge(a,low, cnt, dir);
    }
}


int main(int argc, char **argv){
    if (argc < 2) {
        printf("Error. Input number of threads.\n");
        return 0;
    };

    int s;
    int arg;
    arg = atoi(argv[1]);
    THREAD_MAX = min(max(arg, 1), MAX) - 1;

    printf("Input array size\n");
    scanf("%d", &s);
    char **arr = (char **) malloc(sizeof(char *) * s);
    for (int i = 0; i < s; i++) {
        arr[i] = (char *) malloc(sizeof(char) * 30);
```

```
        scanf("%s", arr[i]);
    }


    Args args;
    args.arr = arr;
    args.l = 0;
    args.r = s;
    args.dir = 1;

    pthread_t first_thread;
    pthread_create(&first_thread, NULL, &bitonicSort, &args);
    k++;
    pthread_join(first_thread, NULL);

    printf("\nResult:\n");
    for (int i = 0; i < s; i++)
        printf("%s\n", arr[i]);
    printf("Number of threads: %d\n", k);
    return 0;
}
```

3. Протокол работы программы

```
masha@masha-VirtualBox:~/2kurs/OS/os_02$ ./a.out 10
Input array size
8
8
7
6
5
4
3
2
1

Result:
1
2
3
4
5
6
7
8
Number of threads: 8
masha@masha-VirtualBox:~/2kurs/OS/os_02$ ./a.out 4
Input array size
```

8
8
7
6
5
4
3
2
1

Result:
1
2
3
4
5
6
7
8
Number of threads: 4
masha@masha-VirtualBox:~/2kurs/OS/os_02$ ./a.out 4
Input array size
2
aaaf
a

Result:
a
aaaf
Number of threads: 2

5. Объяснение работы программы

Поток делит исходный массив на два подмассива, по возрастанию сортируется левый, по убыванию -правый. Далее два массива сливаются в один при помощи bitonicMerge.

Вывод

Потоки применяются для многозадачности и ускорения работы программ, однако, при неправильном использовании только усложняют программу и её отладку, не давая прироста производительности.

Потоки эффективны, когда создание отдельного процесса слишком расточительно. Даже учитывая оптимизацию при создании процесса создаются новые копии страниц памяти, а потоки используют единое пространство имён. В целом создать поток на порядок быстрее, чем новый процесс.