Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 3
по курсу «Операционные системы»

Студент:	Голубев В.С.
Группа:	М8О-206Б-18
Вариант:	3
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва, 2019

Постановка задачи

Отсортировать массив строк при помощи параллельной сортировки слиянием

Операционная система: Unix.

Цель работы:

Приобретение практических навыков в:

- Управлении процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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.

Решение задачи

pthread_create – создает новый поток выполнения в программе

```
int pthread_create(*ptherad_t, const pthread_attr_t *attr, void* (*start_routine)
(void*), void *arg)
```

Функция получает в качестве аргументов указатель на поток, переменную типа pthread_t, в которую, в случае удачного завершения сохраняет id потока. pthread_attr_t – атрибуты потока. В случае если используются атрибуты по умолчанию, то можно передавать NULL. start_routin – это непосредственно та функция, которая будет выполняться в новом потоке. arg – это аргументы, которые будут переданы функции.
Поток может выполнять много разных дел и получать разные аргументы. Для этого функция, которая будет запущена в новом потоке, принимает аргумент типа void*. За счёт этого можно обернуть все передаваемые аргументы в структуру. Возвращать значение можно также через передаваемый аргумент.

Phread_join – дожидается завершения переданного потока, после чего получает его выходное значение и позволяет программе продолжать работу

int pthread_join(thread_t tid, void **status)

Откладывает выполнение вызывающего потока до тех пор, пока не будет выполнен поток thread. Когда pthread_join выполняется успешно, она возвращает 0, если поток явно возвращает значение, оно будет записано в переменную status

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ:

```
vsgolubev@vsgolubev-VirtualBox:~/OS/lab3$ ./lab3 8
Входной массив:
9
8
7
6
5
4
3
2
part- 0
part- 1
part- 2
part- 3
Выходной массив:
2
3
4
5
6
7
8
9
Количество потоков: 4
```

1. Руководство по использованию программы

Компиляция и запуск программного кода в *Ubuntu* :

gcc –pthread –o lab3 la3.c
./lab3

2. Листинг программы

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#include <math.h>

int MAX; // Количество элементов в массиве

int THREAD_MAX; //количество потоков

int part=0; //переменная, которая будет хранить количество потоков

int* a ; //указатель на массив для хранения элементов


void merge(int low, int mid, int high)

{

int n1 = mid - low + 1, nr = high - mid, i, j;
```

```
int* left = calloc(n1,sizeof(int));
int* right = calloc(nr,sizeof(int));

for(i = 0; i < n1; i++)
left[i] = a[i + low];

for(i = 0; i < nr; i++)
right[i] = a[i + mid + 1];

int k = low;
i = j = 0;

// объединяем левую и правую половины в порядке возрастания
while(i < n1 && j < nr)
{
if(left[i] <= right[j])
a[k++] = left[i++];
else
a[k++] = right[j++];
}

// вставляем оставшиеся значения из левой половины
while(i < n1) {
a[k++] = left[i++];
}

// вставляем оставшиеся значения из правой половины
while(j < nr) {
a[k++] = right[j++];
}
}

void merge_sort(int low, int high)
{
// вычисление средней точки массива
int mid = low + ((high - low) / 2);
```

```
if(low < high) {
//сортировка первой половины
merge_sort(low, mid);
// сортировка второй половины
merge_sort(mid + 1, high);
// объединение половинок
merge(low, mid, high);
}
}

//функция сортировки с использованием потоков
void* merge_sort_tread()
{
//получает идентификатор потока
int thread_part = part;
part+=1;
// вычисляем нижнюю и верхнюю границу массива
int low = thread_part * (MAX / THREAD_MAX);
int high = (thread_part + 1) * (MAX / THREAD_MAX) - 1;
// обновление средней точки
int mid = low + (high - low) / 2;
if(low < high)
{
merge_sort(low, mid);
merge_sort(mid + 1, high);
merge(low, mid, high);
}
}

void merge_rec(int tread_m)//рекурсивная функция сбора данных после отработки потоков
{
if(tread_m>THREAD_MAX)
return;
merge_rec(tread_m*2);
int minsize = MAX/tread_m;
for(int i = 0;i < tread_m; i++)
```

```
{
int low = i * minsize;
int high = (i + 1) * (minsize) - 1;
int mid = low + (high - low) / 2;
merge(low, mid, high);
}

}

int main(int argc, char* argv[])
{
MAX = atoi(argv[1]);
THREAD_MAX = MAX/ 2;
a = calloc(MAX,sizeof(int));//выделение памяти под левую половину
printf("Входной массив: \n");
for(int i = 0; i < MAX; i++) {
scanf("%d", &a[i]);
}
pthread_t threads[THREAD_MAX];//создаем массив идентификаторов потока
int status;
for(int i = 0; i < THREAD_MAX; i++)
{
printf("part- %d\n",part);
status = pthread_create(&threads[i], NULL, merge_sort_tread, NULL);
if (status != 0)
printf("main error: can't create thread, status = %d\n", status);
part++;
}
for(int i = 0; i < THREAD_MAX; i++)
pthread_join(threads[i], NULL);
merge_rec(2);
merge(0, (MAX - 1)/2, MAX - 1);
printf("Выходной массив: \n");
for(int i = 0; i < MAX; i++)
printf("%d\n", a[i]);
printf("Количество потоков: %d\n",THREAD_MAX);
```

```
return 0;
}
```

3. Вывод

В современных программах редко можно встретить реализацию через один процесс или поток, часто нам необходимо запустить какую-то стороннюю программу, при помощи уже запущенной нашей. При работе с потоками необходимо быть аккуратным, т.к. отловить различные ошибки сложнее, чем в однопоточных простых программах.