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

Студент: Вахрамян Кирилл Олегович
Группа: М8О–206Б–18
Вариант: 3
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019.

Содержание

Постановка задачи

Составить программу, 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, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.

Отсортировать массив строк при помощи TimSort.

Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: iostream, string.h, pthread.h. В программе используются следующие системные вызовы:

1. **pthread_create** – создает новый поток
2. **pthread_join** – ожидает завершения переданного потока, получает его выходное значение
3. **pthread_mutex_init** – инициализация mutex.
4. **pthread_mutex_destroy –** уничтожение mutex.
5. **pthread_mutex_lock –** блокировка части кода определенным потоком.
6. **pthread_mutex_unlock –** разблокировка части кода определенным потоком.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Ввести размер массива
2. Ввести массив строк
3. Алгоритм представляет собой разбиение на подмассивы размера RUN = 4, затем каждый подмассив сортируется InsertionSort в отдельном потоке, после происходит слияние при помощи функции merge.
4. Программа выводит исходный массив и отсортированный.

Основные файлы программы

main.cpp:

```
#include <iostream>
#include <string>
#include <pthread.h>
const int RUN = 4;
const int CAP = 64;
int NumberOfThreads;

pthread_mutex_t mutex;
void printArray(void* tmp);

struct argumetns {
    std::string arr[CAP];
    int size;
    int left;
    int right;
    int mid;
};



void* insertionSort(void* arr) {
        if (pthread_mutex_lock(&mutex) == -1) {
                perror("Mutex lock error\n");
                exit(-1);
        }
        argumetns* tmp = (argumetns*)arr;

        for (int i = tmp->left + 1; i <= tmp->right; i++) {
                std::string sw = tmp->arr[i];
                int j = i - 1;
```

```
                while (tmp->arr[j] > sw && j >= tmp->left) {
                        tmp->arr[j + 1] = tmp->arr[j];
                        j--;
                }
                tmp->arr[j + 1] = sw;
        }
        if (pthread_mutex_unlock(&mutex) == -1) {
                perror("Mutex unlock error\n");
                exit(-1);
        }
}


void* merge(void * arr) {
        argumetns* tmp = (argumetns*)arr;
        int l, m, r;
        l = tmp->left;
        m = tmp->mid;
        r = tmp->right;
        int len1 = m - l + 1, len2 = r - m;
        std::string right[len2];
        std::string left[len1];
                for (int i = 0; i < len1; i++) {
                        left[i] = tmp->arr[l + i];
                }
                for (int i = 0; i < len2; i++) {
                right[i] = tmp->arr[m + 1 + i];
                }

                int i = 0, j = 0, k = l;
                while (i < len1 && j < len2) {
                        if (left[i] <= right[j]) {
                                tmp->arr[k] = left[i];
```

```
                        i++;
                } else {
                        tmp->arr[k] = right[j];
                        j++;
                }
                k++;
        }
        while (i < len1) {
                tmp->arr[k] = left[i];
                k++;
                i++;
        }
        while (j < len2) {
                tmp->arr[k] = right[j];
                k++;
                j++;
        }
}

void TimSort(argumetns* array) {
        pthread_t threads[NumberOfThreads];
        int j = 0;

        for (int i = 0; i < array->size; i += RUN) {
                if (pthread_mutex_init(&mutex, NULL) == -1) {
                perror("Mutex init error\n");
                exit(-1);
        }
                array->left = i;
                array->right = std::min(i + 3, array->size - 1);
                pthread_create(&threads[j], NULL, insertionSort, (void*)array);
                j++;
```

```
	}

	for (j = 0; j < NumberOfThreads; j++) {
		pthread_join(threads[j], NULL);
	}

	if (pthread_mutex_destroy(&mutex) == -1) {
		perror("Mutex destroy error\n");
		exit(-1);
	}
	for (int size = 1; size < array->size; size = size * 2) {
		for (int left = 0; left < array->size; left  = left + 2 * size) {
			array->mid = left + size - 1;
			array->left = left;
			array->right = std::min(left + 2 * size - 1, array->size - 1);
			merge(array);
		}
	}
}
void printArray(void* tmp) {
	argumetns* arr = (argumetns*)tmp;
	for (int i = 0 ; i < arr->size; i++) {
		std::cout << arr->arr[i] << " ";
	}
	std::cout << std::endl;
}

int main() {
	argumetns* arr = new argumetns;
	int size;
	std::cin >> size;
	NumberOfThreads = size / RUN;
	if (size % RUN != 0) {
```

```
            NumberOfThreads++;
        }

        if (size > CAP) {
            std::cout << "size too big, possible size <= 64\n";
            return 0;
        }
        arr->size = size;
        for (int i = 0; i < arr->size; i ++) {
            std::cin >> arr->arr[i];
        }

        printArray(arr);
        TimSort(arr);
        printArray(arr);
        delete arr;

}
```

Пример работы

```
courage@courage-X550LC:~/OS/os_lab3$ ./a.out
8
qwe
asd
zxc
fgh
tyu
hkhj
ret
xcv
qwe asd zxc fgh tyu hkhj ret xcv
asd fgh hkhj qwe ret tyu xcv zxc
```

```
courage@courage-X550LC:~/OS/os_lab3$ ./a.out
6
Obuhov
Volkov
Ryabov
Nikolaev
Kozlov
Avdeev
Obuhov Volkov Ryabov Nikolaev Kozlov Avdeev
Avdeev Kozlov Nikolaev Obuhov Ryabov Volkov
```

Вывод

Обрел навыки работы с многопоточностью, использовал mutex для синхронизации потоков. Многопоточность является одним из важнейших механизмов в программировании, позволяющий синхронизировать данные программы и налаживать её корректную работу.