УО «Белорусский государственный университет информатики и радиоэлектроники»
Кафедра ПОИТ

Отчет по лабораторной работе №5

по предмету

Операционные системы и системное программирование

Выполнил
Каширский А.Е.

Проверил
Деменковец Д. В.

Группа:
851005

Минск 2020

Код программы

ConsoleApp.cpp

```
#include <stdio.h>
#include "TaskQueue.h"
#include "TaskWorker.h"
#include <thread>
#include <string>
#include <iostream>
#include <fstream>
#include <functional>


#define THREADS_COUNT 6
#define FILE_PATH "C:\\5 semester\\OSaSP\\Lab_5\\test.txt"


#define TYPE std::function<void(void)>


std::vector<std::string>* vecOfStr = new std::vector<std::string>();
std::vector < std::vector<std::string>*>* vectorOfParts = new std::vector <
std::vector<std::string>*>();



bool getFileContent(std::string fileName, std::vector<std::string>*
vecOfStrs)
{

	std::ifstream dict_file(FILE_PATH);
	std::string line;

	if (!dict_file.good()) {
		return false;
	}

	while (std::getline(dict_file, line))
	{
		std::string new_line;
		new_line = line + "\n";
```

```
            if (new_line.size() != 0)
                  vecOfStrs->push_back(new_line);
      }

      return true;
}


void addVector(std::vector<std::string>* vect, TaskQueue queue) {

      queue.addTask([vect]() {
            std::vector<std::string>* copy = vect;
            std::sort(vect->begin(), vect->end());
            });
}

#define Vector std::vector<std::string>
Vector mergeTwo(Vector A, Vector B)
{
      // Get sizes of vectors
      int m = A.size();
      int n = B.size();

      // Vector for storing Result
      Vector D;
      D.reserve(m + n);

      int i = 0, j = 0;
      while (i < m && j < n) {

            if (A[i] <= B[j])
                  D.push_back(A[i++]);
            else
                  D.push_back(B[j++]);
      }
```

```
	// B has exhausted
	while (i < m)
		D.push_back(A[i++]);

	// A has exhausted
	while (j < n)
		D.push_back(B[j++]);

	return D;
}

Vector mergeVectors() {
	Vector tmpVector;
	if (vectorOfParts->size() > 0) {
		tmpVector = *(*vectorOfParts)[0];
	}
	for (int i = 1; i < vectorOfParts->size(); i++) {
		tmpVector = mergeTwo(tmpVector, *(*vectorOfParts)[i]);
	}
	return tmpVector;
}

void outPutVector(Vector vector) {
	for (int i = 0; i < vector.size(); i++) {
		printf("   %s", vector[i].c_str());
	}
}

void splitAndSortVectors(TaskQueue taskQueue, int threadsCount) {
	int onePartCount = floor((((double)vecOfStr->size()) / threadsCount) +
.5);



	for (int i = 0; i < vecOfStr->size(); i += onePartCount) {
```

```
            std::vector<std::string>* newVector = new
std::vector<std::string>;
            vectorOfParts->push_back(newVector);
            for (int j = i; j < i + onePartCount; j++) {
                  if (j < vecOfStr->size()) {
                        std::string str = (*vecOfStr)[j];
                        newVector->push_back(str);
                  }
            }
            addVector(newVector, taskQueue);

      }
}

int main() {

      bool result = getFileContent(FILE_PATH, vecOfStr);

      if (!result) {
            printf("File isn't exist");
            return -1;
      }

      TaskQueue taskQueue;
      TaskWorker taskExecutor(taskQueue);

      int threadsCount = THREADS_COUNT > vecOfStr->size() ? vecOfStr->size() :
THREADS_COUNT;

      splitAndSortVectors(taskQueue, threadsCount);
      taskExecutor.startExecution(threadsCount);

      Vector vector = mergeVectors();
      outPutVector(vector);
}
```

TaskQueue.cpp

```
#include <mutex>
#include <queue>
#include <functional>

typedef std::function<void()> TTask;

class TaskQueue
{
public:
	TaskQueue();
	int addTask(TTask task);
	TTask popTask();
private:
	std::queue<TTask>* tasksQueue = new std::queue<TTask>;
};

std::mutex g_lock;

TaskQueue::TaskQueue()
{
}



int TaskQueue::addTask(TTask task)
{
	g_lock.lock();
	tasksQueue->push(task);
	g_lock.unlock();

	return 1;
}

TTask TaskQueue::popTask()
{
	TTask result;
```

```
	g_lock.lock();
	if (tasksQueue->empty()) {
		result = NULL;
	}
	else {
		result = tasksQueue->front();
		tasksQueue->pop();
	}
	g_lock.unlock();

	return result;
}
```

TaskWorker.cpp

```
#include <thread>
#include "TaskQueue.h"
class TaskWorker
{
private:
	TaskQueue _queue;
	int _maxThreadsCount;
public:
	TaskWorker(TaskQueue queue);

	void startExecution(int maxThreadsCount);
};

TaskWorker::TaskWorker(TaskQueue queue)
{
	_queue = queue;
}



void threadFunction(TaskQueue queue, int count)
{
	std::vector<std::thread> arr;
```

```
	while (count) {
		TTask task = queue.popTask();
		if (task != NULL) {
			std::thread thr(task);
			arr.push_back(move(thr));
			count--;
		}
	}

	for (int i = 0; i < arr.size(); i++) {
		arr[i].join();
	}
}

void TaskWorker::startExecution(int maxThreadsCount) {
	std::thread thr(threadFunction, _queue, maxThreadsCount);
	thr.join();
}
```

Скриншоты работы программы