**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)**

**Курсовой проект по курсу «Операционные системы»**

**Планировщик DAG**

Студент: Белоусов Егор Владимирович

Группа: М8О-207Б-20

Преподаватель: Миронов Е. С.

Дата:

Оценка:

**Цель работы**

1. Приобретение практических навыков в использовании знаний, полученных в течении курса  
2. Проведение исследования в выбранной предметной области

**Задание**

Необходимо реализовать на языке C++ программу, которая принимает на вход конфигурационный файл в формате json с графом из джоб. Граф должен быть неориентированным и ацикличным. Джобы должны запускаться в порядке заданном в графе, так же требуется реализовать синхронизацию для джобов.

**Описание алгоритма**

Я использовал библиотеку json для C++ для того, чтобы обработать полученный конфигурационный файл. Затем я проверяю граф на валидность, для этого требуется несколько раз запустить алгоритмы поиска на графе. Для поддержания джоб создаются потоки, в которых проверяется mutex и затем вызывается скрипт на Python с аргументами, полученными от других джоб.

**Листинг программы**

Программе требуется два файла: json.hpp и main.cpp. json.hpp является библиотекой для работы с json и содержит огромное количество строк, поэтому он не будет сюда включен.

**main.cpp**

#include "pthread.h"

#include "iostream"

#include "string"

#include "queue"

#include "mutex"

#include "thread"

#include "vector"

#include "json.hpp"

#include <fstream>

#include <unistd.h>

using namespace std;

using json = nlohmann::json;

using graph\_t = vector<vector<int>>;

json jobs\_data;

graph\_t graph;

map<string, mutex> allMutexes;

vector<thread> threads;

vector<int> jobDegree;

vector<vector<pair<int, string>>> jobArgs;

int jobsToDo;

int onGoing = 0;

mutex errorMutex;

vector<int> errorJobs;

queue<int> queueNewThreads;

queue<int> doneJobs;

mutex q\_mutex;

void dfs(int vertex, vector<char>& used, const graph\_t &graph, bool& ok) {

    used[vertex] = 1;

    for(int to : graph[vertex]){

        if(used[to] == 1){

            ok = false;

            continue;

        }

        if(used[to] == 2){

            continue;

        }

        dfs(to, used, graph, ok);

    }

    used[vertex] = 2;

}

bool isGraphDAG(const int &n, const graph\_t &graph) {

    graph\_t undirectedGraph(n + 1);

    for (int i = 1; i <= n; ++i) {

        for (int to : graph[i]) {

            undirectedGraph[to].push\_back(i);

            undirectedGraph[i].push\_back(to);

        }

    }

    // check that DAG has only one component

    queue<int> q;

    vector<char> used(n + 1, 0);

    q.push(1);

    used[1] = 1;

    while (!q.empty()) {

        int vertex = q.front();

        q.pop();

        for (int to : undirectedGraph[vertex]) {

            if (!used[to]) {

                used[to] = 1;

                q.push(to);

            }

        }

    }

    for (int i = 1; i <= n; ++i) {

        if (!used[i]) {

            return false;

        }

    }

    //check that graph have no cycles

    used.assign(n + 1, 0);

    bool ok = true;

    for(int i = 1; i <= n; ++i) {

        if(!used[i]) {

            dfs(i, used, graph, ok);

        }

    }

    return ok;

}

void thread\_routine(const int& job\_id){

    errorMutex.lock();

    if(!errorJobs.empty()){

        errorMutex.unlock();

        return;

    }

    errorMutex.unlock();

    bool hasMutex = false;

    string mtx;

    if(jobs\_data["jobs\_description"][job\_id - 1].find("mutex") != jobs\_data["jobs\_description"][job\_id - 1].end()){

        hasMutex = true;

        mtx = jobs\_data["jobs\_description"][job\_id - 1]["mutex"];

    }

    if(hasMutex){

        allMutexes[mtx].lock();

    }

    onGoing++;

    sort(jobArgs[job\_id].begin(), jobArgs[job\_id].end());

    string scriptPath = jobs\_data["jobs\_description"][job\_id - 1]["script"];

    scriptPath = "./jobs/" + scriptPath;

    string command = "python ";

    command += scriptPath + " ";

    command += to\_string(job\_id) + " ";

    for(int i = 0;i < jobArgs[job\_id].size();++i){

        command += jobArgs[job\_id][i].second + " ";

    }

    errorMutex.lock();

    if(!errorJobs.empty()){

        onGoing--;

        errorMutex.unlock();

        return;

    }

    errorMutex.unlock();

    int status = system(command.c\_str());

    if(status != 0){

        errorMutex.lock();

        errorJobs.push\_back(job\_id);

        onGoing--;

        errorMutex.unlock();

        return;

    }

    for(int to : graph[job\_id]){

        jobArgs[to].push\_back({job\_id, "./jobs\_results/" + to\_string(job\_id) + ".txt"});

        jobDegree[to]--;

        if(jobDegree[to] == 0){

            q\_mutex.lock();

            queueNewThreads.push(to);

            q\_mutex.unlock();

        }

    }

    doneJobs.push(job\_id);

    jobsToDo--;

    onGoing--;

    if(hasMutex){

        allMutexes[mtx].unlock();

    }

}

int main(int argc, char \*argv[]) {

    cout << "Enter json file: ";

    string s;

    cin >> s;

    //reading json

    ifstream file\_input(s);

    if(!file\_input.good()){

        cout << "Bad file!\n";

        return 0;

    }

    file\_input >> jobs\_data;

    int jobs\_number = jobs\_data["jobs\_number"];

    jobsToDo = jobs\_number;

    jobDegree.assign(jobs\_number + 1, 0);

    graph.resize(jobs\_number + 1);

    jobArgs.resize(jobs\_number + 1);

    threads.resize(jobs\_number + 1);

    for (int i = 0; i < jobs\_number; ++i) {

        for (auto to : jobs\_data["jobs\_graph"][i]) {

            graph[i + 1].push\_back(to);

            jobDegree[to]++;

        }

    }

    cout << "Checking that graph is DAG...\n";

    if(isGraphDAG(jobs\_number, graph)) {

        cout << "Graph is valid!\n";

    }

    else {

        cout << "Graph is invalid!\n";

        return 0;

    }

    for(int i = 1; i <= jobs\_number; ++i){

        if(jobDegree[i] == 0)queueNewThreads.push(i);

    }

    vector<string> mutexNames = jobs\_data["mutex\_names"];

    for(const string &name : mutexNames){

        allMutexes[name];

    }

    while (jobsToDo > 0) {

        errorMutex.lock();

        if(!errorJobs.empty()){

            cout << "An error occured!\n";

            errorMutex.unlock();

            int tt = 0;

            while(1){

                ++tt;

                errorMutex.lock();

                if(onGoing == 0){;

                    errorMutex.unlock();

                    break;

                }

                errorMutex.unlock();

            }

            break;

        }

        errorMutex.unlock();

        int have = doneJobs.size();

        while(have > 0){

            int jb = doneJobs.front();

            doneJobs.pop();

            cout << "Job " << jb << " is done!\n";

            --have;

        }

        if(queueNewThreads.empty())continue;

        q\_mutex.lock();

        int jobNumber = queueNewThreads.front();

        queueNewThreads.pop();

        thread newJob(thread\_routine, jobNumber);

        threads[jobNumber] = move(newJob);

        threads[jobNumber].detach();

        q\_mutex.unlock();

    }

    int have = doneJobs.size();

    while(have > 0 && errorJobs.empty()){

        int jb = doneJobs.front();

        doneJobs.pop();

        cout << "Job " << jb << " is done!\n";

        --have;

    }

    if(!errorJobs.empty()){

        cout << "Problem in ";

        for(auto jb : errorJobs){

            cout << jb << " ";

        }

        cout << "job.\n";

    }

    else{

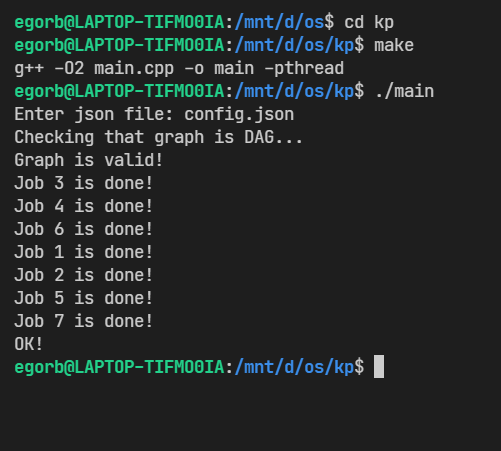
        cout << "OK!\n";

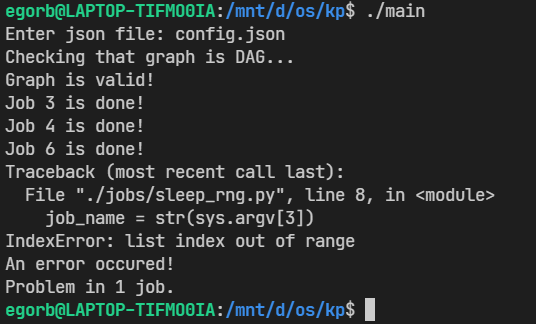
    }

    return 0;

}

**Пример работы**





**Выводы**

Во время выполнения курсового проекта я научился обрабатывать json в C++, закрепил свои знания в потоках и примитивах синхронизации. Также освежил свои знания в алгоритмах на графах, научился использовать Python вместе с C++.

Самым сложным было реализовать нормальное выполнение Python скриптов через C++ и корректную обработку ошибок.