**Московский авиационный институт (национальный исследовательский университет)**

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования

Курсовой проект по предмету "операционные системы"

Студент: Мокеева С.А.

Преподаватель: Соколов А.А.

Группа: М8О-206Б-20

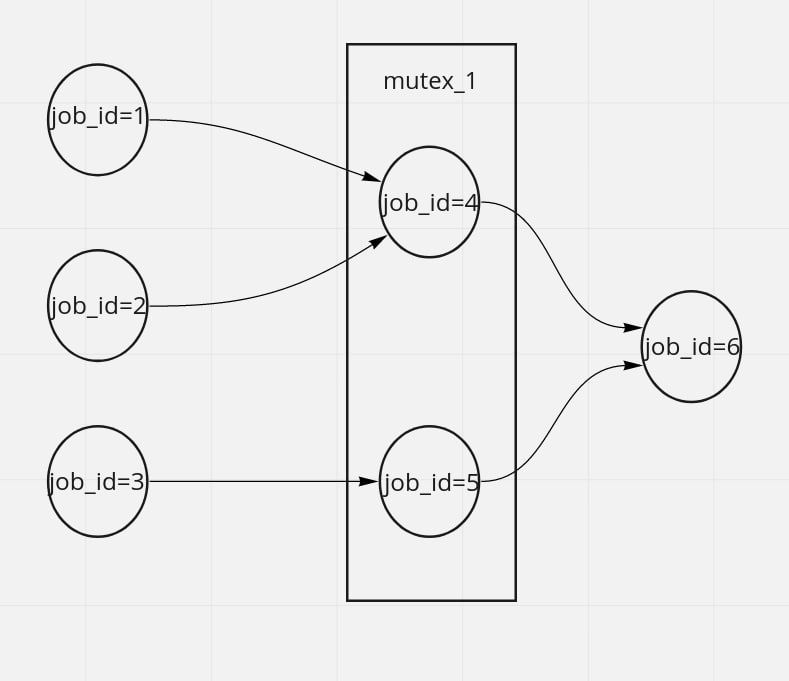
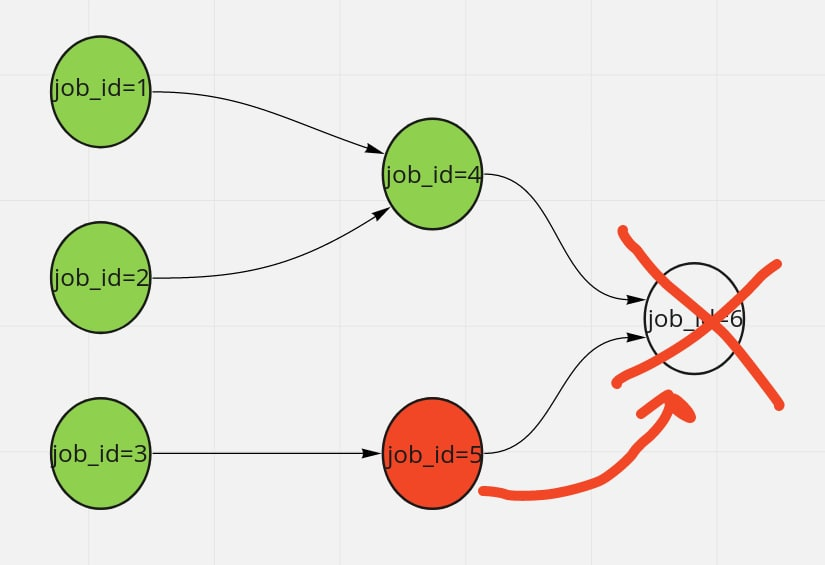
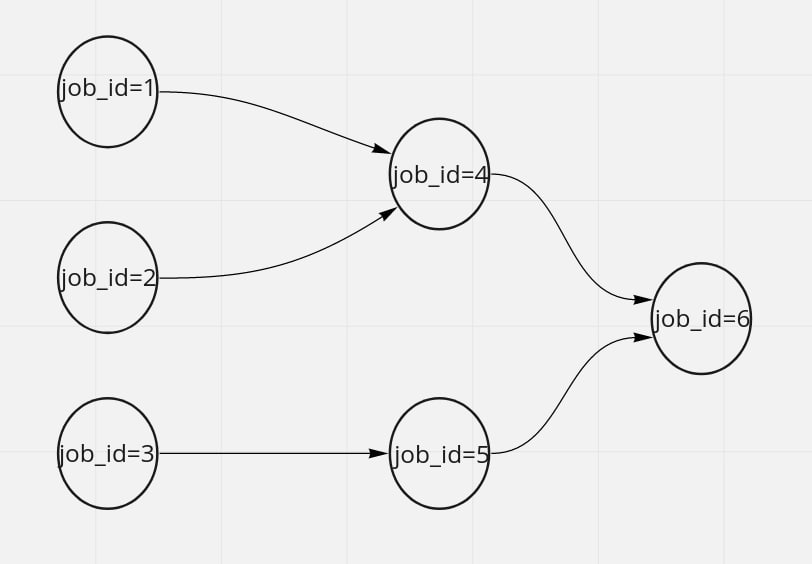
Дата: 12.04.2022

Оценка:

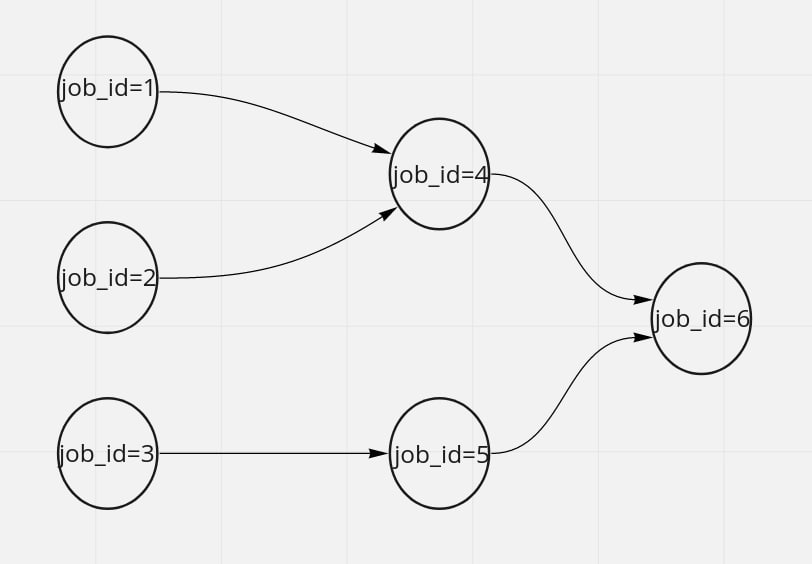
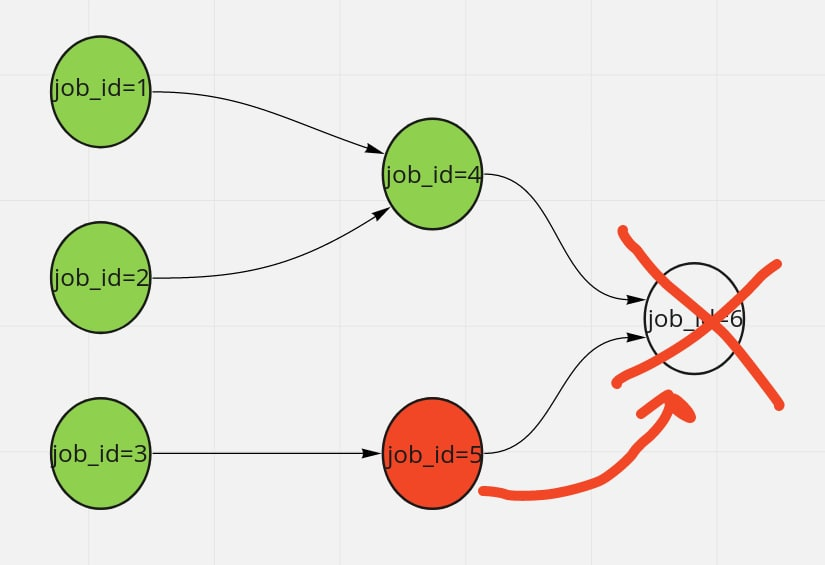
Подпись:

**Москва 2022г.**

**Вариант 6**

Создание планировщика DAG\*’a «джобов» (jobs)\*\*

На языке C\C++ написать программу, которая:

1. По конфигурационному файлу в формате yaml, json или ini принимает спроектированный DAG джобов и проверяет на корректность: отсутствие циклов, наличие только одной компоненты связанности, наличие стартовых и завершающих джоб. Структура описания джоб и их связей произвольная.  
   
2. При завершении джобы с ошибкой, необходимо прервать выполнение всего DAG’а и всех запущенных джоб.   
   

**Реализация**

**main.cpp**

#include <algorithm>

#include <cstdlib>

#include <iostream>

#include <map>

#include <set>

#include <stdexcept>

#include <vector>

#include "json.hpp"

using json = nlohmann::json;

using namespace std;

class OrdinalError : public runtime\_error

{

public:

OrdinalError(const string& what) : runtime\_error(what) {}

};

struct Job {

string id;

string command;

vector<string> ids\_prev;

vector<string> ids\_next;

};

// https://en.wikipedia.org/wiki/Topological\_sorting#Depth-first\_search

class Toposorter {

const map<string, Job>& id2job;

bool is\_used;

set<string> marks\_permanent;

set<string> marks\_temporary;

vector<string> ids\_ordered;

public:

Toposorter(const map<string, Job>& id2job) : id2job(id2job), is\_used(false)

{}

void visit(const string& id) {

if (marks\_permanent.find(id) != marks\_permanent.end()) {

return;

}

if (marks\_temporary.find(id) != marks\_temporary.end()) {

throw OrdinalError("Not a DAG.");

}

marks\_temporary.insert(id);

const Job& job = id2job.find(id)->second;

for (const string& id\_next : job.ids\_next) {

visit(id\_next);

}

marks\_temporary.erase(id);

marks\_permanent.insert(id);

ids\_ordered.push\_back(id);

}

vector<string> run()

{

assert(! is\_used);

is\_used = true;

// Обходим в обратном порядке из-за инверсии дальше (чтобы в итоге

// сортировка была стабильной).

for (auto it = id2job.rbegin(); it != id2job.rend(); it++) {

const string& id = it->first;

if (marks\_permanent.find(id) == marks\_permanent.end()) {

visit(id);

}

}

reverse(ids\_ordered.begin(), ids\_ordered.end());

return ids\_ordered;

}

};

class Componenter {

const map<string, Job>& id2job;

bool is\_used;

set<string> ids\_visited;

public:

Componenter(const map<string, Job>& id2job) : id2job(id2job), is\_used(false)

{}

void visit(const string& id) {

if (ids\_visited.find(id) != ids\_visited.end()) {

return;

}

ids\_visited.insert(id);

const Job& job = id2job.find(id)->second;

for (const string& id\_next : job.ids\_next) {

visit(id\_next);

}

for (const string& id\_prev : job.ids\_prev) {

visit(id\_prev);

}

}

void run()

{

assert(! is\_used);

is\_used = true;

assert(! id2job.empty());

const string& id\_first = id2job.begin()->first;

visit(id\_first);

if (ids\_visited.size() < id2job.size()) {

throw OrdinalError("More than one component.");

}

}

};

vector<string> preprocess\_dag(const map<string, Job>& id2job) {

if (id2job.empty()) {

throw OrdinalError("DAG is empty.");

}

vector<string> ids\_ordered = Toposorter(id2job).run();

Componenter(id2job).run();

return ids\_ordered;

}

map<string, Job> read\_dag()

{

json json\_input;

try {

cin >> json\_input;

} catch (const nlohmann::detail::parse\_error& exc) {

throw OrdinalError(exc.what());

}

if (! json\_input.is\_object()) {

throw OrdinalError("JSON input must be an object.");

}

map<string, Job> id2job;

for (const auto& item : json\_input.items()) {

const string& id = item.key();

try {

id2job.emplace(

id,

Job{id, item.value()["command"], item.value()["deps"]}

);

} catch (const nlohmann::detail::type\_error& exc) {

throw OrdinalError("Job " + id + ": " + exc.what());

}

}

for (auto& pair\_id\_job : id2job) {

Job& job = pair\_id\_job.second;

for (const string& id\_prev : job.ids\_prev) {

const auto& it\_id\_job = id2job.find(id\_prev);

if (it\_id\_job == id2job.end()) {

throw OrdinalError(

"Job " + job.id + ": dependency not found: " + id\_prev

);

}

Job& job\_prev = it\_id\_job->second;

job\_prev.ids\_next.push\_back(job.id);

}

}

return id2job;

}

void execute\_dag(

const map<string, Job>& id2job, const vector<string>& ids\_ordered

) {

for (const string& id : ids\_ordered) {

const auto& it\_id\_job = id2job.find(id);

assert(it\_id\_job != id2job.end());

const Job& job = it\_id\_job->second;

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

cerr << "Job " << id << ": " << job.command << ": ";

bool is\_ok = false;

if (status == -1) {

cerr << "system call failed";

} else {

int exitcode = WEXITSTATUS(status);

cerr << "exit " << exitcode;

is\_ok = exitcode == 0;

}

cerr << '\n';

if (! is\_ok) {

throw OrdinalError("A job failed.");

}

}

}

int main()

{

int stage;

try {

stage = 1;

map<string, Job> id2job = read\_dag();

stage = 2;

vector<string> ids\_ordered = preprocess\_dag(id2job);

stage = 3;

execute\_dag(id2job, ids\_ordered);

} catch (const OrdinalError& exc) {

cerr << exc.what() << '\n';

return stage;

}

}

**Пример работы**

sophie@sophie-VirtualBox:~/os/k$ make

cat task-plus-2-5.json

{

"1": {

"command": "echo one >1",

"deps": []

},

"2": {

"command": "echo two >2",

"deps": []

},

"3": {

"command": "echo three >3",

"deps": []

},

"4": {

"command": "cat 1 2 >4",

"deps": ["1", "2"]

},

"5": {

"command": "cat 2 3 >5",

"deps": ["2", "3"]

},

"6": {

"command": "cat 4 5 >6",

"deps": ["4", "5"]

}

}

( ./main <task-plus-2-5.json 2>&1 || echo "[exit $?]" ) | tee task-plus-2-5.log

Job 1: echo one >1: exit 0

Job 2: echo two >2: exit 0

Job 3: echo three >3: exit 0

Job 4: cat 1 2 >4: exit 0

Job 5: cat 2 3 >5: exit 0

Job 6: cat 4 5 >6: exit 0

cat bad-key.json

{

"1": {

"kommand": "echo one >1",

"deps": []

},

"2": {

"command": "cat 1 >2",

"deps": ["1"]

}

}

( ./main <bad-key.json 2>&1 || echo "[exit $?]" ) | tee bad-key.log

Job 1: [json.exception.type\_error.302] type must be string, but is null

[exit 1]

cat bad-job-command.json

{

"1": {

"command": "echo one >1",

"deps": []

},

"2": {

"command": "echo two >2",

"deps": []

},

"3": {

"command": "echo three >3",

"deps": []

},

"4": {

"command": "cat 1 2 >4",

"deps": ["1", "2"]

},

"5": {

"command": "false",

"deps": ["3"]

},

"6": {

"command": "cat 4 5 >6",

"deps": ["4", "5"]

}

}

( ./main <bad-job-command.json 2>&1 || echo "[exit $?]" ) | tee bad-job-command.log

Job 1: echo one >1: exit 0

Job 2: echo two >2: exit 0

Job 3: echo three >3: exit 0

Job 4: cat 1 2 >4: exit 0

Job 5: false: exit 1

A job failed.

[exit 3]

cat bad-json.json

{

( ./main <bad-json.json 2>&1 || echo "[exit $?]" ) | tee bad-json.log

[json.exception.parse\_error.101] parse error at line 2, column 1: syntax error while parsing object key - unexpected end of input; expected string literal

[exit 1]

cat bad-dep.json

{

"1": {

"command": "echo one >1",

"deps": []

},

"2": {

"command": "cat 1 >2",

"deps": ["3"]

}

}

( ./main <bad-dep.json 2>&1 || echo "[exit $?]" ) | tee bad-dep.log

Job 2: dependency not found: 3

[exit 1]

cat single.json

{

"1": {

"command": "echo one >1",

"deps": []

}

}

( ./main <single.json 2>&1 || echo "[exit $?]" ) | tee single.log

Job 1: echo one >1: exit 0

cat simple-1-2.json

{

"1": {

"command": "echo one >1",

"deps": []

},

"2": {

"command": "cat 1 >2",

"deps": ["1"]

}

}

( ./main <simple-1-2.json 2>&1 || echo "[exit $?]" ) | tee simple-1-2.log

Job 1: echo one >1: exit 0

Job 2: cat 1 >2: exit 0

cat task.json

{

"1": {

"command": "echo one >1",

"deps": []

},

"3": {

"command": "echo three >3",

"deps": []

},

"4": {

"command": "cat 1 2 >4",

"deps": ["1", "2"]

},

"5": {

"command": "cat 3 >5",

"deps": ["3"]

},

"6": {

"command": "cat 4 5 >6",

"deps": ["4", "5"]

},

"2": {

"command": "echo two >2",

"deps": []

}

}

( ./main <task.json 2>&1 || echo "[exit $?]" ) | tee task.log

Job 1: echo one >1: exit 0

Job 2: echo two >2: exit 0

Job 3: echo three >3: exit 0

Job 4: cat 1 2 >4: exit 0

Job 5: cat 3 >5: exit 0

Job 6: cat 4 5 >6: exit 0

cat empty.json

{}

( ./main <empty.json 2>&1 || echo "[exit $?]" ) | tee empty.log

DAG is empty.

[exit 2]

cat bad-json-contents.json

"string"

( ./main <bad-json-contents.json 2>&1 || echo "[exit $?]" ) | tee bad-json-contents.log

JSON input must be an object.

[exit 1]

cat bad-components.json

{

"1": {

"command": "echo one >1",

"deps": []

},

"2": {

"command": "cat 1 >2",

"deps": ["1"]

},

"3": {

"command": "echo three >3",

"deps": []

}

}

( ./main <bad-components.json 2>&1 || echo "[exit $?]" ) | tee bad-components.log

More than one component.

[exit 2]

cat cycle.json

{

"1": {

"command": "cat 2 >1",

"deps": ["2"]

},

"2": {

"command": "cat 1 >2",

"deps": ["1"]

}

}

( ./main <cycle.json 2>&1 || echo "[exit $?]" ) | tee cycle.log

Not a DAG.

[exit 2]

sophie@sophie-VirtualBox:~/os/k$ cat bad-key.json

{

"1": {

"kommand": "echo one >1",

"deps": []

},

"2": {

"command": "cat 1 >2",

"deps": ["1"]

}

}

sophie@sophie-VirtualBox:~/os/k$ cat bad-key.log

Job 1: [json.exception.type\_error.302] type must be string, but is null

[exit 1]

**Вывод**

В ходе данного курсового проекта я научилась делать базовый планировщик job-ов, который проверен на корректность.