Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу

«Операционные системы»

СОЗДАНИЕ ПЛАНИРОВЩИКА «ПРОЦЕССОВ-ЗАДАЧ»

Студент: Слесарчук Василий Анатольевич
Группа: М8О–210Б–22
Вариант: 32
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023.

Постановка задачи

Цель курсового проекта

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

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Создание планировщика DAG’а «джобов» (jobs):
1. По конфигурационному файлу в формате ini принимает спроектированный DAG джобов и на корректность: отсутствие циклов, наличие одной компоненты связанности, наличие стартовых 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.
3. (на оценку 4) Джобы должны запускаться максимально параллельно. Должны быть ограниченны параметром – максимальным числом одновременно выполняемых джоб.
4. (на оценку 5) Реализовать для джобов один из примитивов синхронизации мьютекс\семафор\барьер. То есть в конфиге дать возможность определять имена семафоров (с их степенями)\мьютексов\барьеров и указывать их в определение джобов в конфиге. Джобы указанные с одним мьютексом могут выполняться только последовательно (в любом порядке допустимом в DAG). Джобы указанные с одним семафором могут выполнятся параллельно с максимальным числом параллельно выполняемых джоб равным степени семафору. Джобы указанные с одним барьером имеют следующие свойство – зависимые от них джобы начнут выполнятся не раньше того момента времени, когда выполнятся все джобы с указанным барьером.

Вариант 32: Yaml\ Semaphore

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить структуру и синтаксис yaml файла
2. Придумать реализацию графа
3. Организовать парсер для .yaml файла

4. Написать свой собственный DAG с соответствующими проверками графа.

Основные файлы программы

main.cpp

```
#include <iostream>
#include <vector>
#include <queue>
#include <unistd.h>
#include <yaml-cpp/yaml.h>
#include "jobs.hpp"
#include "jobs_validator.hpp"

using namespace std;
using graph = vector<vector<IJob*> >;

const string CONFIG = "config.yaml";

IJob* function_identifier(string fnc) {
   if(fnc == "GCD") {
      return new GCDJob;
   } else if (fnc == "FibNumberRemainderCount") {
      return new FibNumberRemainderCountJob;
   } else if (fnc == "HanoiTower") {
      return new HanoiTowersJob;
   } else if (fnc == "Sum") {
      return new SumJob;
   } else if(fnc == "DumbFib") {
      return new DumbFibJob;
   } else if(fnc == "SOmanyHorses") {
      return new VoidJob;
   } else {
      return nullptr;
   }
```

```
}

auto main(int argc, char** argv) -> int {

    YAML::Node config = YAML::LoadFile(CONFIG);
    const YAML::Node& jobs = config["Jobs"];



    // Config printing
    vector<vector<IJob*> > tmp_graph;
    vector<IJob*> Jobs;
    for (YAML::const_iterator it = jobs.begin(); it != jobs.end(); ++it) {
        const YAML::Node& job = *it;
        string cur_job_id = job["Job_id"].as<string>();
        cout << "Job_id: " << cur_job_id << endl;
        const int _id = stoi(cur_job_id);

        cout << "Vertices: ";
        vector<IJob*> cur_vertices;
        vector<string> cur_vertices_id = job["Vertices"].as<vector<string> >();
        for (size_t i = 0; i < cur_vertices_id.size(); ++i) {
            cout << cur_vertices_id[i] << ", ";
            IJob* vtx = new IJob(stoi(cur_vertices_id[i]) - 1);
            cur_vertices.push_back(vtx);
        } cout << endl;

        tmp_graph.push_back(cur_vertices);

        cout << "Job: " << job["Job"].as<string>() << endl;
        IJob* current_job = function_identifier(job["Job"].as<string>()); // Set job
function by its name
        if(!current_job) {
            cout << "Wrong job function name!" << endl;
            exit(-1);
```

```
        }
        current_job->id = _id - 1;
        Jobs.push_back(current_job);
    }

    for(size_t i = 0; i < tmp_graph.size(); ++i) {
        for (size_t j = 0; j < tmp_graph[i].size(); ++j) {
            tmp_graph[i][j] = Jobs[tmp_graph[i][j]->id];
        }
    }

    for (size_t i = 0; i < Jobs.size(); ++i) { // Set links to jobs
        if(tmp_graph[i].size() < 1) {
            continue;
        }
        Jobs[i]->next = tmp_graph[i][0];
    }

    // Graph validation
    vector<IJob*> start_jobs;
    vector<IJob*> finish_jobs;

    try {
        if(!only_connectivity_component(tmp_graph)) {
            throw std::logic_error("Graph' vertex hasn't the only one connectivity
component");
        }
        if((finish_jobs = finish_component(tmp_graph, Jobs)).size() <= 0) {
            throw std::logic_error("There is no finish component!");
        }
        if(!without_cycles(tmp_graph, Jobs, finish_jobs)) {
            throw std::logic_error("Graph has cycle");
        }
        if((start_jobs = start_component(tmp_graph, Jobs)).size() <= 0) {
            throw std::logic_error("There is no start component!");
```

```
        }
    }
    catch(const std::exception& e) {
        std::cerr << e.what() << '\n';
    }

    queue<IJob*> tasks;
    for (size_t i = 0; i < start_jobs.size(); ++i) {
        tasks.push(start_jobs[i]);
    }

    while(!tasks.empty()) {

        IJob* cur_job = tasks.front();
        tasks.pop();

        if(!cur_job) {
            cout << "Nullpointer in queue!" << endl;
            break;
        }
        if(cur_job->next) {
            tasks.push(cur_job->next);
        }
        if(cur_job->is_done) {
            continue;
        }
        cout << "Current job id: " << cur_job->id + 1 << endl;
        cur_job->do_job();
        sleep(1);
    }

    return 0;
}
```

jobs.cpp

```
#include <iostream>
#include "jobs.hpp"

using namespace std;

ll GCD(ll a, ll b) {
    if(a == 0 || b == 0) {
        return max(a, b);
    }
    return GCD(max(a, b) % min(a, b), min(a, b));
}

ll FibNumberRemainderCount(ll n) {

    if(n == 0) {
        return 0;
    }
    ll f_prev = 0;
    ll f_cur = 1;
    for (ll i = 0; i < n - 1; ++i) {
        ll tmp = (f_cur + f_prev) % 10;
        f_prev = f_cur % 10;
        f_cur = tmp % 10;
    }
    return f_cur;
}

void HanoiTowers(int n, char from_rod, char to_rod, char
aux_rod) {
    if (n == 0) {
        return;
    }
    HanoiTowers(n - 1, from_rod, aux_rod, to_rod);
```

```
    cout << "We shift the ring from " << from_rod << " rod to "
<< to_rod << " rod..." << endl;
    HanoiTowers(n - 1, aux_rod, to_rod, from_rod);
}

ll Sum(ll a, ll b) {
    return a + b;
}

ll DumbFib(ll n) {
    if(n <= 1) {
        return 1;
    }
    return DumbFib(n-1) + DumbFib(n-2);
}

void SOmanyHorses() {
    cout << "*VOID*" << endl;
    return;
}
```

jobs_validator.cpp

```
#include "jobs_validator.hpp"
#include "jobs.hpp"
#include <iostream>
#include <algorithm>

using namespace std;
using graph = vector<vector<IJob*> >;

bool DFS_cycle(IJob* u, IJob* p, const graph & g, vector<int> &
state, vector<IJob*> & path, vector<IJob*> & cycle) {

    if(u == nullptr || p == nullptr) {
```

```
        return false;
    }
    if(state[u->id] == 2) {
        return false;
    }

    path.push_back(u);
    state[u->id] = 1;
    for(auto v : g[u->id]) {
        if(p != v) {
            if(state[v->id] == 1) {
                cycle.push_back(v);
                int idx = path.size() - 1;

                while(path[idx] != v) {
                    cycle.push_back(path[idx]);
                    --idx;
                }
                reverse(cycle.begin(), cycle.end());
                return true;
            }
            if(DFS_cycle(v, u, g, state, path, cycle)) {
                return true;
            }
        }
    }
    state[u->id] = 2;
    path.pop_back();
    return false;
}

bool without_cycles(graph &g, vector<IJob*> Jobs, vector<IJob*>
&finish_jobs) {
```

```
    for (size_t j = 0; j < finish_jobs.size(); ++j) {
        for(size_t i = 0; i < g.size(); ++i) {

            vector<int> state(g.size(), 0);
            vector<IJob*> path;
            vector<IJob*> cycle;

            bool has_cycle = DFS_cycle(Jobs[i], finish_jobs[j],
g, state, path, cycle);
            if(has_cycle) {
                return false;
            }
        }
    }

    cout << "Graph is without cycles" << endl;
    return true;
}

bool only_connectivity_component(graph& g) {
    for (size_t i = 0; i < g.size(); ++i) {
        if(g[i].size() > 1) {
            return false;
        }
    }
    cout << "Graph has only connectivity components!" << endl;
    return true;
}

vector<IJob*> start_component(graph &g, vector<IJob*> Jobs) {

    // Graph reverse
    graph tmp(g.size());
    for (size_t i = 0; i < g.size(); ++i) {
```

```
        for (size_t j = 0; j < g[i].size(); ++j) {
            tmp[g[i][j]->id].push_back(new IJob(i));
        } cout << endl;
    }

    // Finish components for reversed graph
    vector<IJob*> fcs = finish_component(tmp, Jobs);
    return fcs;
}


vector<IJob*> finish_component(graph &g, vector<IJob*> Jobs) {
    vector<IJob*> ans;
    for (size_t i = 0; i < g.size(); ++i)  {
        if(g[i].size() == 0) {
            ans.push_back(Jobs[i]);
        }
    }
    return ans;
}
```

Пример работы

```
Job_id: 1
Vertices: 4,
Job: GCD
Job_id: 2
Vertices: 4,
Job: FibNumberRemainderCount
Job_id: 3
Vertices: 5,
Job: HanoiTower
Job_id: 4
Vertices: 6,
```

```
Job: Sum
Job_id: 5
Vertices: 6,
Job: DumbFib
Job_id: 6
Vertices: 7,
Job: SOmanyHorses
Job_id: 7
Vertices: 8,
Job: Sum
Job_id: 8
Vertices:
Job: HanoiTower
Job_id: 9
Vertices: 8,
Job: GCD
Job_id: 10
Vertices: 8,
Job: FibNumberRemainderCount
Graph has only connectivity components!
Graph is without cycles


Current job id: 1
GCD(1245324, 74152356) result: 1884
Current job id: 2
FibNumberRemainderCount(100000) result: 5
Current job id: 3
HanoiTowers(5, 1, 3, 2) result:
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
```

```
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 3 rod to 1 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 3 rod to 1 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
Current job id: 9
GCD(1245324, 74152356) result: 1884
Current job id: 10
FibNumberRemainderCount(100000) result: 5
```

```
Current job id: 4
Sum(51232425, 81241671) result: 132474096
Current job id: 5
DumbFib(30) result: 1346269
Current job id: 8
HanoiTowers(5, 1, 3, 2) result:
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 3 rod to 1 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 3 rod to 2 rod...
We shift the ring from 3 rod to 1 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 1 rod to 2 rod...
```

```
We shift the ring from 3 rod to 2 rod...
We shift the ring from 1 rod to 3 rod...
We shift the ring from 2 rod to 1 rod...
We shift the ring from 2 rod to 3 rod...
We shift the ring from 1 rod to 3 rod...
Current job id: 6
*VOID*
Current job id: 7
Sum(51232425, 81241671) result: 132474096
```

Вывод

В ходе курсовой работы я вспомнил и реализовал алгоритмы проверки графа на компоненты связанности и на циклы, а так же разработал программу реализующую парсер yaml файла с последующим преобразованием в граф. Этот курсовой проект собрал в себе все накопленные знания за курс «Операционных систем» и прошлого курса «алгоритмы и структуры данных» поэтому он является показательным результатом моих трудов.