Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу

«Операционные системы»

Студент: Кудрявов Леонид Вадимович
Группа: М8О-208Б-22
Вариант: 21

Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

Репозиторий

https://github.com/Marsha2022/OS.git

Постановка задачи

На языке C\C++ написать программу, которая по конфигурационному файлу в формате json принимает спроектированный DAG джобов и проверяет на корректность: отсутствие циклов, наличие только одной компоненты связанности, наличие стартовых и завершающих джоб.

Общие сведения о программе

Программа написана на C++ на операционную систему Linux. Для парсинга JSON файлов была использована библиотека nlohmann/json (https://github.com/nlohmann/json) .

Общий метод и алгоритм решения

Программа анализирует файл конфигурации и на его основе создает DAG джобов, осуществляет проверку его корректности и, при соблюдении требований, начинает его выполнение. В любой момент времени программа хранит таблицу, в которой для каждого джоба указано, сколько еще не выполненных джобов зависит от него. После выполнения каждого джоба таблица обновляется, и если видим, что для некоторой задачи счетчик достиг нуля, он перемещается в очередь на выполнение.

Исходный код

```
========================== job_dag.hpp ==========================

#pragma once

#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <fstream>

#include "nlohmann/json.hpp"
```

```
namespace cp {

class TDagJobExecutor;

struct TJob {
    std::string name, path;
};

class TJobDag {
private:
    using TMapStringToStrings = std::map<std::string,
std::vector<std::string>>;

    std::map<std::string, TJob> jobs;

    TMapStringToStrings dep;


    TMapStringToStrings rdep;

    static bool Dfs(const std::string &v,
                    std::map<std::string, int>& visited,
                    TMapStringToStrings& dep);

    static bool CheckCorrectness(TJobDag &dag);

    static TMapStringToStrings Inverse(TMapStringToStrings &map);

public:

    friend class TDagJobExecutor;

    TJobDag() = default;
```

```
    TJobDag(const std::vector<TJob>& jobs, const
std::vector<std::pair<std::string, std::string>>& deps);

};

/*

    Example of CORRECT json file:

{
    "path_to_bins": "/path/to/bin/",
    "jobs": [
        {
            "name": "job1",
            "path": "bin/job1"
        },
        {
            "name": "job2",
            "path": "bin/job2"
        }
    ]
    "dependencies": [
        {
            "required": "job1",
            "target": "job2"
        }
    ]

}

*/
class JSONParser {
public:
```

```
    static TJobDag Parse(const std::string &pathToFile);

};

} // namespace cp
========================== job_exec.hpp ==========================

#pragma once

#include <iostream>
#include <optional>

#include <unistd.h>
#include <sys/wait.h>

#include "job_dag.hpp"

namespace cp {

class TSystem {
public:
    static int Exec(const std::string& path);
};

struct LogStack {
    // Stack of completed jobs
    std::vector<std::string> completed;
    size_t wasRead = 0;

    void Push(const std::string &str);

};

class TBasicExecutor {
```

```
private:
    LogStack * log;

public:

    void Execute(const std::string &name, const std::string &path,
LogStack *log);
    TBasicExecutor(LogStack *_log) : log(_log) { }

};

class TDagJobExecutor {
private:

    size_t target, current;

    std::set<std::string> actuallyReadyToBeExecuted;

    LogStack log;
    TBasicExecutor ex;

public:

    TDagJobExecutor() : ex(&log) { }

    bool Execute(TJobDag &dag);

};

}
========================== job_dag.cpp ==========================

#include "job_dag.hpp"
```

```
namespace cp {

bool TJobDag::Dfs(const std::string &v,
                  std::map<std::string, int>& visited,
                  TMapStringToStrings& dep) {
    visited[v] = 1;
    for (const auto& to : dep[v]) {
        if (visited[to] == 1) {
            return true;
        } else if (visited[to] == 0) {
            bool result = Dfs(to, visited, dep);
            if (result) {
                return result;
            }
        }
    }
    visited[v] = 2;
    return false;
}

bool TJobDag::CheckCorrectness(TJobDag &dag) {
    for (const auto& p : dag.jobs) {
        const auto& key = p.first;
        const auto& job = p.second;
        if (key != job.name) {
            return false;
        }
    }
    for (const auto& p : dag.dep) {
        if (dag.jobs.find(p.first) == dag.jobs.end()) {
            return false;
        }
        for (const auto& i : p.second) {
            if (dag.jobs.find(i) == dag.jobs.end()) {
```

```
                return false;
            }
        }
    }
    std::map<std::string, std::vector<std::string>> dep = dag.dep;
    std::map<std::string, int> visited;
    for (const auto& p : dep) {
        visited[p.first] = 0;
    }
    for (const auto& p : dep) {
        if (visited[p.first] == 0) {
            if (Dfs(p.first, visited, dep)) {
                return false;
            }
        }
    }
    return true;
}

TJobDag::TMapStringToStrings TJobDag::Inverse(TMapStringToStrings
&map) {
    TMapStringToStrings result;
    for (const auto& p : map) {
        for (const auto& target : p.second) {
            result[target].push_back(p.first);
        }
    }
    return result;
}

TJobDag::TJobDag(const std::vector<TJob>& jobs, const
std::vector<std::pair<std::string, std::string>>& deps) {
    for (const auto& i : jobs) {
        this->jobs[i.name] = i;
```

```
    }
    for (const auto& p : deps) {
        dep[p.second].push_back(p.first);
    }
    if (!CheckCorrectness(*this)) {
        throw std::logic_error("Bad DAG");
    }
    rdep = Inverse(dep);
}

TJobDag JSONParser::Parse(const std::string &pathToFile) {
        std::ifstream f(pathToFile);
        nlohmann::json jsn = nlohmann::json::parse(f);

        std::string path_to_bins = jsn["path_to_bins"];

        std::vector<TJob> jobs;
        for (const auto& job : jsn["jobs"]) {
            std::string path = path_to_bins +
std::string(job["path"]);
            jobs.push_back({job["name"], path});
        }

        std::vector<std::pair<std::string, std::string> > deps;
        for (const auto& dep : jsn["dependencies"]) {
            deps.push_back({dep["required"], dep["target"]});
        }

        return TJobDag(jobs, deps);
    }

}
========================== job_exec.cpp ==========================
```

```
#include "job_exec.hpp"

#include <atomic>

namespace cp {

int TSystem::Exec(const std::string& path) {
    int pid = fork();
    if (pid == 0) {
        if (execl(path.c_str(), path.c_str(), nullptr) == -1) {
            std::cout << "Can't exec: " << path <<'\n';
        }
    } else if (pid == -1) {
        throw std::logic_error("Can't fork");
    } else {
        int status;
        waitpid(pid, &status, 0);
        return status;
    }
    return 0;
}

void LogStack::Push(const std::string &str) {
    completed.push_back(str);
}

void TBasicExecutor::Execute(const std::string &name, const
std::string &path, LogStack *log) {
    try {
        int status = TSystem::Exec(path);
        if (status != 0) {
            exit(EXIT_FAILURE);
        }
    } catch (...) {
```

```
        exit(EXIT_FAILURE);
    }

    log->Push(name); // Warn about task completed
}

bool TDagJobExecutor::Execute(TJobDag &dag) {
    target = dag.jobs.size();
    current = 0;
    if (target == 0) {
        return true;
    }

    std::set<std::string_view> executionQueue;
    for (const auto& p : dag.jobs) {
        executionQueue.insert(p.first);
    }
    std::map<std::string_view, int> countOfDeps;
    for (const auto& p : dag.dep) {
        countOfDeps[p.first] = p.second.size();
    }

    // First layer
    for (const auto& p : dag.jobs) {
        if (countOfDeps[p.first] == 0) {
            actuallyReadyToBeExecuted.insert(p.first);
            executionQueue.erase(p.first);
        }
    }
    while (true) {
        if (current == target) {
            return true;
        } else {
            {
```

```
                std::vector<std::string> toErase;
                for (const auto& job : actuallyReadyToBeExecuted) {
                    toErase.push_back(job);
                    ex.Execute(job, dag.jobs[job].path, &log);

                }
                for (const auto& job : toErase) {
                    actuallyReadyToBeExecuted.erase(job);
                }
            }
            std::vector<std::string> completed;
            {
                for (size_t i = log.wasRead; i < log.completed.size();
i++) {
                    completed.push_back(log.completed[i]);
                    current++;
                }
                log.wasRead = log.completed.size();
            }
            for (const auto& job : completed) {
                for (const auto& depend : dag.rdep[job]) {
                    countOfDeps[depend]--;
                    if (countOfDeps[depend] == 0) {
                        actuallyReadyToBeExecuted.insert(depend);
                        executionQueue.erase(depend);
                    }
                }
            }

        }
    }
}
```

```
}
========================== main.cpp ==========================

#include <iostream>
#include <fstream>

#include "nlohmann/json.hpp"

#include "job_dag.hpp"
#include "job_exec.hpp"

using json = nlohmann::json;
using namespace cp;

int main(int argc, char ** argv) {
    if (argc < 1) {
        std::cerr << "Missing arguments : path to config file\n";
        exit(EXIT_FAILURE);
    }

    std::string pathToConfig(argv[1]);
    TJobDag dag = JSONParser::Parse(pathToConfig);
    TDagJobExecutor executor;
    executor.Execute(dag);

    std::cout << "Execution finished!\n";
}
```

Демонстрация работы программы

marhall@marshall:~/Desktop/OS_labs/build/cp$ ./cp_mai

n /Desktop/marshall/prog/OS_labs/data/cp/ex1/ex1.json

Started doing job1

Finished doing job1

Started doing job2

Finished doing job2

Started doing job3

Finished doing job3

Execution finished!

Выводы

В ходе выполнения курсового проекта мною были изучены инструменты парсинга конфигурационных файлов в формате “JSON”, также я закрепил знания в области работы с процессами.