Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовая работа по курсу
«Операционные системы»

Студент: Рокотянский А.Е.
Группа: М8О-201Б-21
Вариант: 36
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Sly-al/OS-labs

Постановка задачи

Цель работы

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.

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. При завершении джобы с ошибкой, необходимо прервать выполнение всего DAG'а и всех запущенных джоб.

Общие сведения о программе

Основной код программы представлен в файле plan.cpp . Программа использует заголовочные файлы graph.h и parser.h. Используются следующие системные вызовы:

1. fgets() – чтение символов из потока и сохранение их в строку
2. popen() – открытие канала процесса
3. pclose() – закрытие канала процесса

Помимо этого, используются библиотечные вызовы для работы с потоками и мьютексами.

Общий метод и алгоритм решения

Весь DAG хранится в векторе. Каждый элемент вектора – это узел(структура), в котором хранятся id джоба, команда для выполнения, результат её выполнения и два вектора: один содержит id детей этого джоба, другой – id его родителей. Парсинг выполняется при помощи функции JsonToVector (заголовочный файл parser.h), далее выполняются проверка полученного DAGa на наличие циклов, одной компоненты сильной связности. Проверка выполняется при помощи алгоритма DFS (поиск в глубину) и его модификаций (заголовочный файл graph.h). Если DAG проходит проверки, то мы переходим к его выполнению.

Планирование выполнения DAGa осуществляется при помощи вектора planning. В котором каждый индекс соответствует id джоба. Вектор принимает только 4 значения:

- 0 – джоб не готов к выполнению
- 1 – джоб готов к выполнению
- 2 – джоб выполняется
- 3 – джоб выполнен

Изначально он заполнен нулями. Мы проходимся по нему и всем стартовым джобам ставим 1 в планирование. Потом начинается цикл, который работает до тех пор, пока у финальной

джобы в планировании не станет цифра 3 (значит он выполнен), после чего выведем все выполненные в терминале команды.

Код в циклы выполнен следующим образом каждый раз мы считаем, сколько сейчас выполняется джобов, потом высчитываем разнось с допустимым количеством одновременно выполняемых джобов, именно по этой разнице запускается такое же количество потоков, в каждый из который закидывается джоб, готовый к выполнению

Каждый поток помимо того, что выполняет данный джоб, ещё и производит проверку всех «собратьев» на завершённость, если все они выполнены, то он ставит этому потоку в планирование цифру 1 (готов к выполнению), после чего завершается сам.

Исходный код

graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#include <vector>
#include <iostream>
#include <algorithm>
#include <set>
#include "parser.h"

const int maxn = 1e5;
std::vector<int> component(maxn);
int n;
std::vector<char> color;
std::vector<int> parent;
int cycle_start, cycle_end;

void DFS(int v, int num, const graph& matr) {
    component[v] = num;
    for (int u : matr[v]){
        if (!component[u]){
            DFS(u, num, matr);
        }
    }
    return;
}

bool CheckOneComp(const graph& matr){
    int num = 0;
    for (unsigned long v = 0; v < matr.size(); v++){
        if (!component[v]){
            DFS(v, ++num, matr);
        }
    }
    return num == 1;
}

bool dfs(int v, const std::vector<Node>&matr) {
    color[v] = 1;
    for (int u : matr[v].childId) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u, matr))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
```

```
    color[v] = 2;
    return false;
}

bool FindCycle(const std::vector<Node>&matr) {
    int n = matr.size();
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v, matr))
            break;
    }

    if (cycle_start == -1) {
        return false;
    }
    return true;
}

#endif
```

parser.h

```
#ifndef PARSER_H
#define PARSER_H

#include <iostream>
#include <vector>
#include <jsoncpp/json/json.h>
#include <fstream>
#include <sstream>

using graph = std::vector<std::vector<int> >;
const int NOTREADY = 0;
const int READY = 1;
const int INWORKING = 2;
const int FINISH = 3;

struct Node {
    int id;
    std::vector<int> childId;
    std::string comToExec;
    std::string arguments;
    std::vector<int> parentId;
};

void JsonToVector(const std::string& nameFileJson, std::vector<Node> & vectorOfNodes, std::vector<int>&
planning){
    std::string pathToFile = "/home/alex/Рабочий стол/OS-labs/KP/json/" + nameFileJson +".json";
    std::ifstream file(pathToFile);
    Json::Value actualJson;
    Json::Reader reader;

    reader.parse(file, actualJson);

    vectorOfNodes.resize(actualJson.size());
    planning.resize(actualJson.size());

    for (unsigned int j = 0; j < actualJson.size(); ++j){
        vectorOfNodes[j].id = j;
        std::string childIdString = actualJson["Node"+ std::to_string(j)]["childId"].asString();
        std::stringstream iss( childIdString );
        int number;
        while ( iss >> number){
```

```
            if(number != -1){
                vectorOfNodes[number].parentId.push_back(j);
                vectorOfNodes[j].childId.push_back(number);
            }
        }

        vectorOfNodes[j].comToExec = actualJson["Node"+ std::to_string(j)]["comToExec"].asString();
    }

    return;

}

#endif
```

plan.cpp

```
#include <pthread.h>
#include "parser.h"
#include "graph.h"

pthread_mutex_t mutex;

std::vector<Node> vectorOfNodes;
std::vector<int> planning;

bool Stop = false;

void* ExecUtilits(void* args){

    int amountOfNodes = planning.size();
    std::array<char, 128> buffer;
    std::string result;
    long long er;

    int id = ((Node *)args)->id;
    std::vector<int> childId = ((Node *)args)->childId;
    std::string comToExec = ((Node *)args)->comToExec;

    auto pipe = popen(comToExec.c_str(), "r");
    if (!pipe) throw std::runtime_error("popen() failed!");
    while (!feof(pipe)) {
        if (fgets(buffer.data(), buffer.size(), pipe) != nullptr){
            result += buffer.data();
        }
    }

    int rc = pclose(pipe);

    if (rc != EXIT_SUCCESS) {
        er = pthread_mutex_lock(&mutex);
        if (er){
            return (void*)er;
        }
        Stop = true;
        er = pthread_mutex_unlock(&mutex);
        if (er){
            return (void*)er;
        }
        return NULL;
    }

    vectorOfNodes[id].arguments = result;

    if (id != amountOfNodes - 1){
        for(int k: childId){
            int amountOfParents = vectorOfNodes[k].parentId.size() - 1;
```

```
            int countFinish = 0;
            for (int j: vectorOfNodes[k].parentId){
                if (planning[j] == FINISH){
                    countFinish++;
                }

            }

            if (amountOfParents == countFinish){
                er = pthread_mutex_lock(&mutex);
                if (er){
                    return (void*)er;
                }
                planning[k] = READY;
                er = pthread_mutex_unlock(&mutex);
                if (er){
                    return (void*)er;
                }
            }

        }
    }

    planning[id] = FINISH;
    for(unsigned long i = 0; i < planning.size(); ++i){
            std::cout << planning[i] << ' ';
        }
    std::cout << '\n';

    return NULL;
}

int main(){
    int amountThreads;
    std::string fileJson;

    std::cout<< "Input amount of threads and file with DAG\n";
    std::cin >> amountThreads >> fileJson;

    JsonToVector(fileJson, vectorOfNodes, planning);

    int amountOfNodes = planning.size();
    graph matr(amountOfNodes);

    for(int i = 0; i < amountOfNodes; ++i){
        for (int j : vectorOfNodes[i].childId){
            matr[i].push_back(j);
            matr[j].push_back(i);
        }
    }

    if(!CheckOneComp(matr) || FindCycle(vectorOfNodes) ){
        std::cout<< "There are cycle or more tnan one componet\n";
        return (EXIT_FAILURE);
    }

    for (int i = 0; i < amountOfNodes; ++i){
        if(vectorOfNodes[i].parentId.empty()){
            planning[i] = READY;
        }
    }

    if (pthread_mutex_init(&mutex, NULL) != 0) {
        std::cout << "mutex init failed\n";
        return (EXIT_FAILURE);
    }

    while(planning[amountOfNodes - 1] != FINISH) {
```

```
        int countInWork = 0;
        for (int i = 0; i < amountOfNodes; ++i){
            if(planning[i] == INWORKING){
                countInWork++;
            }
        }
        int dostup = amountThreads - countInWork;
        for (int i = 0; (i < amountOfNodes) && dostup > 0; ++i){
            if(planning[i] == READY){
                dostup--;
                planning[i] = INWORKING;
                pthread_t thread;
                if(int err = pthread_create(&thread, NULL, ExecUtilits, (void *)&vectorOfNodes[i])){
                    std::cout << "Thread create error: " << err << '\n';
                    return (EXIT_FAILURE);
                }
                if(int err = pthread_detach(thread)){
                    std::cout << "Thread detach error: " << err << '\n';
                    return (EXIT_FAILURE);
                }
            }
        }

        if (Stop){
            std::cout << "Job crushed\n";
            return (EXIT_FAILURE);
        }
    }

    for (int i = 0; i < amountOfNodes; ++i){
        std::cout<< "Id = " << i << ' ' << "Result = " << vectorOfNodes[i].arguments <<'\n';
    }
    pthread_mutex_destroy(&mutex);
    return (EXIT_SUCCESS);
}
```

Демонстрация работы программы

```
alex@alex-VirtualBox:~/Рабочий стол/OS-labs/build/KP$ ./kp

Input amount of threads and file with DAG

1

test0

3Id =  03  Result = 3

Id = 1 Result =

Id = 2 Result = .:

CMakeFiles

cmake_install.cmake

CTestTestfile.cmake

h

kp

Makefile

newfile1

newfile2

trash
```

```
./trash:

Id = 3 Result =

Id = 4 Result = Ср 04 янв 2023 10:22:12 MSK

Id = 5 Result =      10      83     518

alex@alex-VirtualBox:~/Рабочий стол/OS-labs/build/KP$ ./kp

Input amount of threads and file with DAG

2

test0

2 0 3 0 2 0

2 0 3 0 3 0

3 2 3 0 3 0

mkdir: невозможно создать каталог «trash»: Файл существует

Job crushed
alex@alex-VirtualBox:~/Рабочий стол/OS-labs/build/KP$ ./kp

Input amount of threads and file with DAG

2

test2

There are cycle or more tnan one componet
```

Выводы

Составлена и отлажена программа на языке C++, реализующая обработку спроектированных DAG of jobs, джобы запускаются максимально параллельно, обрабатываются возможные ошибки.