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: Железнов Илья Васильевич

Группа: М8О–210Б–22

Вариант: 37

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’a «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 37: Ini \ Mutex

**Описание**

В качестве реализации моего курсового проекта было выбрано следующее задание: парсер .ini файла с генерацией Makefile.

Makefile — это текстовый файл, содержащий инструкции по компиляции и связыванию (или сборке) набора файлов исходного кода. Программа (часто называемая программой make) считывает файл makefile и вызывает компилятор, компоновщик и, возможно, другие программы для создания исполняемого файла.

Для написании своей утилиты Make, нужно было сначала разобраться как в целом работает Makefile и как правильно комплировать файлы в среде gcc/g++. Проанализировав его принципы работы и архитектуру внутри, было принято решение написать план к данному курсовуму проекту. Так как сам парсер мы пишем на .ini файле, а .ini – это файл конфигурации, который содержит данные настроек Microsoft Windows и некоторых приложений. Ну или по простому — это обычный текстовый файл, в котором есть некоторые правила оформления.   
Они чересчур просты: есть секции, которые отделяются квадратными скобками, есть параметры, которые задаются путем <parametr\_name = parametr> и есть массивы <arr1 = arr\_val\_1\_1>.

Разобравшись в базовых принципах работы .ini файла нужно реализовать свои правила работы с нашим “Makefile’ом”.

Вот их перечень:  
1. Любая исполняемая команда начинается с имени секции в квадратных закрывающихся скобках ( [b.o] );

2. Тело секции может начинаться с определением команды секции (Command) или обозначения зависимостей (Dep[]), но никак не с именем mutex’а (Mutex);

3. Любая команда, зависимость или имя mutex’а (соотвественно) свое описание должно начинать с двойных кавычек и заканчивать ими. (Command = “g++ main.cpp”);

4. Условная команда <Mutex> обязана всегда быть объявлена (если не хотите, чтобы это поле mutex попало под синхронизацию, просто задайте имя ’mutex\_null’);

5. Никаких других исключений не может быть написано, кроме как все правила с пункта 1-4. Если описание внутри зависимости или команды написано неверно, компилятором сам подскажет вам о ваших ошибках.

И так, ознакомившись с правилами можно приступить к реализации. (Ее детальное описанием см. **Общие сведения о программе**). Так как Makefile внутри себя держит структуру DAG (а именно, зависимости и команды) и является набором правил написания, то мы так же разделим нашу реализацию на два участка работы: написание парсера .ini файла и собственная структура DAG\_Jobs.

**Общие сведения о программе**

Программа состоит из двух основных интерфейсов (parserIni.cpp и dag.cpp). Компилируется наш „Makefile“ следующей командой: ‘g++ main.cpp src/parserIni.cpp src/dag.cpp -o main’. Далее задается аргумент при запуске по следующему принципу: « [ИСПОЛНЯЕМЫЙ\_ФАЙЛ] [ИМЯ\_МЮТЕКСА\_ДЛЯ\_БЛОКИРОВКИ]».

Теперь детальнее разберемся в работе наших функций.

**ParserIni:**

Изначально создается специальная структура *Job*, которая содержит в себе 4 поля: универсальное имя job’ы, команда, список зависимостей, имя мютекса. Затем создадим класс *ParserIni*. У нас есть 2 основные публичные функции для работы с парсером и 5 приватных, которые не нужны пользователю — нужны только для работ функций. Функция *returnResultParsing* полностью считывает файл по определенному алгоритму и заполняет наш список job’s. Функция *getParserRes* является простым геттером нашего поля *\_parserResult*, которое хранит в себе наш вектор job’s.

Теперь кратко разберем каждую из наших вспомогательных функций в приватном поле:

**1. chekingQuotes** — функция, которая проверяет на валидность написание команды в двойных ковычках.

**2. checkIsComandTag** — функция, которая проверяет является ли данная строчка в .ini файле объявлением команды.

**3. checkIsSquareBrackets** — функция, которая проверяет объявления секции, если секция не была объявлена, а в ее теле уже появились, например, команды или зависимости, то выведется ошибка

**4. checkIsDep** — аналогично, с функцией *checkIsComandTag* проверяет принадлежность данной строчки к типу списка зависимостей.

**5. checkIsMutx** – проверяет, является ли данная строчка mutex – именем.

Принцип работы данного модуля программы прост — построчно считываем файл по определенным правилам, записывая его содержимое, если в нем нет синтаксических ошибок, и получаем на выходе распаршенный .ini файл по job’ам.

**Dag:**

Как и в прошлый раз, нам потребуется создать новый тип данных UJob (unit\_job) – вершины нашего графа. В нем будет всего 2 поля — сама job’а и вектор зависимостей, то есть от какой вершины зависит наша вершина.

Сама реализация графа построена на списке зависимостей. Был выбран этот метод из-за своей емкости в нашем случае. Так как наш граф является ациклическим, нам просто не нужно хранить числа выше единицы, значит списки здесь получаются выгодными по памяти.

Для удобства работы был создан класс DAG\_J. Его функционал заключается в следующем: создать граф по заданному имени .ini файла, распечатать созданный граф и запустить выполнение нашего Makefile.

В публичном поле есть единственное поле значения — это *ParserIni \_jobs*. Оно нам нужно для того, чтобы создать по этому типу распаршенные значения.

И так, мы имеем 3 основные функции и 3 приватные, которые нужны лишь для внутреннего использования.

*Public:*

**1. createDag** – функция, которая создает наш граф по содержимому .ini файла;

**2. printDag** – функция, которая выводит на экран условное представление графа с его универсальным именем, mutex-именем и списком зависимостей (если они есть).

**3. runCompile** – данная функция является точкой финального запуска нашей программы. Она начинает всю работу нашего асинхронного запуска команд.

*Private:*

**1. hasCycle** – функция, возвращающая булево значение при наличи цикла в графе. Путем вспомогательной функции проходится по всем вершинам и записывает в стек — посещений информацию о том, что был в данной вершине. Если в вершине мы оказались больше одного раза возвращаем false

**2. hasCycleUtil** – вспомогательная функция для *hasCycle*, чтобы упростить визуальное оформление кода.

**3. executeJob** – функция, которая по индексу job’ы в списке вершин, начинает запуск нашей job’ы с использованием mutex’а, если его имя заданно при запуске.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы Makefile, структуру DAG, правила .ini файла
2. Придумать собственные правила для работы нашего собственного „Makefile“
3. Организовать парсер для .ini файла
4. Написать свой собственный DAG, разобраться в алгоритме обхода в глубину (проверка на наличие цикла в графе).
5. Написать финальную main программу, которая будет запускать наш „Makefile“

**Основные файлы программы**

**parserIni.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <map>

#include <fstream>

#include <sstream>

typedef struct \_job {

std::string name;

std::string command;

std::vector<std::string> dependencies;

std::string mutexName;

} Job;

class ParserIni

{

public:

ParserIni() = default;

~ParserIni() = default;

void returnResultParsing(std::string fileName);

std::vector<Job> getParserRes();

private:

void checkingQuotes(std::string line, std::string tag);

bool checkIsCommandTag(std::string line);

bool checkIsSquareBrackets(std::string line);

bool checkIsDep(std::string line);

bool checkIsMutx(std::string line);

private:

std::vector<Job> \_parseRersult;

};

**parserIni.cpp**

#include "./parserIni.h"

std::string cmd = "Command = ";

std::string dep = "Dep[] = ";

std::string mutx = "Mutex = ";

void ParserIni::returnResultParsing(std::string fileName)

{

std::ifstream file(fileName);

if (!(file.is\_open())) {

std::cerr << "Unable to open file: " << fileName << std::endl;

}

// Temporary variables for working with strings

std::string line;

bool sectionInit = false;

// Counters of position

size\_t countSectionName = 0;

size\_t countSectionCommand = 0;

size\_t countOfDep = 0;

size\_t startPos = 0;

size\_t length = 0;

while (std::getline(file, line)) {

// std::cout << line ;

if (checkIsSquareBrackets(line)) {

startPos = 1;

length = line.size() - 2;

std::string extractedName = line.substr(startPos, length);

\_parseRersult.push\_back({extractedName, {}, {}});

++countSectionName;

countOfDep = 0;

sectionInit = true;

continue;

} else if (checkIsCommandTag(line)) {

if (!sectionInit) {

std::perror("Error! Section is not inited");

exit(EXIT\_FAILURE);

}

checkingQuotes(line, cmd);

startPos = cmd.size() + 1;

length = line.size() - startPos - 1;

std::string extractedCommand = line.substr(startPos, length);

\_parseRersult[countSectionCommand].command = extractedCommand;

sectionInit = false;

} else if (checkIsDep(line)) {

checkingQuotes(line, dep);

startPos = dep.size() + 1;

length = line.size() - startPos - 1;

std::string extractedDep = line.substr(startPos, length);

\_parseRersult[countSectionCommand].dependencies.push\_back(extractedDep);

++countOfDep;

} else if (checkIsMutx(line)) {

checkingQuotes(line, mutx);

startPos = mutx.size() + 1;

length = line.size() - startPos - 1;

if (!(length > 0)) {

throw std::length\_error("Error! Mutex name is undefiend");

}

std::string extractedMutx = line.substr(startPos, length);

\_parseRersult[countSectionCommand].mutexName = extractedMutx;

++countSectionCommand;

} else if ((line.size() != 0)) {

std::perror("Error! Unknow command");

std::cout << "\n\t" << line << std::endl;

exit(EXIT\_FAILURE);

}

}

file.close();

}

std::vector<Job> ParserIni::getParserRes()

{

return this->\_parseRersult;

}

void ParserIni::checkingQuotes(std::string line, std::string tag)

{

if (line[tag.size()] != '"' || line[line.size() - 1] != '"' ) {

std::cout << "Error! Invalid input\n" << "\t" << line << std::endl;

exit(EXIT\_FAILURE);

}

}

bool ParserIni::checkIsCommandTag(std::string line)

{

for (size\_t i = 0; i < cmd.size(); ++i) {

if (line[i] != cmd[i]) {

return false;

}

}

return true;

}

bool ParserIni::checkIsSquareBrackets(std::string line)

{

if (line[0] == '[' || line[line.size() - 1] == ']') {

return true;

}

return false;

}

bool ParserIni::checkIsDep(std::string line)

{

for (size\_t i = 0; i < dep.size(); ++i) {

if (line[i] != dep[i]) {

return false;

}

}

return true;

}

bool ParserIni::checkIsMutx(std::string line)

{

for (size\_t i = 0; i < dep.size(); ++i) {

if (line[i] != mutx[i]) {

return false;

}

}

return true;

}

**dag.h**

#pragma once

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_set>

#include <algorithm>

#include <sys/wait.h>

#include <unistd.h>

#include <mutex>

#include <shared\_mutex>

#include "./parserIni.h"

typedef struct \_job\_unit {

Job j;

std::vector<int> dep;

} UJob;

class DAG\_J

{

public:

ParserIni \_jobs;

DAG\_J() = default;

~DAG\_J() = default;

void createDAG(std::string fileName);

void printDAG();

void runCompile(const std::string & mtxNameLock);

private:

bool hasCycle();

bool hasCycleUtil(int v, std::vector<bool>& visited, std::vector<bool>& recStack);

void executeJob(int jobIndex, std::string mtxNameLock);

private:

std::vector<UJob> \_graph;

std::unordered\_map<std::string, std::mutex> \_mutexes;

};

**dag.cpp**

#include "./dag.h"

std::mutex mtx;

void DAG\_J::createDAG(std::string fileName)

{

\_jobs.returnResultParsing(fileName);

std::vector<Job> vecJobs = \_jobs.getParserRes();

for (const auto& job : vecJobs) {

UJob ujob;

ujob.j = job;

\_graph.push\_back(ujob);

// Проверяем, есть ли имя мьютекса для данной задачи

if (!job.mutexName.empty()) {

// Если есть имя мьютекса, связываем его с соответствующей задачей

\_mutexes[job.mutexName]; // Создаем мьютекс для заданного имени (если его нет)

}

}

for (size\_t i = 0; i < \_graph.size(); ++i) {

for (const auto& dep : \_graph[i].j.dependencies) {

// Находим индекс задачи, от которой зависит текущая задача

auto it = std::find\_if(\_graph.begin(), \_graph.end(), [&dep](const UJob& uj) {

return uj.j.name == dep;

});

if (it != \_graph.end()) {

\_graph[i].dep.push\_back(std::distance(\_graph.begin(), it));

}

}

}

}

void DAG\_J::printDAG()

{

if (\_graph.size() == 0) {

std::perror("Error! graph is empty");

exit(EXIT\_FAILURE);

}

for (size\_t i = 0; i < \_graph.size(); ++i) {

if (\_graph[i].dep.size() == 0) {

std::cout << \_graph[i].j.name << " " << \_graph[i].j.mutexName << '\t';

} else {

std::cout << \_graph[i].j.name << " " << \_graph[i].j.mutexName << " : ";

for (size\_t j = 0; j < \_graph[i].dep.size(); ++j) {

std::cout << \_graph[i].dep[j] << " ";

}

std::cout << ";\t";

}

}

}

bool DAG\_J::hasCycle() {

std::vector<bool> visited(\_graph.size(), false);

std::vector<bool> recStack(\_graph.size(), false);

for (size\_t i = 0; i < \_graph.size(); ++i) {

if (!visited[i] && hasCycleUtil(i, visited, recStack)) {

std::cout << "В вашем графе найден цикл, пожалуйста, исправьте его!" << std::endl;

return true; // Найден цикл

}

}

return false; // Цикл не найден

}

bool DAG\_J::hasCycleUtil(int v, std::vector<bool>& visited, std::vector<bool>& recStack) {

if (!visited[v]) {

visited[v] = true;

recStack[v] = true;

for (const auto& adj : \_graph[v].dep) {

if (!visited[adj] && hasCycleUtil(adj, visited, recStack)) {

return true;

} else if (recStack[adj]) {

return true;

}

}

}

recStack[v] = false; // Убираем вершину из стека рекурсии

return false;

}

void DAG\_J::runCompile(const std::string & mtxNameLock) {

if (hasCycle()) {

exit(EXIT\_FAILURE);

}

std::queue<int> readyJobs;

std::unordered\_set<int> completedJobs;

// Находим все джобы без зависимостей (готовые к выполнению) и добавляем их в очередь

for (size\_t i = 0; i < \_graph.size(); ++i) {

if (\_graph[i].j.dependencies.empty()) {

readyJobs.push(i);

}

}

// Запускаем выполнение джоб в нескольких процессах

while (!readyJobs.empty()) {

std::vector<pid\_t> childPids;

// Ограничим максимальное количество процессов для запуска

int maxProcesses = 5; // Например, 5 процессов

int jobIndex;

while (!readyJobs.empty() && maxProcesses > 0) {

jobIndex = readyJobs.front();

readyJobs.pop();

pid\_t pid = fork();

if (pid < 0) {

std::cout << "Ошибка при создании процесса-потомка." << std::endl;

exit(EXIT\_FAILURE);

}

if (pid == 0) {

// child process

executeJob(jobIndex, mtxNameLock);

exit(EXIT\_SUCCESS);

} else {

// parent process

childPids.push\_back(pid);

--maxProcesses;

}

}

// Ожидаем завершение выполнения дочерних процессов

for (pid\_t pid : childPids) {

int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status)) {

int exitStatus = WEXITSTATUS(status);

if (exitStatus == EXIT\_SUCCESS) {

completedJobs.insert(jobIndex);

}

}

}

// Находим новые готовые джобы для выполнения

auto isJobCompleted = [&](const std::string& dep) {

auto it = std::find\_if(\_graph.begin(), \_graph.end(),

[&dep](const UJob& uj) {

return uj.j.name == dep;

});

return it != \_graph.end() && completedJobs.find(std::distance(\_graph.begin(), it)) != completedJobs.end();

};

for (size\_t i = 0; i < \_graph.size(); ++i) {

if (completedJobs.find(i) == completedJobs.end() &&

std::all\_of(\_graph[i].j.dependencies.begin(), \_graph[i].j.dependencies.end(), isJobCompleted)) {

readyJobs.push(i);

}

}

}

}

void DAG\_J::executeJob(int jobIndex, std::string mtxNameLock) {

std::string compileCommand = \_graph[jobIndex].j.command;

if (!compileCommand.empty()) {

std::cout << "Выполнение команды: " << compileCommand << std::endl;

std::string mutexName = \_graph[jobIndex].j.mutexName;

// Проверка наличия мьютекса в контейнере \_mutexes

if (!mutexName.empty() && \_mutexes.find(mutexName) != \_mutexes.end() && mutexName == mtxNameLock) {

// Блокировка соответствующего мьютекса

std::cout << "Mute: " << mutexName << std::endl;

std::lock\_guard<std::mutex> lock(\_mutexes[mutexName]);

}

// Выполняем команду в процессе-потомке

FILE\* pipe = popen(compileCommand.c\_str(), "r");

if (pipe) {

pclose(pipe);

} else {

std::cout << "Ошибка при выполнении команды компиляции." << std::endl;

exit(EXIT\_FAILURE);

}

} else {

std::cout << "Команда компиляции для джобы не найдена." << std::endl;

exit(EXIT\_FAILURE);

}

}

**Пример работы**

За основу тестов были взяты следующие .ini файлы: **[b.o]**

**Command = "g++ -c ./test/b.cpp"**

**Mutex = "mutex\_1"**

**[c.o]**

**Command = "g++ -c ./test/c.cpp"**

**Mutex = "mutex\_2"**

**[d.o]**

**Command = "g++ -c ./test/d.cpp"**

**Mutex = "mutex\_1"**

**[a.out]**

**Dep[] = "b.o"**

**Dep[] = "c.o"**

**Dep[] = "d.o"**

**Command = "g++ -o a.out b.o c.o d.o"**

**Mutex = "mutex\_3"**

**################################################**

**[b.o]**

**Command = "g++ -c ./test/b.cpp"**

**Mutex = "mutex\_1"**

**[c.o]**

**Command = "g++ -c ./test/c.cpp"**

**Mutex = "mutex\_1"**

**[d.o]**

**Command = "g++ -c ./test/d.cpp"**

**Mutex = "mutex\_1"**

**[e.o]**

**Dep[] = "b.o"**

**Dep[] = "c.o"**

**Command = "ld -relocatable c.o b.o -o e.o"**

**Mutex = "mutex\_2"**

**[f.o]**

**Dep[] = "d.o"**

**Command = "ld -relocatable d.o -o f.o"**

**Mutex = "mutex\_2"**

**[a.out]**

**Dep[] = "e.o"**

**Dep[] = "f.o"**

**Command = "g++ -o a.out b.o c.o d.o"**

**Mutex = "mutex\_3"**

**################################################**

**[list.o]**

**Command = "gcc -c ./test/list.c"**

**Mutex = "mutex\_1"**

**[main.o]**

**Command = "gcc -c ./test/main.c"**

**Mutex = "mutex\_1"**

**[a.out]**

**Dep[] = "list.o"**

**Dep[] = "main.o"**

**Command = "gcc main.o list.o -o main"**

**################################################**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ g++ main.cpp src/dag.cpp src/parserIni.cpp -o main**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./main mutex\_1**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: g++ -c ./test/d.cpp**

**Выполнение команды: g++ -c ./test/b.cpp**

**Mute: mutex\_1**

**Mute: mutex\_1**

**Выполнение команды: g++ -c ./test/b.cpp**

**Mute: mutex\_1**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: g++ -c ./test/b.cpp**

**Mute: mutex\_1**

**Выполнение команды: g++ -o a.out b.o c.o d.o**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./a.out**

**Hello, World**

**Hello, Man!**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ g++ main.cpp src/dag.cpp src/parserIni.cpp -o main**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./a.out**

**Hello, World**

**Hello, Man!**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./main mutex\_2**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: g++ -c ./test/d.cpp**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: ld -relocatable d.o -o f.o**

**Mute: mutex\_2**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: ld -relocatable c.o b.o -o e.o**

**Mute: mutex\_2**

**Выполнение команды: g++ -o a.out b.o c.o d.o**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./main**

**Error! You have entered few arguments**

**Example:**

**./main <MUTEX\_NAME\_TO\_LOCKED>**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./main mutex\_2**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: g++ -c ./test/d.cpp**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: ld -relocatable d.o -o f.o**

**Mute: mutex\_2**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: g++ -c ./test/c.cpp**

**Выполнение команды: g++ -c ./test/b.cpp**

**Выполнение команды: ld -relocatable c.o b.o -o e.o**

**Mute: mutex\_2**

**Выполнение команды: g++ -o a.out b.o c.o d.o**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./a.out**

**Hello, World**

**Hello, Man!**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ g++ main.cpp src/dag.cpp src/parserIni.cpp -o main**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./main mutex\_1**

**Выполнение команды: gcc -c ./test/list.c**

**Mute: mutex\_1**

**Выполнение команды: gcc -c ./test/main.c**

**Mute: mutex\_1**

**Выполнение команды: gcc -c ./test/list.c**

**Mute: mutex\_1**

**Выполнение команды: gcc main.o list.o -o main**

**keinpop@DESKTOP-T6SLHUS:/mnt/c/oc\_cp$ ./main**

**HELP BOARD**

**Command programms:**

**exit Exit to porgramm**

**help Call help board**

**pushf Push front elem**

**insind Insert elemnt by index**

**popind Pop elemnt by index**

**print Print all elemnts in list**

**dellist Delete all elemnent in list**

**vartask Searches for the entered element in the list, if the element is found - deletes the list**

**ENUM COLORS:**

**Red**

**Green**

**Blue**

**Yellow**

**Black**

**Enter command:**

**Вывод**

По итогу своей работы, хочу подвести итоги. Написания своего Makefile очень трудная и большая работа, которая как раз подходит для курсового проекта.

Придумать определенный стэк операторов и реализовав свой личный (не совсем идеальный) парсер очень большая и трудоемкая работа, которая учит программиста азам разработки.

Встретившись впервые с графами, а именно со структурой DAG – это очень тяжелая и кропотливая работа, которая не подвластна с первого раза, но научившись работать с ней, программист может обеспечить себя тем стэком знаний, который не каждый обладает.

Makefile на .ini файле получился хорошим упрощением для компиляции через командную строку. У него есть свои минусы, как и в пограничных тестах ввода в самом файле, так и некоторые нюансы оформления и рефакторинги структуры, но в целом это не влияет на работу самой программы.

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.