Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Курсовой проект по курсу

«Операционные системы»

Группа: М8О-209БВ-24

Студент: Попов П.А,

Преподаватель: Миронов Е.С. (ПМИ)

Оценка: _____

Дата: 24.12.25

Москва, 2025

Постановка задачи

Вариант 39.

1. 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.

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.

* DAG - Directed acyclic graph. Направленный ациклический граф.

** Джоб(Job) – процесс, который зависит от результата выполнения других процессов (если он не стартовый), которые исполняются до него в DAG, и который порождает данные от которых может быть зависят другие процессы, которые исполняются после него в DAG (если он не завершающий).

Вариант 39: Ini\Barrier

Общий метод и алгоритм решения.

Использованные системные вызовы:

- clone() – создание потока
- futex() – синхронизация
- execve() – выполнение команд
- wait4() – ожидание завершения
- open() – открытие файлов
- read() – чтение файлов
- pthread_create() – создать поток
- pthread_mutex_lock() – заблокировать мьютекс
- pthread_mutex_unlock() – разблокировать мьютекс
- sem_wait() – ждать семафор
- sem_post() – освободить семафор
- system() – выполнить команду
- fopen() – открыть файл

- fgets() – прочитать строку

Алгоритм работы программы:

1. Загружается INI-файл с задачами (job'ами) и барьерами, строится DAG (ориентированный ациклический граф).
2. Проверяется DAG на корректность:
    - Отсутствие циклов
    - Связность (одна компонента связности)
    - Наличие стартовых и завершающих задач
3. Запускается планировщик:
    - Создаются потоки для каждой задачи (`pthread_create`)
    - Количество одновременно выполняемых задач ограничено параметром `max_parallel`
4. Задача выполняется:
    - Поток ждет, пока выполнены все зависимости задачи
    - Выполняет команду задачи через `system()`
    - Если задача имеет барьер — приходит к нему и ждет
5. Барьерная синхронизация:
    - Когда все задачи с одинаковым барьером завершились
    - Барьер разблокируется
    - Все задачи барьера продолжают выполнение одновременно
6. Обработка ошибок:
    - Если любая задача завершилась с ошибкой
    - Останавливается выполнение всего DAG
    - Оставшиеся задачи не запускаются
7. Завершение:
    - Все потоки завершаются
    - Ресурсы освобождаются (`sem_destroy`, `pthread_mutex_destroy`)

Код программы

main.c

```
#include <stdio.h>
#include "scheduler.h"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Usage: %s <config.ini>\n", argv[0]);
        return 1;
    }

    if (!parse_ini(argv[1])) {
        printf("Failed to parse config file\n");
        return 1;
    }

    if (job_count == 0) {
        printf("No jobs defined\n");
        return 1;
    }
```

```
    printf("Loaded %d jobs, %d barriers\n", job_count, barrier_count);
    printf("Max parallel jobs: %d\n", max_parallel);

    for (int i = 0; i < job_count; i++) {
        printf("Job %s: command='%s', deps=%d",
               jobs[i].name, jobs[i].command, jobs[i].dep_count);
        if (jobs[i].dep_count > 0) {
            printf(" (");
            for (int d = 0; d < jobs[i].dep_count; d++) {
                printf("%s%s", jobs[i].dependencies[d],
                       d < jobs[i].dep_count - 1 ? ", " : "");
            }
            printf(")");
        }
        printf(", barrier='%s'\n", jobs[i].barrier);
    }

    if (has_cycle()) {
        printf("DAG contains cycles\n");
        return 1;
    }
    printf("No cycles found\n");

    if (!is_single_component()) {
        printf("DAG is not a single connected component\n");
        return 1;
    }
    printf("Single connected component OK\n");

    if (!has_start_and_end()) {
        printf("DAG must have start and end jobs\n");
        return 1;
    }
    printf("Start and end jobs OK\n");

    printf("Starting DAG execution...\n");
    run_dag();

    for (int i = 0; i < barrier_count; i++) {
        sem_destroy(&barriers[i].sem);
        pthread_mutex_destroy(&barriers[i].lock);
    }
    pthread_mutex_destroy(&dag_mutex);

    return global_error;
}
```

parcer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "scheduler.h"

Job jobs[MAX_JOBS];
Barrier barriers[MAX_BARRIERS];
int job_count = 0;
int barrier_count = 0;
int max_parallel = 1;
int global_error = 0;
pthread_mutex_t dag_mutex = PTHREAD_MUTEX_INITIALIZER;
sem_t parallel_sem;

void trim(char* str) {
    int i = 0, j = 0;
    while (str[i] && isspace((unsigned char)str[i])) i++;
    while (str[i]) str[j++] = str[i++];
    str[j] = 0;
    while (j > 0 && isspace((unsigned char)str[j-1])) str[--j] = 0;
}

Barrier* find_barrier(char* name) {
    for (int i = 0; i < barrier_count; i++) {
        if (strcmp(barriers[i].name, name) == 0) return &barriers[i];
    }
    return NULL;
}

int parse_ini(char* filename) {
    FILE* f = fopen(filename, "r");
    if (!f) return 0;

    char line[MAX_LINE];
    char section[MAX_NAME] = "";
    char current_job[MAX_NAME] = "";
    int in_job_section = 0;

    while (fgets(line, MAX_LINE, f)) {
        char* trimmed = line;
        while (*trimmed && isspace((unsigned char)*trimmed)) trimmed++;
        char* end = trimmed + strlen(trimmed) - 1;
        while (end > trimmed && isspace((unsigned char)*end)) *end-- = '\0';

        if (trimmed[0] == ';' || trimmed[0] == '#' || trimmed[0] == '\0') continue;

        if (trimmed[0] == '[' && strchr(trimmed, ']')) {
            char* close_bracket = strchr(trimmed, ']');
```

```
            *close_bracket = '\0';
            strncpy(section, trimmed + 1, MAX_NAME - 1);
            section[MAX_NAME - 1] = '\0';

            if (strcmp(section, "dag") == 0) {
                in_job_section = 0;
            } else if (strncmp(section, "job:", 4) == 0) {
                in_job_section = 1;
                strncpy(current_job, section + 4, MAX_NAME - 1);
                current_job[MAX_NAME - 1] = '\0';

                if (job_count < MAX_JOBS) {
                    strncpy(jobs[job_count].name, current_job, MAX_NAME - 1);
                    jobs[job_count].name[MAX_NAME - 1] = '\0';
                    jobs[job_count].dep_count = 0;
                    jobs[job_count].completed = 0;
                    jobs[job_count].failed = 0;
                    jobs[job_count].barrier[0] = '\0';
                    jobs[job_count].command[0] = '\0';
                    job_count++;
                }
            } else if (strncmp(section, "barrier:", 8) == 0) {
                in_job_section = 0;
                if (barrier_count < MAX_BARRIERS) {
                    strncpy(barriers[barrier_count].name, section + 8, MAX_NAME - 1);
                    barriers[barrier_count].name[MAX_NAME - 1] = '\0';
                    barriers[barrier_count].arrived = 0;
                    barriers[barrier_count].total_jobs = 0;
                    pthread_mutex_init(&barriers[barrier_count].lock, NULL);
                    barrier_count++;
                }
            }
        } else if (strchr(trimmed, '=')) {
            char* equals = strchr(trimmed, '=');
            *equals = '\0';
            char* key = trimmed;
            char* value = equals + 1;

            while (*key && isspace((unsigned char)*key)) key++;
            char* key_end = key + strlen(key) - 1;
            while (key_end > key && isspace((unsigned char)*key_end)) *key_end-- =
'\0';

            while (*value && isspace((unsigned char)*value)) value++;
            char* value_end = value + strlen(value) - 1;
            while (value_end > value && isspace((unsigned char)*value_end))
*value_end-- = '\0';

            if (strcmp(section, "dag") == 0) {
                if (strcmp(key, "max_parallel") == 0) {
```

```
                    max_parallel = atoi(value);
                }
            } else if (in_job_section) {
                for (int i = 0; i < job_count; i++) {
                    if (strcmp(jobs[i].name, current_job) == 0) {
                        if (strcmp(key, "command") == 0) {
                            strncpy(jobs[i].command, value, MAX_LINE - 1);
                            jobs[i].command[MAX_LINE - 1] = '\0';
                        } else if (strcmp(key, "deps") == 0) {
                            if (strlen(value) > 0) {
                                char* dep = strtok(value, ",");
                                while (dep && jobs[i].dep_count < MAX_DEPS) {
                                    char* dep_trim = dep;
                                    while (*dep_trim && isspace((unsigned
char)*dep_trim)) dep_trim++;
                                    char* dep_end = dep_trim + strlen(dep_trim) - 1;
                                    while (dep_end > dep_trim && isspace((unsigned
char)*dep_end)) *dep_end-- = '\0';

                                    if (strlen(dep_trim) > 0) {
strncpy(jobs[i].dependencies[jobs[i].dep_count], dep_trim, MAX_NAME - 1);
jobs[i].dependencies[jobs[i].dep_count][MAX_NAME - 1] = '\0';
                                        jobs[i].dep_count++;
                                    }
                                    dep = strtok(NULL, ",");
                                }
                            }
                        } else if (strcmp(key, "barrier") == 0) {
                            strncpy(jobs[i].barrier, value, MAX_NAME - 1);
                            jobs[i].barrier[MAX_NAME - 1] = '\0';
                        }
                        break;
                    }
                }
            }
        }
    }

    fclose(f);
    return 1;
}
```

validator.c

```
#include <stdio.h>
#include <string.h>
#include "scheduler.h"

int find_job_index(char* name) {
    for (int i = 0; i < job_count; i++) {
```

```
        if (strcmp(jobs[i].name, name) == 0) return i;
    }
    return -1;
}

int has_cycle_util(int v, int visited[], int rec_stack[]) {
    if (!visited[v]) {
        visited[v] = 1;
        rec_stack[v] = 1;
        for (int i = 0; i < jobs[v].dep_count; i++) {
            int dep_idx = find_job_index(jobs[v].dependencies[i]);
            if (dep_idx == -1) continue;
            if (!visited[dep_idx] && has_cycle_util(dep_idx, visited, rec_stack))
                return 1;
            else if (rec_stack[dep_idx])
                return 1;
        }
    }
    rec_stack[v] = 0;
    return 0;
}

int has_cycle() {
    int visited[MAX_JOBS] = {0};
    int rec_stack[MAX_JOBS] = {0};
    for (int i = 0; i < job_count; i++) {
        if (has_cycle_util(i, visited, rec_stack))
            return 1;
    }
    return 0;
}

void dfs(int v, int visited[]) {
    visited[v] = 1;

    for (int d = 0; d < jobs[v].dep_count; d++) {
        int dep_idx = find_job_index(jobs[v].dependencies[d]);
        if (dep_idx != -1 && !visited[dep_idx]) {
            dfs(dep_idx, visited);
        }
    }

    for (int i = 0; i < job_count; i++) {
        for (int d = 0; d < jobs[i].dep_count; d++) {
            if (strcmp(jobs[i].dependencies[d], jobs[v].name) == 0 && !visited[i]) {
                dfs(i, visited);
            }
        }
    }
```

```
}

int is_single_component() {
    if (job_count == 0) return 1;

    int visited[MAX_JOBS] = {0};

    dfs(0, visited);

    for (int i = 0; i < job_count; i++) {
        if (!visited[i]) return 0;
    }

    return 1;
}

int has_start_and_end() {
    int has_start = 0;
    int has_end = 0;

    for (int i = 0; i < job_count; i++) {
        if (jobs[i].dep_count == 0) {
            has_start = 1;
        }

        int is_referenced = 0;
        for (int j = 0; j < job_count; j++) {
            for (int k = 0; k < jobs[j].dep_count; k++) {
                if (strcmp(jobs[j].dependencies[k], jobs[i].name) == 0) {
                    is_referenced = 1;
                    break;
                }
            }
            if (is_referenced) break;
        }

        if (!is_referenced) {
            has_end = 1;
        }
    }

    return has_start && has_end;
}

int all_deps_completed(int idx) {
    for (int i = 0; i < jobs[idx].dep_count; i++) {
        int dep_idx = find_job_index(jobs[idx].dependencies[i]);
        if (dep_idx == -1 || !jobs[dep_idx].completed)
            return 0;
```

```
    }
    return 1;
}
```

executor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include "scheduler.h"

void* job_thread(void* arg) {
    Job* job = (Job*)arg;
    Barrier* bar = NULL;

    pthread_mutex_lock(&dag_mutex);
    if (global_error) {
        pthread_mutex_unlock(&dag_mutex);
        sem_post(&parallel_sem);
        return NULL;
    }

    if (strlen(job->barrier) > 0) {
        bar = find_barrier(job->barrier);
    }
    pthread_mutex_unlock(&dag_mutex);

    printf("Starting job: %s\n", job->name);

    int ret = system(job->command);

    pthread_mutex_lock(&dag_mutex);

    printf("Job %s finished with exit code %d\n", job->name, ret);

    if (ret != 0) {
        if (!global_error) {
            global_error = 1;
            job->failed = 1;
            printf("Job %s failed! Stopping DAG...\n", job->name);
        }
        job->completed = 1;
        pthread_mutex_unlock(&dag_mutex);

        if (bar) {
            pthread_mutex_lock(&bar->lock);
            bar->arrived++;
            if (bar->arrived == bar->total_jobs) {
```

```
                for (int i = 0; i < bar->total_jobs; i++) {
                    sem_post(&bar->sem);
                }
            }
            pthread_mutex_unlock(&bar->lock);
        }

        sem_post(&parallel_sem);
        return NULL;
    }

    job->completed = 1;
    pthread_mutex_unlock(&dag_mutex);

    if (bar) {
        printf("Job %s arriving at barrier %s\n", job->name, bar->name);
        pthread_mutex_lock(&bar->lock);
        bar->arrived++;

        if (bar->arrived == bar->total_jobs) {
            printf("Barrier %s released! All %d jobs arrived\n",
                   bar->name, bar->total_jobs);
            for (int i = 0; i < bar->total_jobs; i++) {
                sem_post(&bar->sem);
            }
        }
        pthread_mutex_unlock(&bar->lock);

        pthread_mutex_lock(&dag_mutex);
        int should_wait = !global_error;
        pthread_mutex_unlock(&dag_mutex);

        if (should_wait) {
            sem_wait(&bar->sem);
            printf("Job %s passed barrier %s\n", job->name, bar->name);
        } else {
            printf("Job %s skipping barrier due to error\n", job->name);
        }
    }

    sem_post(&parallel_sem);
    return NULL;
}

void run_dag() {
    sem_init(&parallel_sem, 0, max_parallel);

    for (int i = 0; i < barrier_count; i++) {
        barriers[i].arrived = 0;
```

```
    barriers[i].total_jobs = 0;
    sem_init(&barriers[i].sem, 0, 0);
}

for (int i = 0; i < job_count; i++) {
    if (strlen(jobs[i].barrier) > 0) {
        Barrier* bar = find_barrier(jobs[i].barrier);
        if (bar) {
            bar->total_jobs++;
        }
    }
}

int remaining = job_count;
int running[MAX_JOBS] = {0};
int started_jobs = 0;

while (remaining > 0 && !global_error) {
    for (int i = 0; i < job_count; i++) {
        if (!jobs[i].completed && !running[i] && all_deps_completed(i)) {
            pthread_mutex_lock(&dag_mutex);
            int can_start = !global_error;
            pthread_mutex_unlock(&dag_mutex);

            if (can_start && sem_trywait(&parallel_sem) == 0) {
                running[i] = 1;
                started_jobs++;
                pthread_create(&jobs[i].thread, NULL, job_thread, &jobs[i]);
                pthread_detach(jobs[i].thread);
                printf("Launched job %s (%d/%d started)\n",
                       jobs[i].name, started_jobs, job_count);
            }
        }
    }

    for (int i = 0; i < job_count; i++) {
        if (running[i] && jobs[i].completed) {
            running[i] = 0;
            remaining--;
        }
    }
}

if (global_error) {
    printf("\n=== DAG INTERRUPTED DUE TO JOB FAILURE ===\n");

    for (int i = 0; i < barrier_count; i++) {
        pthread_mutex_lock(&barriers[i].lock);
        for (int j = 0; j < barriers[i].total_jobs; j++) {
```

```
                sem_post(&barriers[i].sem);
            }
            pthread_mutex_unlock(&barriers[i].lock);
        }

        while (remaining > 0) {
            for (int i = 0; i < job_count; i++) {
                if (running[i] && jobs[i].completed) {
                    running[i] = 0;
                    remaining--;
                }
            }
        }
    } else {
        printf("\n=== DAG COMPLETED SUCCESSFULLY ===\n");
    }

    sem_destroy(&parallel_sem);
}
```

scheduler.h

```
#ifndef SCHEDULER_H
#define SCHEDULER_H

#include <pthread.h>
#include <semaphore.h>

#define MAX_JOBS 100
#define MAX_NAME 50
#define MAX_LINE 256
#define MAX_DEPS 10
#define MAX_BARRIERS 10

typedef struct {
    char name[MAX_NAME];
    char command[MAX_LINE];
    char dependencies[MAX_DEPS][MAX_NAME];
    int dep_count;
    char barrier[MAX_NAME];
    pthread_t thread;
    int completed;
    int failed;
} Job;

typedef struct {
    char name[MAX_NAME];
    sem_t sem;
    int arrived;
    int total_jobs;
```

```
    pthread_mutex_t lock;
} Barrier;

extern Job jobs[MAX_JOBS];
extern Barrier barriers[MAX_BARRIERS];
extern int job_count;
extern int barrier_count;
extern int max_parallel;
extern int global_error;
extern pthread_mutex_t dag_mutex;
extern sem_t parallel_sem;

void trim(char* str);
int find_job_index(char* name);
Barrier* find_barrier(char* name);
int has_cycle();
int is_single_component();
int has_start_and_end();
int all_deps_completed(int idx);
void* job_thread(void* arg);
void run_dag();
int parse_ini(char* filename);

#endif
```

Makefile

```
CC = gcc
CFLAGS = -Wall -pthread
TARGET = scheduler
OBJS = main.o parser.o validator.o executor.o

all: $(TARGET)

$(TARGET): $(OBJS)
	$(CC) $(CFLAGS) -o $@ $(OBJS)

main.o: main.c scheduler.h
	$(CC) $(CFLAGS) -c main.c

parser.o: parser.c scheduler.h
	$(CC) $(CFLAGS) -c parser.c

validator.o: validator.c scheduler.h
	$(CC) $(CFLAGS) -c validator.c

executor.o: executor.c scheduler.h
	$(CC) $(CFLAGS) -c executor.c

clean:
```

```
	rm -f $(OBJS) $(TARGET)

run: all
	./$(TARGET) test.ini
```

test.ini

```
[dag]
max_parallel = 1

[job:minimal]
command = true
deps =
barrier =
```

Протокол работы программы

Тестирование:

```
$ strace -f ./scheduler test.ini
execve("./scheduler", ["./scheduler", "test.ini"], 0x7ffc625c7120 /* 27 vars */) = 0
brk(NULL)                               = 0x5f19686bd000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7d8edd3d8000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=41635, ...}) = 0
mmap(NULL, 41635, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7d8edd3cd000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) =
832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7d8edd000000
mmap(0x7d8edd028000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7d8edd028000
mmap(0x7d8edd1b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7d8edd1b0000
```

mmap(0x7d8edd1ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7d8edd1ff000

mmap(0x7d8edd205000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7d8edd205000

close(3) **= 0**

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7d8edd3ca000

arch_prctl(ARCH_SET_FS, 0x7d8edd3ca740) = 0

set_tid_address(0x7d8edd3caa10) = 591

set_robust_list(0x7d8edd3caa20, 24) = 0

rseq(0x7d8edd3cb060, 0x20, 0, 0x53053053) = 0

mprotect(0x7d8edd1ff000, 16384, PROT_READ) = 0

mprotect(0x5f1957452000, 4096, PROT_READ) = 0

mprotect(0x7d8edd410000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7d8edd3cd000, 41635) **= 0**

getrandom("\x39\x13\xfc\x8c\x56\x44\x31\xf8", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x5f19686bd000

brk(0x5f19686de000) = 0x5f19686de000

openat(AT_FDCWD, "test.ini", O_RDONLY) **= 3**

fstat(3, {st_mode=S_IFREG|0777, st_size=77, ...}) = 0

read(3, "[dag]\r\nmax_parallel = 1\r\n\r\n[job:"..., 4096) = 77

read(3, "", 4096) **= 0**

close(3) **= 0**

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

write(1, "Loaded 1 jobs, 0 barriers\n", 26Loaded 1 jobs, 0 barriers

) = 26

write(1, "Max parallel jobs: 1\n", 21Max parallel jobs: 1

) = 21

write(1, "Job minimal: command='true', dep"..., 48Job minimal: command='true', deps=0, barrier=''

) = 48

write(1, "No cycles found\n", 16No cycles found

) = 16

write(1, "Single connected component OK\n", 30Single connected component OK

```
     ) = 30

     write(1, "Start and end jobs OK\n", 22Start and end jobs OK

     ) = 22

     write(1, "Starting DAG execution...\n", 26Starting DAG execution...

     ) = 26

     rt_sigaction(SIGRT_1, {sa_handler=0x7d8edd099530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7d8edd045330}, NULL, 8)
= 0

     rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

     mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7d8edc7ff000

     mprotect(0x7d8edc800000, 8388608, PROT_READ|PROT_WRITE) = 0

     rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

     clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7d8edcfff990,
parent_tid=0x7d8edcfff990, exit_signal=0, stack=0x7d8edc7ff000, stack_size=0x7fff80,
tls=0x7d8edcfff6c0}strace: Process 592 attached

      => {parent_tid=[592]}, 88) = 592

     [pid   592] rseq(0x7d8edcffffe0, 0x20, 0, 0x53053053 <unfinished ...>

     [pid   591] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

     [pid   592] <... rseq resumed>)         = 0

     [pid   591] <... rt_sigprocmask resumed>NULL, 8) = 0

     [pid   592] set_robust_list(0x7d8edcfff9a0, 24 <unfinished ...>

     [pid   591] write(1, "Launched job minimal (1/1 starte"..., 35 <unfinished ...>

     [pid   592] <... set_robust_list resumed>) = 0

     Launched job minimal (1/1 started)

     [pid   591] <... write resumed>)        = 35

     [pid   592] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

     [pid   592] write(1, "Starting job: minimal\n", 22Starting job: minimal

     ) = 22

     [pid   592] rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7d8edd045330}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0

     [pid   592] rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[],
sa_flags=SA_RESTORER, sa_restorer=0x7d8edd045330}, {sa_handler=SIG_DFL, sa_mask=[],
sa_flags=0}, 8) = 0

     [pid   592] rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
```

[pid 592] **mmap(NULL, 36864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7d8edd3cf000**

[pid 592] rt_sigprocmask(SIG_BLOCK, ~[], [CHLD], 8) = 0

[pid 592] **clone3({flags=CLONE_VM|CLONE_VFORK|CLONE_CLEAR_SIGHAND, exit_signal=SIGCHLD, stack=0x7d8edd3cf000, stack_size=0x9000}, 88strace: Process 593 attached**

<unfinished ...>

[pid 593] rt_sigprocmask(SIG_BLOCK, NULL, ~[KILL STOP], 8) = 0

[pid 593] rt_**sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7d8edd045330}, NULL, 8) = 0**

[pid 593] rt_**sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7d8edd045330}, NULL, 8) = 0**

[pid 593] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 593] **execve("/bin/sh", ["sh", "-c", "--", "true"], 0x7ffdacd51ba0 /* 27 vars */ <unfinished ...>**

[pid 592] <... clone3 resumed>) = 593

[pid 592] **munmap(0x7d8edd3cf000, 36864 <unfinished ...>**

[pid 593] <... execve resumed>) = 0

[pid 592] <... munmap resumed>) = 0

[pid 593] brk(NULL <unfinished ...>

[pid 592] rt_sigprocmask(SIG_SETMASK, [CHLD], <unfinished ...>

[pid 593] <... brk resumed>) = 0x5b1c28af8000

[pid 592] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 593] **mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>**

[pid 592] **wait4(593, <unfinished ...>**

[pid 593] <... mmap resumed>) = 0x7285d505a000

[pid 593] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 593] **openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3**

[pid 593] fstat(3, {st_mode=S_IFREG|0644, st_size=41635, ...}) = 0

[pid 593] **mmap(NULL, 41635, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7285d504f000**

[pid 593] **close(3) = 0**

[pid 593] **openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3**

[pid 593] **read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832**

[pid 593] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid 593] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 593] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

[pid 593] **mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7285d4e00000**

[pid 593] **mmap(0x7285d4e28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7285d4e28000**

[pid 593] **mmap(0x7285d4fb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7285d4fb0000**

[pid 593] **mmap(0x7285d4fff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7285d4fff000**

[pid 593] **mmap(0x7285d5005000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7285d5005000**

[pid 593] **close(3)** = 0

[pid 593] **mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7285d504c000**

[pid 593] arch_prctl(ARCH_SET_FS, 0x7285d504c740) = 0

[pid 593] set_tid_address(0x7285d504ca10) = 593

[pid 593] set_robust_list(0x7285d504ca20, 24) = 0

[pid 593] rseq(0x7285d504d060, 0x20, 0, 0x53053053) = 0

[pid 593] mprotect(0x7285d4fff000, 16384, PROT_READ) = 0

[pid 593] mprotect(0x5b1bf2fbe000, 8192, PROT_READ) = 0

[pid 593] mprotect(0x7285d5092000, 8192, PROT_READ) = 0

[pid 593] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 593] **munmap(0x7285d504f000, 41635) = 0**

[pid 593] getuid() = 1000

[pid 593] **getgid()** = **1000**

[pid 593] **getpid()** = **593**

[pid 593] rt_**sigaction(SIGCHLD, {sa_handler=0x5b1bf2fb3cd0, sa_mask=~[RTMIN RT_1], sa_flags=SA_RESTORER, sa_restorer=0x7285d4e45330}, NULL, 8) = 0**

[pid 593] geteuid() = 1000

[pid 593] getrandom("\x6b\x4c\xa5\x84\x6a\xd8\x18\x64", 8, GRND_NONBLOCK) = 8

[pid 593] brk(NULL) = 0x5b1c28af8000

[pid 593] brk(0x5b1c28b19000) = 0x5b1c28b19000

[pid 593] **getppid()** = **591**

```
     [pid   593] newfstatat(AT_FDCWD, "/mnt/c/Dev/Projects/OS_Labs/CP",
{st_mode=S_IFDIR|0777, st_size=4096, ...}, 0) = 0

     [pid   593] newfstatat(AT_FDCWD, ".", {st_mode=S_IFDIR|0777, st_size=4096, ...}, 0) = 0

     [pid   593] geteuid()                   = 1000

     [pid   593] getegid()                   = 1000

     [pid   593] rt_sigaction(SIGINT, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8)
= 0

     [pid   593] rt_sigaction(SIGINT, {sa_handler=0x5b1bf2fb3cd0, sa_mask=~[RTMIN RT_1],
sa_flags=SA_RESTORER, sa_restorer=0x7285d4e45330}, NULL, 8) = 0

     [pid   593] rt_sigaction(SIGQUIT, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0},
8) = 0

     [pid   593] rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=~[RTMIN RT_1],
sa_flags=SA_RESTORER, sa_restorer=0x7285d4e45330}, NULL, 8) = 0

     [pid   593] rt_sigaction(SIGTERM, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0},
8) = 0

     [pid   593] rt_sigaction(SIGTERM, {sa_handler=SIG_DFL, sa_mask=~[RTMIN RT_1],
sa_flags=SA_RESTORER, sa_restorer=0x7285d4e45330}, NULL, 8) = 0

     [pid   593] exit_group(0)               = ?

     [pid   593] +++ exited with 0 +++

     [pid   591] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=593, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---

     [pid   592] <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 593

     [pid   592] rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7d8edd045330}, NULL, 8) = 0

     [pid   592] rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[],
sa_flags=SA_RESTORER, sa_restorer=0x7d8edd045330}, NULL, 8) = 0

     [pid   592] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

     [pid   592] write(1, "Job minimal finished with exit c"..., 38Job minimal finished with
exit code 0

     ) = 38

     [pid   591] write(1, "\n", 1 <unfinished ...>


     [pid   592] rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>

     [pid   591] <... write resumed>)        = 1

     [pid   592] <... rt_sigprocmask resumed>NULL, 8) = 0

     [pid   591] write(1, "=== DAG COMPLETED SUCCESSFULLY ="..., 35 <unfinished ...>

     === DAG COMPLETED SUCCESSFULLY ===
```

```
[pid   592] madvise(0x7d8edc7ff000, 8368128, MADV_DONTNEED <unfinished ...>
[pid   591] <... write resumed>)        = 35
[pid   592] <... madvise resumed>)      = 0
[pid   591] exit_group(0 <unfinished ...>
[pid   592] exit(0 <unfinished ...>
[pid   591] <... exit_group resumed>)   = ?
[pid   592] <... exit resumed>)         = ?
[pid   592] +++ exited with 0 +++
+++ exited with 0 +++
```

Вывод

В ходе курсового проекта был успешно разработан и реализован планировщик выполнения задач (джобов), организованных в виде направленного ациклического графа (DAG). Программа демонстрирует практическое применение механизмов многопоточности, синхронизации через барьеры и эффективное управление параллельным выполнением задач. Реализованное решение позволяет гибко настраивать зависимости между задачами, контролировать степень параллелизма и обеспечивать корректную обработку ошибок. Планировщик может быть использован для автоматизации сложных вычислительных процессов, требующих упорядоченного выполнения операций с синхронизацией промежуточных результатов, что подтверждает его практическую ценность и соответствие современным требованиям к системному программному обеспечению.