МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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»

Студент: Власко М. М.
Группа: М8О-208Б-23
Преподаватель: Живалев Е. А.
Дата: _____
Оценка: _____
Подпись: _____

Москва, 2024

Тема: Разработка планировщика DAG джобов

Цель работы: Целью курсового проекта являлось приобретение практических навыков в:

- разработке планировщиков задач на основе направленных ациклических графов (DAG);
- работе с конфигурационными файлами (JSON);
- реализации многопоточности и примитивов синхронизации.

Вариант: 35. JSON/Semaphore.

Задачи проекта:

1. Реализовать проверку DAG на корректность, включая отсутствие циклов и наличие одной связной компоненты.
2. Организовать выполнение задач (джобов) DAG с учётом их зависимостей.
3. Реализовать параллельное выполнение джобов с ограничением максимального количества одновременно выполняемых задач.
4. Реализовать поддержку синхронизации через семафоры с возможностью настройки через конфигурационный файл.

Описание решения: Программное решение разработано на языке C++ с использованием библиотеки pthread для работы с потоками. Основные компоненты системы:

1. **Проверка корректности DAG:**
    - На этапе загрузки конфигурационного файла происходит построение графа джобов.
    - Реализована проверка графа на наличие циклов с использованием алгоритма обхода в глубину.
    - Проверяется наличие стартовых и завершающих джобов, а также единственность связной компоненты.
2. **Организация выполнения DAG:**
    - Для каждой джобы хранится информация о её зависимостях и семафоре (если он указан).
    - Выполнение начинается с джобов, не имеющих зависимостей (стартовые).
    - Задачи DAG запускаются в отдельном потоке, и их выполнение координируется с использованием мьютексов и условных переменных.

3. **Параллельное выполнение джобов:**
    - Максимальное количество одновременно выполняемых задач задаётся параметром, передаваемым программе.
    - Реализована очередь готовых к выполнению задач, которая координируется через условные переменные.
4. **Поддержка семафоров:**
    - Семафоры настраиваются через конфигурационный файл. Для каждого семафора указывается его имя и степень параллелизма.
    - Джобы, связанные с одним семафором, ограничены в количестве одновременно выполняемых экземпляров.

Пример реализации основных функций:

```
bool IsCyclicUtil(const int node, std::unordered_map<int, std::vector<int>>&
graph,
        std::unordered_map<int, bool>& visited, std::unordered_map<int,
bool>& recStack) {
    if (!visited[node]) {
        visited[node] = true;
        recStack[node] = true;

        for (const auto& neighbor : graph[node]) {
            if (!visited[neighbor] && IsCyclicUtil(neighbor, graph, visited,
recStack) || recStack[neighbor])
                return true;
        }
    }

    recStack[node] = false;
    return false;
}

bool ValidateDAG(std::unordered_map<int, std::vector<int>>& graph) {
    if (IsCyclic(graph)) {
        std::cerr << "Error: DAG contains cycles" << std::endl;
        return false;
    }
    // Additional validation for start/end jobs
    return true;
}
```

```
void ExecuteJob(const std::string& jobName, sem_t* semaphore,
std::atomic<bool>& errorFlag, const int execTime) {
    if (errorFlag) {
        pthread_cond_broadcast(&queueCV);
        return;
    }

    if (semaphore) {
        sem_wait(semaphore);
    }

    std::cout << "Starting job: " << jobName << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(execTime));

    if (mustBreak == jobName) {
        std::cerr << "Job failed: " << jobName << std::endl;
        errorFlag = true;
    } else {
        std::cout << "Job completed: " << jobName << std::endl;
    }

    if (semaphore) {
        sem_post(semaphore);
    }
}
```

Использование программы:

1. Конфигурационный файл в формате JSON задаёт описание джобов и их зависимостей. Пример:

```
{
  "jobs": [
    {
      "id": 3,
      "name": "d",
      "dependencies": [],
      "semaphore": "sem1",
      "semaphore_limit": 1,
      "time": 4
    },
    {
      "id": 0,
      "name": "a",
      "dependencies": [],
      "semaphore": "sem1",
      "time": 1
    },
    {
      "id": 1,
```

```
      "name": "b",
      "dependencies": []
    },
    {
      "id": 2,
      "name": "c",
      "dependencies": [0, 1, 3]
    }
  ]
}}
```

2. Программа принимает на вход имя конфигурационного файла и максимальное число параллельных задач.
3. После проверки DAG запускается выполнение джобов с учётом их зависимостей и ограничений.

Пример работы:

```
./cp config.json 2

Starting job: b

Starting job: a

Job completed: a

Starting job: d

Job completed: b

Job completed: d

Starting job: c

Job completed: c

Execution completed successfully
```

Вывод: В результате работы программы были достигнуты все поставленные цели. Реализован планировщик DAG с поддержкой параллельного выполнения и синхронизации через семафоры. Программа успешно проверяет корректность графа и выполняет задачи с учётом заданных ограничений. Получены навыки работы с многопоточностью, примитивами синхронизации и обработкой графовых структур.