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

Курсовой проект по курсу
«Операционные системы»

Студент: Абдыкалыков Нурсултан Абдыкалыкович
Группа: М8О-206Б-23
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
_____
Дата: _____
Подпись: _____
_____

Москва, 2024

Постановка задачи

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.

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).
Реализация через ini

Общие сведения о программе

Программа на C++ состоит из одного файла dag.cpp и использует многопоточность для выполнения задач, представленных в виде Directed Acyclic Graph (DAG). Основные компоненты включают структуру Job, которая описывает джобу с её зависимостями и зависимыми задачами, класс DAG для представления графа джобов, класс Parser для парсинга INI-файла с
зависимостями, и класс Scheduler для планирования и выполнения джоб.

Программа использует библиотеки <thread>, <mutex>, <atomic> и <chrono> для многопоточности, синхронизации, управления флагами и симуляции выполнения задач. Функция validateDAG проверяет граф на циклы, связность и наличие стартовых и завершающих джоб. Основная функция main считывает конфигурационный файл, парсит его, валидирует граф и запускает планировщик.

Программа эффективно выполняет задачи в порядке их зависимостей, используя потоки для параллельного выполнения готовых джоб. При возникновении ошибки выполнение прерывается. Код структурирован, легко читаем и поддерживаем.

Общий метод и алгоритм решения
Общий метод и алгоритм решения включают несколько классов, каждый с определёнными функциями. Класс DAG хранит граф задач в unordered_map, добавляет рёбра через addEdge, проверяет циклы (hasCycle) и связность (isConnected), а также возвращает стартовые (getStartJobs) и завершающие задачи (getEndJobs). Класс Parser парсит INI-файл, строя граф на основе
зависимостей. Функция validateDAG проверяет граф на циклы, связность и корректность, завершая программу при ошибках.

Класс Scheduler планирует выполнение задач, создавая отдельные потоки для каждой джобы. Работа задач моделируется с задержкой (std::this_thread::sleep_for). Планировщик отслеживает зависимости, запуская задачи только после выполнения их предшественников. Для потокобезопасности используются мьютексы, а атомарная переменная контролирует завершение при ошибках. Программа эффективно выполняет задачи в порядке их зависимостей с использованием многопоточности.

Демонстрация работы программы
./dag config
DAG прошел валидацию успешно. Стартовые джобы: 1 2
Завершающие джобы: 3
Запуск джобы 1
Джоба 1 завершена успешно. Запуск джобы 2
Джоба 2 завершена успешно. Запуск джобы 3
Джоба 3 завершена успешно.

Вывод:
В ходе выполнения данной лабораторной работы мне удалось закрепить навыки работы с потоками и мьютексами, а также углубить понимание многопоточного программирования в C++. Я научился эффективно управлять параллельным выполнением задач, используя синхронизацию для обеспечения потокобезопасности. Помимо этого, я освежил знания в области дискретной математики и теории графов, которые были применены при реализации алгоритмов, таких как обход графа в глубину (DFS) для проверки на циклы и связность. Также я получил опыт работы с парсингом INI-файлов, что позволило мне автоматизировать процесс построения графа задач на основе входных данных. В процессе работы я столкнулся с необходимостью оптимизации управления ресурсами, что помогло мне лучше понять, как избегать состояний гонки и утечек памяти. В целом, лабораторная работа позволила мне не только улучшить навыки программирования, но и развить умение анализировать и решать сложные задачи, связанные с параллельными вычислениями и обработкой данных