ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Кафедра программной инженерии

Микропроект №2

по дисциплине «Архитектура вычислительных систем»

Тема: «разработка многопоточной программы с использованием семафоров»

Вариант №5

Пояснительная записка

Исполнитель: студент 2 курса,

Шалаева Марина Андреевна

Группа БПИ191

Москва 2020

СОДЕРЖАНИЕ

СТРУКТУРА РАБОТЫ

1) Четыре .bat файла для упрощения тестирования программы;
2) .cpp файл с исходным кодом работы;
3) .exe файл
4) Три теста «testN.txt», где N – порядковый номер теста;
5) Пояснительная записка.pdf – отчет о проделанной работе в формате pdf;
6) Пояснительная записка.docx – отчет о проделанной работе в формате docx;
7) Результаты тестирования.pdf – отчет о результатах тестирования программы.

ТЕКСТ ЗАДАНИЯ

Тема №5:

Задача о каннибалах. Племя из n дикарей ест вместе из большого горшка, который вмещает m кусков тушеного миссионера. Когда дикарь хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе дикарь будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает. Создать многопоточное приложение, моделирующее обед дикарей. При решении задачи пользоваться семафорами.

МОДЕЛЬ ВЫЧИСЛЕНИЯ

Программа была разработана на языке C++ в среде разработки Visual Studio.

Для работы с входными данными была использована файловая система. Все входные файлы имеют вид «testN.txt», где N – номер теста. Основная работа по считыванию данных из файла происходит в функции **void workWithFiles()**. В первой строке каждого файла располагаются два числа, введенные через пробел: **m** – максимальное количество кусков тушеного миссионера в горшке – и **n** – количество дикарей в племени. Далее в каждой строчке располагается команда типа «**cannibal N is hungry**», где **N** – это номер каннибала в племени. Оба числа должны быть целочисленными и иметь значение, строго большее нуля; также на **N** есть ограничение сверху: оно должно быть меньше либо равно количеству дикарей в племени.

Выходные данные реализованы с помощью вывода сообщений в консоль.

Формат взаимодействия с программой – ввод данных в консоль в формате «<имя .exe файла> <путь к входному файлу>». При некорректно заданных входных параметрах в консоль выводится сообщение о верном формате ввода.

Для реализации племени каннибалов было принято решение использовать поле типа **int n**, олицетворяющее количество дикарей, а также поле **std::vector<int> currentCannibalNumber**, которое является списком, содержащим в себе номера каннибалов, которые едят из горшка, в считанной из входного файла последовательности.

Для реализации горшочка с едой было принято решение использовать два поля типа **int**: **m** и **currentNumOfPeices**, где **m** – максимальное количество кусков тушеного миссионера в горшке, **currentNumOfPeices** – количество кусков миссионера в горшке на момент обращения к данной переменной.

Для реализации работы потоков была подключена библиотека **<thread>**. Было создано поле **int numberOfThreads**, представляющее из себя количество исполняемых потоков, эквивалентное числу строчек с командами в исходном файле; поле **std::mutex mut**, являющуюся двоичным семафором для обработки критических секций; а также в функции **void workWithThreads()** создается переменная **std::vector<std::thread> threads(numberOfThreads)**, представляющая из себя список исполняемых потоков, размер которого задается с помощью поля **numberOfThreads**, которое инициализируется после завершения работы по считыванию информации из входного файла.

Для анализа и демонстрации работы приложения было предусмотрено измерение времени работы программы с использованием функции **clock()** из библиотеки **<ctime>**, а также вывод индексов потоков в каждом выводимом на экран сообщении.

При разработке программы была использована парадигма параллельного программирования взаимодействующие равные и парадигма портфеля задач.

Взаимодействующие равные – модель, в которой исключен не занимающийся непосредственными вычислениями управляющий поток.

Распределение работ в таком приложении либо фиксировано заранее, либо динамически определяется во время выполнения. Одним из распространенных способов динамического распределения работ является «портфель задач». Портфель задач, как правило, реализуется с помощью разделяемой переменной, доступ к которой в один момент времени имеет только один процесс.

Вычислительная задача делится на конечное число подзадач. Как правило, каждая подзадача должна выполнить однотипные действия над разными данными. Подзадачи нумеруются, и каждому номеру определяется функция, которая однозначно отражает номер задачи на соответствующий ему набор данных. Создается переменная, которую следует выполнять следующей. Каждый поток сначала обращается к портфелю задач для выяснения текущего номера задачи, после этого увеличивает его, потом берет соответствующие данные и выполняет задачу, затем обращается к портфелю задач для выяснения следующего номера задачи.

То есть поток получает задачу из портфеля и пока задача остается не выполненной, поток ее решает, а затем снова получает задачу из портфеля.[1]

В случае данного приложения количество потоков, как и эквивалентное ему число подзадач, определяется после считывания информации из файла, поскольку нам важно количество исходных команд. Портфелем задач в данном случае является горшок, из которого едят дикари, уменьшая тем самым общее число кусков тушеного миссионера в горшочке. Каждой подзадачей, выполняемой одним потоков, является функция **void feedHungryCannibal(const int& currentCommandDigit)**, в которой происходят проверка наличия каннибала с таким номером в племени, уменьшение количества кусков миссионера в горшке и вывод соответствующих сообщений. Аргумент **currentCommandDigit** является индексом строки из файла, с помощью которого можно узнать, номер какого каннибала передавался в конкретной строчке. В случае, если горшочек оказывается пуст, создается

временный поток и вызывается функция **void cookDinner()**. Было принято решение о создании нового потока по причине того, что предугадать количество вызовов данной функции заранее крайне проблематично. Новый созданный поток «заполняет» горшок едой (увеличивает переменную **currentNumOfPeices** до **m**), после чего происходит его удаление с помощью функции **delete**.

Для синхронизации параллельно работающих задач и для защиты критических секций был разработан такой метод управления параллельными процессами, как семафор.[2] Семафор был предложен Дейкстрой в 1965 году. Это очень важный метод управления параллельными процессами с использованием простого целочисленного значения, известного как семафор. Семафор — это просто переменная, которая неотрицательна и разделена между потоками. Эта переменная используется для решения проблемы критической секции и для синхронизации процессов в многопроцессорной среде.

Семафоры бывают двух типов:

- Двоичный семафор — это также известно как блокировка мьютекса. Он может иметь только два значения — 0 и 1. Его значение инициализируется равным 1. Он используется для реализации решения проблемы критического сечения с несколькими процессами.
- Подсчет семафора — его значение может варьироваться в неограниченном домене. Он используется для управления доступом к ресурсу, который имеет несколько экземпляров.[3]

В случае разработанного мной приложения во избежание обработки одной и той же команды двумя разными потоками был предусмотрен двоичный семафор мьютекс **std::mutex mut** из библиотеки **<thread>**. Команда **mut.lock()** находится в начале критической секции, а команда **mut.unlock()** находится в конце критической секции; используется данный прием в функциях **void feedHungryCannibal(const int& currentCommandDigit)** и **void cookDinner()**.

ТЕСТИРОВАНИЕ

В качестве входных аргументов программа принимает один аргумент, являющийся именем входного файла или путем к нему. Предъявляемые требования: параметр должен содержать в себе корректный путь к файлу или его название. В случае ввода некорректных данных программа выводит пользователю сообщение о том, в каком формате должны вводиться входные данные и завершает свою работу.

Для ускорения тестирования были разработаны .bat файлы для тестирования корректных и некорректных входных данных. Для работы с .bat файлами необходимо открыть командную строку, изменить директорию на папку, содержащую .exe файл, «перетащить» нужный .bat файл в консоль и нажать Enter.

ИСТОЧНИКИ

1) Парадигмы параллельного программирования [Электронный ресурс] //URL: https://pro-prof.com/forums/topic/parallel-programming-paradigms (Дата обращения: 9.12.2020, режим доступа: свободный).
2) Семафор [Электронный ресурс] //URL: https://ru.wikipedia.org/wiki/Семафор_(программирование) (Дата обращения: 10.12.2020, режим доступа: свободный).
3) Семафоры в синхронизации процессов [Электронный ресурс] //URL: http://espressocode.top/semaphores-in-process-synchronization/ (Дата обращения: 10.12.2020, режим доступа: свободный).
4) Мьютекс [Электронный ресурс] //URL: https://medium.com/nuances-of-programming/с-мьютекс-пишем-наш-первый-код-для-многопоточной-среды-543a3d60ef30 (Дата обращения: 10.12.2020, режим доступа: свободный).
5) Многопоточное программирование [Электронный ресурс] //URL: http://softcraft.ru/edu/comparch/practice/thread/02-sync/ (Дата обращения: 9.12.2020, режим доступа: свободный).