**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Паралельнные алгоритмы»**

Тема: Реализация потокобезопасных структур данных с блокировками

|  |  |  |
| --- | --- | --- |
| Студент гр. 0304 |  | Максименко Е.М. |
| Преподаватель |  | Сергеева Е.И |

Санкт-Петербург

2023

## Цель работы.

Изучение способов реализации потокобезопасных структур данных с блокировками. Изучение примитивов синхронизации межпоточного взаимодействия.

## Задание.

Реализовать итерационное (потенциально бесконечное) выполнение подготовки, обработки и вывода данных по шаблону “производитель -потребитель” (на основе лаб. 1 (части 1.2.1 и 1.2.2) ).

Обеспечить параллельное выполнение потоков обработки готовой порции данных, подготовки следующей порции данных и вывода предыдущих полученных результатов.

Использовать механизм “условных переменных”.

**2.1**

Использовать очередь с “грубой” блокировкой.

**2.2**

Использовать очередь с “тонкой” блокировкой.

## Выполнение работы.

1. Реализован класс ThreadSafeQueue для потокобезопасной работы с очередью. Класс ThreadSafeQueue предоставляет минимальный интерфейс очереди, который содержит такие методы, как push и waitAndPop. Реализовано 2 варианта данного класса: очередь с «грубыми» блокировками и очередь с «тонкими» («мелкогранулярными») блокировки.

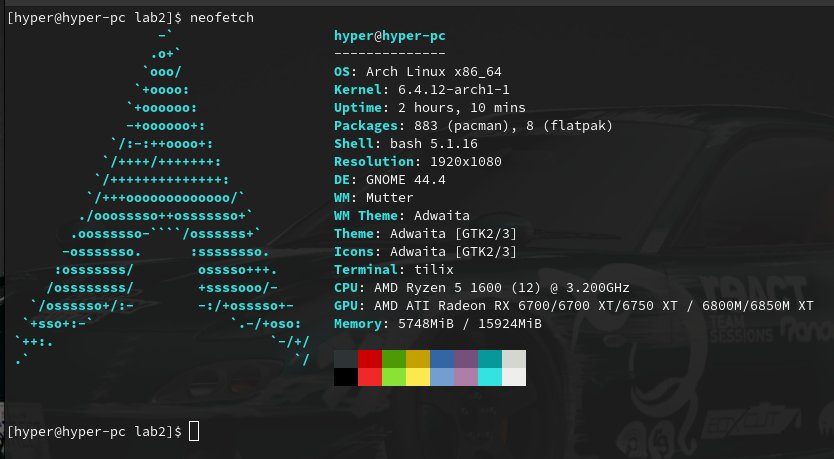
Очередь с «грубыми» блокировками представляет собой обертку над очередью из стандартной библиотеки, операции над которой синхронизируются с помощью мьютекса и условной переменной. Для выполнения операции добавления и извлечения мьютекс блокируется. Операция добавления push сигнализирует условной переменной о том, что очередь не пуста. Операция popAndWait проверяет, пуста ли очередь. Если очередь пуста, то выполнение блокируется до момента сигнализирования условной переменной. Когда очередь не пуста, элемент извлекается из очереди.

Очередь с «тонкими» блокировками представляет собой однонаправленный связный список, который поддерживает операции добавления в конец (push) и извлечения из начала (popAndWait). Для синхронизации операций с очередью используется два мьютекса (для головы и для хвоста) и условная переменная. Выполнение операции добавления элемента требует блокировки мьютекса хвоста и изменения значения текущего хвостового узла и привязывания нового хвостового узла, после чего следует сигнализации условной переменной. Выполнение операции извлечения требует больших усилий: для работы условной переменной блокируется мьютекс головы очереди, после чего ожидается условная переменная, в предикате которой проверяется пустота списка (данная операция требует блокировки еще и мьютекса хвоста). Когда очередь не пуста, данные извлекаются, причем мьютекс головы до конца извлечения остается заблокированным.

2. Для реализации паттерна производитель-потребитель реализован класс ThreadFactory. Данный класс поддерживает операции добавления производителя и потребителя, а также добавления и извлечения задачи в очереди. Производитель производит задачу и добавляет ее в очередь. Потребитель извлекает из очереди задачу и выполняет ее. В данной работе в качестве задачи производителя выступает генерация двух матриц, в качестве задачи потребителя — их умножение.

3. Измерение времени работы программы для очереди с «грубыми» и «тонкими» блокировками в зависимости от количества производителей и потребителей.

Для измерения зависимости времени выполнения программы с различными типами блокировок использовалась платформа, параметры которой отражены на рис. 1.

Рисунок 1. Параметры платформы, на которой производятся измерения

Для усреднения результатов программы запускаются 100 раз. Время, представленное в таблицах, указывается суммарное для 100 запусков. Количество задач, выполняемых программой — 6000.

Таблица 1. Время выполнения программы с «грубыми» блокировками.

|  |  |  |
| --- | --- | --- |
| **Количество потоков:**  **Производители/Потребители** | **Real Time, сек** | **Sys. Time, сек** |
| 2/2 | 17.679 | 5.880 |
| 12/12 | 6.144 | 9.849 |
| 12/1 | 7.216 | 3.607 |
| 1/12 | 33.199 | 6.725 |
| 500/500 | 7.180 | 8.721 |
| 500/10 | 6.134 | 9.076 |
| 10/500 | 8.721 | 19.396 |

Таблица 2. Время выполнения программы с «тонкими» блокировками.

|  |  |  |
| --- | --- | --- |
| **Количество потоков:**  **Производители/Потребители** | **Real Time, сек** | **Sys. Time, сек** |
| 2/2 | 18.131 | 5.306 |
| 12/12 | 4.566 | 4.111 |
| 12/1 | 7.451 | 3.059 |
| 1/12 | 33.723 | 6.501 |
| 500/500 | 7.183 | 7.410 |
| 500/10 | 6.069 | 7.317 |
| 10/500 | 5.702 | 9.728 |

Как видно по табл. 1 и табл. 2, очередь с «тонкими» блокировками не во всех ситуациях превосходит очередь с «грубыми» блокировками (по Real Time): при малом количестве потоков (2/2), когда потоки не сильно (по сравнению с 500/500) конкурируют за мьютекс, на первый план выходят особенности реализации структуры данных. При небольшом количестве потоков (12/12, 12/1, 1/12) конкуренция за мьютекс возрастает, но все еще невелика, что позволяет обеим реализациям работать примерно за одинаковое время (Real Time).