МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Параллельные алгоритмы»
Тема: Реализация потокобезопасных структур данных с блокировками

Студент гр. 0303 _____ Морозов А.Ю.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучение способов реализации потокобезопасных структур данных с блокировками.

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.

В отчёте: сравнить производительность 2.1. и 2.2 в зависимости от количества производителей и потребителей.

Выполнение работы.

В ходе выполнения лабораторной работы были реализованы шаблонные классы *ThreadSafeQueue* и *FineGrainedQueue* представляющие собой очередь с «грубой» блокировкой и очередь с «тонкой» блокировкой соответственно.

Очередь с «грубой» блокировкой имеет методы добавления элемента *add(T element)* и извлечения элемента *T get()*. Синхронизация обеспечивается с помощью мьютекса и условной переменной. Для извлечения и добавления элемента необходимо заблокировать мьютекс. В случае добавления элемента посылается сигнал условной переменной о том, что очередь не пуста. В случае извлечения элемента необходимо проверить очередь на наличие в ней элементов. Если элементов нет, нужно дождаться сигнала от условной переменной.

Очередь с «тонкой» блокировкой представляет собой односвязный список, который поддерживает операции добавления элемента *add(T element)* и извлечения элемента *T get()*. Синхронизация обеспечивается использованием

двух мьютексов (для головы и для хвоста) и условной переменной. Для добавления элемента в очередь требуется заблокировать мьютекс хвоста, изменить значение текущего хвостового узла и привязать новый хвостовой узел, после чего следует послать сигнал условной переменной.

В случае извлечения элемента необходимо заблокировать мьютекс головы очереди, после чего нужно дождаться сигнала от условной переменной, в предикате которой проверяется наличие элементов в очереди. Для этой проверки нужно заблокировать еще и мьютекс хвоста. Когда в очереди есть элементы, можно извлекать данные. При этих операциях мьютекс головы списка остается заблокированным до конца выполнения извлечения.

Исследование.

Исследование производительности в зависимости от количества производителей и потребителей производилось при помощи запуска программ в течение 100 миллисекунд на матрицах 20 * 20 при разном количестве потоков-производителей и потоков-потребителей. Количество выполненных умножений считалось как среднее среди 100 вызовов программ. Вызовы и подсчет производились скриптом, написанным на языке Python (см. рис. 1).

```python
from subprocess import Popen, PIPE

generated = []
resolved = []
outputed = []

print("Input program name:")
program = input()
print("Input number of starts:")
starts = int(input())
print()

for _ in range(starts):
    process = Popen(["./" + program], stdout=PIPE)
    (output, err) = process.communicate()
    data = output.decode("utf-8").split("\n")[0:-1]
    generated.append(int(data[0].split()[1]))
    resolved.append(int(data[1].split()[1]))
    outputed.append(int(data[2].split()[1]))
    process.wait()


def average(arr):
    return sum(arr) / len(arr)


print("Generated: ", average(generated))
print("Resolved: ", average(resolved))
print("Outputed: ", average(outputed))
```

Рисунок 1 – Скрипт для подсчета количества проведенных умножений.

Результаты для очереди с «грубой» блокировкой представлены в таблице (см. табл. 1).

Таблица 1 – Результаты для очереди с «грубой» блокировкой.

Производители	Потребители	Умножения
1	1	723
3	3	788
10	10	854
2	5	999
5	2	595

Результаты для очереди с «тонкой» блокировкой представлены в таблице (см. табл. 2).

Таблица 2 – Результаты для очереди с «тонкой» блокировкой.

Производители	Потребители	Умножения
1	1	722
3	3	1056
10	10	1191
2	5	1232
5	2	795

Из проведенного исследования можно сделать вывод, что программа, использующая очередь с «тонкой» блокировкой работает быстрее. Это связано с тем, что очередь с «тонкой» блокировкой позволяет уменьшить количество блокировок. Особенно эта разница заметна при большом количестве потоков.

Выводы.

В ходе лабораторной работы были изучены способы реализации потокобезопасных структур данных с блокировками. Были написаны 2 программы, выполняющие одну и ту же задачу, но использующие разные очереди: с «грубой» и «тонкой» блокировкой. Также было проведено исследование производительности задач в зависимости от количества потоков-производителей и потоков-потребителей. Был сделан вывод о выигрыше по времени в задаче, использующей очередь с «тонкой» блокировкой.