Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ОТЧЕТ
по лабораторной работе №6
на тему
“Работа с файлами, отображенными в память”

Выполнил: студент группы 350501
М.А. Василюк

Проверил: старший преподаватель каф. ЭВМ
Л.П. Поденок

Минск 2025

СОДЕРЖАНИЕ

1 ЗАДАНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

1.1 Цель работы

Написать многопоточную программу sort_index для сортировки вторичного индексного файла таблицы базы данных, работающую с файлом с использованием отображение файлов в адресное пространство процесса. Программа должна запускаться следующим образом:

```
$ sort_index memsize blocks threads filename
```

Параметры командной строки:

memsize – размер рабочего буфера, кратный размеру страницы (getpagesize(2))

blocks – порядок (количество блоков) разбиения буфера

threads – количество потоков (от k до N), где k – количество процессорных ядер, N – максимальное количество потоков (k <= N <= 8k). filename – имя файла

Количество блоков должно быть степенью двойки и превышать количество потоков не менее, чем в 4 раза (при k = 4 количество блоков не должно быть менее 16). Соответственно, раз- мер файла должен удовлетворять указанным ограничениям.

Для целей тестирования следует написать программу gen , которая будет генерировать неотсортированный индексный файл, и программу view для отображения индексного файла на stdout.

1.2 Исходные данные к работе

Алгоритм программы генерации

Генерируемый файл представляет собой вторичный индекс по времени и состоит из заголовка и индексных записей фиксированной длины.

Индексная запись имеет следующую структуру:

```
struct index_s {
double  time_mark; // временная метка (модифицированная юлианская дата) uint64_t recno; // номер записи в таблице БД (первичный индекс)
};
```

Заголовок представляет собой следующую структуру

```
struct index_hdr_s {
uint64_t recsords; // количество записей
struct index_s idx[]; // массив записей в количестве records
};
```

Временная метка определяется в модифицированный юлианских днях. Целая часть лежит в пределах от 15020.0 (1900.01.01-0:0:0.0) до «вчера». Дробная – это часть дня (0.5 – 12:0:0.0). Для генерации целой и дробной частей временной метки следует использовать системный генератор случайных чисел rand(3) или rand_r(3).

Первичный индекс, как вариант, может заполняться последовательно,

начиная с 1, но может быть случайным целым > 0 (в программе сортировки не используется).

Размер индекса в записях должен быть кратен 256 и кратно превышать планируемую выделенную память для отображения. Размер индекса и имя файла указывается при запуске программы генерации.

Алгоритм программы сортировки

1) Основной поток запускает `threads` потоков, сообщая им адрес буфера, размер блока `memsize`/`blocks`, и их номер от 1 до `threads` - 1, используя возможность передачи аргумента для `start_routine`. Порожденные потоки останавливаются на барьере, ожидая прихода основного.

2) Основной поток с номером 0 открывает файл, отображает его часть размером `memsize` на память и синхронизируется на барьере. Барьер «открывается» и все `threads` потоков входят на равных в фазу сортировки.

3) Фаза сортировки

С каждым из блоков связана карта (массив) отсортированных блоков, в которой изначально блоки с 0 по `threads`-1 отмечены, как занятые.

Поток n начинает с того, что выбирает из массива блок со своим номером и его сортирует, используя `qsort`(3). После того, как поток отсортировал свой первый блок, он на основе конкурентного захвата мьютекса, связанного с картой, получает к ней эксклюзивный доступ, отмечает следующий свободный блок, как занятый, освобождает мьютекс и приступает к его сортировке.

Если свободных блоков нет, синхронизируется на барьере. После прохождения барьера все блоки будут отсортированы.

4) Фаза слияния

Поскольку блоков степень двойки, слияния производятся парами в цикле.

Поток 0 сливает блоки 0 и 1, поток 1 – блоки 2 и 3, и так далее.

Для отметки слитых пар и не слитых используется половина карты. Если для потока нет пары слияния, он синхронизируется на барьере.

В результате слияния количество блоков, подлежащих слиянию сокращается в два раза, а размер их в два раза увеличивается.

После очередного прохождения барьера количество блоков, подлежащих слиянию, станет меньше количества потоков. В этом случае распределение блоков между потоками осуществляется на основе конкурентного захвата мьютекса, связанного с картой. Потоки, котором не досталось блока, синхронизируются на барьере.

Когда осталась последняя пара, все потоки с номером не равным нулю синхронизируются на барьере, о поток с номером 0 выполняет слияние последней пары.

После слияния буфер становится отсортирован и подлежит сбросу в файл (`munmap`()).

Если не весь файл обработан, продолжаем с шага 2.

Если весь файл обработан, основной поток отправляет запрос отмены порожденным потокам, выполняет слияние отсортированных частей файла и завершается.

Как вариант, потоки, которым не досталось блоков для слияния, завершаются.

2 ВЫПОЛНЕНИЕ РАБОТЫ

2.1 Описание алгоритма выполнения работы

Программа sort_index представляет собой многопоточное приложение, разработанное для операционной системы Fedora и предназначенное для сортировки вторичного индексного файла таблицы базы данных. Для достижения эффективной обработки больших объемов данных программа использует механизм отображения файлов в адресное пространство процесса, что позволяет минимизировать операции ввода-вывода и обеспечить прямой доступ к данным как к оперативной памяти; в основе алгоритма лежит парадигма разделяемой памяти и многопоточности POSIX, где главный поток выполняет ключевые функции управления и координации: он принимает аргументы командной строки, определяющие параметры работы программы, такие как размер рабочего буфера, количество блоков для разделения данных и количество потоков для параллельной обработки, открывает файл, отображает его части в память с использованием системного вызова mmap, создает рабочие потоки, , координирует процесс сортировки и слияния отсортированных блоков, используя барьеры и мьютексы для синхронизации потоков и обеспечения корректного обмена данными между ними, , если его размер превышает размер выделенного буфера, многократно отображая в память различные участки файла; потоки-исполнители, в свою очередь, отвечают за непосредственное выполнение алгоритма сортировки: они получают от главного потока указатели на выделенные им блоки данных, сортируют их, применяя алгоритм быстрой сортировки (qsort), и участвуют в последующих этапах слияния отсортированных блоков в более крупные последовательности, синхронизируя свою работу между собой и с главным потоком с помощью механизмов синхронизации POSIX для обеспечения целостности данных и предотвращения гонок.

2.2 Описание основных функций

Функция worker()

Функция worker() реализует логику работы каждого потока-исполнителя. Каждый поток получает уникальный идентификатор и указатель на область памяти, содержащую данные для сортировки. Потоки синхронизируются с использованием барьера, после чего каждый поток сортирует выделенный ему блок данных с помощью функции qsort(). Для эффективного использования ресурсов и распараллеливания процесса, потоки динамически захватывают и сортируют свободные блоки данных. После завершения сортировки всех блоков, потоки участвуют в фазе слияния, объединяя отсортированные блоки в более крупные последовательности. Этот процесс слияния также синхронизируется с использованием барьера и мьютекса для обеспечения корректности и предотвращения

конфликтов при доступе к разделяемым данным.

Функция compare()

Функция compare() определяет порядок сравнения двух элементов индекса (struct index_s) и используется функцией qsort() для сортировки блоков данных. Она принимает два указателя на элементы индекса и возвращает отрицательное значение, если первый элемент меньше второго, положительное значение, если первый элемент больше второго, и ноль, если элементы равны. Сравнение основывается на значении time_mark (временной метки) каждого элемента индекса, обеспечивая сортировку данных в хронологическом порядке.

Функция merge()

Функция merge() выполняет слияние двух отсортированных последовательностей элементов индекса в одну отсортированную последовательность. Она принимает указатели на начало двух входных последовательностей (a и b), их размеры (na и nb) и указатель на буфер для хранения результата слияния (dst). Функция сравнивает элементы из обеих входных последовательностей и копирует меньший элемент в результирующий буфер, последовательно перемещаясь по входным и выходному буферам. После обработки всех элементов одной из входных последовательностей, оставшиеся элементы другой последовательности копируются в результирующий буфер.

Функция main()

Функция main() служит точкой входа в программу sort_index. Её основная задача — обработка аргументов командной строки, инициализация необходимых ресурсов и запуск процесса сортировки. main() анализирует переданные аргументы, определяя размер рабочего буфера, количество блоков для разделения данных, количество потоков и имя файла, подлежащего сортировке. Затем открывает указанный файл, отображает его части в память с использованием mmap, инициализирует барьер и мьютекс для синхронизации потоков, создает потоки-исполнители и запускает процесс сортировки, координируя его выполнение. По завершении сортировки main() выполняет необходимые операции очистки, включая уничтожение барьера и мьютекса, снятие отображения памяти и закрытие файла.

3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
parallels@fedora-linux-40:~/Documents/tar_work-
ing_dir/ВасилюкМ.А./Lab06/build/debug$ ./gen file.dat 8
parallels@fedora-linux-40:~/Documents/tar_work-
ing_dir/ВасилюкМ.А./Lab06/build/debug$
```

Рисунок 1 – пример создания файла

```
parallels@fedora-linux-40:~/Documents/tar_work-
ing_dir/ВасилюкМ.А./Lab06/build/debug$ ./view file.dat
Records: 8
 45478.37293  1
 38748.99565  2
 23393.10063  3
 44472.12785  4
 52501.62376  5
 25551.51476  6
 15386.41490  7
 16231.12218  8
```

Рисунок 2 – пример чтения файла

```
parallels@fedora-linux-40:~/Documents/tar_work-
ing_dir/ВасилюкМ.А./Lab06/build/debug$ ./sort_index 4096 8
2 file.dat
[Main] records = 8
[Thread 1] started
[Thread 0] started
[Thread 0] sorted initial block 0
[Thread 0] sorted block 1
[Thread 0] sorted block 2
[Thread 0] sorted block 3
[Thread 0] sorted block 4
[Thread 0] sorted block 5
[Thread 0] sorted block 6
[Thread 0] sorted block 7
[Thread 1] merging blocks 0 and 1 (step 1)
[Thread 1] merging blocks 2 and 3 (step 1)
[Thread 1] merging blocks 4 and 5 (step 1)
[Thread 1] merging blocks 6 and 7 (step 1)
[Thread 0] merging blocks 0 and 2 (step 2)
[Thread 0] merging blocks 4 and 6 (step 2)
[Thread 1] merging blocks 0 and 4 (step 3)
[Main] final merge of blocks 0 to 4 and 4 to 8
```

Рисунок 3 – пример сортировки

4 ВЫВОД

В ходе выполнения данной работы была разработана программа `sort_index`, демонстрирующая эффективную сортировку больших объемов данных посредством применения многопоточности `POSIX` и механизма отображения файлов в память. Программа обеспечивает распараллеливание процесса сортировки, используя несколько потоков для одновременной обработки различных частей входного файла, что значительно ускоряет общее время выполнения. Для координации работы потоков и обеспечения целостности данных применяются барьеры и мьютексы. Каждый поток выделяет, сортирует, а затем сливает блоки данных. Разработанное решение наглядно иллюстрирует принципы эффективной организации параллельных вычислений, управления жизненным циклом потоков и применения примитивов многопоточности `POSIX` для решения задачи, требующей интенсивной обработки данных.