Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

Лабораторная работа №4
по курсу
«Операционные системы и системное программирование»
на тему
«Работа с файлами, отображенными в память»

Выполнил: студент группы 350501
Русак Г.Д.
Проверил: старший преподаватель каф. ЭВМ
Поденок Л.П.

Минск 2025

1 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

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 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. Соответственно, размер 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`.

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 блокамemsize/blocks, и их номер от 1 до threads - 1, используя возможность передачи аргумента длястart_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 и1, 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.

Потоки, которым не досталось блоков для слияния, завершаются.

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ

2.1 Общая структура

Разрабатываемое приложение осуществляет внешнюю сортировку бинарного файла, содержащего записи со временными метками. Работа строится на следующих принципах:

- Использование памяти, отображённой в адресное пространство процесса через mmap;
- Многопоточная сортировка блоков данных;
- Итеративное слияние отсортированных блоков по степеням двойки;
- Синхронизация потоков с использованием `pthread_barrier_t` и `pthread_mutex_t`.

Программа состоит из трёх утилит:

- Генератор входного файла;
- Основной модуль сортировки (`main`);
- Чтение результата.

2.2 Алгоритм сортировки

1) Инициализация:

- Проверка корректности аргументов командной строки: размер данных (кратно 256), число блоков (чётное), число потоков (от 2 до 8000);
- Расчёт размера одного блока: `block_size = size / blocks`;
- Инициализация барьера и мьютекса.

2) Чтение и отображение файла:

- Файл открывается в режиме `rb+`, затем отображается в память с помощью `mmap`;
- Указатель `cur` инициализируется на начало массива структур `index_record_t`.

3) Многопоточная сортировка блоков:

- Главный поток и дополнительные `threads - 1` потоков (через `pthread_create`) параллельно сортируют блоки с помощью qsort;
- Сортировка осуществляется по полю `time_mark`.

4) Итеративное слияние блоков

- Блоки объединяются парами с шагом `mergeStep`;
- Слияние производится в отдельных буферах (`left`, `right`), после чего результат копируется обратно.

5) Завершение

- Ожидание завершения всех потоков через `pthread_join`;
- Освобождение ресурсов (`munmap`, `fclose`, `pthread_mutex_destroy`, `pthread_barrier_destroy`).

2.3 Работа потоков

Каждый поток выполняет функцию `sort_in_memory`, в которой:

- Ждёт синхронизации через барьер;
- Последовательно получает блоки для сортировки через защищённый мьютексом указатель cur;
- После сортировки участвует в многократных фазах слияния блоков;
- По завершении всех итераций — освобождает память и синхронизируется с другими потоками.

2.4 Используемые структуры

При разработке использовались следующие структуры:

1) `index_record_t` представляет запись индекса, содержащую два поля:
- `double time_mark` — временная метка для сортировки;
- `uint64_t recno` — номер записи (уникальный идентификатор).

2) `index_hdr_t` представляет заголовок индекса. Поля структуры:
- `uint64_t  records` — количество записей в файле;
- `index_record_t* idx` — указатель на массив записей индекса.

3) `thread_args` представляет структуру для передачи параметров потока сортировки:
- `index_record_t buf` — указатель на буфер данных для сортировки;
- `int block_size` — размер блока для сортировки;
- `int thread_num` — идентификатор потока.

4) `file_sort_args` структуру для передачи параметров сортировки файла. Поля структуры:
- `int block_size` — размер блока данных для сортировки;
- `int threads` — количество потоков для сортировки;
- `char* file_name` — имя файла, который нужно отсортировать.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Система состоит из нескольких ключевых модулей, каждый из которых выполняет определенную функцию.

Модуль чтения файла (`read`) выполняет следующие функции:

1) Инициализация системы. Открытие файла с данными и выделение памяти для хранения заголовка и записей индекса;

2) Чтение количества записей (`records`) и индекса записей (`idx`) из файла;

3) Печать временной метки каждой записи индекса на экран;

4) Закрытие файла и освобождение выделенной памяти.

Модуль генерации файла (`generator`) выполняет следующие функции:

1) Инициализация системы. Открытие файла и выделение памяти для хранения заголовка и индекса;

2) Генерация случайных данных. Генерация случайных записей с временными метками и заполнение массива индекса;

3) Запись в файл. Запись количества записей и индекса в файл;

4) Завершение работы. Закрытие файла и освобождение памяти, занятой массивом индекса.

Модуль сортировки (`func`) выполняет следующие функции:

1) Открытие файла `open_file_or_exit()`;

2) Получение размера файла `get_file_size()`;

3) Инициализация синхронизации `init_barrier_mutex()`;

4) Сортировка блоков `sort_block()`;

5) `Слияние блоков merge_blocks()`;

6) Сортировка в памяти `sort_in_memory()`;

7) Сортировка файла в памяти `sort_file_in_memory()`.

Модуль управления многопоточной сортировкой (`sort_index`) выполняет следующие функции:

1) Проверка аргументов командной строки;

2) Инициализация синхронизации;

3) Подготовка параметров сортировки;

4) Создание потока сортировки;

5) Ожидание завершения работы потока;

6) Освобождение ресурсов.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА ПРОЕКТА

Порядок сборки и запуска состоит в следующем:

1) Клонировать репозиторий, используя команду

`$git clone https://github.com/Everolfe/lab06-osasp`,

или разархивировать каталог с проектом;

2) Перейти в каталог с проектом

`$cd lab06-osasp`,

или

`$cd "Русак Г.Д./lab06"`;

3) Собрать проект используя `make`;

4) После сборки проекта можно использовать, прописав

```
$./build/release/generator 256 testfile
$./build/release/generator 256 testfile
$./build/release/sort_index 256 16 4 testfile
```

5 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
grusak@fedora:~/tar_working_dir/Русак           Г.Д./lab06$
./build/release/generator 256 testfile
grusak@fedora:~/tar_working_dir/Русак           Г.Д./lab06$
./build/release/read testfile | wc
    256     256    3328
grusak@fedora:~/tar_working_dir/Русак           Г.Д./lab06$
./build/release/sort_index 256 32 6 testfile
Sort in 5 thread.
Sort in 0 thread.
Sort in 3 thread.
Merging in 3 thread.
Merging in 0 thread.
Sort in 4 thread.
Merging in 4 thread.
Sort in 1 thread.
Merging in 1 thread.
Sort in 2 thread.
Merging in 2 thread.
Merging in 5 thread.
grusak@fedora:~/tar_working_dir/Русак           Г.Д./lab06$
./build/release/read testfile | wc
    256     256    3328
15700.220266
15755.613021
15783.051887
16210.638403
16272.211343
17669.453553
17674.620127
17756.851019
17837.070162
17928.001238
18005.577014
18053.610498
18233.794225
18309.325602
18393.463333
18565.090023
18692.388819
18709.364282
19190.480972
19339.529225
19401.840324
19418.912986
19716.674815
20130.437037
20141.170868
```