Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

Лабораторная работа №6
по курсу
«Операционные системы и системное программирование»
на тему
«Работа с файлами, отображенными в память»

Выполнил: студент группы 350501
Слепица О.Н.
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 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 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

Приложение выполняет многопоточную сортировку записей бинарного файла с временными метками, используя итеративное слияние и память, отображённую в адресное пространство процесса через mmap. Работа строится на следующих принципах:

– Использование памяти, отображённой в адресное пространство процесса через `mmap`;

– Многопоточная сортировка блоков данных;

– Итеративное слияние отсортированных блоков;

– Синхронизация потоков с использованием `pthread_barrier_t` и `pthread_mutex_t`.

Программа состоит из трёх утилит:

– Генератор входного файла;

– Основной модуль сортировки;

– Утилита для чтения и проверки результата.

2.2 Алгоритм сортировки

1) Инициализация:

– Проверка аргументов командной строки (`memsize`, `blocks`, `threads`, `filename`);

– Расчёт размера одного блока: `records_per_block = memsize / blocks`;

– Инициализация барьера (`pthread_barrier_t`) и мьютекса (`pthread_mutex_t`).

2) Чтение и отображение файла:

– Открытие файла в режиме `O_RDWR`;

– Получение размера файла через `fstat`;

– Проверка данных (`header_size + data_size == file_size`);

– для отображения данных в память.

3) Многопоточная сортировка блоков:

– Главный поток создаёт `threads - 1` потоков через `pthread_create`;

– Каждый поток сортирует выделенный блок (`qsort` по `time_mark`);

– Потоки получают новые блоки через `pthread_mutex_t`.

4) Итеративное слияние блоков:

– Блоки объединяются парами с шагом `merge_size *= 2`;

– Слияние производится в временный буфер;

– Результаты копируются обратно в отображённую память.

5) Завершение:

– Ожидание завершения всех потоков через `pthread_join`;

– Освобождение ресурсов (`munmap`, `fclose`, `pthread_mutex_destroy`, `pthread_barrier_destroy`);

– Финальная проверка отсортированных данных.

2.3 Работа потоков

Каждый поток выполняет функцию `thread_worker`, в которой:

– Ждёт синхронизации;

– Последовательно получает блоки через `pthread_mutex_t`;

– Сортирует блок (`qsort`);

– Выполняет итеративное слияние блоков;

– По завершении всех итераций — освобождает память и синхронизируется с другими потоками.

2.4 Используемые структуры

При разработке использовались следующие структуры:

1) `index_s` представляет запись индекса, содержащую два поля:

– `double time_mark` – временная метка для сортировки;

– `uint64_t recno` – номер записи (уникальный идентификатор).

2) `index_hdr_s` представляет заголовок индекса. Поля структуры:

– `uint64_t records` – количество записей в файле;

– `index_x idx[]` – массив записей индекса.

3) `thread_info_t` представляет структуру для передачи параметров потока сортировки:

– `size_t block_size` – размер блока;

– `index_s *buffer` – указатель на данные;

– `int blocks` – число блоков;

– `int threads` – число потоков;

– `pthread_barrier_t *barrier` – барьер синхронизации;

– `pthread_mutex_t *map_mutex` – мьютекс для блокировки;

– `int *block_map` – карта отсортированных блоков;

– `int *merge_map` – карта слияния.

4) `file_sort_args` структура 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 определённые функции.

Модуль чтения файла (`read`) отвечает за анализ и проверку содержимого файла индексов:

1) Инициализация. Открытие файла и отображение его в память (`mmap`);

2) Чтение заголовка. Извлечение количества записей (`records`) и массива индексов (`idx`);

3) Вывод данных;

4) Завершение работы. Закрытие файла и освобождение памяти.

Модуль генерации файла (`generator`) отвечает за создание тестового бинарного файла с записями:

1) Инициализация. Открытие файла и выделение памяти;

2) Генерация данных с временными метками (`time_mark`);

3) Запись в файл;

4) Освобождение памяти и закрытие файла.

Модуль сортировки (`func`) содержит вспомогательные алгоритмы сортировки и слияния:

1) Сортировка записей (`qsort`);

2) Слияние блоков (`merge_blocks`);

3) Финальное слияние (`merge_sorted_chunks`). Итеративное объединение всех блоков в единую отсортированную последовательность;

5) Проверка упорядоченности (`is_sorted`).

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

1) Проверка аргументов командной строки;

2) Инициализация потоков;

3) Сортировка блоков.;

4) Итеративное слияние;

5) Ожидание завершения работы потоков;

6) Освобождение ресурсов.

4 ПОРЯДОК СБОРКИ И ЗАПУСКА ПРОЕКТА

Порядок сборки и запуска состоит в следующем:

1) Разархивировать каталог с проектом;

2) Перейти в каталог с проектом:

```
$cd "Слепица О.Н./lab06";
```

3) Собрать проект используя `make`;

4) После сборки проекта можно использовать, прописав

```
$./build/release/generator 1000000 testfile
```

```
$./build/release/view testfile
```

```
$./build/release/sort_index 4096000 16 4 testfile
```

5 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
helga@fedora:~/tar_working_dir/Слепица О.Н./lab06$
./build/release/generator 1000000 test.idx
Adjusting record count to 1000192 to make it a multiple of 256
Generated index file 'test.idx' with 1000192 records (15.26 MB)
helga@fedora:~/tar_working_dir/Слепица О.Н./lab06$
./build/release/sort_index 4096000 16 4 test.idx
File: test.idx
File size: 16003080 bytes (15.26 MB)
Total records from header: 1000192
Memory mapping size: 4096000 bytes (3.91 MB)
Blocks: 16
Records per block: 16000
Threads: 4
Processed chunk of 256000 records (3.91 MB). Remaining: 744192
records
Processed chunk of 256000 records (3.91 MB). Remaining: 488192
records
Processed chunk of 256000 records (3.91 MB). Remaining: 232192
records
Processed chunk of 232192 records (3.54 MB). Remaining: 0
records
Performing final merge of all chunks...
Verifying final sort...
File is SORTED
Sorting completed successfully.
helga@fedora:~/tar_working_dir/Слепица О.Н./lab06$
./build/release/view test.idx
Index file: test.idx
Total records: 1000192
File size: 15.26 MB
Time Mark       Record Number
------------------------------------
32457.23335     1
38083.82505     2
20248.72978     3
49508.01738     4
48272.32288     5
54397.55479     6
27101.38598     7
49021.51052     8
18630.87765     9
49353.77304     10
...
31686.47641     1000192              (last record)

The index is NOT SORTED (499928 violations)
```