СОДЕРЖАНИЕ

1 Условия лабораторной работы

Кооперация потоков для высокопроизводительной обработки больших файлов.

Изучаемые системные вызовы: pthread_barrier_init(), pthread_barrier_destroy(), pthread_barrier_wait(), mmap(), munmap().

Задание

Написать многопоточную программу sort_index для сортировки вторичного индексного файла таблицы базы данных, работающую с файлом в двух режимах: read()/write() и с использованием отображение файлов в адресное пространство процесса. Программа должна запускаться следующим образом:

sort_index memsize granul threads filename

Параметры командной строки:

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

blocks – порядок разбиения буфера

threads – количество потоков (от k до N)

k – количество ядер

N – максимальное количество потоков (8k)

filename – имя файла

Количество блоков должно быть степенью двойки и превышать количество потоков.

Для целей тестирования следует написать программу gen для генерации неотсортированного индексного файла и программу view для отображения индексного файла на stdout.

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
} index_record;
```

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 используется системный генератор случайных чисел (random(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 Описание алгоритмов и решений

Программа генерации индексного файла (gen) имеет следующий алгоритм.

Шаг 1. Программа принимает два аргумента:

– num_records: количество записей (должно быть положительным и кратно 256).

– filename: имя файла для записи.

Шаг 2. Проверка входных данных. Проверяется, что количество записей больше нуля и делится на 256.

Шаг 3. Файл открывается для записи в бинарном режиме.

Шаг 4. В файл записывается заголовок, содержащий количество записей.

Шаг 5. Выделяется массив структур индексных записей.

Шаг 6. Инициализация генератора случайных чисел.

Шаг 7. Вычисление границ временной метки.

Шаг 8. Генерация каждой записи.

Шаг 9. Запись массива в файл.

Шаг 10. Завершение работы. Освобождается выделенная память, файл закрывается.

Программа просмотра индексного файла (view) имеет следующий алгоритм.

Шаг 1. Ввод параметров. Программа принимает имя файла в качестве единственного аргумента.

Шаг 2. Файл открывается для чтения в бинарном режиме.

Шаг 3. Чтение заголовка. Считываются первые 8 байт файла, которые содержат количество записей.

Шаг 4. Последовательное чтение записей. Для каждой из считанных записей читается структура записи, а затем выводится на стандартный вывод с форматом отображения значений.

Шаг 5. Завершение работы. После вывода всех записей файл закрывается, и программа завершает свою работу.

Программа сортировки индексного файла (sort_index) обрабатывает файл сегментами, сортируя каждый сегмент с помощью многопоточной обработки и последующего объединения отсортированных блоков.

Шаг 1. Ввод параметров. Программа принимает четыре параметра:

– memsize: размер рабочей области (обладание должно быть кратно размеру страницы).

– granul: порядок разбиения буфера (число блоков = 2^granul).

– threads: число потоков сортировки.

– filename: имя файла.

Шаг 2. Валидация параметров.

Шаг 3. Считывается размер файла. Получается системный размер страницы.

Шаг 4. Инициализация логического смещения. Переменная logical_offset устанавливается в 0 и будет обновляться по мере обработки сегментов.

Шаг 5. Обработка сегментов файла. Цикл повторяется, пока не обработан весь файл:

Шаг 5.1. Определение размера сегмента

Шаг 5.2. Выравнивание для mmap. aligned_offset: логическое смещение округляется до ближайшей границы страницы. adjustment: разница между логическим смещением и выровненным смещением. map_length: вычисляется как adjustment + effective_length.

Шаг 5.3. Проверка минимального размера сегмента

Шаг 5.4. Отображение файлового сегмента в память.

Шаг 5.5. Подсчет числа записей в сегменте

Шаг 6. Фаза сортировки.

Шаг 6.1. Определение начала данных.

Шаг 6.2. Инициализация структуры сортировки

Шаг 6.3. Запуск потоков сортировки

Шаг 6.4. Каждый поток сортирует блок с номером, равным его идентификатору, затем получает и сортирует следующие свободные блоки (по нарастающему индексу).

Шаг 6.5. После обработки всех блоков потоки синхронизируются на барьере.

Шаг 7. Фаза слияния.

Шаг 7.1. Инициализация структуры слияния.

Шаг 7.2. Запуск потоков слияния.

Шаг 7.3. В цикле:

Шаг 7.3.1. Определяется количество пар для слияния; если идентификатор потока меньше количества пар, он сливает блоки с номерами 2i и 2i+1, используя временный буфер.

Шаг 7.3.2. Все потоки ожидают синхронизации на барьере после каждой фазы слияния. Один поток обновляет общее число блоков и удваивает размер блока.

Шаг 7.3.3. Цикл повторяется до тех пор, пока не останется один отсортированный блок.

Шаг 7.4. Ожидается завершение работы всех потоков слияния.

Шаг 8. Запись и завершение обработки сегмента.

Шаг 8.1. Запись результата в файл.

Шаг 8.2. Очистка ресурсов сегмента

Шаг 8.3. Обновление логического смещения

Шаг 9. Повтор цикла с шага 5. Переход к следующему сегменту, пока весь файл не обработан.

Шаг 10. Завершение работы

3 Функциональная структура проекта

Проект состоит из нескольких модулей. Описание структуры программы сортировки приведено ниже.

Модуль index определяет базовую структуру индексной записи, которая используется на всех этапах обработки (как входной тип для сортировки, слияния и записи).

Структура index_s представляет одну индексную запись.

```
struct index_s {
    double time_mark; // временная метка
    uint64_t recno;   // номер записи в таблице БД
};
```

Модуль scan отображает файл в память. Выполняет начальную подготовку данных для сортировки, включая извлечение заголовка и подготовку массива индексных записей.

Структура scan_data содержит дескриптор файла, указатель на отображённую память, размер отображения, количество записей и указатель на массив записей.

```
typedef struct {
    int fd;                 // Дескриптор файла
    void *map_ptr;          // Адрес отображённой области
    size_t memsize;         // Размер отображённой области
    uint64_t total_records; // Количество записей (из заголовка)
    struct index_s *records; // Указатель на массив записей
(начало после заголовка)
} scan_data;
```

scan_file_segment() – функция отображения заданного сегмента файла в память с учетом выравнивания.

free_scan_data() – освобождает ресурсы, связанные с отображённой областью.

Модуль sort предназначен для многопоточной сортировки блоков индексных записей. Каждому потоку изначально назначается свой блок, затем при помощи мьютекса остальные свободные блоки получают для сортировки.

Структура sort_data содержит указатель на записи, размер блока, количество блоков, индекс следующего свободного блока, мьютекс и барьер.

```
typedef struct {
    struct index_s *records;    // Массив записей, который нужно
сортировать блоками
    size_t block_size;          // Число записей в одном блоке
    int num_blocks;             // Общее число блоков
    int next_block;               // Следующий свободный блок для
сортировки
    pthread_mutex_t mutex;      // Мьютекс для защиты next_block
    pthread_barrier_t barrier;  // Барьер для синхронизации
завершения сортировки
} sort_data;
```

Структура sort_thread_arg передаёт идентификатор потока и указатель на sort_data.

```
typedef struct {
    int thread_id;      // Идентификатор потока (0,1,...)
    sort_data *sd;      // Указатель на общую структуру параметров сортировки
} sort_thread_arg;
```

sort_init() инициализирует структуру сортировки.

sort_blocks() – функция, которую выполняют потоки для сортировки назначенных блоков с использованием qsort.

Модуль merge отвечает за слияние отсортированных блоков. После этапа сортировки блоков, несколько потоков параллельно сливают пары блоков в один, до получения одного общего отсортированного блока.

Структура merge_data содержит указатель на записи, текущий размер блока, число оставшихся блоков, мьютекс и барьер.

```
typedef struct {
    struct index_s *records;
    size_t block_size;    // размер одного блока (в записях)
    int num_blocks;       // текущее количество блоков
    pthread_mutex_t  mutex;        //  резерв  для  возможного динамического распределения (не используется в этом упрощённом варианте)
    pthread_barrier_t barrier;  // синхронизация фаз слияния
} merge_data;
```

merge_thread_arg содержит идентификатор потока, общее число потоков и указатель на merge_data.

```
typedef struct {
    int thread_id;
    int num_threads;
    merge_data *md;
} merge_thread_arg;
```

merge_init() инициализирует структуру для слияния (создает мьютекс и барьер).

merge_two_sorted_blocks() объединяет две отсортированные части в один массив.

merge_blocks_phase() – функция, выполняемая потоками, которая сливает назначенные пары блоков и синхронизирует обновление общего состояния.

Модуль write предназначен для записи отсортированного результата обратно в файл. Синхронизирует изменения, сделанные в отображённой области памяти, с содержимым файла.

write_sorted_result() выполняет синхронизацию и обеспечивает запись изменений на диск.

Модуль finish очищает и разрушает синхронизационные объекты.

finish_cleanup_prep(): разрушает глобальный барьер, используемый в начальной фазе.

finish_cleanup_sort(sort_data *sd): разрушает барьер и мьютекс внутри структуры сортировки.

finish_cleanup_merge(merge_data *md): разрушает барьер и мьютекс внутри структуры слияния.

Программа генерации (gen) выполняет генерацию неотсортированного индексного файла с заданным числом записей.

Программа просмотра (view) выполняет просмотр содержимого индексного файла. Программа читает заголовок и последовательно выводит данные каждой индексной записи на стандартный вывод.

4 Порядок сборки и тестирования

Для сборки используется Makefile, содержащий следующие ключевые элементы:

– Исходные файлы находятся в каталоге ./src.

– Заголовочные файлы расположены в каталоге ./include.

– В зависимости от типа сборки создаётся каталог ./out/debug для debug-версии или ./out/release для release-версии.

– Переменная BUILD задаётся как debug по умолчанию. При запуске make release устанавливается оптимизированная сборка.

– Флаги компиляции задаются через переменные COMMON_CFLAGS (определения, предупреждения, стандарт C11) и дополняются флагами отладки (-g -O0) или оптимизации (-O2).

Для сборки в debug-режиме необходимо выполнить в терминале:

```
make
```

Это соберёт программы в ./out/debug/.

Для сборки в release-режиме необходимо выполнить в терминале:

```
make release
```

Это соберёт программы в ./out/release/.

Для очистки сборки необходимо выполнить в терминале:

```
make clean
```

Для тестирования используется скрипт test_debug.sh и test_release.sh.

```
#!/bin/bash
# Скрипт для тестирования всего конвейера проекта

# 1. Генерация файла с заданным числом записей.
#    В данном случае 131072 записей (131072 кратно 256 и размер файла будет ~2МБ)
echo "Генерация файла с 131072 записями..."
out/release/gen 131072 test/testfile

# 2. Первичный просмотр файла.
#    Выводим только первые 10 строк и последние 10 строк.
echo "Просмотр файла до сортировки (первая часть):"
out/release/view test/testfile | head -n 10
```

```
echo "Просмотр файла до сортировки (последняя часть):"
out/release/view test/testfile | tail -n 10

# 3. Сортировка файла.
#    Параметры: memsize = 1048576 байт (1 МБ), granul = 4 (2^4=16
блоков), threads = 8.
echo "Сортировка файла..."
out/release/sort_index 1048576 4 8 test/testfile

# 4. Просмотр файла после сортировки.
echo "Просмотр файла после сортировки (первая часть):"
out/release/view test/testfile | head -n 10
echo "Просмотр файла после сортировки (последняя часть):"
out/release/view test/testfile | tail -n 10
```

5 Методика и результаты тестирования

Ниже приведён порядок тестирования всего проекта, отражённый в скриптах test_*.sh.

Шаг 1. Генерация файла с индексными записями.

Программа генерации (gen) запускается с параметрами 131072 (число записей) и testfile (имя выходного файла).

```
echo "Генерация файла с 131072 записями..."
out/debug/gen 131072 testfile
```

Ожидается создание тестового индексного файла. Число записей выбрано таким образом, что оно кратно 256 и размер файла получается примерно 2 МБ.

Шаг 2. Первичный просмотр файла до сортировки.

Программа просмотра (view) запускается для файла testfile. Вывод команды разделён на две части:

– Первая 10 строк выводятся с помощью head -n 10.

– Последние 10 строк выводятся с помощью tail -n 10.

```
echo "Просмотр файла до сортировки (первая часть):"
out/debug/view testfile | head -n 10
echo "Просмотр файла до сортировки (последняя часть):"
out/debug/view testfile | tail -n 10
```

Ожидаемый результат: на экране должны быть выведены первые и последние 10 строк неотсортированного файла.

```
Просмотр файла до сортировки (первая часть):
Number of records: 131072
Record 1: time_mark = 26830.606183, recno = 1
Record 2: time_mark = 53419.786616, recno = 2
Record 3: time_mark = 35381.408752, recno = 3
Record 4: time_mark = 50720.205705, recno = 4
Record 5: time_mark = 55970.732981, recno = 5
Record 6: time_mark = 25659.985967, recno = 6
Record 7: time_mark = 52001.384628, recno = 7
Record 8: time_mark = 50807.173442, recno = 8
Record 9: time_mark = 29646.322070, recno = 9
Просмотр файла до сортировки (последняя часть):
Record 131063: time_mark = 21670.380860, recno = 131063
Record 131064: time_mark = 32822.024793, recno = 131064
Record 131065: time_mark = 44272.576357, recno = 131065
Record 131066: time_mark = 15624.006506, recno = 131066
Record 131067: time_mark = 51994.259787, recno = 131067
Record 131068: time_mark = 60581.253881, recno = 131068
Record 131069: time_mark = 57880.217038, recno = 131069
Record 131070: time_mark = 35254.036550, recno = 131070
Record 131071: time_mark = 36647.096622, recno = 131071
Record 131072: time_mark = 55616.321716, recno = 131072
```

3. Сортировка файла.

Программа сортировки (sort_index) запускается с параметрами:

– memsize = 1048576 байт (1 МБ) – размер рабочей области для отображения файла.

– granul = 4 – что означает разбиение области на 2^4 = 16 блоков.

– threads = 8 – число потоков, задействованных в сортировке.

– testfile – имя файла для сортировки.

```
echo "Сортировка файла..."
out/debug/sort_index 1048576 4 8 testfile
```

Ожидаемый результат: файл testfile после сортировки содержит отсортированный массив индексных записей, где каждая запись упорядочена по значению time_mark.

```
Сортировка файла...
Отсортированный результат успешно записан в файл.
```

Шаг 4. Просмотр файла после сортировки.

После завершения сортировки программа просмотра запускается снова в два этапа:

– Первая 10 строк выводятся с помощью head -n 10.

– Последние 10 строк выводятся с помощью tail -n 10.

```
echo "Просмотр файла после сортировки (первая часть):"
out/debug/view testfile | head -n 10
echo "Просмотр файла после сортировки (последняя часть):"
out/debug/view testfile | tail -n 10
```

Ожидаемый результат: просмотр файла покажет упорядоченный по возрастающей временной метке массив: первые 10 строк демонстрируют минимальные значения, а последние 10 – максимальные.

```
Просмотр файла после сортировки (первая часть):
Number of records: 131072
Record 1: time_mark = 15021.383548, recno = 59543
Record 2: time_mark = 15021.498715, recno = 42557
Record 3: time_mark = 15022.045726, recno = 43796
Record 4: time_mark = 15022.660836, recno = 40583
Record 5: time_mark = 15023.385767, recno = 3555
Record 6: time_mark = 15023.588835, recno = 36439
Record 7: time_mark = 15023.627069, recno = 19218
```

```
Record 8: time_mark = 15024.825050, recno = 8303
Record 9: time_mark = 15025.405966, recno = 62906
Просмотр файла после сортировки (последняя часть):
Record 131063: time_mark = 60802.100405, recno = 98092
Record 131064: time_mark = 60802.165519, recno = 123243
Record 131065: time_mark = 60802.371751, recno = 77113
Record 131066: time_mark = 60803.137121, recno = 125121
Record 131067: time_mark = 60806.141560, recno = 74124
Record 131068: time_mark = 60806.562920, recno = 117307
Record 131069: time_mark = 60808.317404, recno = 92142
Record 131070: time_mark = 60808.664820, recno = 84578
Record 131071: time_mark = 60810.251820, recno = 122053
Record 131072: time_mark = 60810.494503, recno = 76742
```