Министерство образования Республики Беларусь
Учреждение Образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных машин

Лабораторная работа № 6
«Работа с файлами, отображенными в память»

Проверил: Поденок Л.П.
Выполнил: ст. гр. 350501
Маслаков Н.А.

Минск 2025

1. УСЛОВИЕ ЛАБАРАТОРНОЙ 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, 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;
uint64_t recno;
};
```

Заголовок представляет собой следующую структуру

```
struct index_hdr_s {
uint64_t
recsords;
struct index_s idx[];
};
```

Временная метка определяется в модифицированный юлианских днях. Целая часть лежит в пределах от 15020.0 (1900.01.01-0:0:0.0) до «вчера»8. 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 напамять и синхронизируется на барьере. Барьер «открывается» и все threads потоков входят наравных в фазу сортировки.

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.

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.

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

В рамках нашей работы мы реализовали три утилиты: `gen`, `view` и `sort_index`, каждая из которых отвечает за отдельный этап обработки вторичного индексного файла. Утилита `gen` порождает неупорядоченный файл, начинающийся с 64-битного заголовка, хранящего общее число записей, а затем массива структур с полями `time_mark` (модифицированная юлианская дата) и `recno` (номер записи). Для генерации временных меток мы используем вызовы `rand_r` с инициализацией по комбинации `getpid()` и индекса записи, а дробную часть дня получаем из деления на число секунд в сутках. Размер выходного файла гарантированно кратен 256 и заранее задаётся пользователем.

Утилита `view` загружает весь файл с помощью `mmap` и выводит на `stdout` содержимое записей в текстовом формате, используя спецификатор `PRIu64` для корректного отображения 64-битного поля `recno`. Благодаря отображению файла в память чтение идёт без лишних системных вызовов на каждый доступ, что позволяет быстро выводить даже большие файлы.

Основная разработка посвящена программе `sort_index`, которая разбивает файл на чанки по `memsize` байт и обрабатывает их по очереди. Каждый чанк дополнительно делится на `blocks` равных блоков, а внутри одного чанка запускается ровно `threads` потоков, включая главный, который выступает как «поток 0». Потоки синхронизируются на первом `pthread_barrier_wait`, ожидании которого завершается `mmap` и подготовка структур. Затем каждый поток сначала выполняет `qsort` своего блока, после чего в цикле под защитой `pthread_mutex` динамически захватывает и сортирует оставшиеся блоки до тех пор, пока карта `block_map` не опустеет. Второй барьер гарантирует, что все блоки чанка отсортированы.

После локальной сортировки запускается фаза многоуровневого слияния. В каждом раунде потоки, чей `thread_id` меньше числа пар, сначала статически сливают свои пары блоков, а затем остальные пары распределяются динамически по карте `merge_map`, защищённой тем же мьютексом. Между началом и концом каждого раунда используется `pthread_barrier_wait`, чтобы обеспечить консистентность данных. Как только все подмассивы чанка сведены в единый отсортированный массив, главный поток вызывает `msync(..., MS_SYNC)` для принудительной записи обратно в файл и `munmap` для отключения отображения. После этого смещение `data_offset` увеличивается на `memsize`, и цикл повторяется вплоть до конца файла.

3. ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА

Утилита `sort_index.`

`static void  usage().`

Функция `usage` выводит информацию о верных аргументах программы при ошибках ввода.

`static void *worker_thread(void *arg).`

Функция `worker_thread` используется для задания поведения потоков при сортировке и слиянии.

Принимает аргумент `arg` - структура `thread_arg_t`, которая содержит айди потока и его размер блока.

`static int  cmp_index_s(const void *a, const void *b).`

Функция `cmp_index_s` используется как компоратор для `qsort`.

`static void  do_merge(size_t idx, size_t span, struct index_s *base).`

Функция `do_merge` используется слияния пары отсортированных подмассивов.

Принимает аргументы:

1) `size_t idx`. Порядковый номер пары.
2) `size_t span`. Длина каждого подмассива.
3) `index_s *base`. Указатель на начало чанка в памяти.

Утилита `gen.`

`static double current_mjd(void).`

Функция `current_mjd` используется для получения текущего времени в юлианских днях.

В мейне данной утилиты генерация файла с заполнениям его записями согласно условию работы.

Утилита `view.`

В мейне данной утилиты происходит отображение файла в память и его считывание и вывод в консоль.

4. ПОРЯДОК СБОРКИ И ЗАПУСКА

1) Перейти в каталог проекта.

```
$ cd 'Маслаков Н.А./lab06'
```

2) Собрать проект с помощью `make`. по умолчанию сборка происходит в режиме отладки.

```
$ make
```

3) Сгенерировать файл с записями.

```
$ build/debug/gen 512 data.idx
```

4) Запустить сортировку.

```
$ build/debug/sort_index 4096 256 20 data.idx
```

5. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

```
     ~/lab06$ valgrind --leak-check=full --show-leak-kinds=all --
track-origins=yes build/debug/gen 512 data.idx
==9210== Memcheck, a memory error detector
==9210== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et
al.
==9210== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright
info
==9210== Command: build/debug/gen 512 data.idx
==9210==
==9210==
==9210== HEAP SUMMARY:
==9210==     in use at exit: 0 bytes in 0 blocks
==9210==   total heap usage: 13 allocs, 13 frees, 5,527 bytes
allocated
==9210==
==9210== All heap blocks were freed -- no leaks are possible
==9210==
==9210== For lists of detected and suppressed errors, rerun with: -s
==9210== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
0)

     ~/lab06$ valgrind --leak-check=full --show-leak-kinds=all --
track-origins=yes  build/debug/view data.idx
==9215== Memcheck, a memory error detector
==9215== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et
al.
==9215== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright
info
==9215== Command: build/debug/view data.idx
==9215==
  0: time = 40404.565465, recno = 1
  1: time = 41503.811884, recno = 2
  2: time = 16256.942103, recno = 3
  3: time = 32699.114642, recno = 4
  4: time = 47199.883120, recno = 5
  5: time = 37327.734802, recno = 6
  6: time = 28053.706911, recno = 7
…
508: time = 21643.959345, recno = 509
509: time = 60209.286938, recno = 510
510: time = 21199.739047, recno = 511
511: time = 43393.431010, recno = 512
```

```
==9215==
==9215== HEAP SUMMARY:
==9215==     in use at exit: 0 bytes in 0 blocks
==9215==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==9215==
==9215== All heap blocks were freed -- no leaks are possible
==9215==
==9215== For lists of detected and suppressed errors, rerun with: -s
==9215== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
0)

     ~/lab06$ valgrind --leak-check=full --show-leak-kinds=all --
track-origins=yes build/debug/sort_index 4096 256 20 data.idx
==9217== Memcheck, a memory error detector
==9217== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et
al.
==9217== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright
info
==9217== Command: build/debug/sort_index 4096 256 20 data.idx
==9217==
OK: memsize=4096, blocks=256, threads=20, records=512

--- Chunk | offset=8 (page=0,page_offset=8,len=4104) ---
[T12] local sort done
[T19] local sort done
[T 5] local sort done
[T 0] local sort done
[T 4] local sort done
[T 1] local sort done
[T 2] local sort done
[T11] local sort done
[T 9] local sort done
[T 3] local sort done
[T18] local sort done
[T 8] local sort done
[T 7] local sort done
[T15] local sort done
[T14] local sort done
[T10] local sort done
[T16] local sort done
[T17] local sort done
[T13] local sort done
[T 6] local sort done
[T 0] start merge round 0 (pairs=128)
```

```
[T 0] merge round 0 done
[T 0] start merge round 1 (pairs=64)
[T 0] merge round 1 done
[T 0] start merge round 2 (pairs=32)
[T 0] merge round 2 done
[T 0] start merge round 3 (pairs=16)
[T 0] merge round 3 done
[T 0] start merge round 4 (pairs=8)
[T 0] merge round 4 done
[T 0] start merge round 5 (pairs=4)
[T 0] merge round 5 done
[T 0] start merge round 6 (pairs=2)
[T 0] merge round 6 done
[T 0] start merge round 7 (pairs=1)
[T 0] merge round 7 done
>>> Chunk | offset=8 sorted and written back <<<

--- Chunk | offset=4104 (page=4096,page_offset=8,len=4104) ---
[T10] local sort done
[T 3] local sort done
[T 6] local sort done
[T12] local sort done
[T16] local sort done
[T 5] local sort done
[T17] local sort done
[T 9] local sort done
[T 2] local sort done
[T 4] local sort done
[T14] local sort done
[T13] local sort done
[T 7] local sort done
[T 1] local sort done
[T 8] local sort done
[T 0] local sort done
[T18] local sort done
[T11] local sort done
[T15] local sort done
[T19] local sort done
[T 0] start merge round 0 (pairs=128)
[T 0] merge round 0 done
[T 0] start merge round 1 (pairs=64)
[T 0] merge round 1 done
[T 0] start merge round 2 (pairs=32)
[T 0] merge round 2 done
```

```
[T 0] start merge round 3 (pairs=16)
[T 0] merge round 3 done
[T 0] start merge round 4 (pairs=8)
[T 0] merge round 4 done
[T 0] start merge round 5 (pairs=4)
[T 0] merge round 5 done
[T 0] start merge round 6 (pairs=2)
[T 0] merge round 6 done
[T 0] start merge round 7 (pairs=1)
[T 0] merge round 7 done
>>> Chunk | offset=4104 sorted and written back <<<
==9217==
==9217== HEAP SUMMARY:
==9217==     in use at exit: 0 bytes in 0 blocks
==9217==   total heap usage: 733 allocs, 733 frees, 129,280 bytes
allocated
==9217==
==9217== All heap blocks were freed -- no leaks are possible
==9217==
==9217== For lists of detected and suppressed errors, rerun with: -s
==9217== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
0)
```