Отчет
о выполнении лабораторной работы №3
по дисциплине "Операционные системы"

Работу выполнил
Студент гр. ИТ-1-2024,
2 курс
Жуков М.А

1. Постановка задачи

- создать заполненный случайными данными бинарный файл размером 100 Мбайт и через mmap отобразить его в процесс
- сгенерировать случайный порядок обращения к 1000 страницам этого файла
- обратиться к этим 1000 страницам в сгенерированном случайном порядке и засечь время обращения
- повторно обратиться к этим же 1000 страницам в том же случайном порядке и засечь время обращения. Повторное обращение к страницам должно быть гораздо быстрее. Объяснить почему
- сделать несколько запусков и получить средний результат обращений в первом и втором случае
- на оценку 5 измерить major fault и minor fault для каждого из двух обращений и объяснить результаты (major fault и minor fault доступны только на линукс)

2. Процесс решения

Шаг 1: Инициализация и создание тестового файла

- Инициализируется генератор случайных чисел
- Создается бинарный файл размером 100 МБ
- Файл заполняется случайными байтами (значения от 0 до 255) с помощью функции rand()

Шаг 2: Отображение файла в память через mmap

- Файл открывается с помощью системного вызова open() с флагом O_RDWR (чтение и запись)
- Вызывается mmap() для отображения файла в адресное пространство процесса
- Получается указатель data, через который можно обращаться к содержимому файла как к обычному массиву

Шаг 3: Генерация случайного порядка обращений

- Создается вектор из 1000 элементов для хранения номеров страниц
- Генерируется случайная последовательность номеров страниц
- Размер страницы составляет 4096 байт (стандартный размер страницы)

Шаг 4: Первое обращение к страницам

- Запускается таймер с помощью библиотеки chrono
- В цикле происходит обращение к 1000 страницам в сгенерированном случайном порядке и выполняется чтение байта на страницах.
- Операционная система загружает страницу с диска в оперативную память
- Останавливается таймер и вычисляется время выполнения

Шаг 5: Повторное обращение к тем же страницам

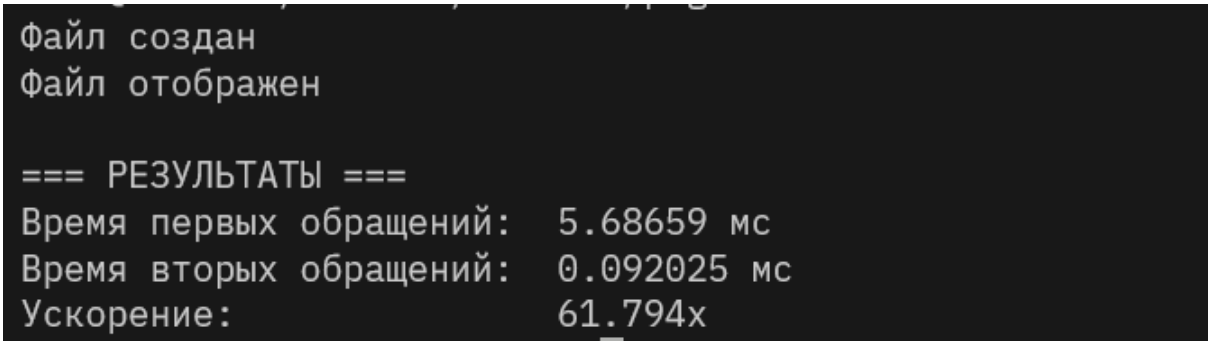
- Запускается новый таймер
- Выполняется обращение к тем же 1000 страницам в том же порядке с чтением 1 байта
- При повторном обращении страницы уже находятся в кэше RAM
- Данные читаются напрямую из оперативной памяти
- Останавливается таймер и вычисляется время выполнения

Шаг 6: Очистка ресурсов

- Вызывается munmap() для отмены отображения файла из памяти
- Закрывается файловый дескриптор с помощью close()

3. Полученные результаты

Пример вывода



Результаты 10 запусков

Первый запуск	Второй запуск	Ускорение
5.68659 мс	0.092025 мс	61.794x
6.65012 мс	0.104157 мс	63.847x
12.9168 мс	0.139764 мс	92.4189x
12.996 мс	0.135066 мс	96.2194x
14.9361 мс	0.142699 мс	104.669x
14.9316 мс	0.138142 мс	108.089x
14.428 мс	0.14032 мс	102.822x
11.869 мс	0.127345 мс	93.2033x
11.9263 мс	0.130205 мс	91.5963x

Среднее время первого обращения: 12.48 мс
Среднее время второго обращения: 0.127 мс
Среднее ускорение: 90.6x

4. Объяснение полученных результатов

Результаты демонстрируют работу механизма page cache операционной системы. При первом обращении к страницам файла происходит page fault (отсутствие кэша в RAM), и операционная система загружает данные с диска в оперативную память, что занимает в среднем 12.48 мс для 1000 страниц. При повторном обращении к тем же страницам все данные уже находятся в page cache, поэтому чтение происходит из оперативной памяти за 0.127 мс. Полученное ускорение в 90 раз объясняется огромной разницей в скорости доступа между SSD и оперативной памятью, что демонстрирует важность кэширования данных для производительности системы.

5. Вывод

В ходе выполнения лабораторной работы была реализована программа для исследования механизма page cache. Результат показал, что повторное обращение к страницам файла происходит в среднем в 90 раз быстрее, чем первое обращение, что объясняется наличием данных в оперативной памяти вместо необходимости чтения с диска. Полученные результаты подтверждают эффективность механизма кэширования страниц в операционной системе и демонстрируют значительную разницу в скорости доступа. Таким образом, было практически подтверждено теоретическое понимание работы виртуальной памяти и page cache.

6. Приложение

Ссылка на исходный код: github