ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“Научно-образовательная корпорация ИТМО”

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

Лабораторная работа №2

по дисциплине “Операционные системы”

Вариант:
Linux | Clock

Выполнил:
Сафарзаде Джунайд
Группа: P3309, ИСУ: 372828
Преподователь:
Гиниятуллин Арслан

Санкт-Петербург, 2025

Задание

Для оптимизации работы с блочными устройствами в ОС существует кэш страниц с данными, которыми мы производим операции чтения и записи на диск. Такой кэш позволяет избежать высоких задержек при повторном доступе к данным, так как операция будет выполнена с данными в RAM, а не на диске (вспомним пирамиду памяти).

В данной лабораторной работе необходимо реализовать блочный кэш в пространстве пользователя в виде динамической библиотеки (dll или so). Политику вытеснения страниц и другие элементы задания необходимо получить у преподавателя.

При выполнении работы необходимо реализовать простой API для работы с файлами, предоставляющий пользователю следующие возможности:

1. Открытие файла по заданному пути файла, доступного для чтения. Процедура возвращает некоторый хэндл на файл. Пример:
   `int lab2_open(const char *path).`
2. Закрытие файла по хэндлу. Пример:
   `int lab2_close(int fd).`
3. Чтение данных из файла. Пример:
   `ssize_t lab2_read(int fd, void buf[.count], size_t count).`
4. Запись данных в файл. Пример:
   `ssize_t lab2_write(int fd, const void buf[.count], size_t count).`
5. Перестановка позиции указателя на данные файла. Достаточно поддержать только абсолютные координаты. Пример:
   `off_t lab2_lseek(int fd, off_t offset, int whence).`
6. Синхронизация данных из кэша с диском. Пример:
   `int lab2_fsync(int fd).`

Операции с диском разработанного блочного кеша должны производиться в обход page cache используемой ОС.

В рамках проверки работоспособности разработанного блочного кэша необходимо адаптировать указанную преподавателем программу-загрузчик из ЛР 1, добавив использование кэша. Запустите программу и убедитесь, что она корректно работает. Сравните производительность до и после.

Исходный код

https://github.com/dzschnd/os-lab-2

Результат работы программ-нагрузчиков

Для анализа производительности реализованного кэша я разработал две программы- нагрузчики io-lat-read и io-lat-write, сравнив с их помощью задержки без использования кэша и с использованием моей реализации.

```
=== Starting IO Lat Read Benchmark without caching ===

Overall Stats:
Average read latency: 0.00122964 seconds
Minimum read latency: 0.00102553 seconds
Maximum read latency: 0.00491156 seconds

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 94.24    0.036160          18      2005           read
  2.88    0.001104          10       106           openat
  1.60    0.000615           5       106           close
  0.60    0.000229           8        26           mmap
  0.22    0.000085          12         7           mprotect
  0.14    0.000053          53         1           write
  0.08    0.000031           4         7           fstat
  0.07    0.000025          25         1           munmap
  0.03    0.000012           4         3           brk
  0.03    0.000012           6         2           pread64
  0.03    0.000011          11         1           prlimit64
  0.02    0.000006           6         1           arch_prctl
  0.02    0.000006           6         1           getrandom
  0.02    0.000006           6         1           rseq
  0.01    0.000005           5         1           futex
  0.01    0.000005           5         1           set_tid_address
  0.01    0.000005           5         1           set_robust_list
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           getcwd
------ ----------- ----------- --------- --------- ----------------
100.00    0.038370          16      2274         1 total
```

```
=== Starting IO Lat Read Benchmark with caching ===

Overall Stats:
Average read latency: 0.000618458 seconds
Minimum read latency: 0.000485575 seconds
Maximum read latency: 0.00246527 seconds

Cache hits: 2397
Cache misses: 103

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.06    0.001483          14       100           fsync
 21.90    0.001013           9       106           openat
 13.06    0.000604           5       105           pread64
 12.69    0.000587           5       106           close
  7.52    0.000348          13        26           mmap
  6.92    0.000320         320         1           execve
  1.56    0.000072          10         7           mprotect
  0.97    0.000045           6         7           fstat
  0.86    0.000040          13         3           write
  0.63    0.000029           5         5           read
  0.48    0.000022          22         1           munmap
  0.32    0.000015           5         3           brk
  0.15    0.000007           7         1         1 access
  0.13    0.000006           6         1           futex
  0.13    0.000006           6         1           getrandom
  0.11    0.000005           5         1           getcwd
  0.11    0.000005           5         1           arch_prctl
  0.11    0.000005           5         1           set_tid_address
  0.11    0.000005           5         1           prlimit64
  0.11    0.000005           5         1           rseq
  0.09    0.000004           4         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.004626           9       479         1 total
```

```
=== Starting IO Lat Write Benchmark without caching ===

Overall Stats:
Average write latency: 0.17672 seconds
Minimum write latency: 0.162517 seconds
Maximum write latency: 0.215694 seconds

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.66    4.607327          22    204801           write
  0.21    0.009482          94       100           unlink
  0.07    0.003203          30       106           openat
  0.02    0.000966           9       106           close
  0.02    0.000950           9       100           fsync
  0.01    0.000386         386         1           execve
  0.01    0.000375          14        26           mmap
  0.00    0.000080          11         7           mprotect
  0.00    0.000045           6         7           fstat
  0.00    0.000030           6         5           read
  0.00    0.000020          20         1           munmap
  0.00    0.000020           6         3           brk
  0.00    0.000010           5         2           pread64
  0.00    0.000008           8         1         1 access
  0.00    0.000007           7         1           getcwd
  0.00    0.000005           5         1           arch_prctl
  0.00    0.000005           5         1           futex
  0.00    0.000005           5         1           set_robust_list
  0.00    0.000005           5         1           prlimit64
  0.00    0.000005           5         1           getrandom
  0.00    0.000005           5         1           rseq
  0.00    0.000004           4         1           set_tid_address
------ ----------- ----------- --------- --------- ----------------
100.00    4.622943          22    205274         1 total
```

```
=== Starting IO Lat Write Benchmark with caching ===

Overall Stats:
Average write latency: 0.0231366 seconds
Minimum write latency: 0.0190702 seconds
Maximum write latency: 0.0443672 seconds

Cache hits: 179200
Cache misses: 25700

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 84.26    0.582274          22     25700           pwrite64
 13.83    0.095593           3     25702           pread64
  0.95    0.006538          65       100           unlink
  0.43    0.002986          28       106           openat
  0.29    0.001994           9       200           fsync
  0.16    0.001073          10       106           close
  0.05    0.000317          12        26           mmap
  0.01    0.000066           9         7           mprotect
  0.01    0.000048           6         7           fstat
  0.01    0.000036          12         3           write
  0.00    0.000029           5         5           read
  0.00    0.000026           5         5           brk
  0.00    0.000018          18         1           munmap
  0.00    0.000008           8         1         1 access
  0.00    0.000006           6         1           getcwd
  0.00    0.000005           5         1           getrandom
  0.00    0.000004           4         1           futex
  0.00    0.000004           4         1           prlimit64
  0.00    0.000003           3         1           arch_prctl
  0.00    0.000003           3         1           set_tid_address
  0.00    0.000003           3         1           rseq
  0.00    0.000002           2         1           set_robust_list
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.691036          13     51978         1 total
```

Из вывода скрипта, запускающего программы, можно сделать следующие выводы:

Чтение:
С кэшем средняя задержка уменьшилась на 49.7%
максимальная - на 49.8%,
минимальная - на 51.4%,
попадания кэша составили 95.9%

Запись:
средняя задержка уменьшилась на 53.8%
максимальная - на 35.4%,
минимальная - на 56.7%,
попадания кэша составили 87.5%

Вывод

За время работы я реализовал блочный кэш в пространстве пользователя в виде динамической библиотеки- интерфейс, поддерживающий операции записи, чтения, открытия и закрытия файлов, синхронизации с диском и перестановки указателя на данные файлы.
Я глубже ознакомился с организацией кэша, изучил ряд политик вытеснения страниц, также реализовал в своей библиотеке политику вытеснения Clock.