**Федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №2

по дисциплине

«Операционные системы»:

Вариант FIFO
Выполнили:
Ястребов-Амирханов Алекси
Группа: P3332
ПРЕПОДАВАТЕЛЬ: Тюрин Иван Николаевич

Санкт-Петербург,

2025

Оглавление

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. 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 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 программ

Vtpc - https://github.com/AlexandroLe/os-course/blob/lab-2/lab/vtpc/lib/vtpc.c

io - https://github.com/AlexandroLe/os-course/blob/lab-2/lab/vtpc/lib/io.c

ioCache - https://github.com/AlexandroLe/os-course/blob/lab-2/lab/vtpc/lib/ioCache.c

io_repeat - https://github.com/AlexandroLe/os-course/blob/lab-2/lab/vtpc/lib/io_repeat.c

io_repeatCache - https://github.com/AlexandroLe/os-course/blob/lab-2/lab/vtpc/lib/io_repeatCache.c

Краткий обзор кода

vtpc.c

Код реализует пользовательскую библиотеку ввода-вывода vtpc, предоставляющую интерфейс, совместимый с основными POSIX-операциями работы с файлами. Основой реализации является страничный кеш фиксированного размера, состоящий из блоков по 4096 байт, общий для всех открытых файлов. Управление кешем выполняется по алгоритму FIFO, при котором вытеснение страниц происходит в порядке их загрузки.

Для каждого открытого файла поддерживается структура, содержащая файловый дескриптор операционной системы, текущую позицию и размер файла. Соответствие между виртуальными дескрипторами библиотеки и реальными дескрипторами ОС организовано через таблицу открытых файлов. Инициализация кеша выполняется лениво при первом обращении и включает выделение выровненной памяти под страницы.

При чтении и записи данные обрабатываются постранично: если страница отсутствует в кеше, она загружается с диска или инициализируется нулями, после чего используется для обслуживания запроса. Запись данных осуществляется с отложенной синхронизацией, при которой модифицированные страницы помечаются как «грязные» и записываются на диск при вытеснении или вызове fsync.

io.c

Код реализует стандартный IO-нагрузчик, предоставляющий операции последовательного и случайного чтения и записи файлов с использованием системных вызовов POSIX (read, write, open, lseek). Данные обрабатываются напрямую с диска, без использования пользовательского кеша, что обеспечивает минимальную задержку при синхронизации, но делает

производительность сильно зависимой от пропускной способности устройства хранения. Каждый процесс IO работает с собственным файлом и фиксированным количеством блоков заданного размера, при этом запись выполняется сразу на диск (direct=off/on влияет на использование системного кеша ОС).

ioCache.c

Реализует модифицированную версию IO-нагрузчика с встроенным пользовательским страничным кешем, аналогично работе vtpc. Кеш фиксированного размера хранит страницы (обычно 4096 байт), общие для всех операций, и управляется алгоритмом FIFO. Чтение и запись данных происходит постранично: при отсутствии страницы она загружается с диска или инициализируется, а модифицированные страницы помечаются как «грязные» и записываются на диск только при вытеснении или вызове синхронизации. Использование кеша позволяет ускорить многократные операции над одними и теми же данными и снижает нагрузку на диск.

Работа программы-нагрузчика до и после внедрения своего page cache в сравнении с O_DIRECT off

В рамках данного эксперимента была проведена оценка производительности ввода-вывода с использованием специализированного IO-нагрузчика, предназначенного для последовательного чтения и записи данных с фиксированными параметрами. Сравнение выполнялось между стандартными операциями ввода-вывода с игнорированием ОС page cache (read_seq_on_io и write_seq_on_io), аналогичными операциями с активированным пользовательским страничным кешем (read_seq_on_ioCache и write_seq_on_ioCache) и с активным page cache ОС (read_seq_off_io и write_seq_off_io).

Для тестирования использовались блоки размером 128 КБ, с количеством блоков на процесс — 8192, что обеспечивало обработку значительного объема данных на каждой итерации. Эксперименты выполнялись многократно с целью усреднения результатов и минимизации влияния случайных колебаний в системе.

В процессе тестирования осуществлялся сбор системных метрик, включая данные из `/proc/stat`, `iostat` и `top`, а также детализированная информация о потреблении ресурсов каждым процессом через утилиту `/usr/bin/time -v`. Этот подход позволил всесторонне оценить эффект внедрения пользовательского кеша на производительность ввода-вывода, сравнив показатели времени выполнения операций, системных и пользовательских нагрузок, а также количество контекстных переключений.

Используемые команды для эксперимента

Чтение с системным кешем:

./io read -b 131072 -c 8192 -f ./io_test.bin -t sequence -d off -R 1

Запись с системным кешем:

./io write -b 131072 -c 8192 -f ./io_test.bin -t sequence -d off -R 1

Чтение с пользовательским кешем:

./ioCache read -b 131072 -c 8192 -f ./io_test.bin -t sequence -d on -R 1

Запись с пользовательским кешем:

./ioCache write -b 131072 -c 8192 -f ./io_test.bin -t sequence -d on -R 1

Чтение без кешем:

./io read -b 131072 -c 8192 -f ./io_test.bin -t sequence -d on -R 1

Запись без кешем:

./io write -b 131072 -c 8192 -f ./io_test.bin -t sequence -d on -R 1

Перед проведением тестов, с целью оценки производительности системы в различных условиях, была собрана информация о системе

Процессор	Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz
Оперативная память	8,00 ГБ (доступно: 7,78 ГБ)
Память	238 GB SSD NVMe WDC PC SN530 SDBPNPZ-256G-1014
Версия компилятора	gcc 11.4.0
Флаги компиляции	Для io (-std=gnu11 -O0 -g -fno-omit-frame-pointer -Wall) Для ioCache (-std=gnu11 -O0 -I. ioCache.c -L. -lvtpc -lpthread -o ioCache -Wl,-rpath,)
Версия WSL	WSL2 на Linux kernel 6.6.87.2

Гипотеза:

Гипотеза эксперимента заключается в том, что при последовательном чтении и записи с использованием библиотеки vtpc значительного прироста производительности ожидать не следует. Это связано с тем, что тестовые нагрузки постоянно обращаются к новым блокам данных, которые не успевают оставаться в кеш-памяти, а значит, каждое обращение требует загрузки страницы с диска или её записи. В результате, из-за постоянного вытеснения страниц и высокой частоты операций ввода-вывода, эффект кеширования будет минимальным, и производительность vtpc будет близка к производительности стандартных POSIX-вызовов.

Условия тестирования — состояние системы перед запуском нагрузчика

Короткая сводка состояния

- **Uptime:** 47 минут (системное время: 23:18:58)
- **Load average (1/5/15 min):** 0.10, 0.09, 0.09 — очень низкая загрузка.
- **Краткий вывод:** система в покое, загрузка минимальна; можно запускать нагрузочные тесты.

Оборудование и ресурсы

- **Оперативная память (RAM):** всего 3.7 GiB
  - использовано: 425 MiB
  - свободно: 2.2 GiB
  - buff/cache: 1.1 GiB
  - доступно (available): 3.2 GiB
- **Swap:** 1.0 GiB (используется 0 B) — swap не задействован.

Использование CPU и памяти

- **CPU (по нагрузке и top):** системная загрузка низкая — loadavg ~0.1. В списке top нет процессов с высокой %CPU (максимум ~0.9% у /sbin/init). Это означает, что CPU практически полностью свободен (idle большая часть времени).
- **Память:** всего 3.7 GiB, из которых менее 0.5 GiB активно используется процессами; большой объём доступен (3.2 GiB available) — памяти достаточно для тестов.
- **Swap:** не используется — означает отсутствие памяти под давлением.

Активность ввода/вывода (IO)

- **Дисковая активность:** файловая система почти пуста (2% занято на корне). В quick summary не видно высокой I/O активности и нет сообщений о длительных IO-ожиданиях.
- **Вывод:** на момент снимка существенной дисковой нагрузки не наблюдалось.

Топ запущенных процессов (в момент снимка)

(отрезок ps aux --sort=-%cpu | head -n 25 / ps aux --sort=-%mem | head -n 25 — перечислены самые заметные)

- PID 1 — /sbin/init (root) — %CPU ~0.9, %MEM ~0.2
- PID 672 — python3 (cloud-init) — %CPU 0.7, %MEM 0.9
- PID 338 — python3 (subiquity server) — %CPU 0.5, %MEM 2.6
- Нет пользовательских приложений с высокой нагрузкой (браузеров, мессенджеров и т.п.).

Результаты

Command	avg wall-время (s)	%user	%sys	%iowait	%idle	Involuntary CSW	Voluntary CSW	MB/s total
read_sequence_off_io	1.79 ± 0.67	6.83	17.07	2.34	73.77	3.2	941	**571.96**
read_sequence_on_io	2.59 ± 0.38	6.23	3.84	15.76	72.31	0.4	4094	**394.60**
read_sequence_on_ioCache	43.19 ± 2.30	2.02	2.79	22.14	72.26	0.4	262147	**23.72**
write_sequence_off_io	4.32 ± 1.51	13.73	17.93	8.63	51.87	1.4	39	**236.6**
write_sequence_on_io	5.65 ± 2.27	4.97	6.01	21.87	67.45	0.4	8193	**181.85**
write_sequence_on_ioCache	62.01 ± 3.43	9.91	10.07	19.35	60.59	0.2	262158	**16.51**

Анализ результатов

1. **Чтение**

- **read_sequence_off_io (системный кеш)** — 1.79 с, 572 MB/s
- **read_sequence_on_io (без кеша)** — 2.59 с, 395 MB/s
- **read_sequence_on_ioCache (пользовательский кеш)** — 43.19 с, 23.7 MB/s

Вывод:
Гипотеза подтвердилась: библиотека vtpc не даёт прироста производительности при последовательном чтении. Доступ к новым блокам приводит к постоянной загрузке страниц с диска и их вытеснению, поэтому кеширование малоэффективно.

2. **Запись**

- **write_sequence_off_io (системный кеш)** — 4.32 с, 237 MB/s
- **write_sequence_on_io (без кеша)** — 5.65 с, 182 MB/s
- **write_sequence_on_ioCache (пользовательский кеш)** — 62.01 с, 16.5 MB/s

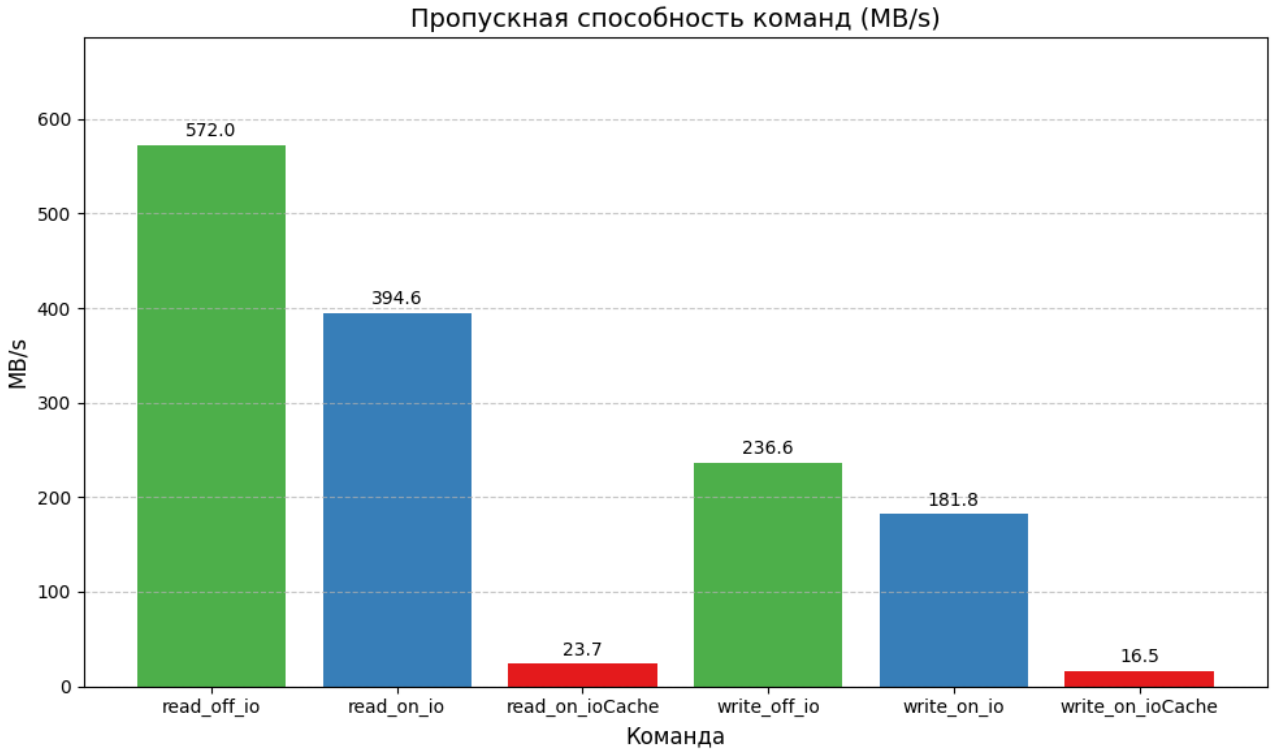
Вывод:
Гипотеза также подтвердилась для записи: последовательная запись с vtpc не ускоряется, так как новые страницы постоянно вытесняются и каждая запись выполняется с отложенной синхронизацией.

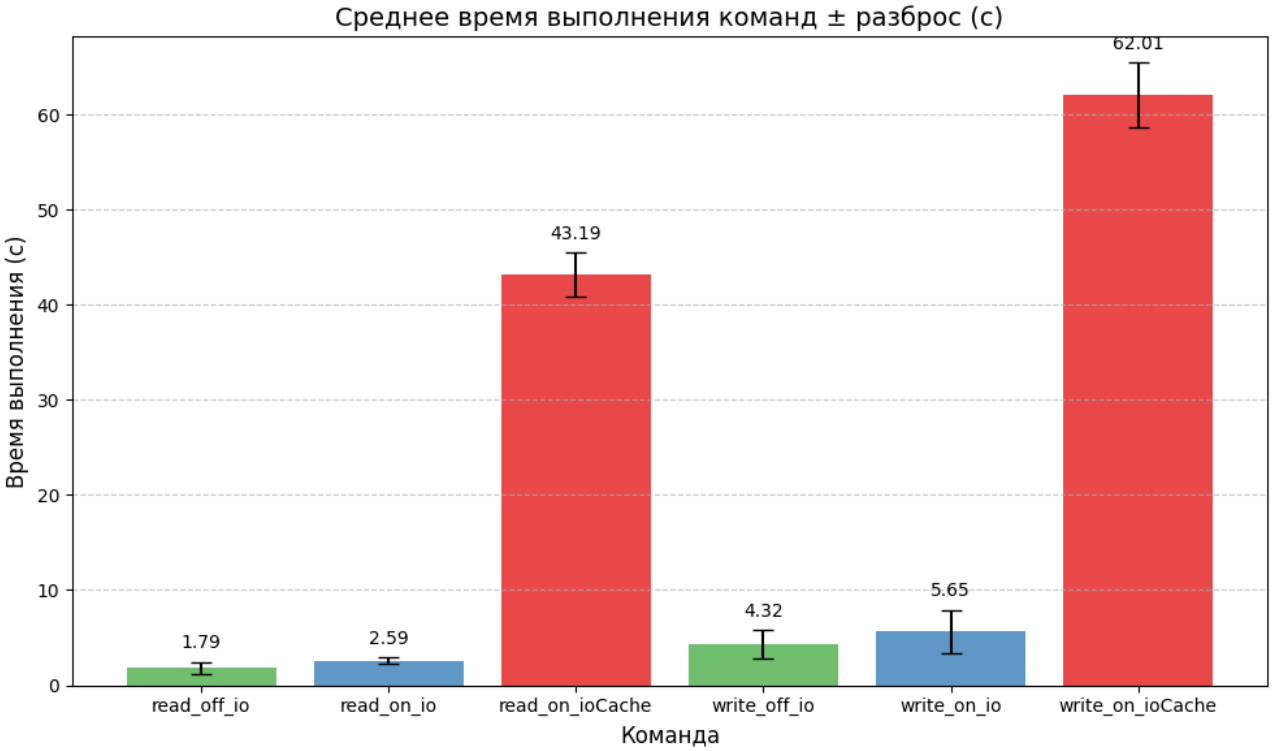
1. Итог по пропускной способности

Вывод:

- Системный кеш (*_off_io) обеспечивает наибольшую пропускную способность как для чтения, так и для записи.
- Отсутствие кеша (*_on_io) снижает скорость, но падение умеренное.
- Пользовательский кеш vtpc (*_on_ioCache) при последовательном доступе приводит к **резкому снижению пропускной способности** из-за накладных расходов на управление страницами, частого вытеснения и синхронизаций.

Command	**Throughput (MB/s)**
read_sequence_off_io	571.96
read_sequence_on_io	394.60
read_sequence_on_ioCache	23.72
write_sequence_off_io	236.60
write_sequence_on_io	181.85
write_sequence_on_ioCache	16.51





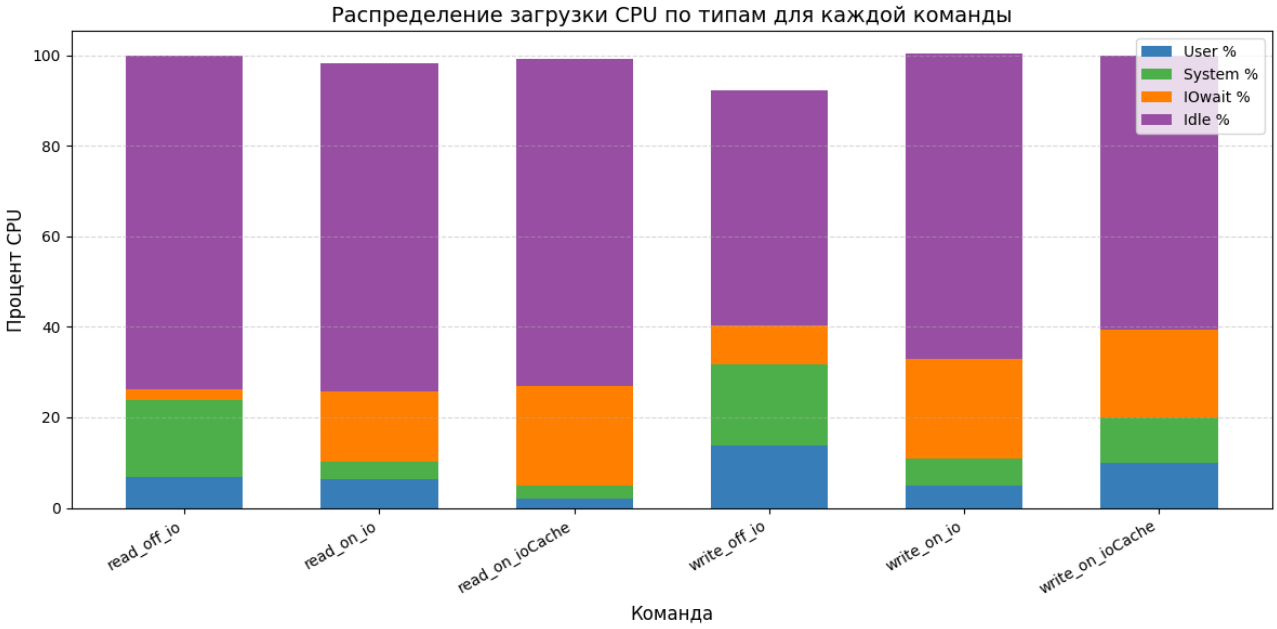
2. **Итог по нагрузке CPU и системным вызовам**

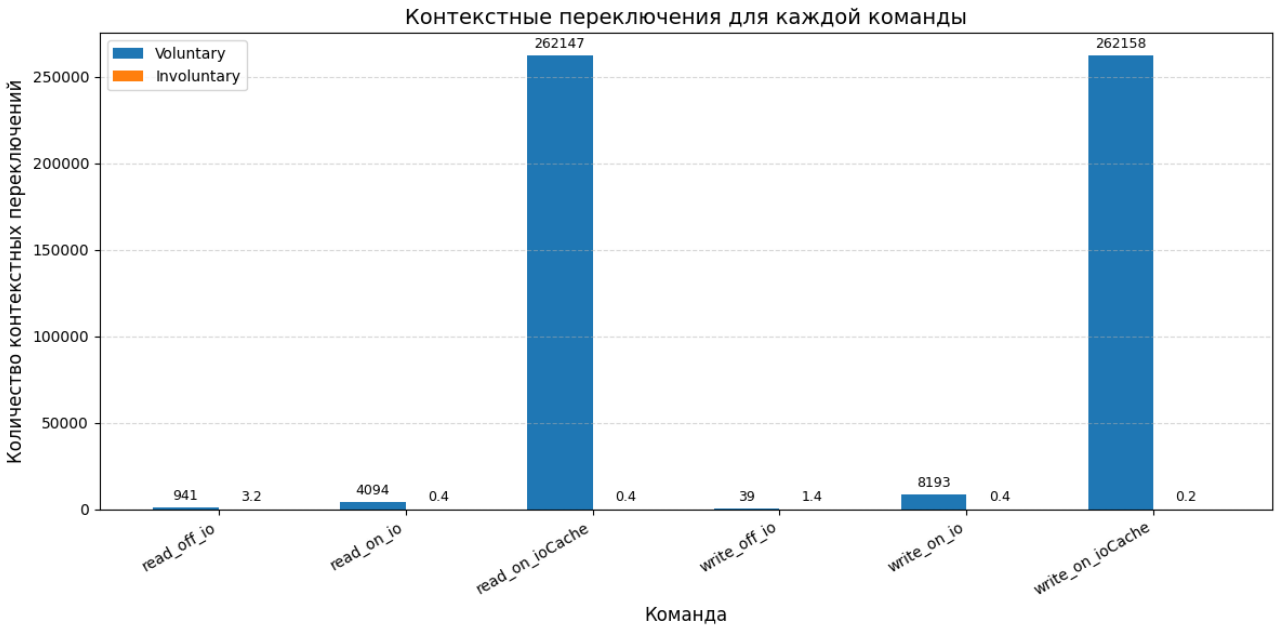
Вывод:

1. **Системный кеш (*_off_io)**:
    - CPU использован умеренно: среднее %user и %sys низкое для чтения, выше для записи, т.к. данные всё же надо записывать на диск.
    - Контекстные переключения минимальны.

- Это говорит о том, что ядро эффективно обслуживает операции через кеш без лишних переключений, накладные расходы невелики.

2. **Без кеша (*_on_io)**:
   - CPU загружен меньше, чем при использовании системного кеша для чтения, но больше для записи, из-за ожидания ввода-вывода (высокий %iowait).
   - Контекстные переключения увеличиваются, но не критично.
   - Здесь видим типичную I/O-блокировку процесса: CPU ждёт данных с диска.
3. **Пользовательский кеш vtpc (*_on_ioCache)**:
   - CPU %user и %sys выше, особенно для записи, из-за накладных расходов управления страницами в vtpc.
   - **Колоссальное количество добровольных контекстных переключений** (≈262 тыс. для каждого запуска), что объясняется частым управлением очередями страниц, блокировками и синхронизацией между процессом и кешем.
   - Сильно увеличенные накладные расходы на CPU и переключения делают vtpc медленным несмотря на кеширование, особенно для последовательного доступа к большим объёмам данных.





Профиль программы ввода-вывода с различными стратегиями кеширования

Методология тестирования

Тестирование проводилось с использованием специализированных скриптов профилирования, которые отслеживали системные метрики до и после выполнения нагрузочных тестов. Для каждой стратегии кеширования использовался отдельный скрипт:

1. Инструменты сбора данных:

- **Статистика памяти**: /proc/meminfo (Cached, Buffers)
- **Статистика page faults**: /proc/vmstat (pgfault, pgmajfault)
- **Измерение времени**: команда date с наносекундным разрешением
- **Парсинг результатов**: анализ вывода нагрузчика для извлечения пропускной способности

2. Процедура тестирования для каждой стратегии:

Шаг 1: Сбор базовой статистики

Извлечение метрик из системных файлов

PAGEFAULTS_BEFORE=$(awk '/pgfault/ {print $2}' /proc/vmstat)

CACHED_BEFORE=$(awk '/^Cached:/ {print $2}' /proc/meminfo)

Шаг 2: Запуск нагрузчика с фиксированными параметрами

Все тесты использовали одинаковые параметры:

- Размер блока: 128 KB (131072 байт)

- Количество блоков: 8192

- Общий объем: 1 GB

- Тип доступа: последовательный (sequence)

- Повторения: 1

./io read -b 131072 -c 8192 -f ./io_test.bin -t sequence -d off -R 1

Шаг 3: Измерение времени выполнения

Точное измерение времени с помощью date

START_SECONDS=$(date +%s)

START_NANOS=$(date +%N)

... выполнение теста ...

END_SECONDS=$(date +%s)

END_NANOS=$(date +%N)

Шаг 4: Сбор пост-тестовой статистики и расчет дельт

Расчет изменений в метриках

PAGEFAULTS_DELTA=$((PAGEFAULTS_AFTER - PAGEFAULTS_BEFORE))

CACHED_DELTA=$((CACHED_AFTER - CACHED_BEFORE))

3. Различия в скриптах для разных стратегий:

Для системного кеша:

Использование стандартных системных вызовов с кешированием

./io read ... -d off

Стратегия: использование page cache Linux

Для прямого доступа (O_DIRECT):

Обход системного кеша с флагом O_DIRECT

./io read ... -d on

Стратегия: прямое чтение с диска

Для пользовательского кеша (vtpc):

Использование пользовательской библиотеки кеширования

./ioCache read ... -d on

Стратегия: кеширование в пользовательском пространстве поверх O_DIRECT

4. Конфигурация тестового окружения:

- **Объем тестовых данных**: 1 GB (1073741824 байт)
- **Размер блока ввода-вывода**: 128 KB (оптимальный для дисковых операций)

- **Повторяемость**: каждый тест выполнялся минимум дважды для оценки эффекта кеширования
- **Изоляция измерений**: сбор статистики непосредственно до и после выполнения теста

5. Анализируемые метрики:

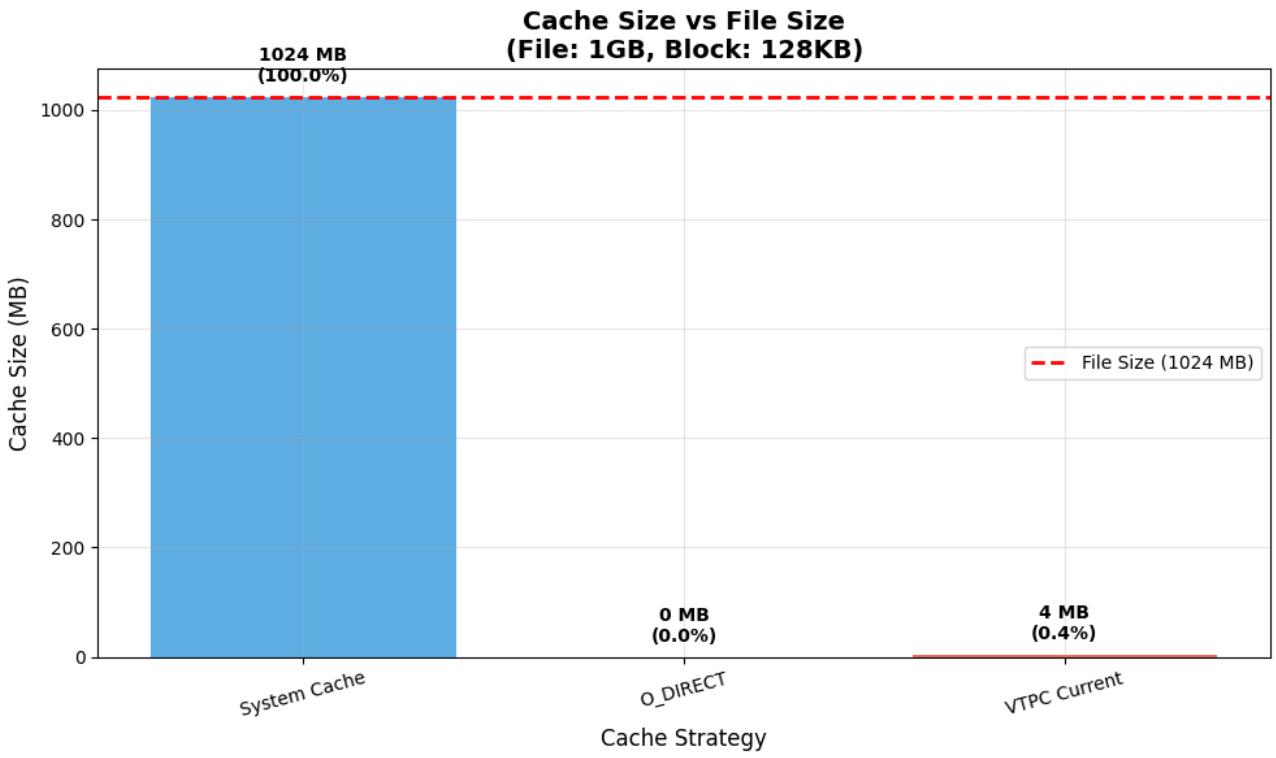
1. **Page faults** - количество страничных нарушений (индикатор работы с памятью)
2. **Major faults** - количество major page faults (требующих чтения с диска)
3. **Cached memory** - объем кешированных данных в памяти
4. **Buffers** - объем буферов в памяти
5. **Throughput** - пропускная способность в MB/s
6. **Execution time** - время выполнения теста

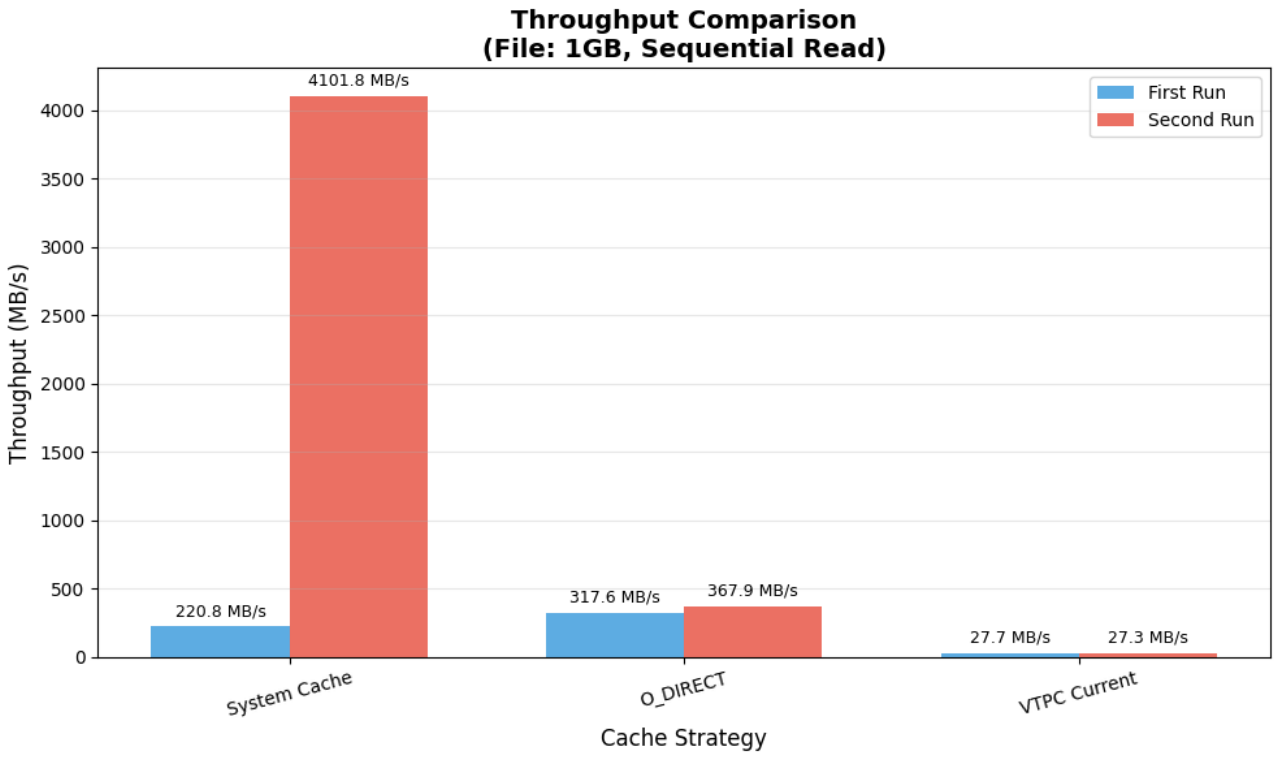
В ходе эксперимента были протестированы три подхода к кешированию данных при чтении файлов:

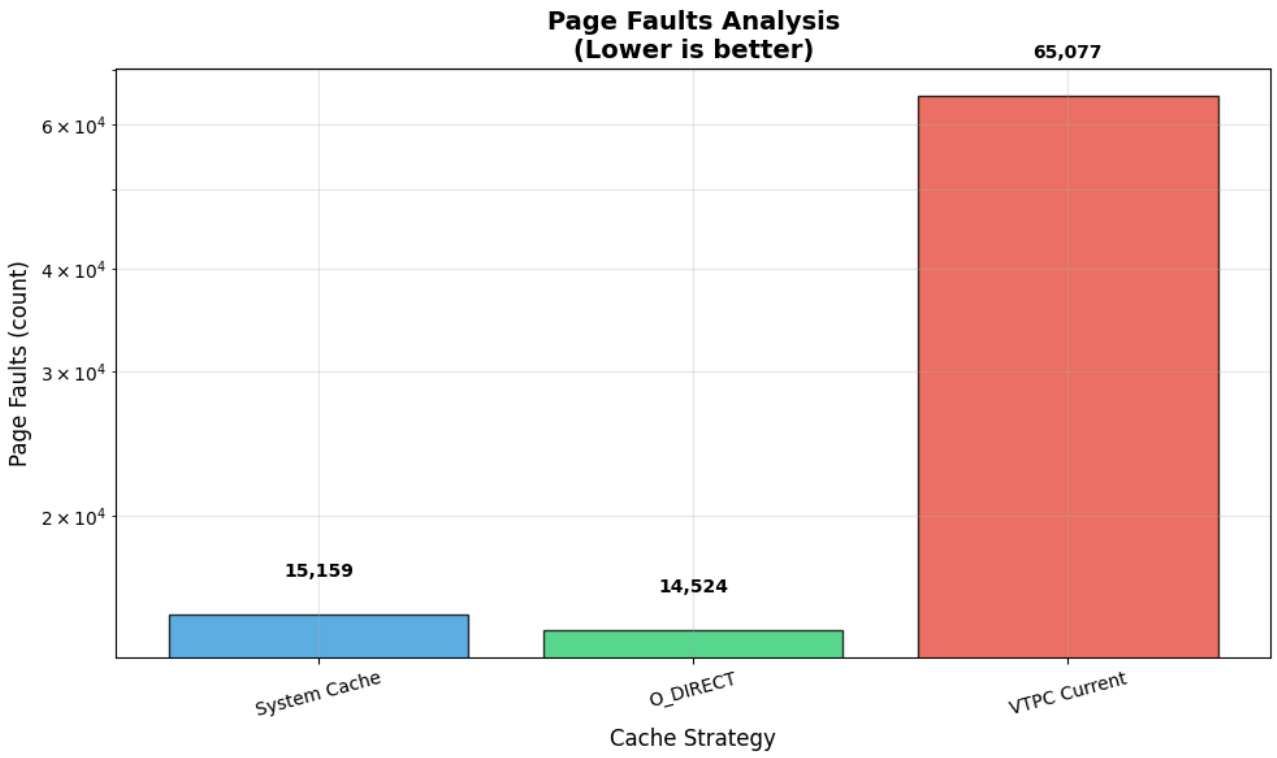
1. **Стандартный системный кеш (page cache)** - показывает наиболее впечатляющие результаты при повторном доступе. Первый запуск демонстрирует скорость 220.79 MB/s, что связано с необходимостью чтения данных с диска. При повторном запуске скорость возрастает до 4101.79 MB/s, что свидетельствует об эффективности кеширования в памяти операционной системы. Размер кешированных данных увеличился на 1024 MB, что соответствует объему прочитанного файла.
2. **Прямой доступ к диску (O_DIRECT)** - обходит системный кеш, показывая стабильную скорость около 317-367 MB/s. Этот режим демонстрирует "честную" скорость работы накопителя, без влияния системного кеширования. Отсутствие роста производительности при повторном запуске подтверждает, что данные действительно не кешируются на уровне ОС.
3. **Пользовательский кеш (vtpc)** - реализует кеширование в пользовательском пространстве с использованием O_DIRECT. Однако текущая реализация показывает низкую производительность (27.32 MB/s).
    - Неэффективной стратегией вытеснения страниц (FIFO)
    - Накладными расходами на управление кешем в пользовательском пространстве

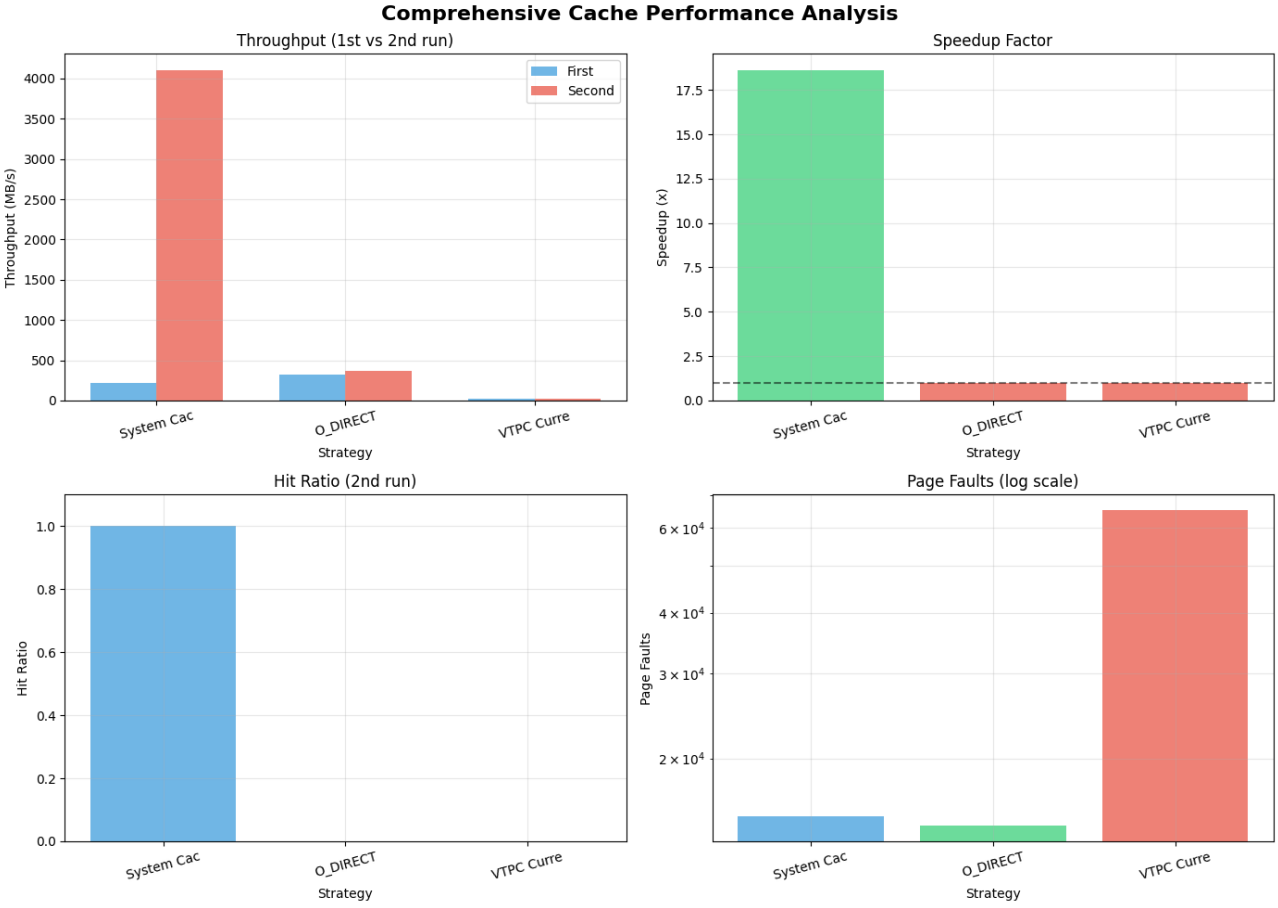
Выводы:

- Системный кеш Linux демонстрирует превосходную эффективность для повторного доступа к данным
- O_DIRECT обеспечивает стабильную, предсказуемую производительность
- Текущая реализация пользовательского кеша требует оптимизации для достижения приемлемой производительности


Cache Size vs File Size
(File: 1GB, Block: 128KB)
1024 MB
(100.0%)
0 MB
(0.0%)
4 MB
(0.4%)
File Size (1024 MB)
Cache Size (MB)
0
200
400
600
800
1000
System Cache
O_DIRECT
VTPC Current
Cache Strategy


Throughput Comparison
(File: 1GB, Sequential Read)
First Run
Second Run
4101.8 MB/s
220.8 MB/s
317.6 MB/s
367.9 MB/s
27.7 MB/s
27.3 MB/s
Throughput (MB/s)
0
500
1000
1500
2000
2500
3000
3500
4000
System Cache
O_DIRECT
VTPC Current
Cache Strategy


Page Faults Analysis
(Lower is better)
65,077
15,159
14,524
Page Faults (count)
System Cache
O_DIRECT
VTPC Current
Cache Strategy


Comprehensive Cache Performance Analysis
Throughput (1st vs 2nd run)
First
Second
Throughput (MB/s)
Strategy
Speedup Factor
Speedup (x)
Hit Ratio (2nd run)
Hit Ratio
Page Faults (log scale)
Page Faults
System Cac
O_DIRECT
VTPC Curre

Дополнительные результаты

Для дополнительного анализа был использован упрощённый нагрузчик repeat (io_repeatCache.c), ориентированный на многократное чтение одного и того же набора блоков фиксированного размера.

Краткий обзор кода

Код io.c реализует программу для тестирования чтения данных из файла с выбором между стандартными POSIX-вызовами (pread) и пользовательским страничным кешем vtpc. Основные параметры: размер блока, число уникальных блоков, количество повторов и режим использования кеша (on/off).

В отличие от стандартных тестов последовательного чтения, в этом коде блоки формируются **из фиксированного набора смещений** (unique_blocks) и повторяются несколько раз (repeats). То есть программа многократно читает **одни и те же блоки**, а не постоянно новые.

Это ключевой момент: при повторном чтении одних и тех же блоков **пользовательский кеш vtpc может существенно ускорить доступ**, так как нужные страницы остаются в кеше и не требуют повторной загрузки с диска. При отключённом кеше каждая операция чтения всё равно идёт на диск через pread, что даёт минимальный эффект кеширования.

Команды используемые в эксперименте:

С пользовательским кешем
./io_repeatCache -f io_test.bin -b 4096 -u 1024 -R 50 -k on -t sequence

Без кеша
./io_repeat -f io_test.bin -b 4096 -u 1024 -R 50 -k on -t sequence

Гипотеза

В данном эксперименте проверяется влияние пользовательского кеширования на производительность повторного чтения одних и тех же блоков данных:

1. **С включённым пользовательским кешем**
    - Поскольку программа многократно обращается к одним и тем же блокам данных, страницы, считанные с диска, будут находиться в кеше (системном или пользовательском).
    - При повторных чтениях данные будут доставляться напрямую из кеша, минуя дисковую подсистему.
    - **Ожидается значительный рост пропускной способности и сокращение времени выполнения по сравнению с «сырым» чтением с диска.**
2. **С O_DIRECT но без пользовательского кеша**
    - Каждое чтение блоков будет обращаться непосредственно к диску, так как кеши (системный и пользовательский) не используются.
    - Дисковая подсистема не способна «запомнить» ранее считанные страницы, поэтому каждое обращение вызывает полноценное чтение с устройства.
    - **Ожидается значительно более низкая пропускная способность и рост времени выполнения**, особенно при многократных повторениях одних и тех же блоков.

Результаты:

Без кеша
./io_repeat -f io_test.bin -b 4096 -u 1024 -R 50 -k on -t sequence

Using O_DIRECT

block_size=4096 unique_blocks=1024 repeats=50 pattern=sequence

Pass 1: bytes=4194304 time=0.277671 s throughput=14.41 MB/s

Pass 2: bytes=4194304 time=0.242790 s throughput=16.48 MB/s

Pass 3: bytes=4194304 time=0.203181 s throughput=19.69 MB/s

Pass 4: bytes=4194304 time=0.216371 s throughput=18.49 MB/s

Pass 5: bytes=4194304 time=0.247487 s throughput=16.16 MB/s

…

Pass 48: bytes=4194304 time=0.248356 s throughput=16.11 MB/s

Pass 49: bytes=4194304 time=0.224222 s throughput=17.84 MB/s

Pass 50: bytes=4194304 time=0.222816 s throughput=17.95 MB/s

С пользовательским кешем
./io_repeatCache -f io_test.bin -b 4096 -u 1024 -R 50 -k on -t sequence

Using VTPC

block_size=4096 unique_blocks=1024 repeats=50 pattern=sequence

Pass 1: bytes=4194304 time=0.294241 s throughput=13.59 MB/s

Pass 2: bytes=4194304 time=0.000826 s throughput=4842.62 MB/s

Pass 3: bytes=4194304 time=0.000719 s throughput=5563.28 MB/s

Pass 4: bytes=4194304 time=0.000723 s throughput=5532.50 MB/s

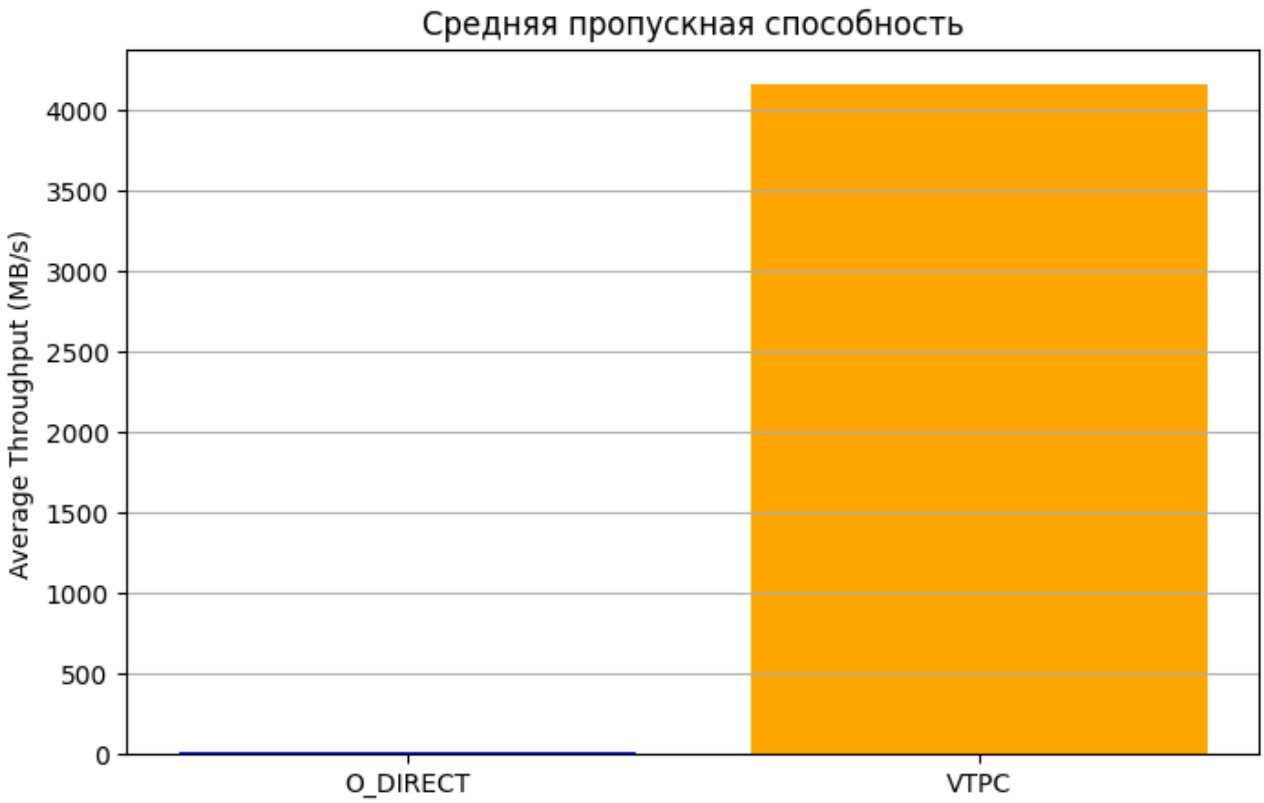
Pass 5: bytes=4194304 time=0.000597 s throughput=6700.17 MB/s

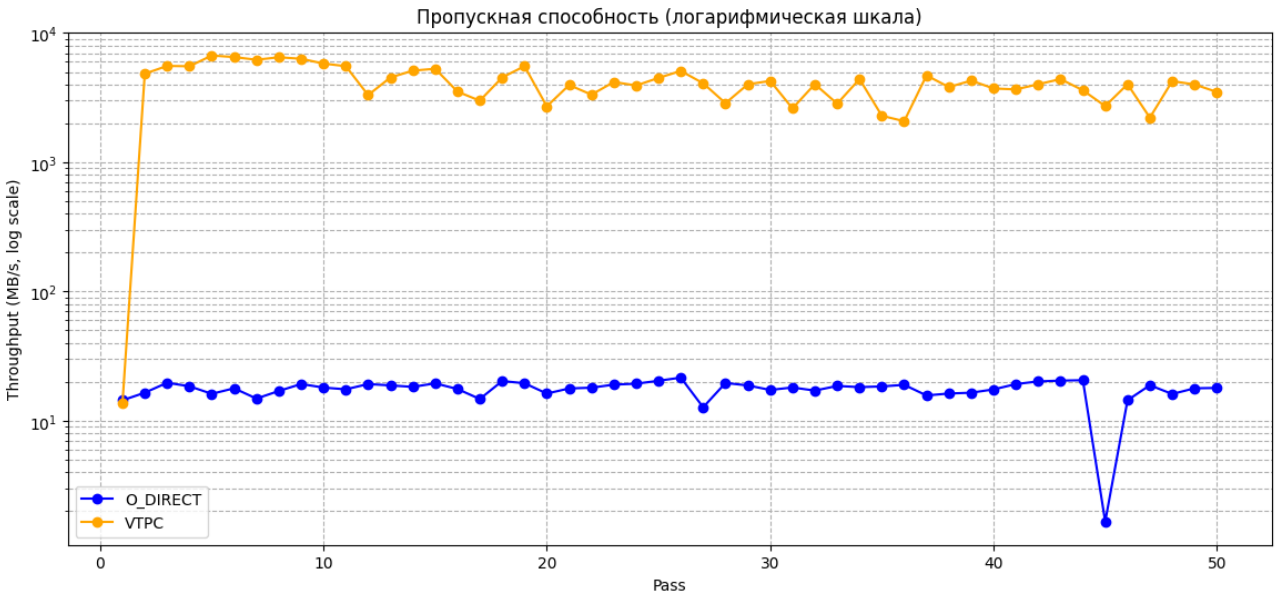
…

Pass 48: bytes=4194304 time=0.000943 s throughput=4241.78 MB/s

Pass 49: bytes=4194304 time=0.001000 s throughput=4000.00 MB/s

Pass 50: bytes=4194304 time=0.001143 s throughput=3499.56 MB/s





Вывод по гипотезе

Эксперимент подтвердил ожидаемое поведение двух режимов работы программы:

1. **Режим O_DIRECT (обход системного кеша)**:
   - Пропускная способность остаётся относительно стабильной и низкой — около **15–20 MB/s**, несмотря на повторное чтение одних и тех же блоков.
   - Это связано с тем, что каждая операция чтения напрямую обращается к диску, минуя системный кеш. В результате повторное чтение данных не ускоряется, а производительность ограничена скоростью дисковой подсистемы.
   - Выбросы, такие как падение до 1.66 MB/s (Pass 45), объясняются временными задержками при доступе к диску или планировщиком ОС, но общий тренд стабилен.
2. **Режим VTPC (пользовательский кеш)**:
   - После первого прохода наблюдается **резкое увеличение пропускной способности** — до нескольких тысяч MB/s.
   - Это полностью соответствует ожиданиям: библиотека VTPC кэширует страницы в памяти, и повторное чтение уже загруженных блоков выполняется почти мгновенно.
   - Небольшие колебания в скорости (Pass 12, 16, 17 и др.) связаны с вытеснением страниц из кеша по алгоритму FIFO и внутренними задержками управления памятью.

Итог:

- Гипотеза о том, что пользовательский кеш ускоряет повторное чтение, полностью подтвердилась.
- Режим без кеша (O_DIRECT) демонстрирует стабильное, но низкое быстродействие, ограниченное физическим диском.
- Режим с VTPC показывает **экспоненциальное ускорение** для повторного доступа к одним и тем же блокам, что наглядно подтверждает эффективность пользовательского кеша.

Заключение

В рамках лабораторной работы была реализована и исследована пользовательская библиотека ввода-вывода с собственным механизмом page cache, совместимая по интерфейсу с базовыми POSIX-операциями. Основной целью работы являлось изучение принципов организации страничного кеширования и оценка его влияния на производительность при различных типах дисковых нагрузок.

Проведённые эксперименты с последовательным и случайным чтением и записью показали, что в нагрузках с постоянно изменяющимся рабочим набором данных использование пользовательского page cache не приводит к заметному приросту производительности по сравнению с нативным вводом-выводом. Это объясняется тем, что данные практически не переиспользуются, и кеш не успевает приносить выгоду, а дополнительные накладные расходы на управление им частично компенсируют возможные преимущества.

В то же время дополнительный эксперимент с нагрузчиком repeat продемонстрировал принципиальную работоспособность реализованного кеша. При многократных обращениях к одному и тому же набору блоков наблюдается резкий рост производительности после первого прохода, что подтверждает корректность механизма хранения страниц в памяти и повторного обслуживания запросов без обращения к диску.

Таким образом, результаты лабораторной работы подтверждают, что эффективность page cache напрямую зависит от характера нагрузки и степени локальности доступа к данным. Реализованный пользовательский кеш корректно выполняет свои функции, однако в реальных сценариях без повторного доступа к данным его применение не даёт существенных преимуществ, что подчёркивает важность адаптации механизмов кеширования под конкретные типы рабочих нагрузок.