Университет ИТМО
Факультет проргаммной инженерии и компьютерной техники

Лабораторная работа №2
по диспицлине
"Операционные системы"

Выполнила: **Завацкая Дарьвя Вадимовна**
Группа: **P3334**
Вариант: **CLOCK, io-lat-write**
Преподаватель: **Смирнов Виктор Игоревич**

г. Санкт-Петербург, 2025

1 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.

Ограничения

- Программа (комплекс программ) должна быть реализован на языке C или C++.
- Запрещено использовать высокоуровневые абстракции над системными вызовами. Необходимо использовать, в случае Unix, процедуры libc.

2 Листинг кода

2.1 Основные программы

Листинг 1: Cache

```
#include "../include/cache.h"

Cache cache;
void cache_init() {
    cache.clock_hand = 0;
    for (int i = 0; i < CACHE_SIZE; i++) {
        cache.pages[i].fd = -1;
        cache.pages[i].file_offset = -1;
        cache.pages[i].used = 0;
        cache.pages[i].modified = 0;
    }
}

// Удаление данных по fd
void cache_clear_fd(int fd) {
    for (int i = 0; i < CACHE_SIZE; i++) {
        if (cache.pages[i].fd == fd) {
            cache.pages[i].fd = -1;
            cache.pages[i].file_offset = -1;
            cache.pages[i].used = 0;
            cache.pages[i].modified = 0;
        }
    }
}

// Поиск страницы в кэше
int cache_lookup(int fd, off_t offset){
    for (int i = 0; i  < CACHE_SIZE; i++) {
        if (cache.pages[i].fd == fd && cache.pages[i].file_offset
           == offset) {
            cache.pages[i].used = 1;
            return i;
        }
    }
    return -1;
}

// Поиск свободной страницы в кэше
int cache_find_free_page() {
    for (int i = 0; i < CACHE_SIZE; ++i) {
        if (cache.pages[i].used == 0) {
            return i;  // Свободная страница
        }
    }
    return -1;  // Свободных страниц нет
}

// Замещение страниц по алгоритму CLOCK
int cache_replace() {
```

```
49      while (true) {
50          int i = cache.clock_hand;
51          if (cache.pages[i].used == 0) {
52              // Если страница изменялась, перед удалением нужно записать
                    изменения на диск
53              if (cache.pages[i].modified == 1) {
54                  _lseek(cache.pages[i].fd, cache.pages[i].
                        file_offset, SEEK_SET);
55                  _write(cache.pages[i].fd, cache.pages[i].data,
                        BLOCK_SIZE);
56              }
57              return i;
58          }
59          cache.pages[i].used = 0;
60          cache.clock_hand = (cache.clock_hand + 1) % CACHE_SIZE;
61      }
62  }
63
64  // Чтение данных из файла
65  void cache_read(int fd, off_t offset, char *buf, size_t count) {
66      size_t bytes_read = 0;
67
68      while (bytes_read < count) {
69          int i = cache_lookup(fd, offset + bytes_read);
70
71          if (i != -1) {
72              // Если страница найдена в кэше, копируем данные
73              std::memcpy(buf + bytes_read, cache.pages[i].data,
                    BLOCK_SIZE);
74              bytes_read += BLOCK_SIZE;
75          } else {
76              // Если страница не найдена в кэше, нужно ее загрузить
77              i = cache_find_free_page();
78              if (i == -1) {
79                  i = cache_replace();  // Используем CLOCK для замены
80              }
81
82              cache.pages[i].fd = fd;
83              cache.pages[i].file_offset = offset + bytes_read;
84              cache.pages[i].used = 1;
85              cache.pages[i].modified = 0;
86
87              // Читаем данные с диска в кэш
88              _lseek(fd, offset + bytes_read, SEEK_SET);
89              _read(cache.pages[i].fd, cache.pages[i].data,
                    BLOCK_SIZE);
90
91              // Копируем данные с кэша в буфер
92              std::memcpy(buf + bytes_read, cache.pages[i].data,
                    BLOCK_SIZE);
93              bytes_read += BLOCK_SIZE;
94          }
95      }
96  }
97
98  // Запись данных в кэш
```

```
99  void cache_write(int fd, off_t offset, const char *buf, size_t
        count) {
100     size_t bytes_written = 0;
101     while (bytes_written < count) {
102         int i = cache_lookup(fd, offset + bytes_written);
103         if (i == -1) {
104             i = cache_find_free_page();
105             if (i == -1) {
106                 // Вызываем CLOCK если нет свободных страниц
107                 i = cache_replace();
108             }
109             cache.pages[i].fd = fd;
110             cache.pages[i].file_offset = offset + bytes_written;
111         }
112 
113         cache.pages[i].used = 1;
114         cache.pages[i].modified = 1;
115 
116         // Копируем данные с буфера в кэш
117         size_t page_offset = (offset + bytes_written) % BLOCK_SIZE;
118         size_t to_copy = std::min(count - bytes_written, BLOCK_SIZE
                - page_offset);
119         std::memcpy(cache.pages[i].data + page_offset, buf +
                bytes_written, to_copy);
120         bytes_written += to_copy;
121 
122     }
123 }
124 
125 
126 // Запись измененных страниц на диск
127 void cache_flush(int fd) {
128     for (int i = 0; i  < CACHE_SIZE; i++) {
129         if (cache.pages[i].fd == fd && cache.pages[i].modified ==
                1) {
130             _lseek(fd, cache.pages[i].file_offset, SEEK_SET);
131             _write(fd, cache.pages[i].data, BLOCK_SIZE);
132             cache.pages[i].modified = 0;
133         }
134     }
135 }
```

Листинг 2: Api для работы с файлами

```
1  #include "../include/file_worker.h"
2  #include "../include/cache.h"
3 
4  std::unordered_map<int, FileInfo> file_map;
5 
6  // Открытие файла по заданному пути файла, доступного для чтения
7  int lab2_open(const char *path) {
8      HANDLE file = CreateFileA(path,
9                                GENERIC_READ | GENERIC_WRITE,
10                               FILE_SHARE_READ,
11                               NULL,
12                               OPEN_EXISTING,
```

```
13                                  FILE_FLAG_NO_BUFFERING,
14                                  NULL);
15      if (file == INVALID_HANDLE_VALUE) {
16          DWORD error = GetLastError();
17          std::cerr << "Error opening file: " << error << std::endl;
18          return -1;
19      }
20
21      int fd = ::_open_osfhandle((intptr_t) file, _O_RDWR);
22      file_map[fd] = {0};
23      return fd;
24  }
25
26  // Закрытие файла по хэндлу
27  int lab2_close(int fd) {
28      cache_flush(fd);
29      cache_clear_fd(fd);
30      file_map.erase(fd);
31
32      HANDLE file = (HANDLE) _get_osfhandle(fd);
33      if (file == INVALID_HANDLE_VALUE) {
34          std::cerr << "Error: invalid file descriptor" << std::endl;
35          return -1;
36      }
37
38      if (_close(fd) == -1) {
39          std::cerr << "Error closing file descriptor" << std::endl;
40          return -1;
41      }
42      return 0;
43  }
44
45  // Чтение данных из файла
46  ssize_t lab2_read(int fd, void *buf, size_t count) {
47      off_t offset = file_map[fd].current_offset;
48      cache_read(fd, offset, (char *) buf, count);
49      file_map[fd].current_offset += count;
50      return count;
51  }
52
53  // Запись данных в файл
54  ssize_t lab2_write(int fd, const void *buf, size_t count) {
55      off_t offset = file_map[fd].current_offset;
56      cache_write(fd, offset, (const char *) buf, count);
57      file_map[fd].current_offset += count;
58      return count;
59  }
60
61  // Перестановка позиции указателя на данные файла
62  off_t lab2_lseek(int fd, off_t offset, int whence) {
63      if (whence != SEEK_SET) {
64          std::cerr << "Error: Only SEEK_SET is supported" << std::
                endl;
65          return -1;
66      }
67
```

```
68      if (file_map.find(fd) == file_map.end()) {
69          std::cerr << "Error: Invalid file descriptor" << std::endl;
70          return -1;
71      }
72
73      file_map[fd].current_offset = offset;
74      return offset;
75  }
76
77  // Синхронизация данных из кэша с диском
78  int lab2_fsync(int fd) {
79      cache_flush(fd);
80      HANDLE hFile = (HANDLE)_get_osfhandle(fd);
81      if (!FlushFileBuffers(hFile)) {
82          std::cerr << "Error flushing buffers" << std::endl;
83      }
84      return 0;
85  }
```

Листинг 3: IO-lat-write реализация с кэшированием

```
1  #include "../include/io_lat_write.h"
2  #include "../include/cache.h"
3  #include "../include/file_worker.h"
4  #include "../include/tools.h"
5
6  void IOLatWriteWithCash(int iterations, const std::string& filePath
       ) {
7      const int block_size = 1024; // размер блока 1K
8      std::vector<char> data(block_size, 'A');
9
10     int fd = lab2_open(filePath.c_str());
11     if (fd == -1) {
12         std::cerr << "Error opening file: " << filePath << std::
               endl;
13         return;
14     }
15
16     // Генератор случайных чисел для выбора случайных смещений в файле
17     std::random_device rd;
18     std::mt19937 gen(rd());
19     std::uniform_int_distribution<> dis(0, 1023);  // Генерация
           случайных смещений
20
21     auto start_time = std::chrono::high_resolution_clock::now();
22
23     // Цикл записи данных в файл заданное количество раз
24     for (int i = 0; i < iterations; ++i) {
25         int offset = dis(gen);  // Генерация случайного смещения
26
27         lab2_lseek(fd, offset, SEEK_SET);
28         lab2_write(fd, data.data(), block_size);
29     }
30     lab2_fsync(fd);
31
32     auto end_time = std::chrono::high_resolution_clock::now();
```

```
33      auto elapsed_time = std::chrono::duration_cast<std::chrono::
            milliseconds>(end_time - start_time).count();
34      std::cout << "average write time per iteration: " <<
            static_cast<double>(elapsed_time) / iterations << " [ms]" <<
             std::endl;
35
36      lab2_close(fd);
37  }
38
39  #ifndef TESTING
40  int main(int argc, char* argv[]) {
41      if (argc != 3) {
42          std::cerr << "Usage: io-lat-write <outputFile> <number of
                iterations>" << std::endl;
43          return 1;
44      }
45
46      int iterations = std::stoi(argv[2]);
47      IOLatWriteWithCash(iterations, argv[1]);
48      return 0;
49  }
50  #endif
```

2.2 Тесты

Листинг 4: Cache test

```
1   #include <gtest/gtest.h>
2   #include "../../include/cache.h"
3
4   class CacheTest : public ::testing::Test {
5   protected:
6       void SetUp() override {
7           cache_init();
8       }
9   };
10
11  TEST_F(CacheTest, ReadWriteTest) {
12      const char test_data1[BLOCK_SIZE] = "First data";
13      const char test_data2[BLOCK_SIZE] = "Second data";
14      char buffer[BLOCK_SIZE];
15
16      cache_write(0, 0, test_data1, BLOCK_SIZE);
17      cache_write(0, BLOCK_SIZE, test_data2, BLOCK_SIZE);
18
19      cache_read(0, 0, buffer, BLOCK_SIZE);
20      EXPECT_EQ(memcmp(test_data1, buffer, BLOCK_SIZE), 0);
21
22      cache_read(0, BLOCK_SIZE, buffer, BLOCK_SIZE);
23      EXPECT_EQ(memcmp(test_data2, buffer, BLOCK_SIZE), 0);
24  }
25
26  TEST_F(CacheTest, ReplaceTest) {
27      const int num_pages = 66;  // Количество страниц превышающее размер
            кэша (64)
28      char test_data[num_pages][BLOCK_SIZE];
```

```
29      char buffer[BLOCK_SIZE];
30
31      // Заполняем тестовые данные
32      for (int i = 0; i < num_pages; i++) {
33          memset(test_data[i], 'A' + (i % 26), 1);
34      }
35
36      // Заполняем полностью кэш данными
37      for (int i = 0; i < CACHE_SIZE; i++) {
38          off_t offset = i * BLOCK_SIZE;
39          cache_write(0, offset, test_data[i], BLOCK_SIZE);
40      }
41
42      // Записываем дополнительные страницы, вызывая замещение в кэше
43      for (int i = (num_pages - CACHE_SIZE); i > 0; i--) {
44          off_t offset = (num_pages - i) * BLOCK_SIZE;
45          cache_write(0, offset, test_data[num_pages - i], BLOCK_SIZE
                );
46      }
47
48      // Проверяем корректность данных в замещенных страницах
49      for (int i = (num_pages - CACHE_SIZE); i > 0; i--) {
50          cache_read(0, (num_pages - i) * BLOCK_SIZE, buffer,
                BLOCK_SIZE);
51          EXPECT_EQ(memcmp(test_data[(num_pages - i)], buffer,
                BLOCK_SIZE), 0);
52      }
53      // Проверяем, что оставшиеся страницы не были заменены
54      for (int i = num_pages - CACHE_SIZE; i < num_pages; i++) {
55          cache_read(0, i * BLOCK_SIZE, buffer, BLOCK_SIZE);
56          EXPECT_EQ(memcmp(test_data[i], buffer, BLOCK_SIZE), 0);
57      }
58  }
```

Листинг 5: Api test

```
1   #include <gtest/gtest.h>
2   #include "../../include/file_worker.h"
3   #include "../../include/cache.h"
4   #include "../../include/tools.h"
5   #include <fstream>
6   #include <cstdio>
7
8   class TestEnvironment : public ::testing::Environment {
9   public:
10      static constexpr const char *test_filename = "test_fw.txt";
11
12      void SetUp() override {
13          std::ofstream test_file(test_filename);
14          ASSERT_TRUE(test_file) << "Failed to create test file";
15          test_file.close();
16
17          cache_init();
18      }
19
20      void TearDown() override {
```

```
21          std::remove(test_filename);
22      }
23  };
24
25  // Регистрируем глобальное окружение
26  ::testing::Environment* const test_env = ::testing::
        AddGlobalTestEnvironment(new TestEnvironment());
27
28  TEST(FileWorkerTest, Lab2OpenCloseTest) {
29      int fd = lab2_open(TestEnvironment::test_filename);
30      ASSERT_NE(fd, -1) << "Failed to open test file";
31      EXPECT_EQ(lab2_close(fd), 0);
32  }
33
34  TEST(FileWorkerTest, Lab2WriteTest) {
35      int fd = lab2_open(TestEnvironment::test_filename);
36      ASSERT_NE(fd, -1);
37
38      const char test_data[] = "good day, good life";
39      ssize_t bytes_written = lab2_write(fd, test_data, sizeof(
            test_data) - 1);
40      EXPECT_EQ(bytes_written, sizeof(test_data) - 1);
41
42      lab2_lseek(fd, 0, SEEK_SET);
43
44      char buffer[BLOCK_SIZE] = {};
45      lab2_read(fd, buffer, sizeof(test_data) - 1);
46
47      EXPECT_EQ(memcmp(test_data, buffer, sizeof(test_data) - 1), 0);
48      EXPECT_EQ(lab2_close(fd), 0);
49  }
50
51  TEST(FileWorkerTest, Lab2SeekTest) {
52      int fd = lab2_open(TestEnvironment::test_filename);
53      ASSERT_NE(fd, -1);
54
55      const char test_data[] = "data after seek";
56      lab2_write(fd, test_data, sizeof(test_data) - 1);
57      lab2_fsync(fd);
58
59      EXPECT_EQ(lab2_lseek(fd, 0, SEEK_SET), 0);
60      EXPECT_EQ(lab2_lseek(fd, 1, SEEK_SET), 1);
61
62      EXPECT_EQ(lab2_close(fd), 0);
63  }
64
65  TEST(FileWorkerTest, Lab2FsyncTest) {
66      int fd = lab2_open(TestEnvironment::test_filename);
67      ASSERT_NE(fd, -1);
68
69      const char test_data[] = "hell";
70      lab2_write(fd, test_data, sizeof(test_data) - 1);
71      EXPECT_EQ(lab2_fsync(fd), 0);
72
73      EXPECT_EQ(lab2_close(fd), 0);
74  }
```

Листинг 6: IO-lat-write test

```
#include <gtest/gtest.h>
#include <chrono>
#include <fstream>
#include <iostream>
#include <cstdio>

#include "../../include/file_worker.h"
#include "../../include/cache.h"
#include "../../include/io_lat_write.h"


TEST(IOTest, CachePerformance) {
    std::string file_path = "test_file_io.txt";
    int iterations = 1000;

    std::ofstream test_file(file_path);
    test_file.close();

    auto start_time_no_cache = std::chrono::high_resolution_clock::
        now();
    IOLatWrite(iterations, file_path);
    auto end_time_no_cache = std::chrono::high_resolution_clock::
        now();
    auto elapsed_time_no_cache = std::chrono::duration_cast<std::
        chrono::milliseconds>(end_time_no_cache -
        start_time_no_cache).count();

    auto start_time_with_cache = std::chrono::high_resolution_clock
        ::now();
    IOLatWriteWithCash(iterations, file_path);
    auto end_time_with_cache = std::chrono::high_resolution_clock::
        now();
    auto elapsed_time_with_cache = std::chrono::duration_cast<std::
        chrono::milliseconds>(end_time_with_cache -
        start_time_with_cache).count();

    std::cout << "Time without cache: " << elapsed_time_no_cache <<
         " ms" << std::endl;
    std::cout << "Time with cache: " << elapsed_time_with_cache <<
        " ms" << std::endl;

    std::remove(file_path.c_str());
    ASSERT_LT(elapsed_time_with_cache, elapsed_time_no_cache);
}
```

3 Результаты измерений и метрик

IO-lat-write

Запуск программы-нагрузчика без кэширования

```
shell>.\io-lat-write "D:\dasha\UNI\OS\io.txt" 10000
average write time per iteration: 1.2712 [ms]
```

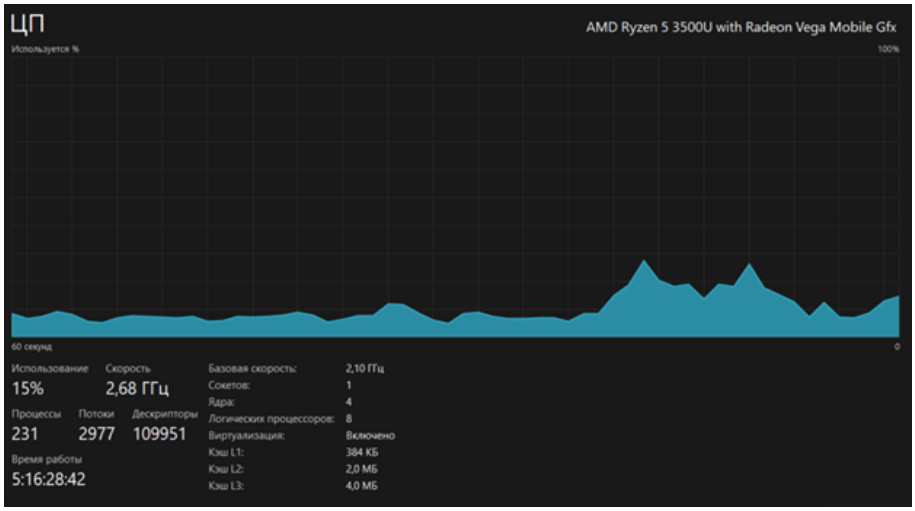

Рис. 1: Нагрузка ЦП

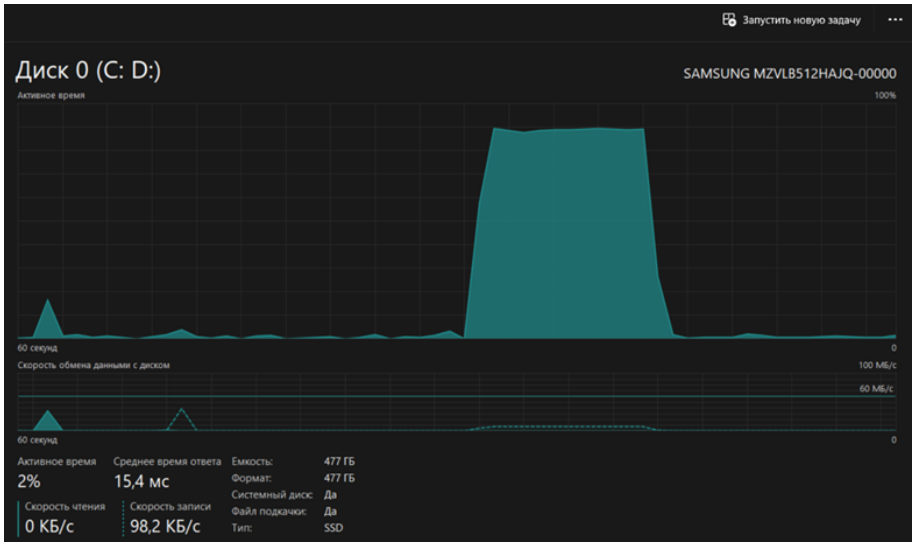

Рис. 2: Нагрузка диск

Рис. 3: IO-lat-write properties (Performance)

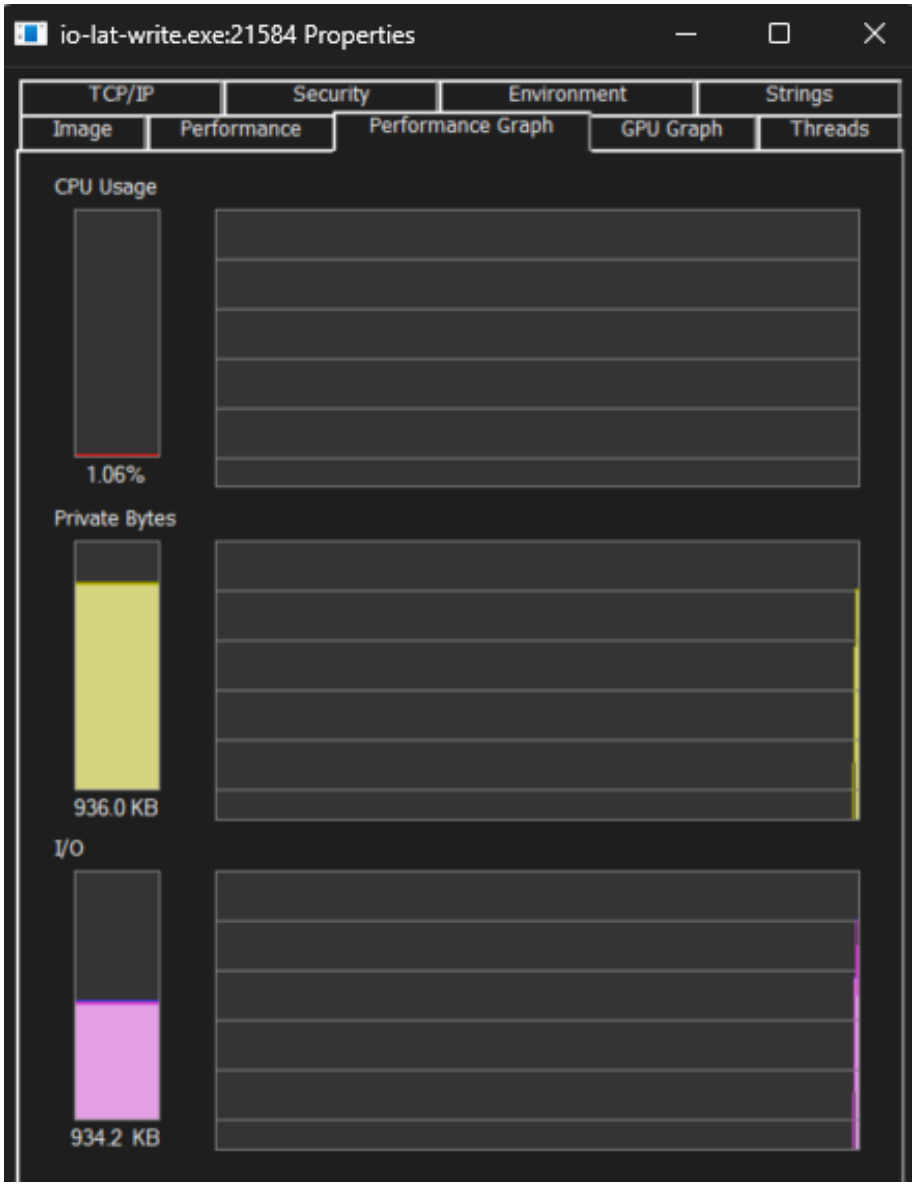

Рис. 4: IO-lat-write properties (Performance Graph)

Полученные результаты:

- **Время выполнения:** Программа выполняется в пределах предполагаемого диапазона (2000–10000 мс), но с небольшим превышением.
- **ЦП:** Процессор использовался в небольшом объеме (15-20%), что подтверждает низкую нагрузку на вычисления, поскольку работа программы в основном направлена на записи на диск.
- **IO Performance:** Задержки и высокая нагрузка на диск (90%) объясняются частыми операциями записи на диск в случайные места файла. Ожидания ввода-вывода высоки из-за случайных смещений, что замедляет процесс записи.

Запуск программы-нагрузчика с кэшированием

```
shell>.\io_lat_write_w_cash test_io.txt 10000
average write time per iteration: 0.0037 [ms]
```

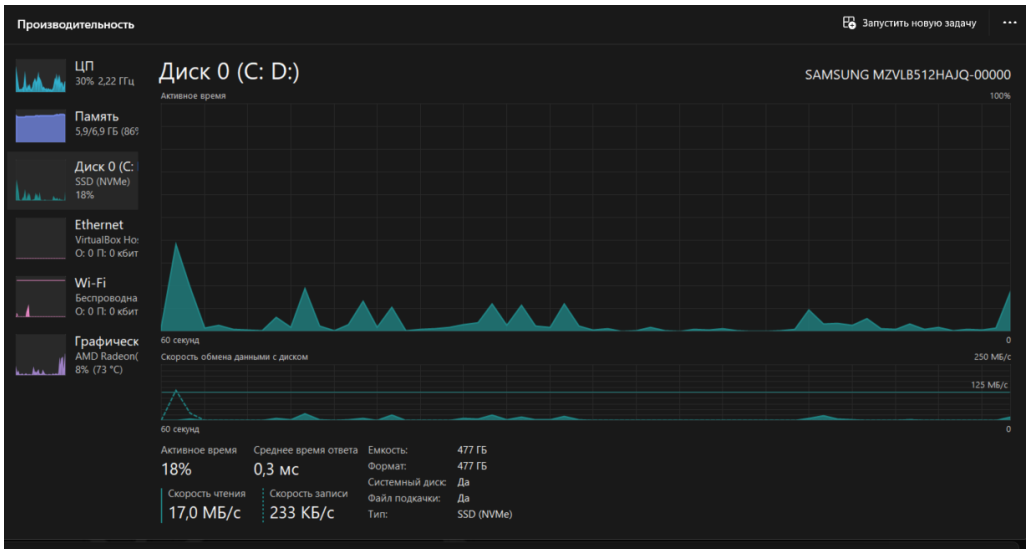

Рис. 5: Нагрузка диск

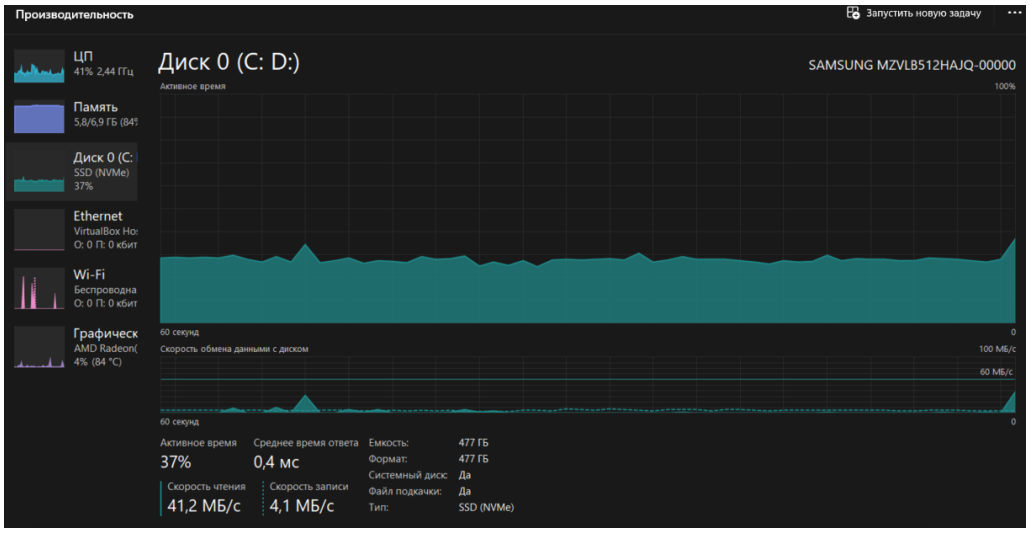

Рис. 6: Нагрузка на диск при переполненном кэше

Полученные результаты:

Отследить подробнее процесс программы не удалось, т.к. она очень быстро завершается.

- **Время выполнения:** Программа с использованием кэша выполняется намного быстрее чем программа, которая результаты каждой итерация записывает на диск.

- **ЦП:** Процессор использовался также в небольшом объеме (15-20%), что подтверждает низкую нагрузку на вычисления, поскольку работа программы в основном направлена на записи на диск.
- **IO Performance:** Нагрузки на диск не наблюдалось, но при переполнении кэша и последующим замещением страниц при каждой итерации нагрузка на диск составляет 20-30% .

4 Вывод

Во время выполнения лабораторной работы я изучила работу кэша, реализовала свою собственную систему кэширования, API для работы с файлами и протестировала программу-нагрузчик из ЛР1. Как и ожидалось, программа с кэшированием выполнилась значительно быстрее, чем с прямой записью на диск. Самым сложным этапом оказалось реализовать правильное смещение внутри кэша, а также разработать рабочие тесты, которые покрывают не только основную функциональность, но и различные сценарии работы системы.