



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет имени  
Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа**  
**по дисциплине «Операционные системы»**  
**на тему:**  
**Буферизованный и небуферизованный ввод-вывод**

Студент группы ИУ7-64Б

\_\_\_\_\_  
(Подпись, дата)

Сухилина А. А.  
(Фамилия И.О.)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Рязанова Н. Ю.  
(Фамилия И.О.)

Москва, 2025

# Структура FILE

Описание структуры FILE

```
struct _IO_FILE;

/* The opaque type of streams.
   This is the definition used elsewhere. */
typedef struct _IO_FILE FILE;
```

Описание структуры \_IO\_FILE

```
struct _IO_FILE
{
    int _flags; /*High-order word is _IO_MAGIC; rest is flags. */

    /* The following pointers correspond to the C++ streambuf protocol. */
    char *_IO_read_ptr; /* Current read pointer */
    char *_IO_read_end; /* End of get area. */
    char *_IO_read_base; /* Start of putback+get area. */
    char *_IO_write_base; /* Start of put area. */
    char *_IO_write_ptr; /* Current put pointer. */
    char *_IO_write_end; /* End of put area. */
    char *_IO_buf_base; /* Start of reserve area. */
    char *_IO_buf_end; /* End of reserve area. */

    /* The following fields are used to support backing up and undo. */
    char *_IO_save_base; /* Pointer to start of non-current get area. */
    char *_IO_backup_base; /* Pointer to first valid character of backup area */
    char *_IO_save_end; /* Pointer to end of non-current get area. */

    struct _IO_marker *_markers;
    struct _IO_FILE *_chain;

    int _fileno;
    int _flags2;
    __off_t _old_offset; /* This used to be _offset but it's too small. */

    /* 1+column number
       of pbase(); 0 is unknown. */
    unsigned short _cur_column;
    signed char _vtable_offset;
    char _shortbuf[1];

    _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

## Первая программа

### Однопоточная программа

```
#include <stdio.h>
#include <fcntl.h>
int main()
{
    int fd = open("alphabet.txt", O_RDONLY);

    FILE *fs1 = fdopen(fd, "r");
    char buff1[20];
    setvbuf(fs1, buff1, _IOFBF, 20);

    FILE *fs2 = fdopen(fd, "r");
    char buff2[20];
    setvbuf(fs2, buff2, _IOFBF, 20);

    int flag1 = 1, flag2 = 2;
    while(flag1 == 1 || flag2 == 1)
    {
        char c;
        flag1 = fscanf(fs1, "%c", &c);
        if (flag1 == 1) {
            fprintf(stdout, "%c", c);
        }
        flag2 = fscanf(fs2, "%c", &c);
        if (flag2 == 1) {
            fprintf(stdout, "%c", c);
        }
    }
    return 0;
}
```

Результат работы программы:

Aubvcwdxeyfzghijklmnopqrst

## Анализ работы программы

Системный вызов `open()` создаёт новый файловый дескриптор для файла «`alphabet.txt`», открытого только на чтение (`O_RDONLY`), и возвращает индекс соответствующей записи в таблице файловых дескрипторов. Затем с помощью функции `fdopen()` создаются две структуры типа `FILE` (`fs1` и `fs2`), которые ссылаются на один и тот же файловый дескриптор. С помощью `setvbuf()` для каждого потока (`fs1` и `fs2`) задаются собственные буферы (`buff1` и `buff2` размером 20 байт) и тип буферизации `_IOFBF` (Fully buffered — полная буферизация). Это означает, что чтение из файла будет происходить блоками по 20 символов, а не по одному символу за раз. В цикле `while` поочерёдно выполняются вызовы `fscanf()` для чтения символов из `fs1` и `fs2`.

- При первом вызове `fscanf(fs1, ...)` происходит заполнение `buff1` первыми 20 символами файла, а указатель позиции `f_pos` перемещается на 21-й символ.
- Затем `fscanf(fs2, ...)` заполняет `buff2` следующими 20 символами (если они есть), и `f_pos` перемещается дальше.

После каждого успешного чтения (`flag1 == 1` или `flag2 == 1`) символ выводится с помощью `fprintf()`. Оба потока (`fs1` и `fs2`) используют один файловый дескриптор, но у каждого свой буфер и указатель позиции, из-за чего символы выводятся поочерёдно. Программа сначала читает символ 'a' из `fs1`, затем символ 'u' из `fs2`.

## Многопоточная программа

```
#include <stdio.h>
#include <fcntl.h>
#include <pthread.h>

void* reader_thread(void* arg) {
    FILE* fs = (FILE*)arg;
    int flag = 1;
    char c;

    while (flag == 1) {
        flag = fscanf(fs, "%c", &c);
        if (flag == 1)
        {
            fprintf(stdout,"%c",c);
        }
    }

    return NULL;
}

int main()
{
    int fd = open("alphabet.txt",O_RDONLY);
    FILE *fs1 = fdopen(fd,"r");
    char buff1[20];
    setvbuf(fs1,buff1,_IOFBF,20);

    FILE *fs2 = fdopen(fd,"r");
    char buff2[20];
    setvbuf(fs2,buff2,_IOFBF,20);

    pthread_t thread;
    if (pthread_create(&thread, NULL, reader_thread, (void*)fs2) != 0) {
        perror("Failed to create thread");
        fclose(fs1);
        fclose(fs2);
        close(fd);
        return 1;
    }

    int flag = 1;
    char c;
    while (flag == 1) {
        flag = fscanf(fs1, "%c", &c);
        if (flag == 1) {
            fprintf(stdout,"%c",c);
        }
    }
    pthread_join(thread, NULL);
    fclose(fs1);
    fclose(fs2);
    close(fd);
    return 0;
}
```

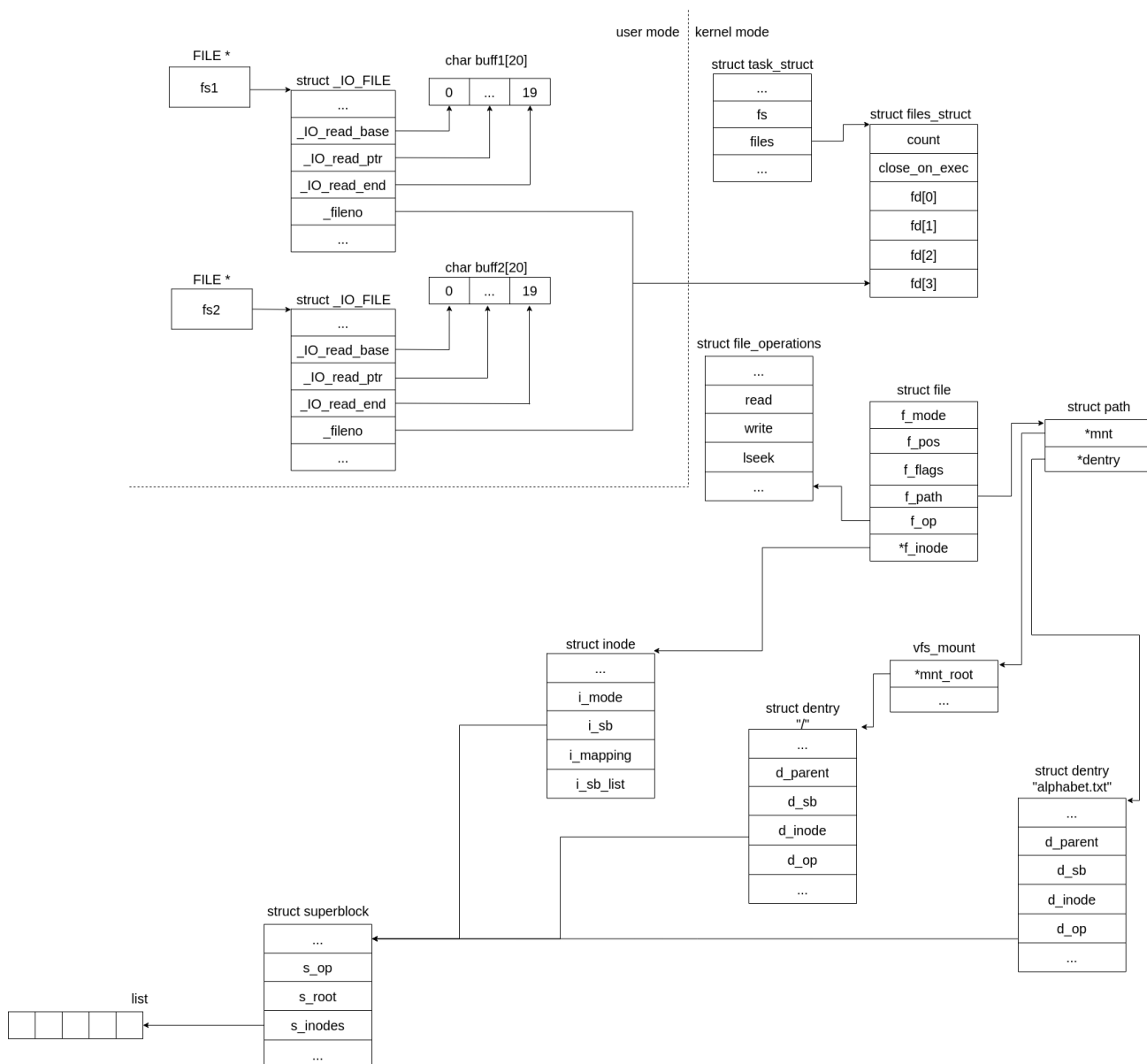
Результат работы программы:

Abcdefghijklmnopqrstuvwxyz

## Анализ работы программы

Первый поток обрабатывает первым, из-за временных затрат на создание второго потока.

Схема связи структур в первой программе



## Вторая программа

### Однопоточная программа

```
#include <fcntl.h>

int main()
{
    char c;
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);
    int flag1 = 1, flag2 = 2;
    while(flag1 != 0 || flag2 != 0)
    {
        flag1 = read(fd1, &c, 1);
        if (flag1 != 0)
        {
            write(1, &c, 1);
        }
        flag2 = read(fd2, &c, 1);
        if (flag2 != 0)
        {
            write(1, &c, 1);
        }
    }
    close(fd1);
    close(fd2);
    return 0;
}
```

Результат работы программы:

```
AAbbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz
```

### Анализ работы программы

Программа открывает файл alphabet.txt дважды, создавая два дескриптора (fd1 и fd2). Каждый дескриптор имеет свой указатель позиции в файле. В цикле программа поочередно читает по одному символу через fd1 и fd2, выводя каждый считанный символ на экран. В результате каждый символ из файла выводится дважды: сначала прочитанный через fd1, затем через fd2.

## Многопоточная программа

```
#include <fcntl.h>
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

void* reader_thread(void* arg) {
    int fd = (int)(long)arg;
    char c;

    while (read(fd, &c, 1) > 0) {
        write(1, &c, 1);
    }

    return NULL;
}

int main() {

    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, reader_thread, (void*)(long)fd1);
    pthread_create(&thread2, NULL, reader_thread, (void*)(long)fd2);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    close(fd1);
    close(fd2);
    return 0;
}
```

Результат работы программы

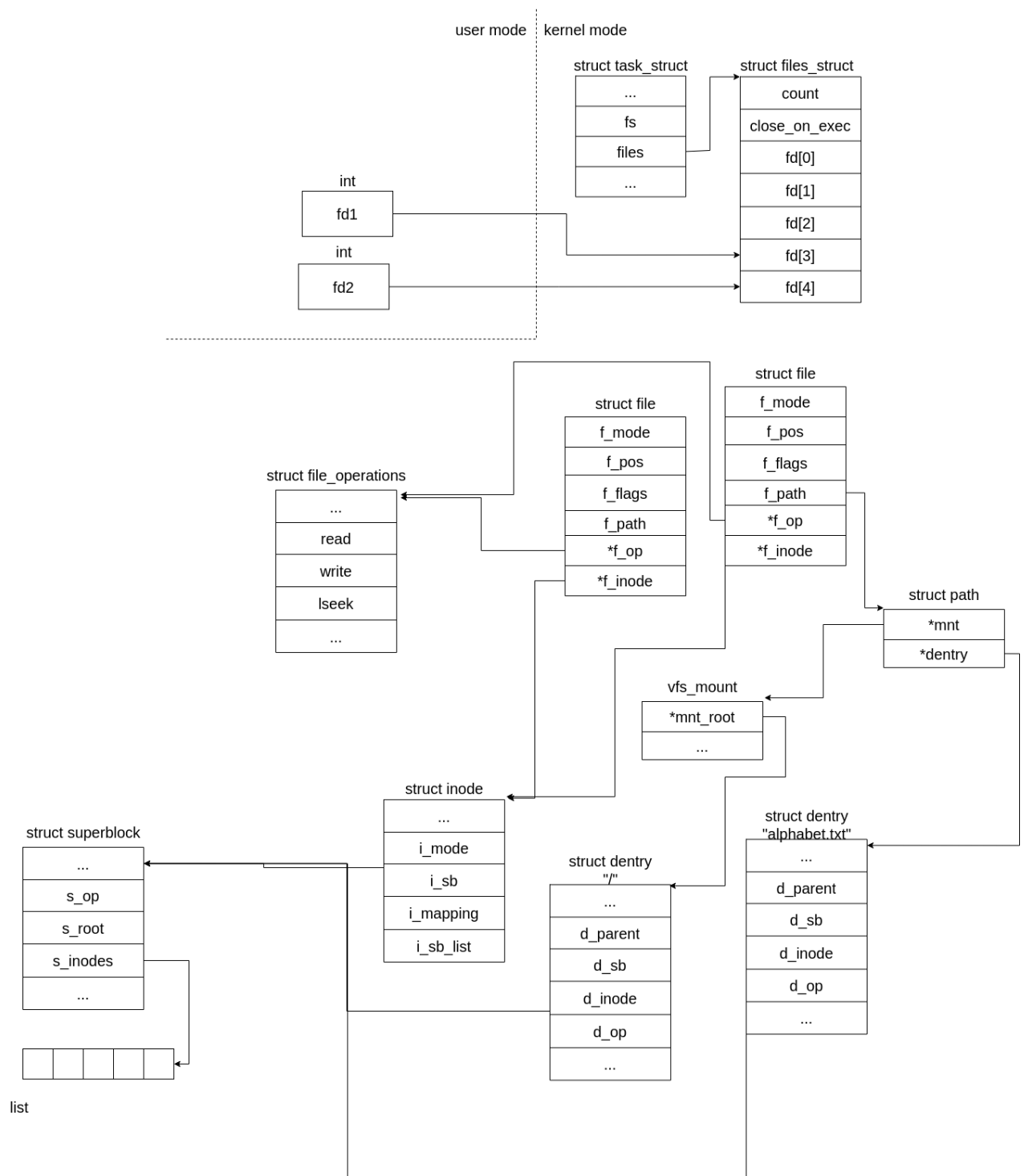
```
AbcdefgAhbicijdkelfmgnhoipjqkrlsmtnuovpqxrysztuvwxyz
```

## Анализ работы программы

Программа создает два файловых дескриптора (fd1 и fd2) для одного и того же файла alphabet.txt, каждый со своим указателем позиции. Затем она создает два потока, каждый из которых читает файл через свой дескриптор и выводит символы. Каждый символ файла будет прочитан и выведен дважды - по одному разу каждым потоком. Потоки работают параллельно, поэтому порядок вывода символов предугадать невозможно.



## Схема структур для второй программы



## Третья программа

### Структура stat

```
struct stat {  
    dev_t st_dev ; /* ID of device containing file */  
    ino_t st_ino ; /* inode number */  
    mode_t st_mode ; /* protection */  
    nlink_t st_nlink ; /* number of hard links */  
    uid_t st_uid ; /* user ID of owner */  
    gid_t st_gid ; /* group ID of owner */  
    dev_t st_rdev ; /* device ID ( if special file ) */  
    off_t st_size ; /* total size , in bytes */  
    blksize_t st_blksize ; /* blocksize for file system I/ O */  
    blkcnt_t st_blocks ; /* number of 512 B blocks allocated */  
    time_t st_atime ; /* time of last access */  
    time_t st_mtime ; /* time of last modification */  
    time_t st_ctime ; /* time of last status change */  
};
```

## Однопоточная реализация(без флага O-APPEND)

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>

void file_info(int fd)
{
    struct stat statbuff;
    stat("result", &statbuff);
    printf("st_ino: %ld ", statbuff.st_ino);
    printf("st_size: %ld ", statbuff.st_size);
    printf("pos: %ld \n", lseek(fd, 0, SEEK_CUR));
}

int main()
{
    FILE* fs1 = fopen("res", "w");
    int fd1 = fileno(fs1);
    file_info(fd1);
    FILE* fs2 = fopen("res", "w");
    int fd2 = fileno(fs2);
    file_info(fd2);
    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c%2) {
            fwrite(&c, sizeof(char), 1, fs1);
        }
        else {
            fwrite(&c, sizeof(char), 1, fs2);
        }
    }

    fclose(fs1);
    file_info(fd1);
    fclose(fs2);
    file_info(fd2);

    return 0;
}
```

Результат работы программы:

```
inode: 1467025 size: 0 pos: 0
inode: 1467025 size: 0 pos: 0
inode: 1467025 size: 13 pos: -1
inode: 1467025 size: 13 pos: -1

bdfhjlnprtvxz
```

## Анализ программы

Функция `forep()` вызывается дважды, она возвращает два указателя на `FILE` - `fs1` и `fs2`. Оба этих указателя ссылаются на один и тот же файловый дескриптор в системной таблице открытых файлов. Существует три ситуации, когда буфер записи сбрасывается в основной файл:

- Когда буфер полностью заполняется
- При явном вызове `fflush()` (принудительная запись)
- При закрытии файла с помощью `fclose()`

В случае, когда файл закрывается, сначала записываются 13 байт из буфера `fs1`, а затем данные из буфера `fs2`. При этом информация из буфера `fs2` может перезаписать содержимое файла. Чтобы избежать потери данных, при открытии файла рекомендуется использовать флаг `O-APPEND`, который гарантирует, что все записи будут добавляться в конец файла, а не перезаписывать существующее содержимое.

## Однопоточная реализация(с флагом O-APPEND)

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>

void file_info(int fd)
{
    struct stat statbuff;
    stat("res", &statbuff);
    printf("st_info: %ld ", statbuff.st_ino);
    printf("st_size: %ld ", statbuff.st_size);
    printf("pos: %ld \n", lseek(fd, 0, SEEK_CUR));
}

int main()
{
    int fd1 = open("res", O_WRONLY | O_CREAT | O_APPEND, 0666);
    file_info(fd1);
    int fd2 = open("res", O_WRONLY | O_APPEND);
    file_info(fd2);

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c%2) {
            write(fd1, &c, sizeof(char));
        }
        else {
            write(fd2, &c, sizeof(char));
        }
    }

    close(fd1);
    file_info(fd1);
    close(fd2);
    file_info(fd2);

    return 0;
}
```

Результат работы программы:

```
inode: 1458927 size: 0 pos: 0
inode: 1458927 size: 0 pos: 0
inode: 1458927 size: 26 pos: -1
inode: 1458927 size: 26 pos: -1

abcdefghijklmnopqrstuvwxyz
```

## Анализ программы

Программа дважды вызывает системный вызов `open()` для файла `alphabet.txt`, получая два файловых дескриптора (`fd1` и `fd2`). Оба вызова используют флаг `O_APPEND`, который гарантирует, что при любой операции записи файловые указатели автоматически будут устанавливаться в конец файла. Это предотвращает ситуацию, когда данные могут перезаписываться или теряться при параллельной работе с файлом.

## Схема связи структур в третьей программе

