C Programming

4_files

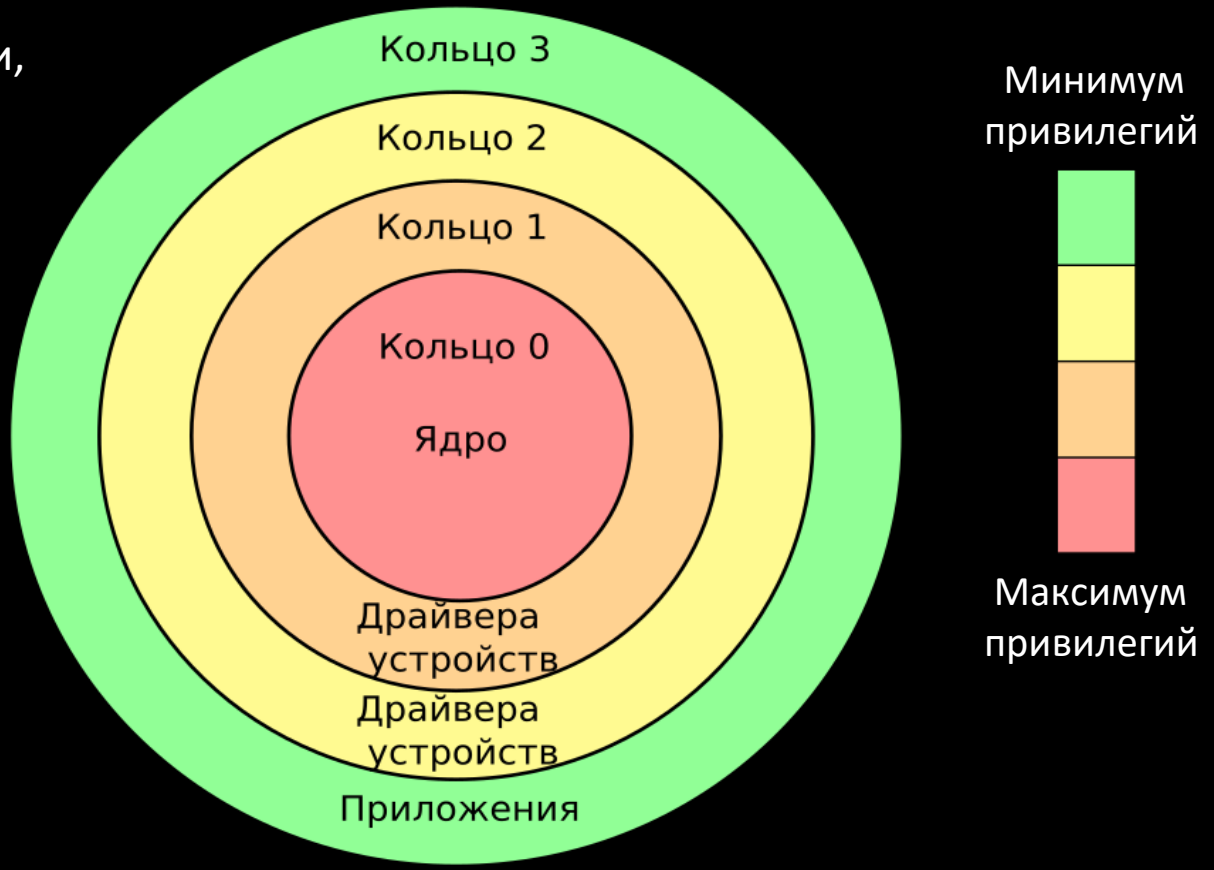
Files

Прежде чем пойдем к файлам неплохо будет копнуть глубже и понять пару вещей про ОС

Protection Rings

Кольца защиты — архитектура информационной безопасности и функциональной отказоустойчивости, реализующая аппаратное разделение системного и пользовательского уровней привилегий.

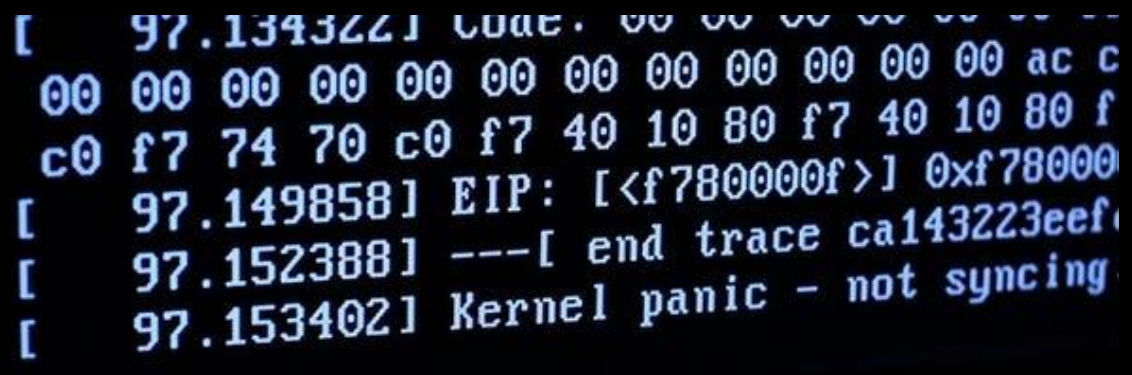
https://en.wikipedia.org/wiki/Protection_ring

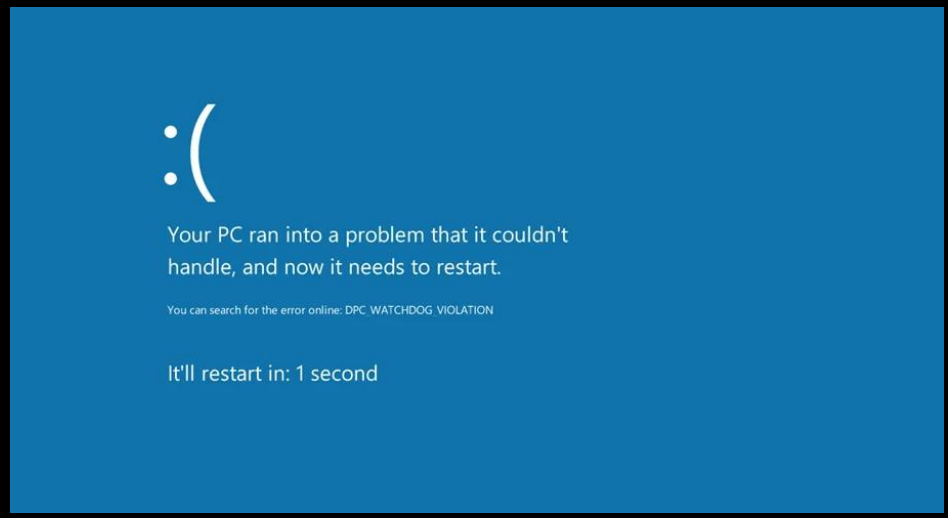


В ОС есть разделение на области (кольца), в каждом из которых есть свой уровень привилегий для исполняемых программ. Самое внешнее кольцо - менее привилегированное, самое внутреннее наоборот. Самый высокий уровень привилегий у ядра. Реализуется в процессорах и ОС (процессор понимает с каким уровнем работает благодаря разным режимам)

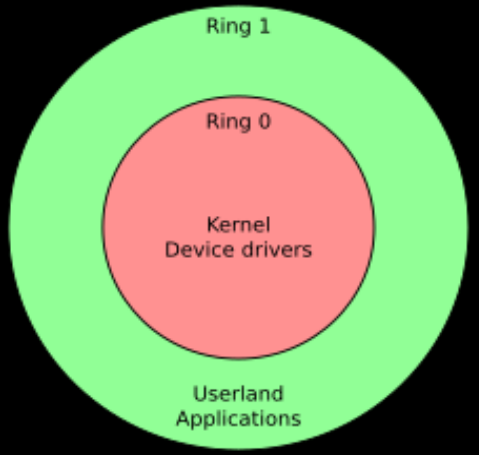
Kernel

Ядро́ (англ. kernel) ОС — это центральная часть ОС, которая управляет основными ресурсами компьютера, такими как процессор, память и устройства ввода-вывода. Можно представить его как "посредника" между аппаратным обеспечением (железо) и программами, который обеспечивает их взаимодействие. Ядро отвечает за выполнение задач, таких как запуск программ, управление памятью, обработка запросов от устройств и многое другое.





Kernel и User space



В современных ОС обычно используется две области - kernel space и user space. Kernel space - пространство ядра, а user space - пользовательское пространство.

Такое разделение позволяет изолировать пользовательские приложения от критически важного ядра операционной системы, что защищает систему от неисправностей и потенциальных угроз безопасности

Kernel Space (пространство ядра):

- Это область памяти, где выполняется код ОС и ядра, а также драйверы устройств.
- В этом пространстве выполняются привилегированные операции, такие как управление памятью, управление процессами и взаимодействие с аппаратным обеспечением.
- Программы, работающие в ядерном пространстве, имеют полный доступ ко всем ресурсам системы. Это означает, что ошибки в таком коде могут привести к сбоям всей системы (поможет перезапуск системы).

User Space (пространство пользователя):

- Это область памяти, где выполняются пользовательские приложения и программы.
- Программы в пространстве пользователя не имеют прямого доступа к аппаратному обеспечению и ресурсам системы. Они должны взаимодействовать с ядром через системные вызовы (syscall).
- Это разделение повышает безопасность системы, т.к. сбой в программе не приводит к сбою ядра и всей ОС (не нужно перезапускать систему).

Syscall (Системный вызов)

Систе́мный вы́зов (англ. system call) в программировании и вычислительной технике — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

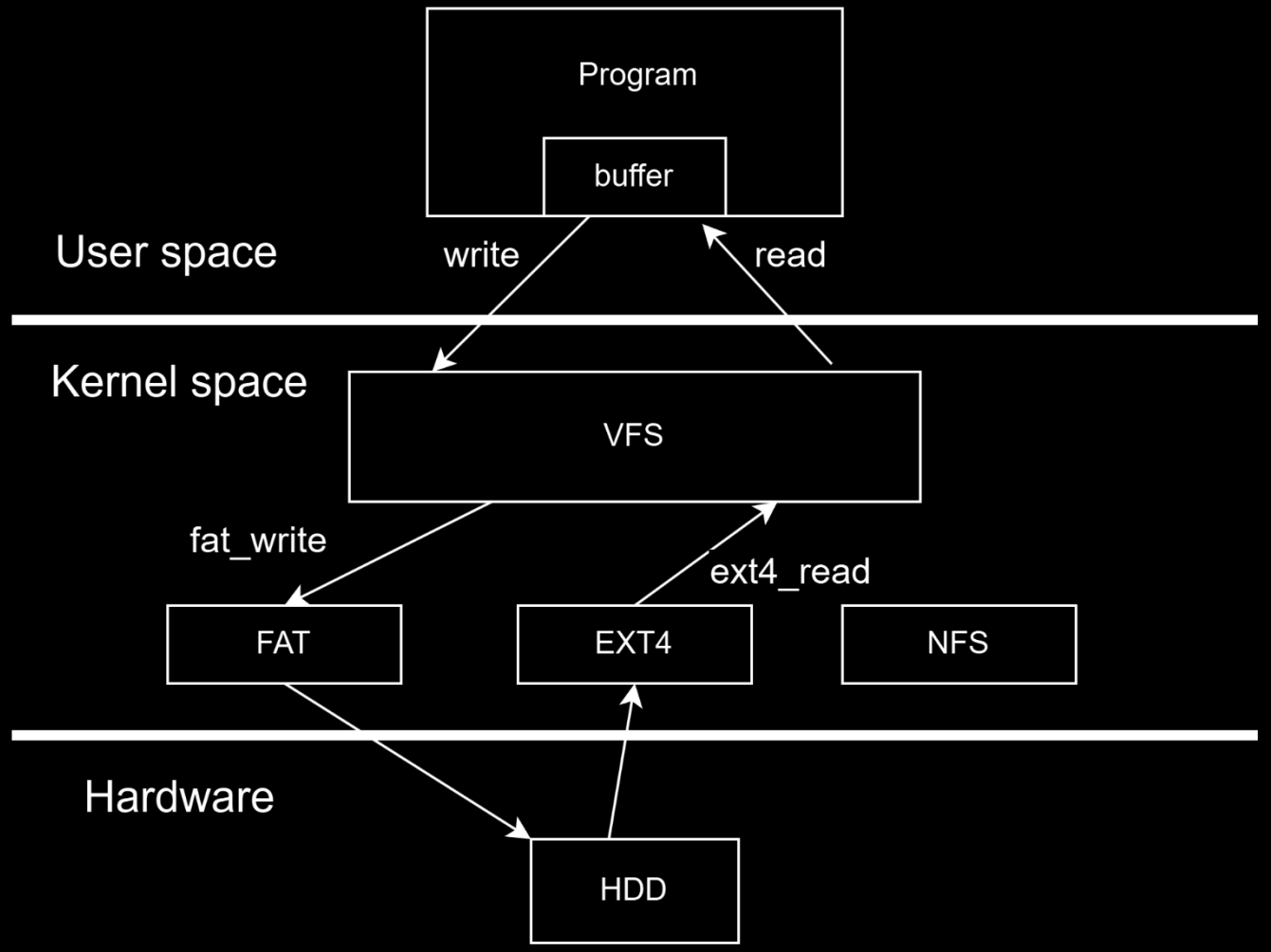
Важно понимать про системные вызовы:

- Если мы захотим что-либо сделать с железом, например, записать в файл на жесткий диск, то мы должны это сделать через системный вызов, т.к. наша программа будет выполняться в user space и не имеет привилегий доступа к жесткому диску.

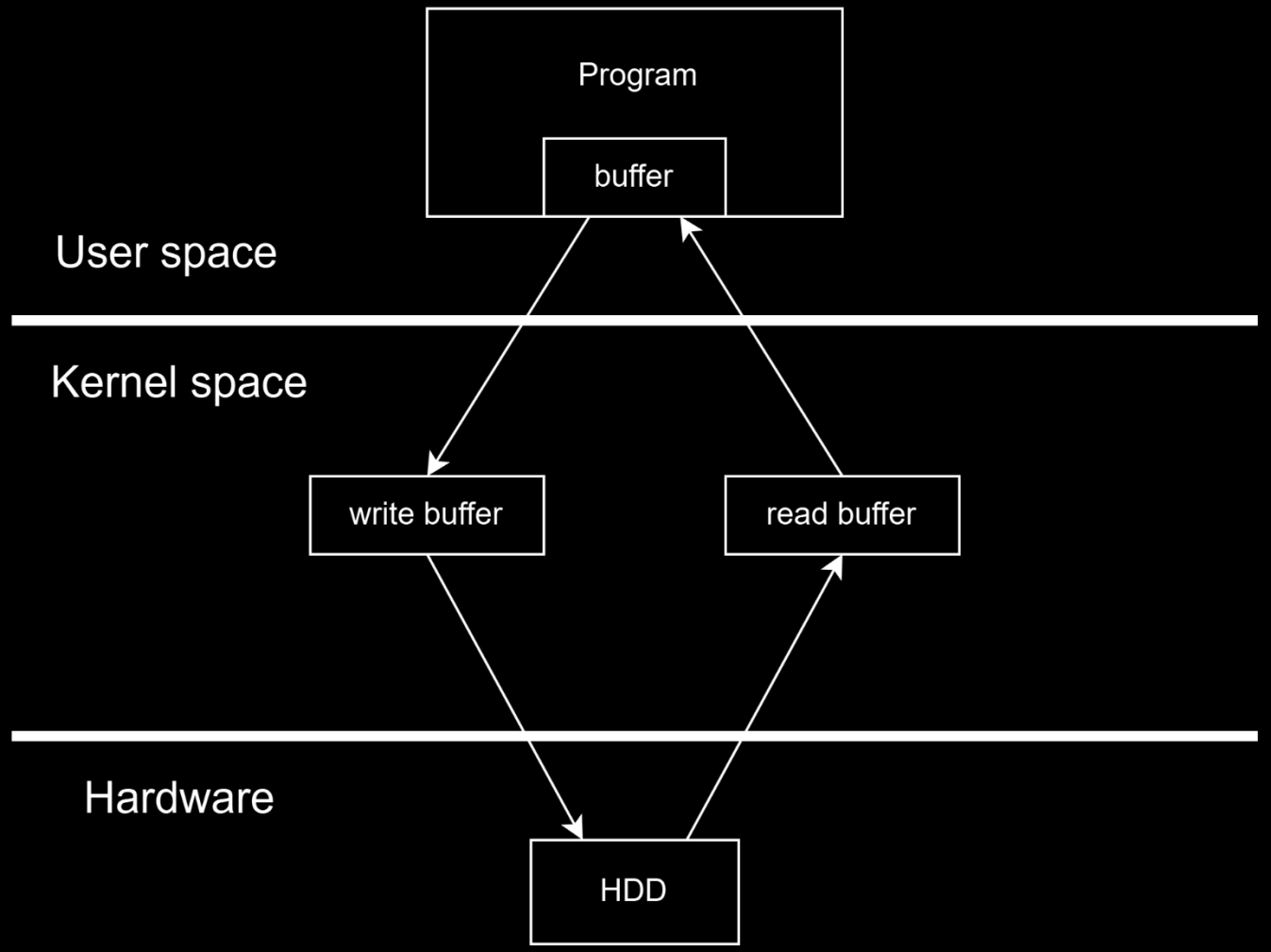
- При исполнении системного вызова происходит переключение контекста - это процесс, при котором ОС меняет состояние выполнения программы с пользовательского уровня на уровень ядра или наоборот. Переключение контекста не самая быстрая вещь, особоенно если очень часто происходят, и по возможности кол-во системных вызовов стоит уменьшать для ускорения выполнения программ на CPU.

- Системные вызовы - API ядра, т.е. это интерфейс из функций, позволяющий userland программам работать например с тем же железом (записать что-либо на HDD).

FILE I/O (Файловый ввод/вывод)



FILE I/O (вопрос)



Промежуточный буфер плохо?

FILE I/O (ответ)

- Запись/чтение из HDD является узким местом (bottle neck - горлышко бутылки. Благодаря копированию память-память (из user в kernel space) наша программа не повиснет на записи на жесткий диск из-за ожидания жесткого диска

- В системе у нас много процессов и каждая хочет что-то писать/читать на жесткий диск. Ядро управляет этими записями, сортирует их как надо (чтобы сделать как можно меньше записей на жесткий диск) и в нужный момент времени когда ядро (диспетчер ввода/вывода) будет готов отложено от этих программ сделает все эти записи на жесткий диск

- Ядро читает из жесткого диска в свой буфер наперед

Функции для работы с файловым I/O

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int close(int fd);

ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);

void sync(void);
```

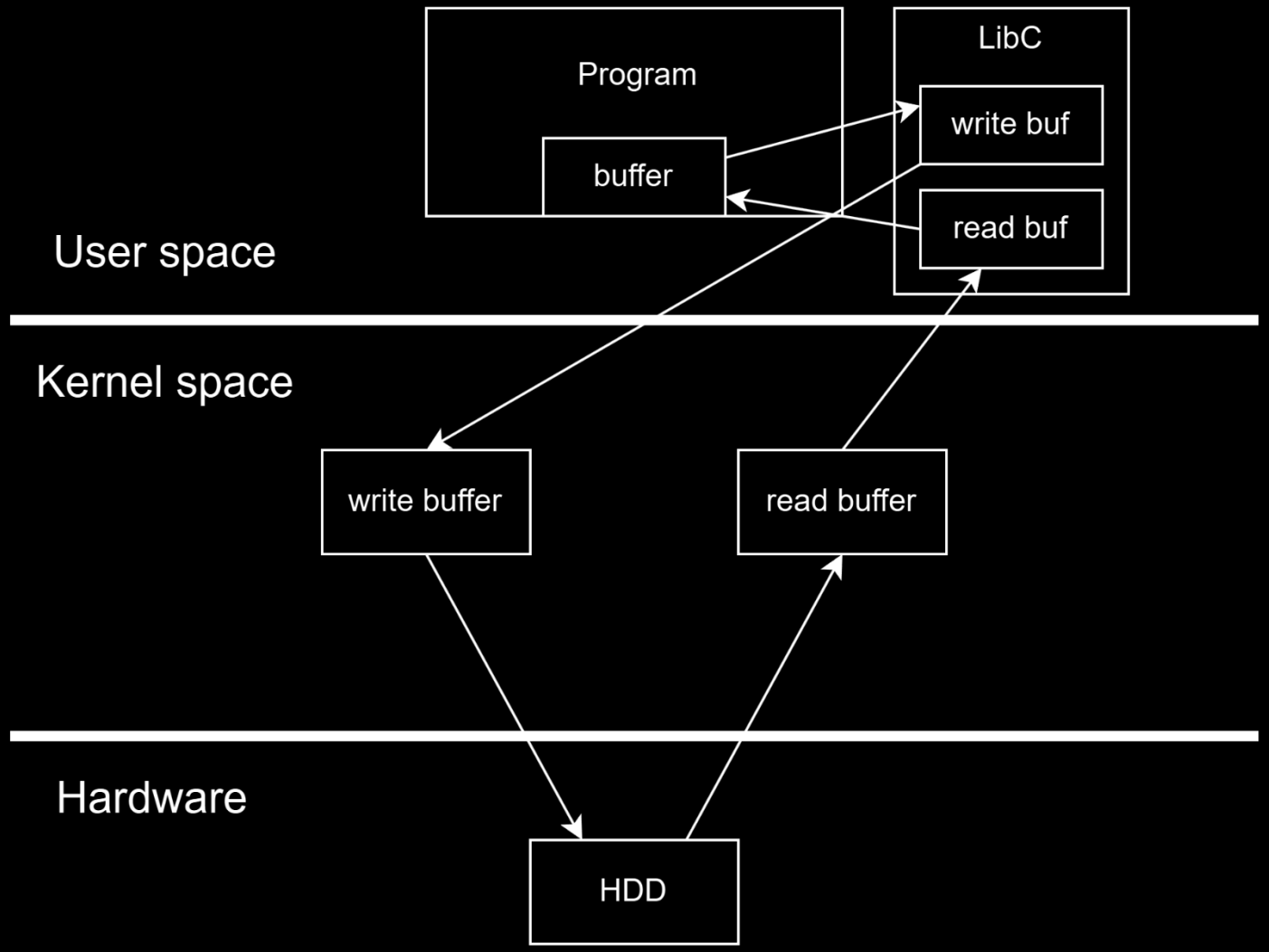
- open - открыть файл, вернется файловый дескриптор
- close - закрыть файл

- read - чтение из файла
- write - запись в файл
- lseek - сместить каретку (указатель внутри файла на каком байте файла мы находимся)

- sync - слить данные из ядра на жесткий диск

Файловый дескриптор, он же FD — это целое число, которое ОС использует для представления открытых файлов или других объектов ввода/вывода (например, сокетов, канала связи и т.д.)

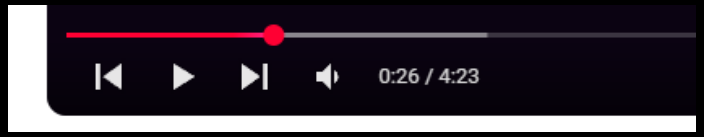
Buf I/O (Буферизированный ввод/вывод)

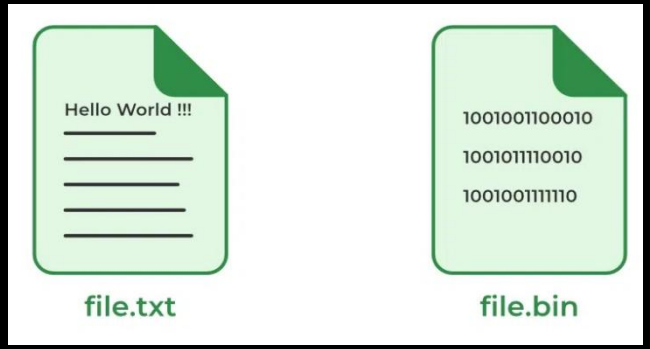


В файловом вводе/выводе все понятно, но здесь то зачем еще один промежуточный буфер?

Файлы

- Многие буферизированные I/O функции имеют букву f в имени: fwrite, fopen, fclose, fprintf и т.д.
- В Си файлы делится на 2 типа - текстовые и бинарные.
- Бинарные файлы могут хранить любые данные, включая числа, строки, изображения, звуковые файлы и другие.
- Текстовые файлы, с другой стороны, предназначены для хранения текстов, состоящих из букв, цифр, общих и специальных символов.





Text files (текстовые файлы)

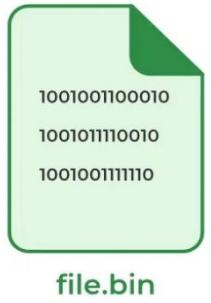
```
FILE *file;

file = fopen("/tmp/my_file.txt", "r");

if (file == NULL) {
    perror("Error opening file");
    exit(1);
}

// ф-ии чтения

fclose(file);
```

Закрывает файл, где stream - это указатель на FILE, возвращенный fopen(). Если возвращен 0, то это означает, что операция закрытия выполнена успешно, а если EOF (End Of File), то, значит, была ошибка

Режимы открытия файлов

Если добавить букву 'b' к любому из режимов, то файл будет открываться в двоичном режиме (например rb, ab+, wb и т.д.)
Таблица режимов открытия файла:

Режимы открытия	Описание
r	Ищет файл. Если файл открыт успешно, fopen() загружает его в память и устанавливает указатель, указывающий на первый символ в нем. Если файл не может быть открыт, fopen() возвращает NULL.
w	Открыть для записи в текстовом режиме. Если файл существует, его содержимое перезаписывается. Если файл не существует, создается новый файл. Возвращает NULL, если невозможно открыть файл.
a	Ищет файл. Если файл открыт успешно, fopen() загружает его в память и устанавливает указатель, указывающий на последний символ в нем. Открывается только в режиме добавления. Если файл не существует, создается новый файл. Возвращает NULL, если файл не удалось открыть.
r+	Ищет файл. Он успешно открыт fopen() загружает его в память и устанавливает указатель, указывающий на первый символ в нем. Возвращает NULL, если файл не удалось открыть.
w+	Ищет файл. Если файл существует, его содержимое перезаписывается. Если файл не существует, создается новый файл. Возвращает NULL, если невозможно открыть файл.
a+	Ищет файл. Если файл открыт успешно, fopen() загружает его в память и устанавливает указатель, указывающий на последний символ в нем. Он открывает файл как в режиме чтения, так и в режиме добавления. Если файл не существует, создается новый файл. Возвращает NULL, если файл не удалось открыть.



Функции чтения/записи для текстовых файлов

```
// Функции чтения из файла

int fscanf(FILE *stream, const char *format, ...);

char *fgets(char *s, int size, FILE *stream);

int fgetc(FILE *stream);
```

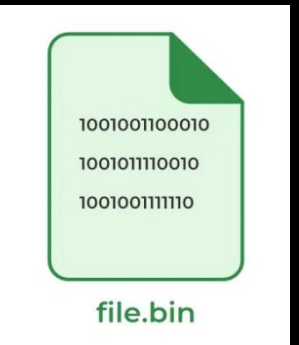
```
// Функции записи в файл

int fprintf(FILE *stream, const char *format, ...);

int fputs(const char *s, FILE *stream);

int fputc(FILE *stream);
```

Функции чтения/записи для бинарных файлов

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Общие функции для буферизированных файлов

```
int fseek(FILE *stream, long offset, int whence);

long ftell(FILE *stream);

void rewind(FILE *stream);

int fflush(FILE *stream);
```



Теперь немного live code..

https://github.com/kruffka/C-Programming