ПРОГРАММИРОВАНИЕ

Файловый ввод-вывод

Преподаватель:

Ст. преп. Кафедры ВС,

Перышкова Евгения Николаевна

Что такое файл?

Файл представляет собой именованный раздел памяти, расположенный обычно на диске.

Что такое файл для *операционной системы*?

Как представляются файлы *в программе на языке С*?

Что такое файл для ОС?

•Файлы в обычном смысле: файлы, которые хранятся на жестком диске (можно считывать из них и записывать в них информацию);

•Экран монитора: файл, в который можно выводить информацию (отобразится на экране монитора);

•Клавиатура: файл, из которого можно считывать информацию;

•Принтер: файл, в который можно выводить информацию (печать текста);

•Модем: файл, из которого можно считывать информацию и в который можно записывать информацию (обмен информации по сети);

Что такое файл?

Файл представляет собой непрерывную последовательность байт, каждый из которых может быть прочитано индивидуально.

Стандарт ANSI C предлагает два способа представления файла:

1. Текстовое представление
2. Двоичное представление

Текстовое и двоичное представление

1. Текстовое представление

Представление о содержимом файла может отличатся от того, что храниться в файле.

2. Двоичное представление

Каждый байт файла доступен из программы.

Текстовые и бинарные файлы

Рассмотрим строку:

```
fopen("file1.txt", "w");
```

`fopen("file1.txt", "w")` - откроет файл как текстовый файл;

`fopen("file1.txt", "wb")` - откроет файл как бинарный файл

Особенности работы с файлами

```
FILE *f = fopen("file1.txt", "r");
```

Зачем закрывать файловый поток?

Особенности работы с файлами

```
FILE *f = fopen("file1.txt", "r");
```

Зачем закрывать файловый поток?

```
fclose(f);
```

Последние данные из буфера не запишутся в файл. Отсутствие этой команды может привести к потере данных в файле, который был открыт для записи (добавления).

Зачем закрывать файловый поток, открытый для чтения?

Особенности работы с файлами

Зачем закрывать файловый поток, открытый для чтения?

1. Может привести к ограничению доступа к файлу для других программ;

2. В любой ОС есть ограничение на количество одновременно открытых файлов;

Важность работы с буфером

При аварийном завершении, последний протокол действий с файлом не запишется.

```
fprintf(file, "%d", data); //данные записались в буфер
fflush(file); //данные из буфера попали в файл
```

Стандартные файлы

Программа на языке C открывает сразу 3 стандартных файла:

1. Стандартный ввод (`scanf, getchar, gets`)
2. Стандартный вывод (`printf, putchar, puts`)
3. Стандартный вывод ошибок

Стандартные файлы

```
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

Эквивалентная запись:

```
scanf(…) = fscanf( stdin, …)
printf(…) = fprintf( stdout, …)
```

Стандартные файлы

Вывод на экран для тестирования:

```
//FILE *f = fopen(...);
FILE *f = stdin;

fscanf(f, ...)
fprintf(f, ...)
```

Стандартные файлы

```
FILE *stderr;
```

По умолчанию выводит данные на экран

Небуферизованный файл (поток)

Вместо:

```
fprintf(stdout, ...);
fflush(stdout);
```

Можно:

```
fprintf(stderr, ...);
```

Чтение символа из потока

```
int fgetc(FILE *f);
```

- `FILE *f` -- файл, из которого читаем;

Функция применяется для чтения из потока.

В случае успеха, `fgetc` возвращает следующий байт или символ из потока в зависимости от того, какой тип файла открыт.

В противном случае, `fgetc` возвращает `EOF`.

Запись в поток

```
int fwrite ( const char * array, size_t size, size_t count, FILE * stream );
```

Функция `fwrite` записывает блок данных в поток.

Таким образом запишется массив элементов `array` в текущую позицию в потоке.

Для каждого элемента запишется `size` байт.

Индикатор позиции в потоке изменится на число байт, записанных успешно.

Возвращаемое значение будет равно `count` в случае успешного завершения записи.

В случае ошибки возвращаемое значение будет меньше `count`.

Чтение из потока

```
int fread ( const char * array, size_t size, size_t count, FILE * stream );
```

Функция `fread` читает блок данных из потока.

Таким образом в массив элементов `array` будут полученны данные из потока.

Для каждого элемента требуется `size` байт.

Индикатор позиции в потоке изменится на число байт, считанных успешно.

Возвращаемое значение будет равно `count` в случае успешного завершения записи.

В случае ошибки возвращаемое значение будет меньше `count`.

Позиционирование в потоке

```
int fseek(FILE *f, long offset, int flag);
```

- `FILE *f` -- файл, в котором передвигаемся;
- `long offset` -- количество байтов для отступа, отступ производится в соответствии с 3-м параметром;
- `int flag` -- позиция, от которой будет совершен отступ; в стандартной библиотеке C для этого параметра определены 3 константы:

`SEEK_SET` -- начало файла;
`SEEK_CUR` -- текущая позиция;
`SEEK_END` -- конец файла;

`int fseek()` -- сама функция возвращает ноль, если операция прошло успешно, иначе возвращается ненулевое значение.

Позиционирование в потоке

```
long int ftell(FILE *f);
```

функция может определить текущее положение в файле, который открыт для чтения

Файловая переменная

Файлы – динамические структуры данных, хранящиеся на внешних запоминающих устройствах.

Файловая переменная – структура данных, связывающая программу с некоторым файлом на диске.

Дисциплина последовательного доступа – ограниченный **набор специальных операций**.

Операции последовательного доступа

Опер.	Описание
f = *opennew*(*n*) *f* = *openold*(*n*)	связывание программы и последовательности *x*, расположенной в файле с именем *n*
close(*f*)	закрытие файлового соединения
write(*f* , *c*)	Запись нового элемента *c* в текущую позицию последовательности *x*, описываемую файловой переменной *f*.
c = *read*(*f*)	Чтение элемента, расположенного в текущей позиции последовательности *x*, описываемой файловой переменной *f*, в ячейку *c*.
set(*f*, *p*)	Установка текущей позиции последовательности *x*, описываемой файловой переменной *f*, в значение смещения *p*.

Операция open

$x_L = < >$
$x_R = < >$

$f = opennew(n_1)$

$f = openold(n_2)$

$x_L = < >$
$x_R = x$

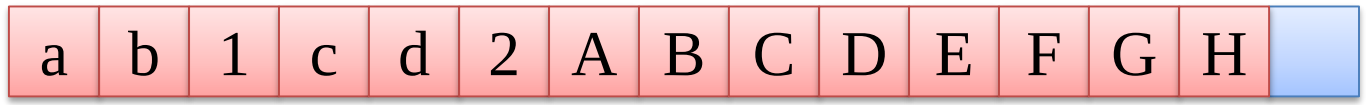
Программа

HDD (жесткий диск)

Файл n_1

x

Файл n_2



Операция *write*

$x_L = <>$
$x_R = <>$
$x = x_L \,\&\, x_R$

$f = opennew(n_1)$

Программа

HDD (жесткий диск)

Операция *write*

$x_L = <>$
$x_L = <>$
$x = x_L \,\&\, x_R$

$f = opennew(n_1)$

***write*(*f* , 'a')**

$x_L = <'a'>$
$x_R = <>$
$x = x_L \,\&\, x_R$

Программа

HDD (жесткий диск)

Операция *write*

$x_L = <>$
$x_R = <>$
$x = x_L \,\&\, x_R$

$f = opennew(n_1)$

write(f , 'a')

$x_L = <'a'>$
$x_R = <>$
$x = x_L \,\&\, x_R$

$x_L = <'ab'>$
$x_R = <>$
$x = x_L \,\&\, x_R$

write(f , 'b')

Программа

HDD (жесткий диск)

Операция *write*

$x_L = <>$
$x_R = <>$
$x = x_L \,\&\, x_R$

$f = opennew(n_1)$

***write*(*f* , 'a')**

$x_L = <'a'>$
$x_R = <>$
$x = x_L \,\&\, x_R$

$x_L = <'ab'>$
$x_R = <>$
$x = x_L \,\&\, x_R$

***write*(*f* , 'b')**

$x_L = <'abc'>$
$x_R = <>$
$x = x_L \,\&\, x_R$

***write*(*f* , 'c')**

Программа

HDD (жесткий диск)

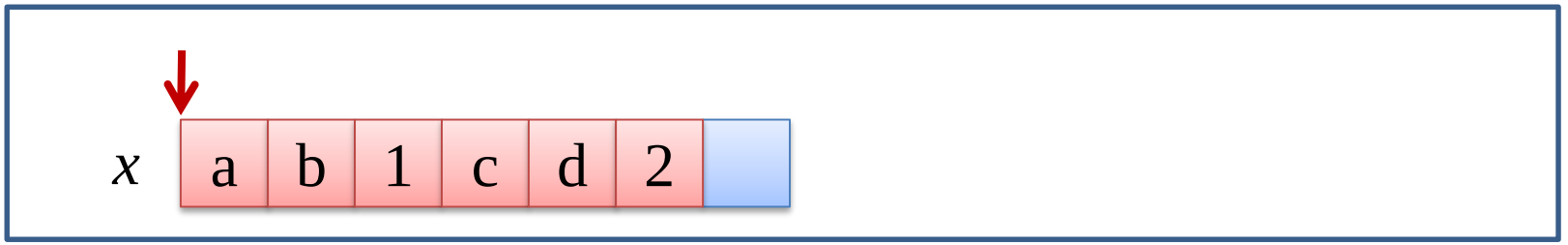
Операция *read*

x_L = < >
x_R = < 'ab1cd2' >
$x = x_L$ & x_R

$f = openold(n_2)$

Программа

HDD (жесткий диск)



Операция *read*

x_L = < >
x_R = < 'ab1cd2' >
$x = x_L$ & x_R

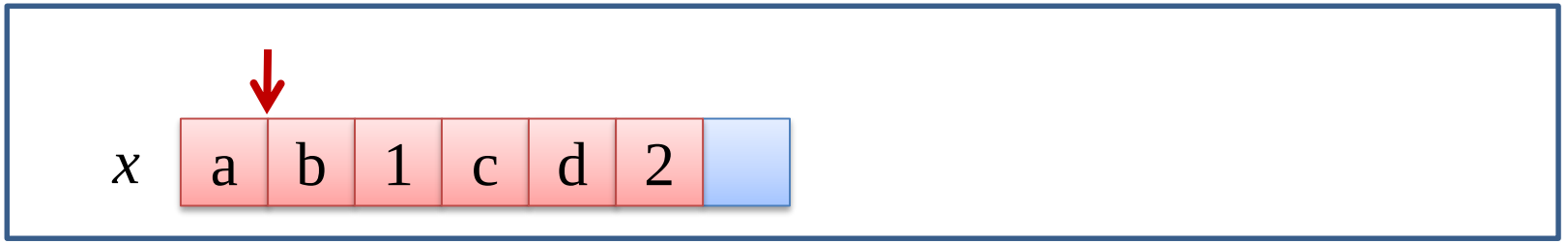
x_L = < 'a' >
x_R = < 'b1cd2' >
$x = x_L$ & x_R

$f = openold(n_2)$

$c = read(f)$

Программа

HDD (жесткий диск)



Операция *read*

x_L = < >
x_R = < 'ab1cd2' >
$x = x_L$ & x_R

$f = openold(n_2)$

x_L = < 'a' >
x_R = < 'b1cd2' >
$x = x_L$ & x_R

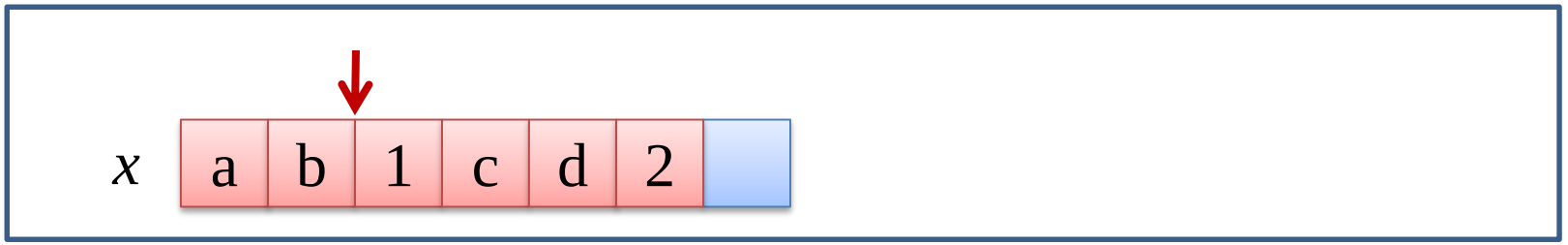
$c = read(f)$

x_L = < 'ab' >
x_R = < '1cd2' >
$x = x_L$ & x_R

$c = read(f)$

Программа

HDD (жесткий диск)



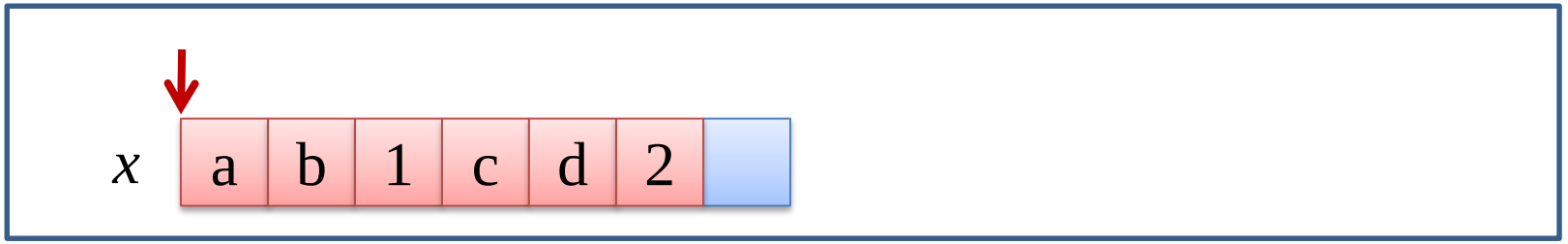
Операция *set*

x_L = < >
x_R = < 'ab1cd2' >
$x = x_L$ & x_R

$f = openold(n_2)$

Программа

HDD (жесткий диск)



Операция *set*

x_L = < >
x_R = < 'ab1cd2' >
$x = x_L$ & x_R

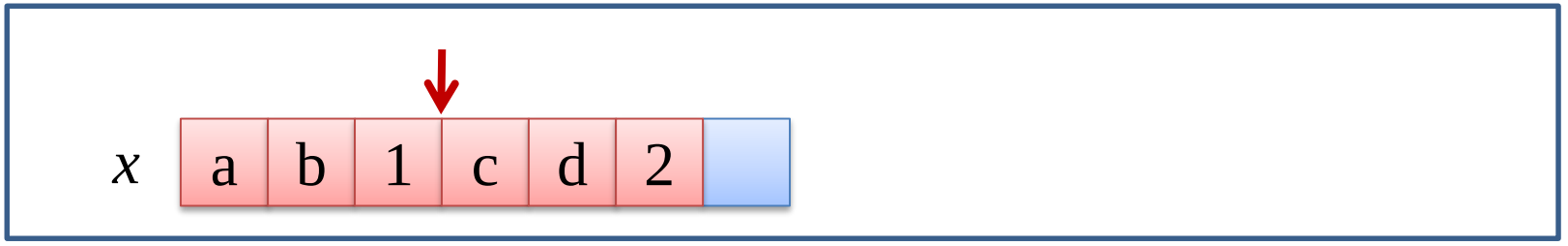
x_L = < 'ab1' >
x_R = < 'cd2' >
$x = x_L$ & x_R

$f = openold(n_2)$

$c = set(f, 3)$

Программа

HDD (жесткий диск)



Средства низкоуровневого ввода-вывода языка СИ

```
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
```

O_RDONLY
O_WRONLY
O_RDWR

```
#include <unistd.h>
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);
```

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

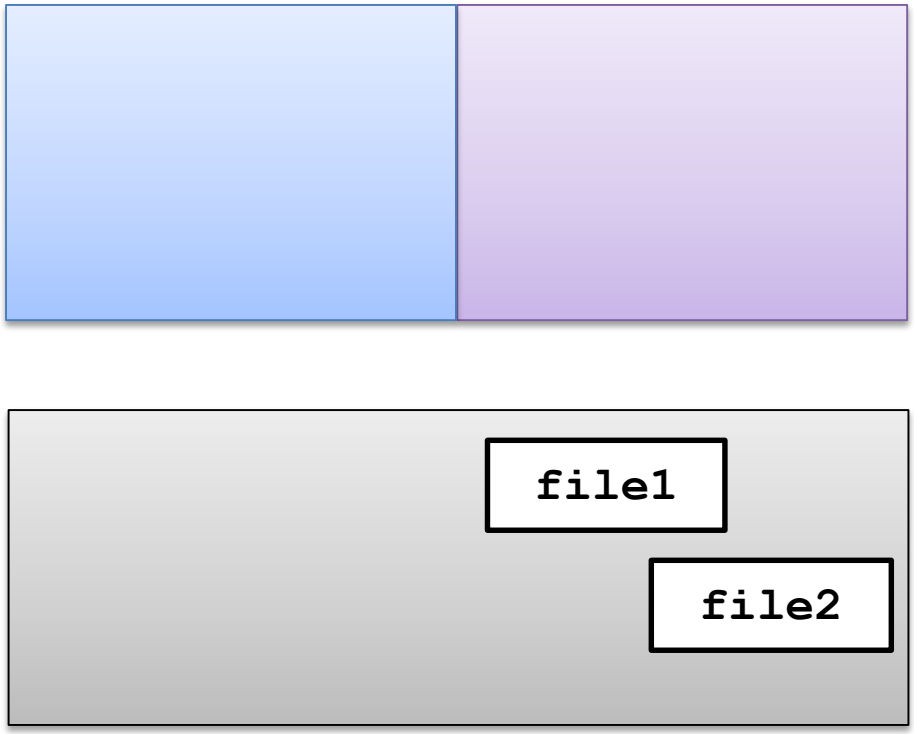
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```



Пример использования низкоуровневого ввода-вывода

Код программы

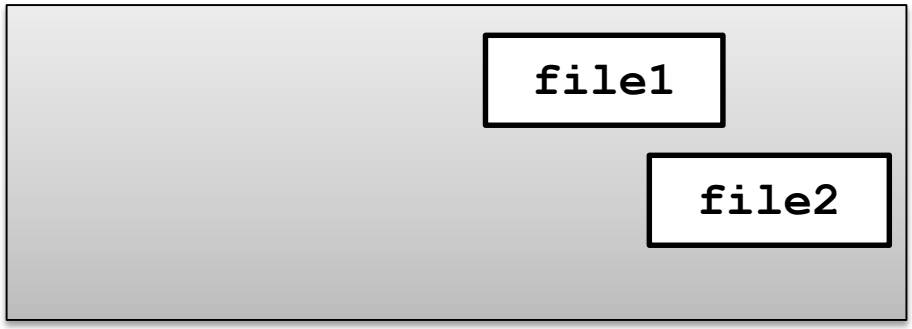
```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```

Память программы

Память ОС



Файловая система

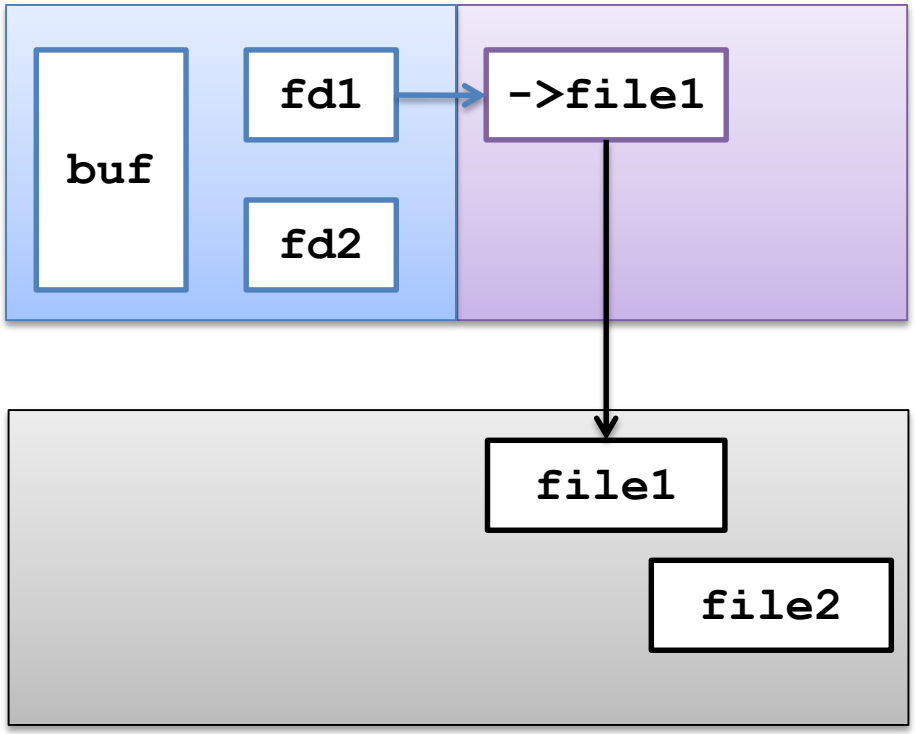
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```



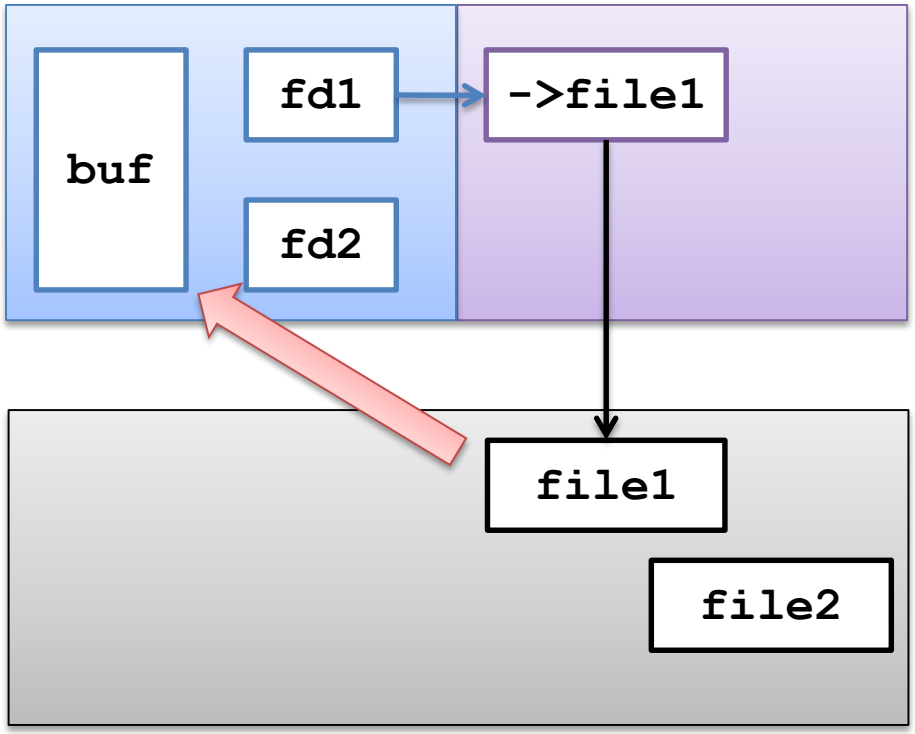
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
                ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```



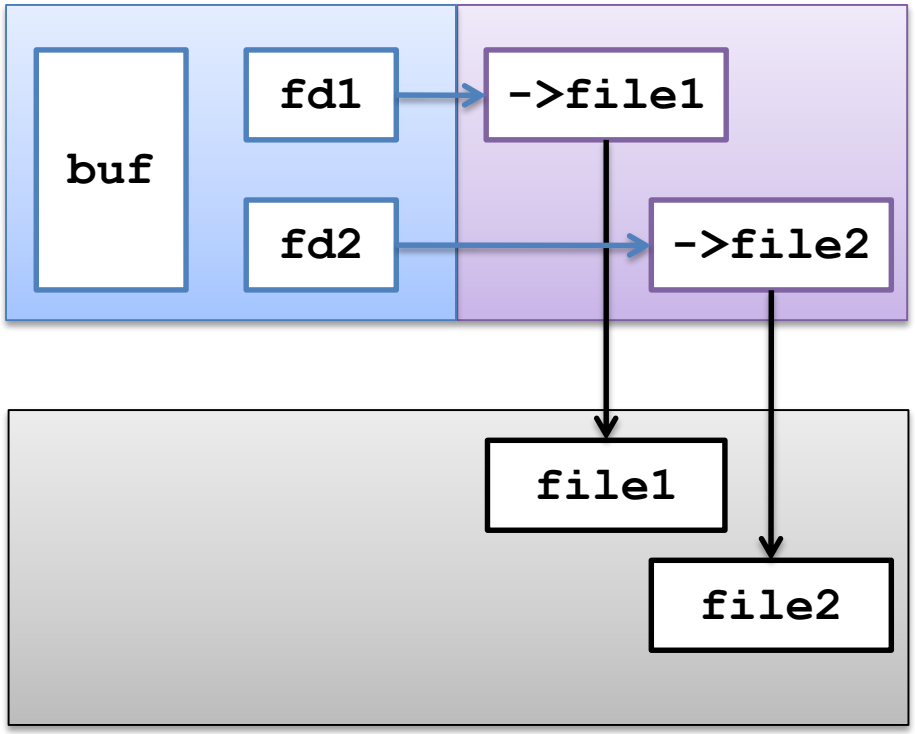
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
                ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```



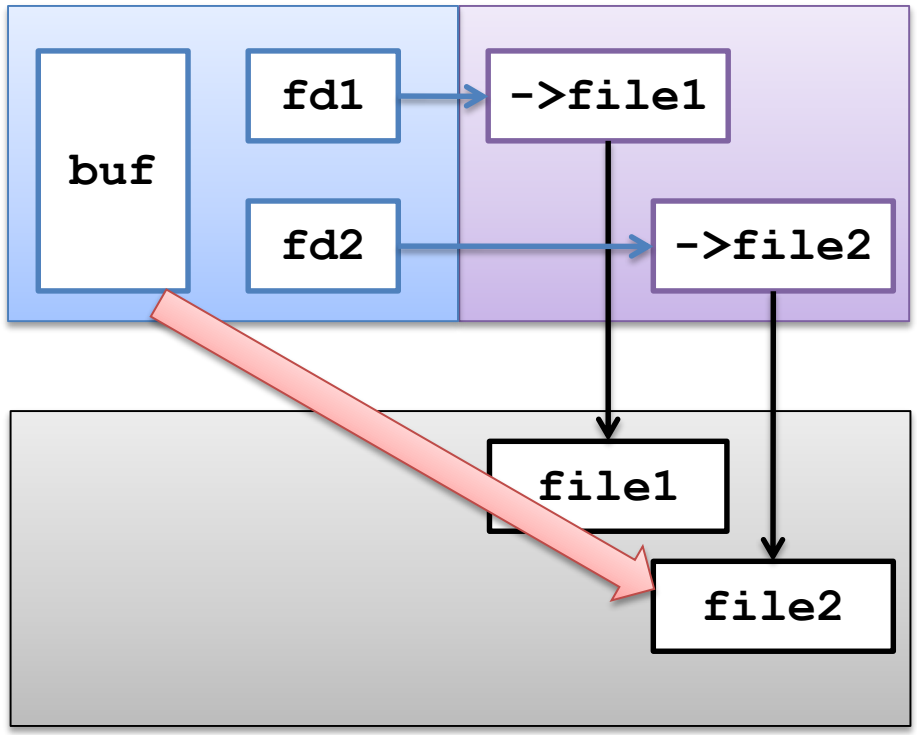
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
                ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```



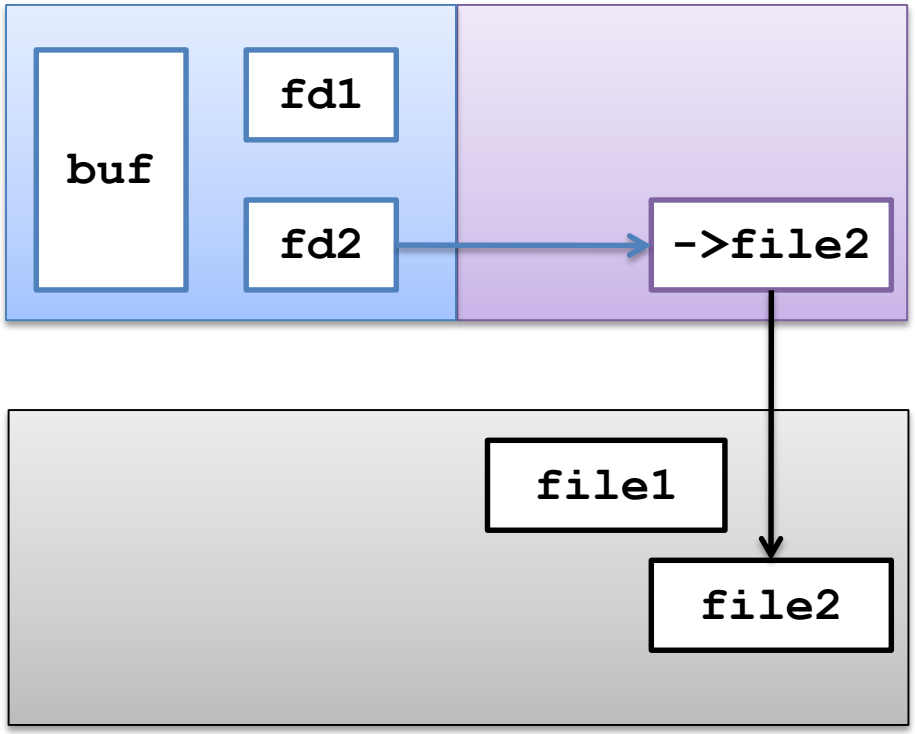
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```



Пример использования низкоуровневого ввода-вывода

Код программы

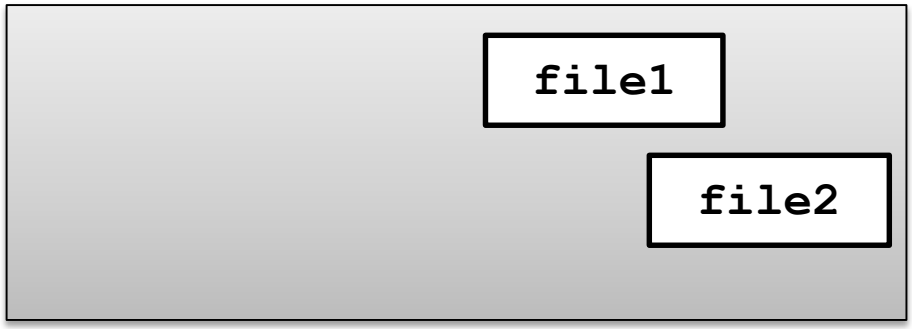
```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);

fd2=open("file2",O_WRONLY);
ret2=write(fd2,buf,ret1);

close(fd1);
close(fd2);
```

Память программы

Память ОС



Файловая система

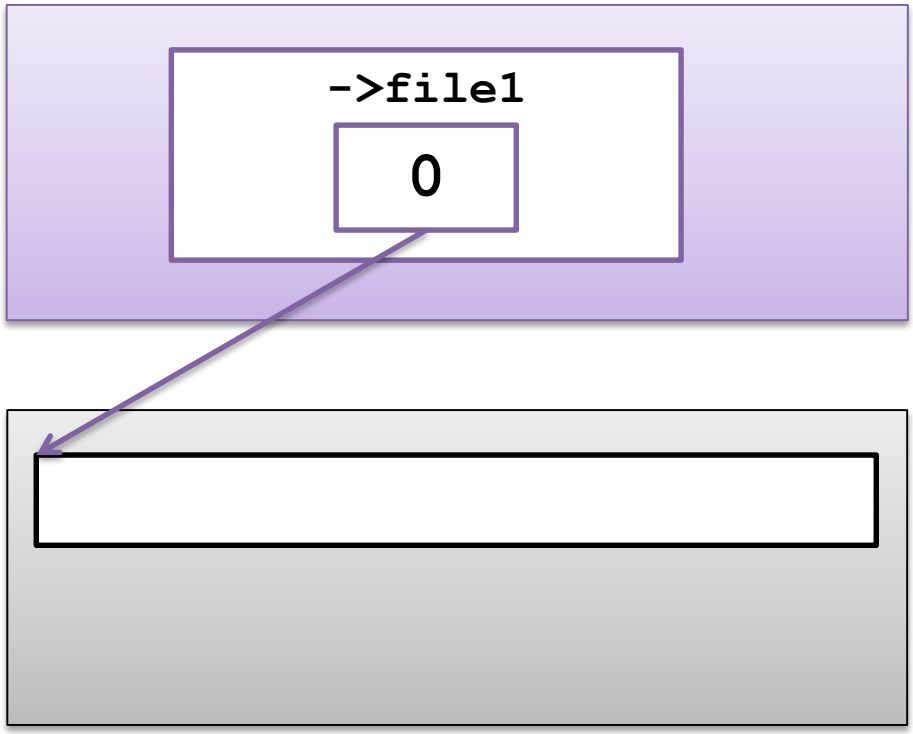
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);
ret1 = read(fd1,buf,10);

lseek(fd1,40,SEEK_SET);

ret1 = read(fd1,buf,10);
close(fd1);
```



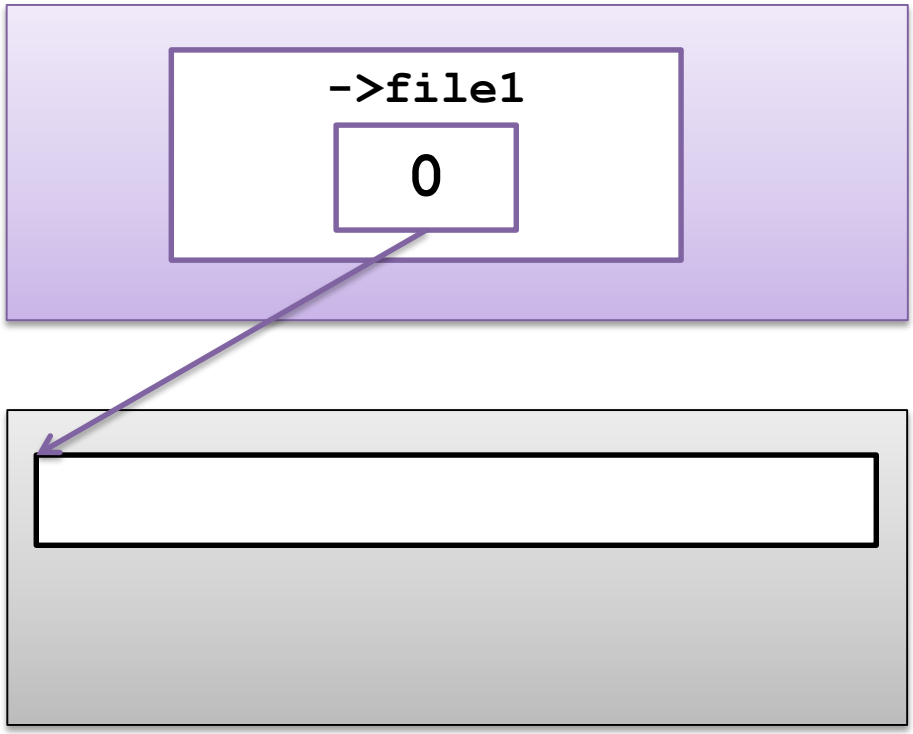
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);
ret1 = read(fd1,buf,10);

lseek(fd1,40,SEEK_SET);

ret1 = read(fd1,buf,10);
close(fd1);
```



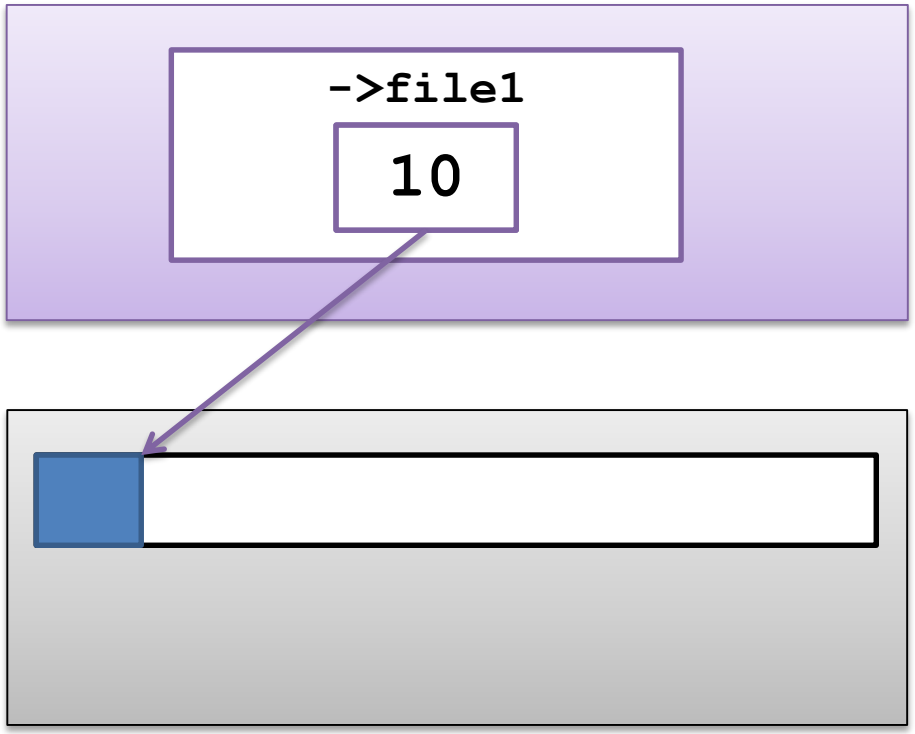
Пример использования низкоуровневого ввода-вывода

Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);
ret1 = read(fd1,buf,10);

lseek(fd1,40,SEEK_SET);

ret1 = read(fd1,buf,10);
close(fd1);
```



Пример использования низкоуровневого ввода-вывода

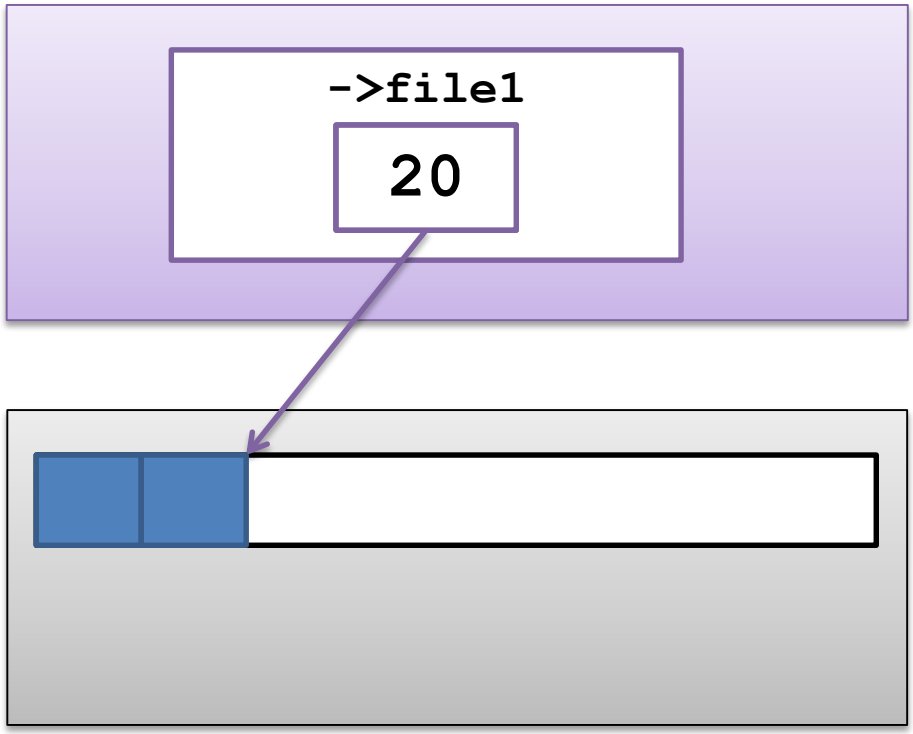
Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);
ret1 = read(fd1,buf,10);

lseek(fd1,40,SEEK_SET);

ret1 = read(fd1,buf,10);
close(fd1);
```

Память ОС



Файловая система

Пример использования низкоуровневого ввода-вывода

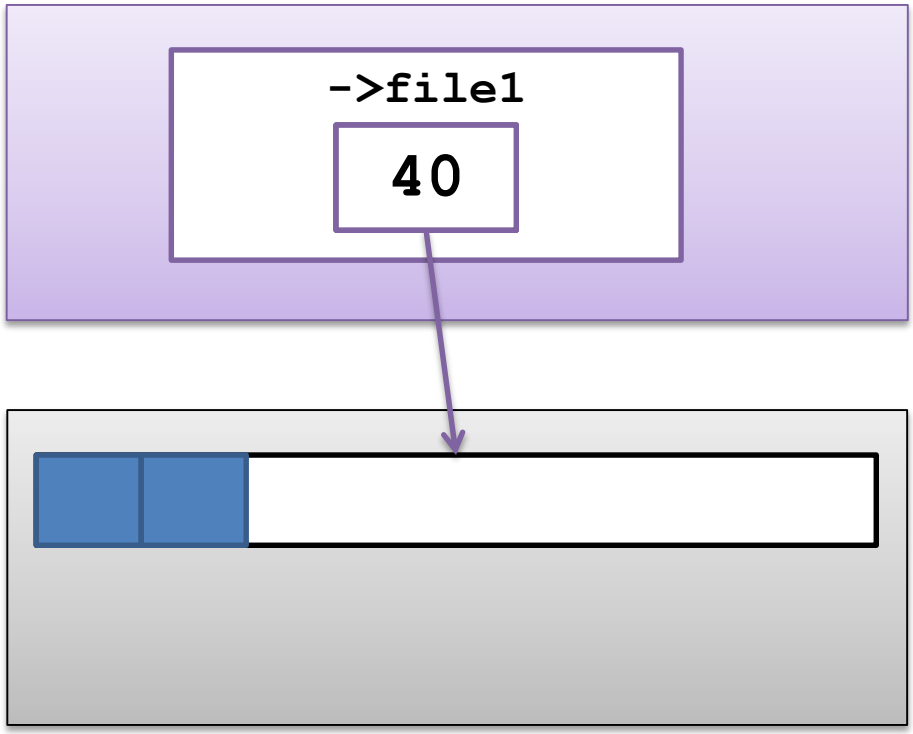
Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);
ret1 = read(fd1,buf,10);

lseek(fd1,40,SEEK_SET);

ret1 = read(fd1,buf,10);
close(fd1);
```

Память ОС



Файловая система

Пример использования низкоуровневого ввода-вывода

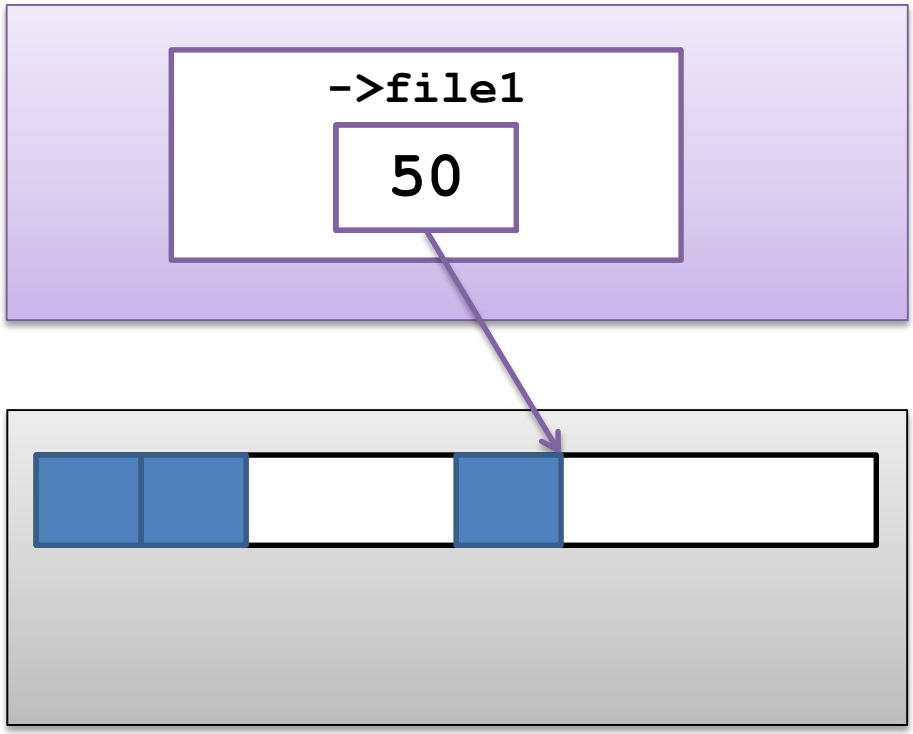
Код программы

```
int fd1, fd2;
int ret1,ret2;
char buf[256];
              ...
fd1=open("file1",O_RDONLY);
ret1 = read(fd1,buf,10);
ret1 = read(fd1,buf,10);

lseek(fd1,40,SEEK_SET);

ret1 = read(fd1,buf,10);
close(fd1);
```

Память ОС



Файловая система

Высокоуровневый интерфейс ввода-вывода языка СИ

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

Управление связанными программными объектами

```
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *fp);
```

Чтение/запись данных

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

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

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

Позиционирование в потоке

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

Пример использования высокоуровневого ввода-вывода

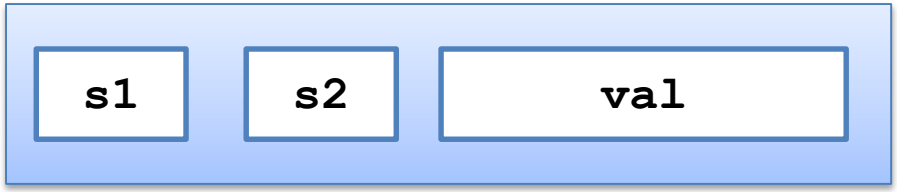
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
              ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Ядро ОС

Станд. библиотека GLibC

Файловая система

Пример использования высокоуровневого ввода-вывода

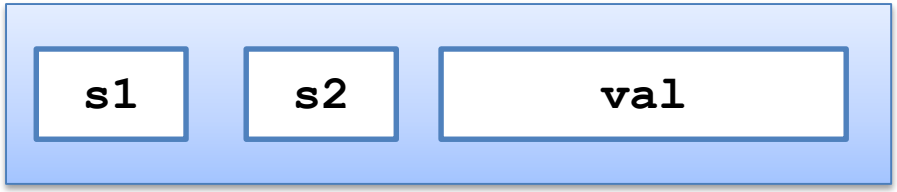
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
              ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Ядро ОС

Станд. библиотека GLibC

Файловая система

Пример использования высокоуровневого ввода-вывода

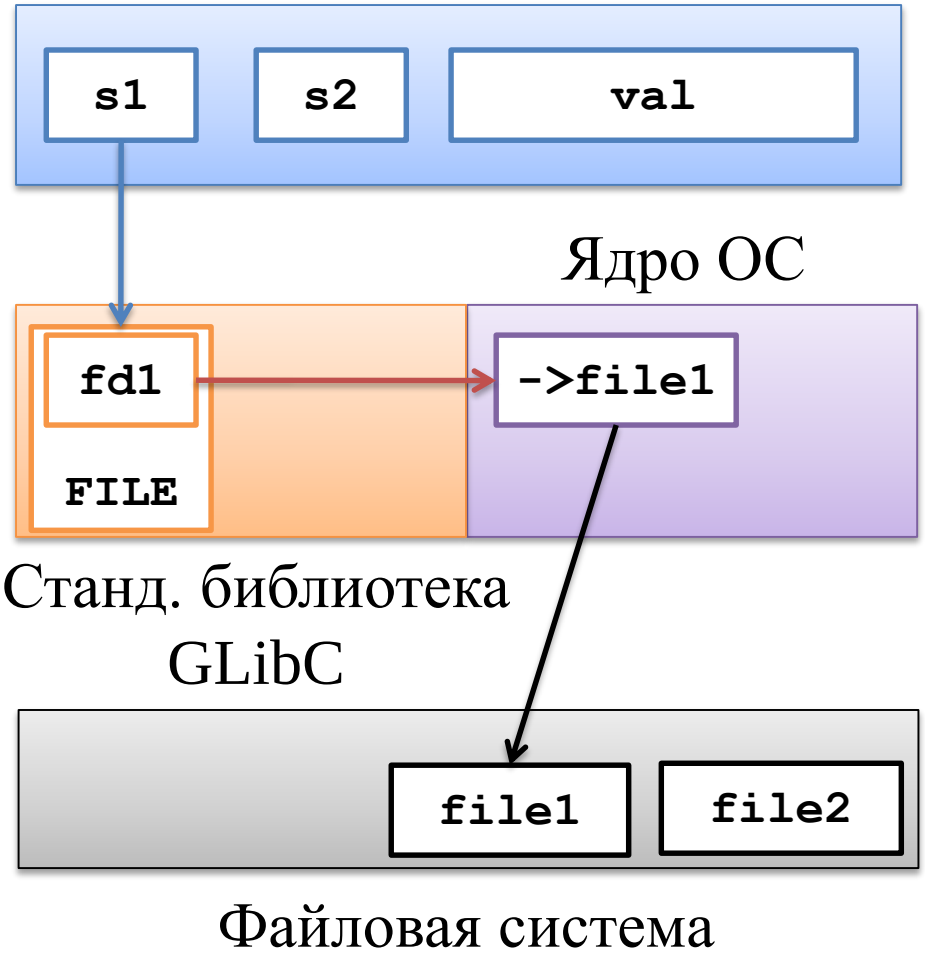
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

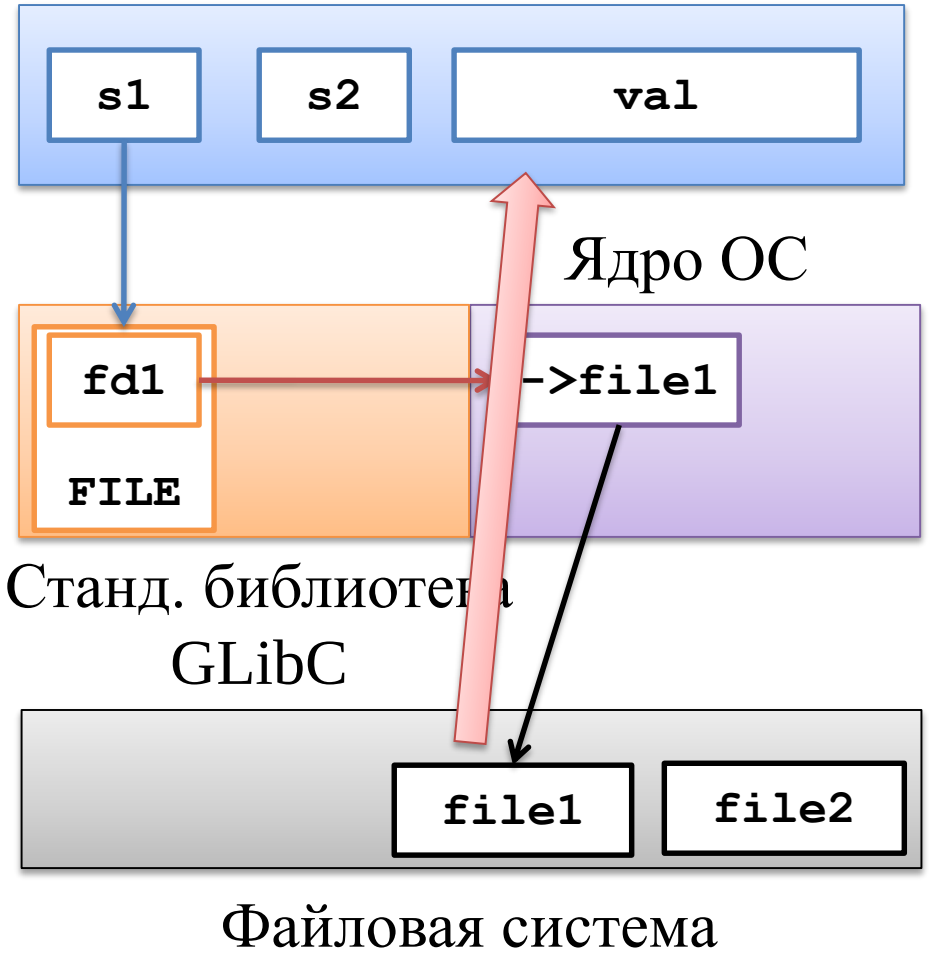
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

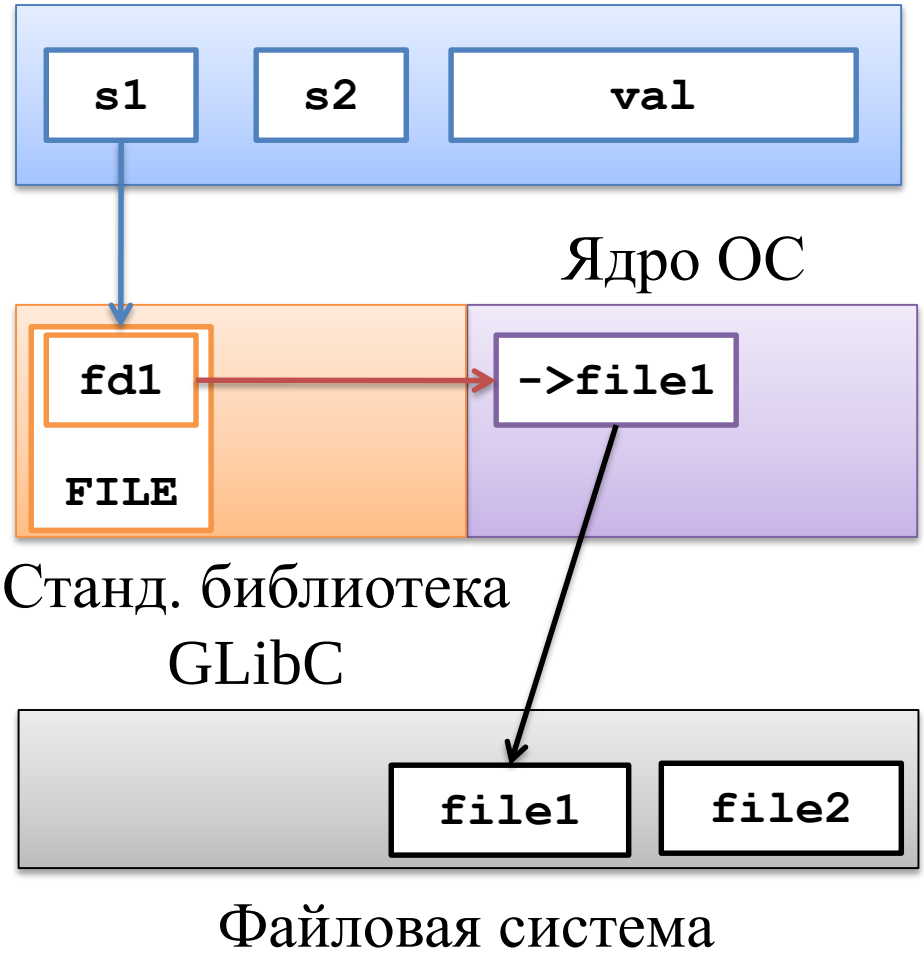
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

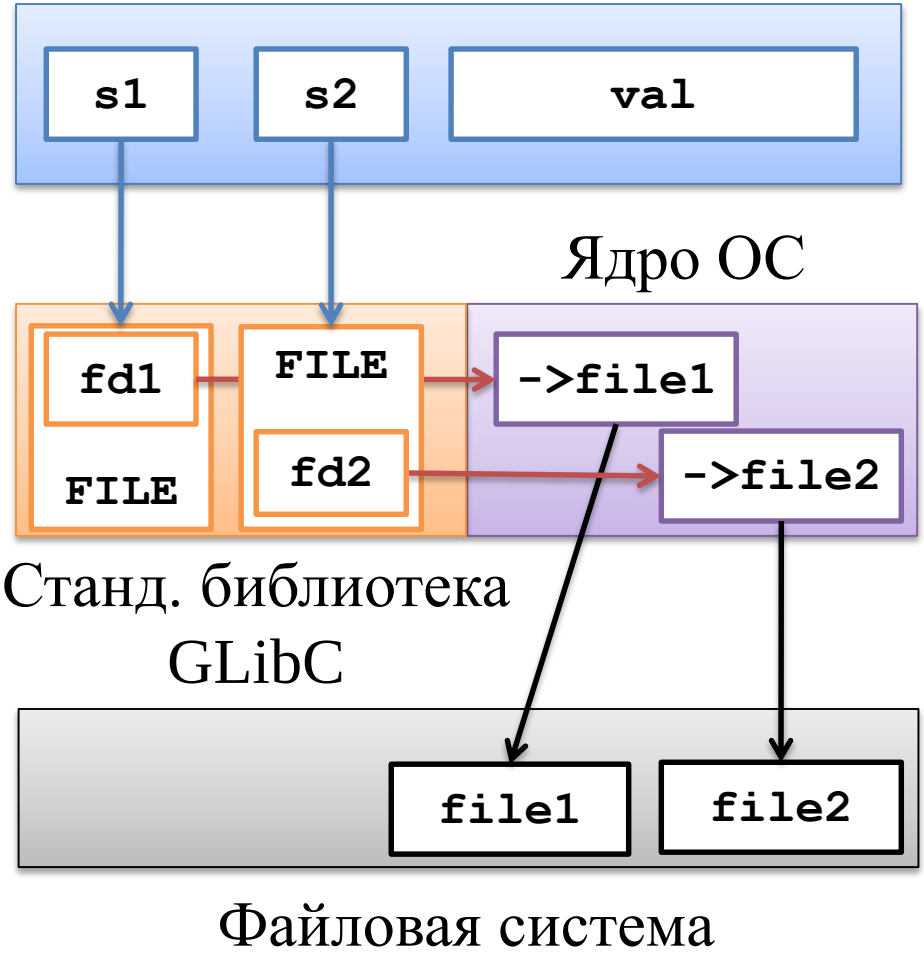
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

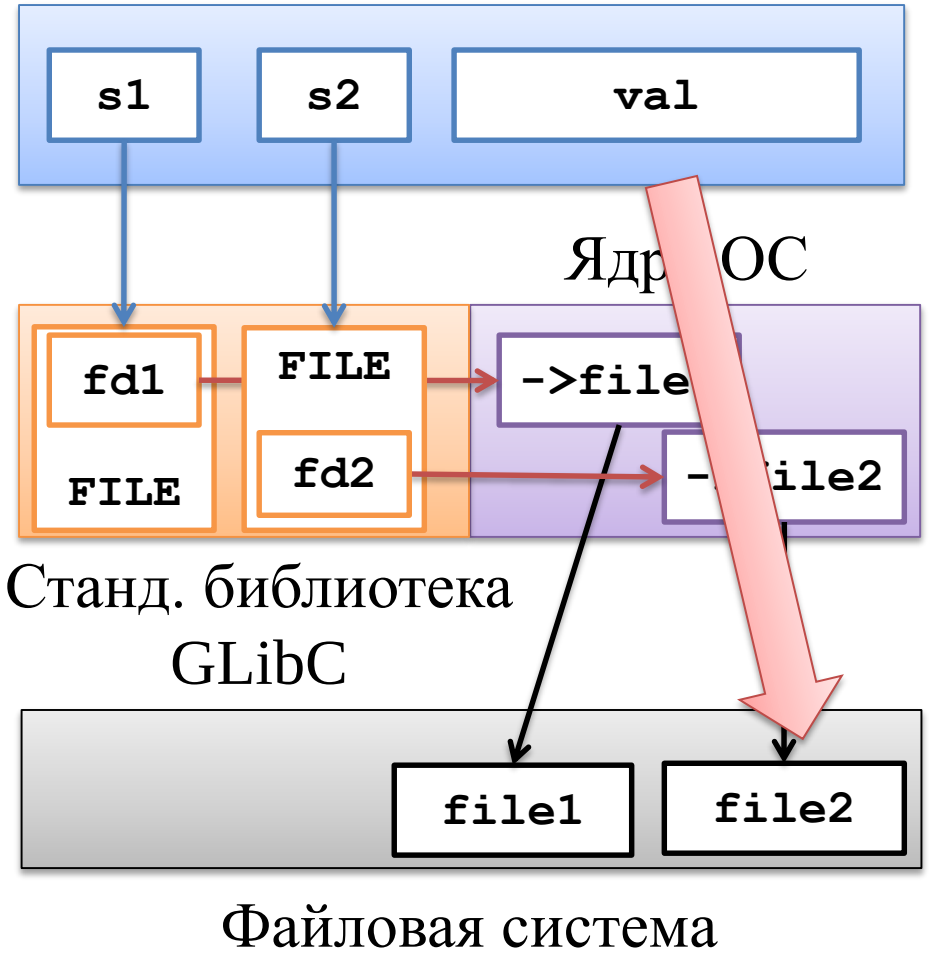
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

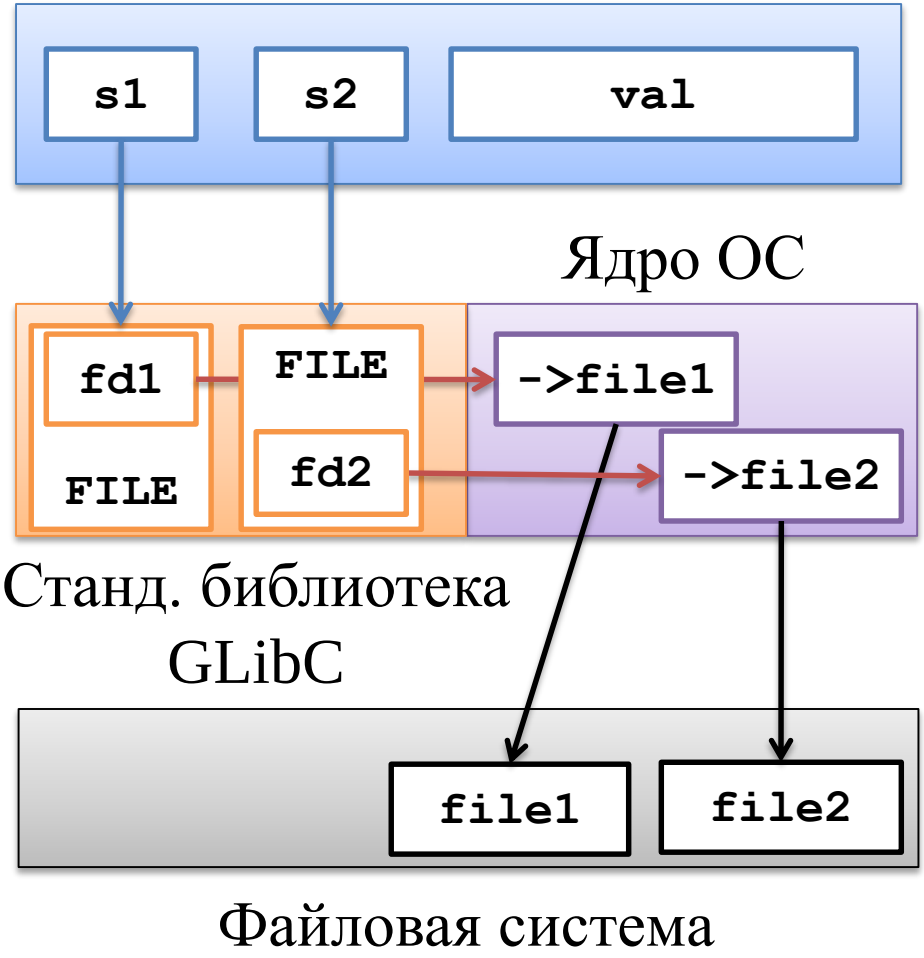
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

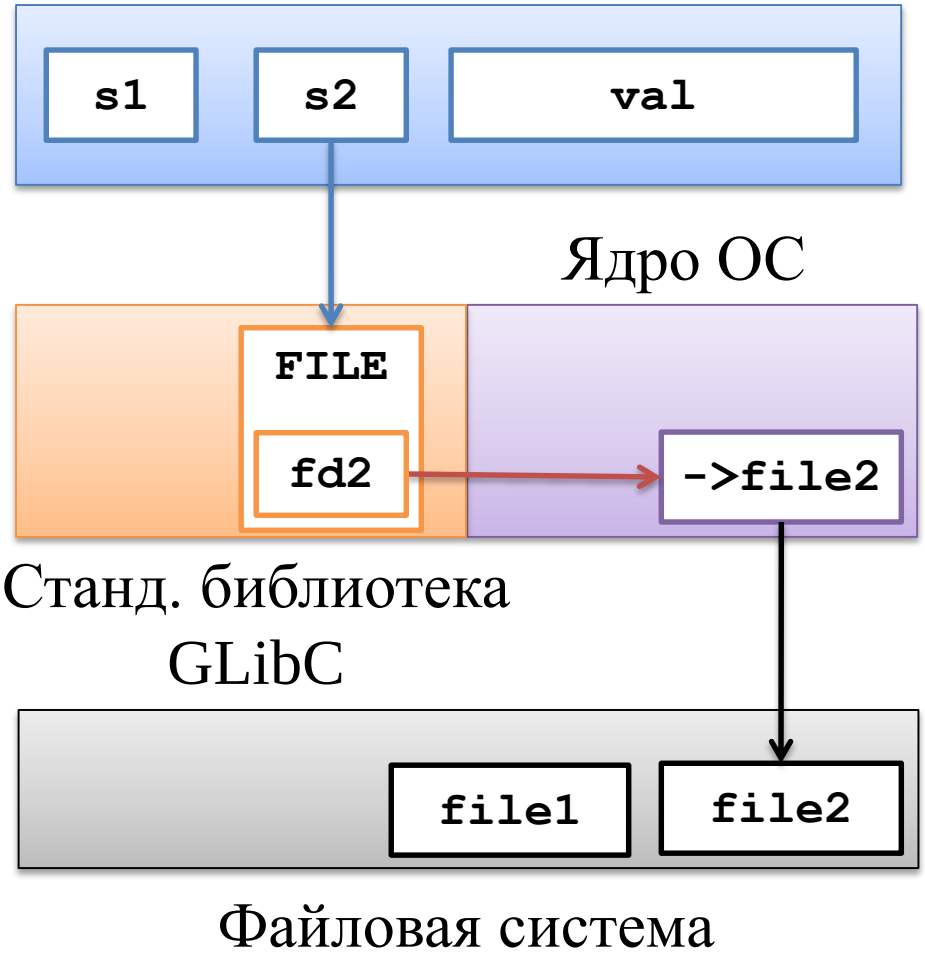
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
              ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Пример использования высокоуровневого ввода-вывода

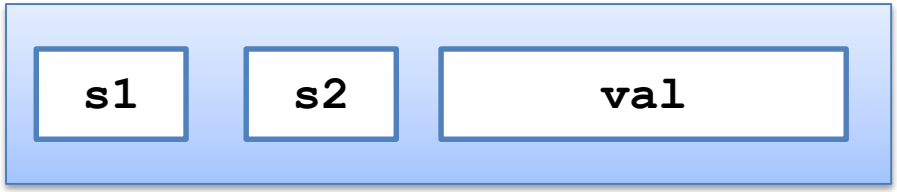
Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
              ...
s1 = fopen("file1","r");
fscanf(s1,"%d", &val);

s2=fopen("file2","w");
fprintf(s2,"%d",val);

fclose(fd1);
fclose(fd2);
```

Программа



Ядро ОС

Станд. библиотека GLibC

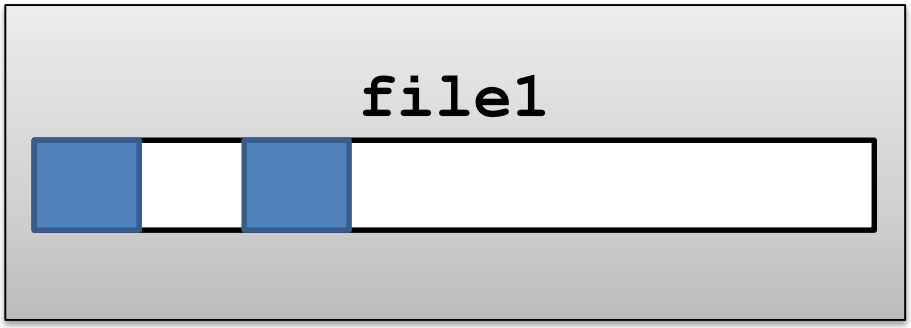
Файловая система

Пример использования высокоуровневого ввода-вывода

Код программы

```
FILE *s1, *s2;
int val;
char buf[256];
                ...
s1 = fopen("file1","r");
fscanf(s1,"%d",&val);
fseek(s1,4, SEEK_CUR);
fscanf(s1,"%d",&val);

fclose(s1);
```



Файловая система

Литература

1. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. – М.: Мир, - 360 с., ил.