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 № 5

Тема Буферизованный и не буферизованный ввод-вывод

Студент Сушина А.Д.

Группа ИУ7-61б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю,

Москва.
2020 г

Оглавление

1. Задание на лабораторную работу

В лабораторной работе анализируется результат выполнения трех программ. Программы демонстрируют открытие одного и того же файла несколько раз. Реализация открытия файла в одной программе несколько раз выбрана для простоты. Такая ситуация возможна в системе, когда один и тот же файл несколько раз открывают разные процессы. Но для получения ситуаций аналогичных тем, которые демонстрируют приведенные программы надо было бы синхронизировать работу процессов. При выполнении асинхронных процессов такая ситуация вероятна и ее надо учитывать, чтобы избежать потери данных или получения неверного результата при выводе в файл.

- Проанализировать работу приведенных программ и объяснить результаты их работы.
- Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции fopen(). Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции fprintf() то первый дескриптор, то – второй.

2. Ход работы

1. Первая программа

Код первой программы представлен на листинге 1.

Листинг 1. TestCIO.c

```
#include <stdio.h>
#include <fcntl.h>
/*
On my machine, a buffer size of 20 bytes
translated into a 12-character buffer.
Apparently 8 bytes were used up by the
stdio library for bookkeeping.
 */
int main()
{
  // have kernel open connection to file alphabet.txt
  int fd = open("alphabet.txt",O_RDONLY);
  // create two a C I/O buffered streams using the above connection
  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);
```

```
  // read a char & write it alternatingly from fs1 and fs2
  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 представлен на рисунке 1.

```
nastya@Nastya:~/iu7/sem6/os/lab5$ ./testCIO.o
Aubvcwdxeyfzghijklmnopqrstnastya@Nastya:~/iu7/sem6/os/lab5$
```

Рис 1. Результат работы программы testCIO.o

Системный вызов open() создает дескриптор файла в системной таблице файлов, открытых процессом, и запись в системной таблице открытых файлов. В этой таблице хранится информация о файле, а именно позиция для чтения/записи в файл, режим открытия файла. После создаются два объекта типа FILE функцией fopen, которые ссылаются на созданный файловый дескриптор. Вызов функции setvbuf меняет тип буферизации на полную буферизацию по 20 байт. При первом вызове fscanf(fs1,"%c",&c); в буфер buff1 считываются первые 20 символов (Abcdefghijklmnopqrst), в переменную с записывается, а затем выводится с помощью fprintf, символ ‘A’. При первом вызове fscanf(fs2,"%c",&c);, в буфер buff2 считываются оставшиеся в файле символы – uvwxyz (в с записывается символ ‘u’). Затем считывание происходит из буферов, соответвующих потоку данных. Так как считывание происходит по очереди, сначала символы из первого и второго буфера чередуются, а затем, когда во втором буфере не остается символов, выводятся оставшиеся символы из первого буфера. В результате получается строка, представленная на рисунке 1.

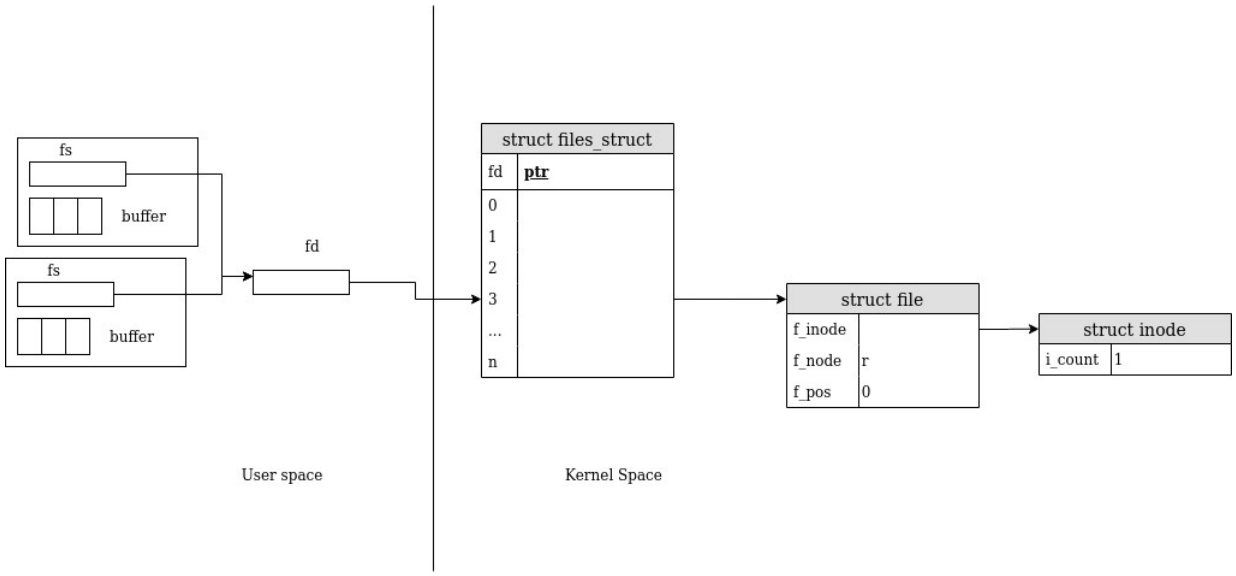

Рис 2. Рисунок, демонстрирующий созданные дескрипторы и связь между ними

2. Вторая программа

Код второй программы представлен на листинге 2. Код переписан без использования break.

Листинг 2. testKernelIO.c

```
#include <fcntl.h>
#include <unistd.h>
int main()
{
    char c;
    // have kernel open two connection to file alphabet.txt
    int fd1 = open("alphabet.txt",O_RDONLY);
    int fd2 = open("alphabet.txt",O_RDONLY);
    // read a char & write it alternatingly from connections fs1 & fd2
    int flag = 1;
    while(flag)
    {
        if (read(fd1,&c,1) == 1) {
            write(1,&c,1);
            if (read(fd2,&c,1) == 1) {
                write(1,&c,1);
            } else {
                flag = 0;
            }
```

```
        } else {
            flag = 0;
        }
    }
    return 0;
}
```

Результат выполнения программы представлен на рисунке 3.

```
nastya@Nastya:~/iu7/sem6/os/lab5$ ./testKernelIO.o
AAbbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
```

Рис 3. Результат программы testKernelIO.c

При вызове системного вызова open() создается дескриптор файла в системной таблице файлов, открытых процессом и запись в системной таблице открытых файлов. В цикле с помощью функции read и write происходит посимвольное чтение из файла. Каждая запись имеет свой указатель позиции в файле, поэтому, при вызове read() для обоих дескрипторов по очереди, оба указателя проходят по всем позициям файла, и каждый символ считывается и выводится по два раза.

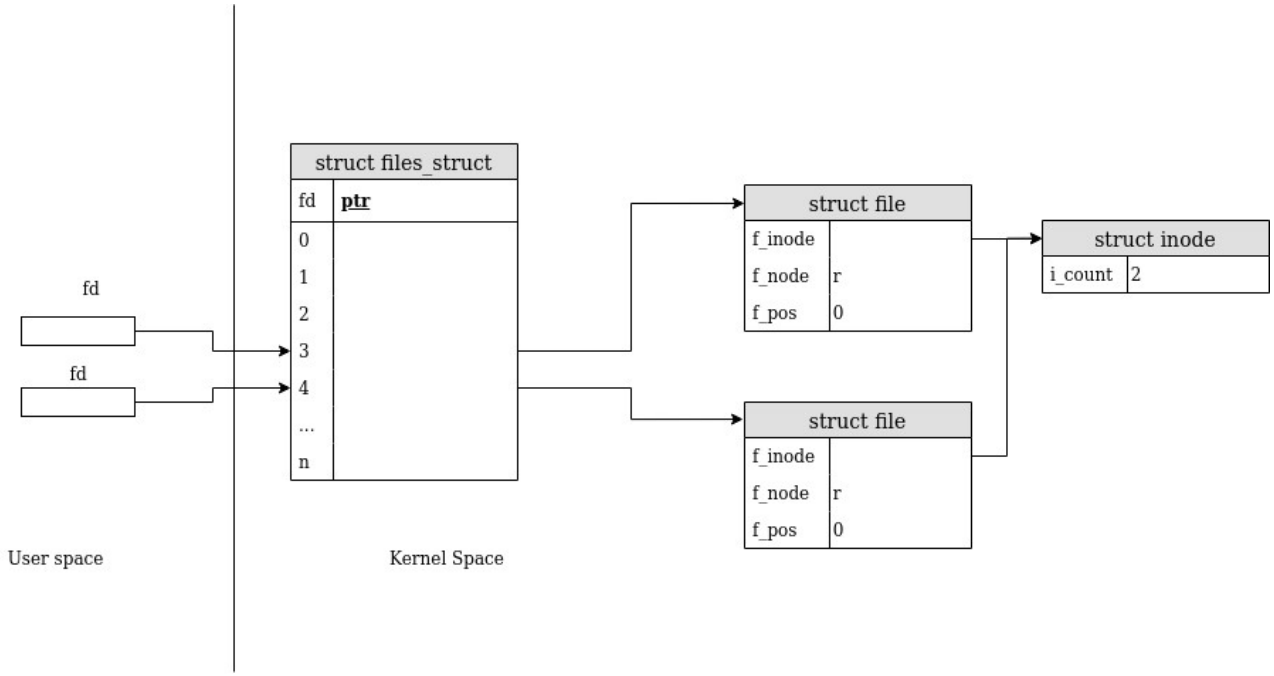

Рис 4. Схема связей дескрипторов 2.

3. Третья программа

Код программы представлен на листинге 3.

Листинг 3. testFopen.c

```c
#include <stdio.h>
int main() {
  FILE* fd[2];
  fd[0] = fopen("output.txt","w");
  fd[1] = fopen("output.txt","w");
  int n = 0;
  for(char c = 'a'; c <= 'z'; c++)
  {
      fprintf(fd[n], "%c", c);
      if (n == 0) {
        n = 1;
      } else {
        n = 0;
      }
  }
  fclose(fd[0]);
  fclose(fd[1]);
}
```

Результат работы программы представлен на рисунке 5.

```
nastya@Nastya:~/iu7/sem6/os/lab5$ ./testFopen.o
nastya@Nastya:~/iu7/sem6/os/lab5$ cat output.txt
bdfhjlnprtvxznastya@Nastya:~/iu7/sem6/os/lab5$
```

Рис 5. Результат работы программы test.Fopen.c

С помощью двух вызовов fopen() создаются два дескриптора файла в таблице файлов, открытых процессом, и две записи в системной таблице открытых файлов. Функция fprintf буферизует данные, следовательно запись производится в два разных буфера. Таким образом в первый буфер попадают нечетные символы, а во второй — четные. При выполнении fclose вызывается fflush, которая переписывает буфер в файл. Таким образом, при первом вызове fclose() осуществляется запись в файл данных из первого буфера, то есть нечетных символов. Затем при втором вызове fclose() осуществляется перезапись данных файла из второго буфера. В результате в файле содержатся данные из второго буфера.

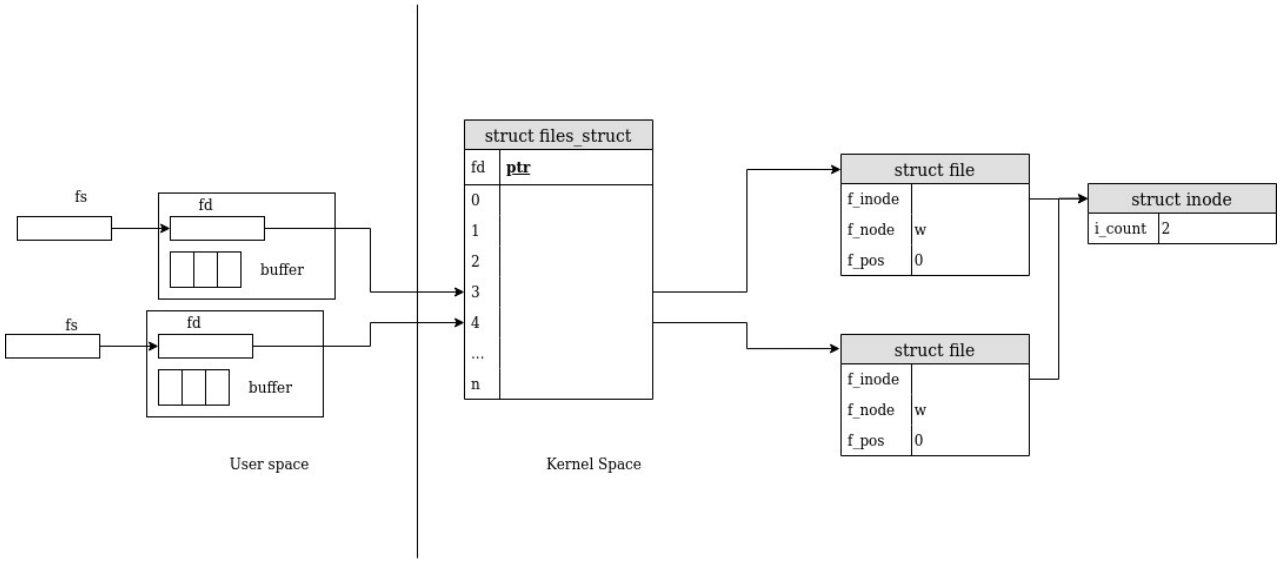

Рис 6. Схема связей дескрипторов 3.

4. Структура FILE

Листинг 1. Структура FILE

```
struct _IO_FILE {
int _flags; /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags
/* The following pointers correspond to the C++ streambuf protocol. */
/* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
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;
#if 0
int _blksize;
#else
```

```
int _flags2;
#endif
_IO_off_t _old_offset; /* This used to be _offset but it's too small. */
#define __HAVE_COLUMN /* temporary */
/* 1+column number of pbase(); 0 is unknown. */
unsigned short _cur_column;
signed char _vtable_offset;
char _shortbuf[1];
/* char* _save_gptr; char* _save_egptr; */
_IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
struct _IO_FILE_complete
{
struct _IO_FILE _file;
#endif
#if defined _G_IO_IO_FILE_VERSION && _G_IO_IO_FILE_VERSION == 0x20001
_IO_off64_t _offset;
# if defined _LIBC || defined _GLIBCPP_USE_WCHAR_T
/* Wide character stream stuff. */
struct _IO_codecvt *_codecvt;
struct _IO_wide_data *_wide_data;
struct _IO_FILE *_freeres_list;
void *_freeres_buf;
# else
void *__pad1;
void *__pad2;
void *__pad3;
void *__pad4;
# endif
size_t __pad5;
int _mode;
/* Make sure we don't get into trouble again. */
char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
#endif
};
typedef struct _IO_FILE FILE;
```