

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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 5

Дисциплина	Операционные системы.
Тема	Буферизованный и небуферизованный ввод-вывод
Студент	Куприй А. А.
Группа	ИУ7-63Б
Преподаватель	Рязанова Н.Ю.

Москва, 2020 г.

ВВЕДЕНИЕ

Цель: проанализировать особенности работы функций ввода-вывода в UNIX/Linux.

Задание: 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.

1 Теоретические сведения

Структура FILE:

Листинг 1.1 — Структура `_IO_FILE`

```
1 struct _IO_FILE
2 {
3     int _flags;          /* High-order word is _IO_MAGIC; rest is flags. */
4     /* The following pointers correspond to the C++ streambuf protocol. */
5     char *_IO_read_ptr;  /* Current read pointer */
6     char *_IO_read_end;  /* End of get area. */
7     char *_IO_read_base; /* Start of putback+get area. */
8     char *_IO_write_base; /* Start of put area. */
9     char *_IO_write_ptr; /* Current put pointer. */
10    char *_IO_write_end;  /* End of put area. */
11    char *_IO_buf_base;   /* Start of reserve area. */
12    char *_IO_buf_end;    /* End of reserve area. */
13    /* The following fields are used to support backing up and undo. */
14    char *_IO_save_base; /* Pointer to start of non-current get area. */
15    char *_IO_backup_base;
16    /* Pointer to first valid character of backup area */
17    char *_IO_save_end; /* Pointer to end of non-current get area. */
18    struct _IO_marker *_markers;
19    struct _IO_FILE *_chain;
20    int _fileno;
21    int _flags2;
22    __off_t _old_offset; /* This used to be _offset but it's too small. */
23    /* 1+column number of pbase(); 0 is unknown. */
24    unsigned short _cur_column;
25    signed char _vtable_offset;
26    char _shortbuf[1];
27    _IO_lock_t *_lock;
28    #ifdef _IO_USE_OLD_IO_FILE
29    };
```

Листинг 1.2 — typedef в файле FILE.h

```
1 typedef struct _IO_FILE FILE;
```

2 Практическая часть

Проанализировать работу приведенных программ и объяснить результат их работы.

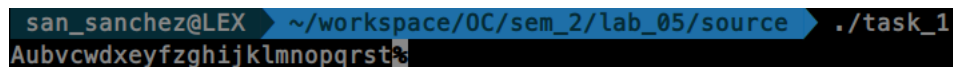
2.1 Задание №1

Первая программа:

Листинг 2.1 — Текст программы первого задания

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 int main()
5 {
6     int fd = open("alphabet.txt", O_RDONLY);
7
8     FILE *fs1 = fdopen(fd, "r");
9     char buff1[20];
10    setvbuf(fs1, buff1, _IOFBF, 20);
11
12    FILE *fs2 = fdopen(fd, "r");
13    char buff2[20];
14    setvbuf(fs2, buff2, _IOFBF, 20);
15
16    int flag1 = 1, flag2 = 2;
17
18    while(flag1 == 1 || flag2 == 1)
19    {
20        char c;
21
22        flag1 = fscanf(fs1, "%c", &c);
23        if (flag1 == 1)
24            fprintf(stdout, "%c", c);
25
26        flag2 = fscanf(fs2, "%c", &c);
27        if (flag2 == 1)
28            fprintf(stdout, "%c", c);
29    }
30
31    return 0;
32 }
```

Результат:



```
san_sanchez@LEX > ~/workspace/0C/sem_2/lab_05/source > ./task_1
Aubvcwdxeyfzghijklmnopqrst
```

Рисунок 2.1 — Скришот результата работы первой программы.

Системный вызов `open()` создаёт файловый дескриптор, который открывается только на чтение, указатель устанавливается на начало файла. В результате чего создаётся запись в общесистемной таблице открытых файлов.

Затем с помощью функции `fdopen()` создаются два разных объекта структуры `FILE`, которые ссылаются на один файловый дескриптор `fd` и `setvbuf()` устанавливает тип буферизации блоком, размер которого 20 байт.

Далее в цикле происходит чтение и вывод. Системная функция `fscanf()` возвращает -1, если число прочитанных символов равно нулю, и 1 в ином случае.

При первом вызове `fscanf()` буфер ввода структуры `FILE` заполняется либо до конца буфера, либо до конца файла. Поэтому буфер заполняется первыми 20 символами и значение текущей позиции смещается.

Так как `fs1` и `fs2` ссылаются на одну и ту же запись в системной таблице открытых файлов, значение позиции файла одинаковое, из чего следует, что при следующем вызове `fscanf()` буфер ввода структуры `fs2` считает оставшиеся символы из файла. В результате получается строка, в которой символы чередуются из первого и второго буфера.

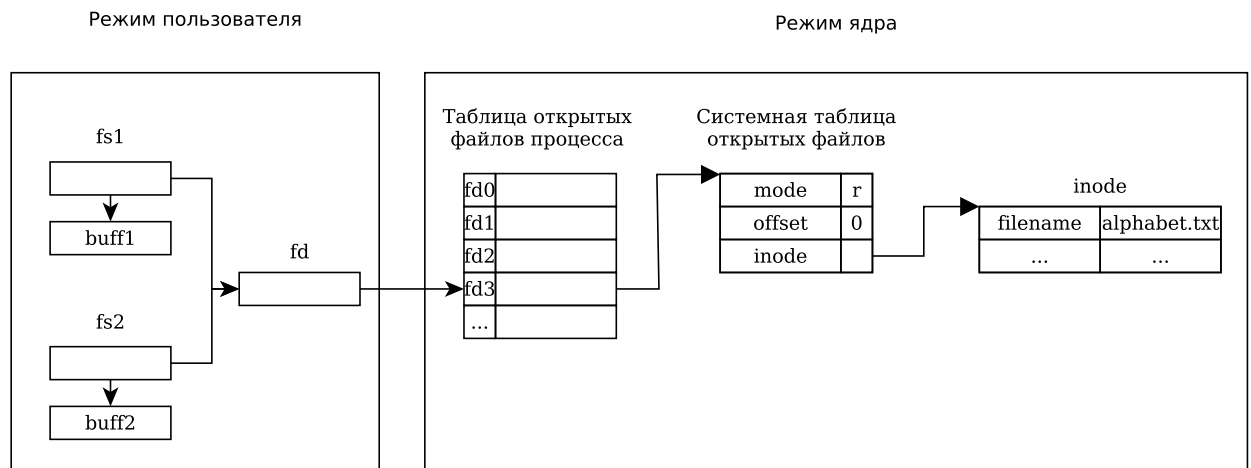


Рисунок 2.2 — Схема связи дескрипторов первой программы.

2.2 Задание №2

Вторая программа:

Листинг 2.2 — Текст программы второго задания

```

1  #include <stdio.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4
5  int main()
6  {
7      char c;
8      int cond = 1;
9      int fd1 = open("alphabet.txt", O_RDONLY);
10     int fd2 = open("alphabet.txt", O_RDONLY);
11
12     while(cond)
13     {
14         if ((cond = read(fd1, &c, 1)) == 1)
15         {
16             write(1, &c, 1);
17         }
18
19         if ((cond = read(fd2, &c, 1)) == 1)
20         {
21             write(1, &c, 1);
22         }
23     }
24

```

```

25     return 0;
26 }

```

Результат:

Рисунок 2.3 — Скришот результата работы второй программы.

Системный вызов `open()` создаёт 2 разных файловых дескриптора для открытого файла, 2 записи в общесистемной таблице открытых файлов. Так как файловые дескрипторы разные, то у каждого имеется своя текущая позиция файла. В результате получается строка, в которой символы дублируются.

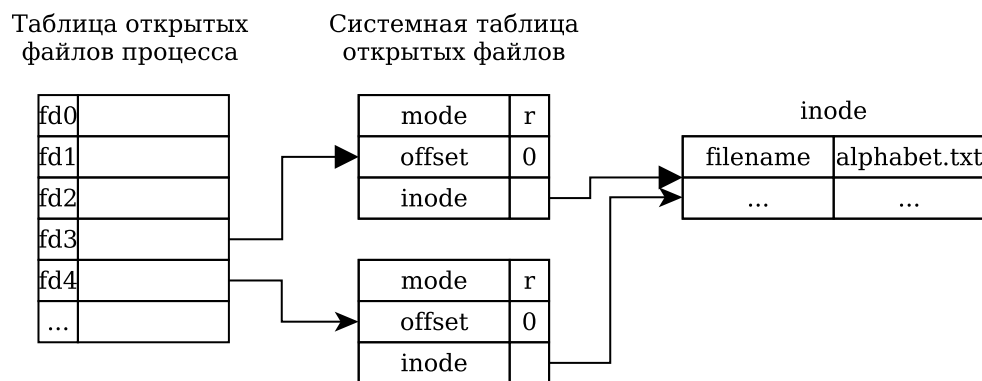


Рисунок 2.4 — Схема связи дескрипторов второй программы.

2.3 Задание №3

Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции `fdopen()`. Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции `fprintf()` то первый дескриптор, то – второй. Результат прокомментировать.

Листинг 2.3 — Текст программы третьего задания

```

1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <errno.h>
4 #include <string.h>

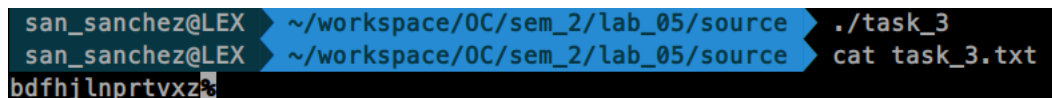
```

```

5
6 int main()
7 {
8     FILE *fd1 = fopen("task_3.txt", "w");
9     if (fd1 == NULL)
10    {
11        printf("%s", strerror(errno));
12        return errno;
13    }
14
15    FILE *fd2 = fopen("task_3.txt", "w");
16    if (fd1 == NULL)
17    {
18        printf("%s", strerror(errno));
19        return errno;
20    }
21
22    for(char c = 'a'; c <= 'z'; c++)
23    {
24        if (c % 2)
25        {
26            fprintf(fd1, "%c", c);
27        }
28        else
29        {
30            fprintf(fd2, "%c", c);
31        }
32    }
33
34    fclose(fd1);
35    fclose(fd2);
36
37    return 0;
38 }

```

Результат:



```

san_sanchez@LEX ~/workspace/0C/sem_2/lab_05/source ./task_3
san_sanchez@LEX ~/workspace/0C/sem_2/lab_05/source cat task_3.txt
bdfhjlnprtvxz%

```

Рисунок 2.5 — Скришот результата работы третьей программы.

Функция `open()` создаёт 2 разных файловых дескриптора и две независимые позиции в файле указывают на начало. Далее в цикле поочередно происходит запись символов от `a` до `z` в буферизированные потоки. В первый

записываются символы на нечётных позициях, а во второй - чётных. Смещение текущей позиции файла независимо для двух различных дескрипторов. Функция `fprintf()` выполняет буферизированный вывод - запись в файл происходит при вызове функции `fclose()`, `fflush()` или при полном заполнении буфера. Функция `fclose(fd1)` очищает поток, на который указывает `fd1` (запись любых буферизованных выходных данных с помощью `fflush()`), и закрывает файловый дескриптор. Далее выполняется функция `fclose(fd2)`, которая своим выполнением перезапишет данные, записанные `fclose(fd1)`. В результате получается строка, состоящая из символов английского алфавита, стоящих на чётных позициях.

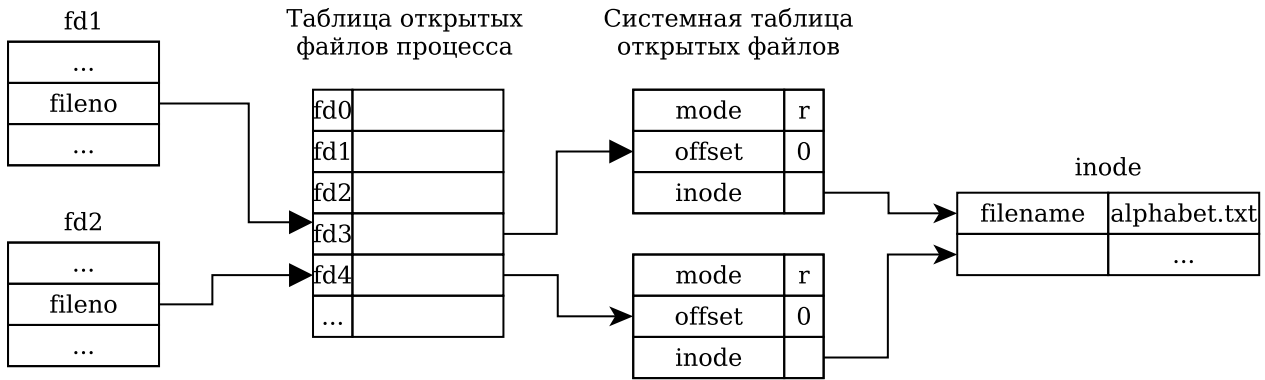


Рисунок 2.6 — Схема связи дескрипторов третьей программы.

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были проанализированный особенности работы функции ввода-вывода в UNIX/LINUX.