|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

**по курсу “Операционные системы”**

**по теме “Буферизованный и не буферизованный ввод-вывод”**

|  |  |
| --- | --- |
| Студент: | Уласик Е.А. |
| Группа: | ИУ7-61 |
| Преподаватель: | Рязанова Н.Ю. |

1. *г.*

**Задача:** анализ особенностей работы функций ввод-вывода в Unix/Linux.

# Программа testCIO

Содержание файла alphabet.txt:

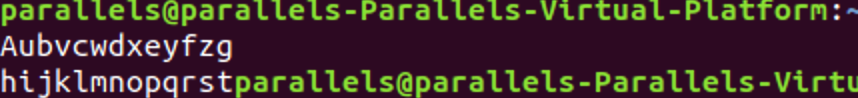
Abcdefghijklmnopqrstuvwxyz

Листинг программы:

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

Листинг 1. Листинг программы testCIO.c

Результат работы:



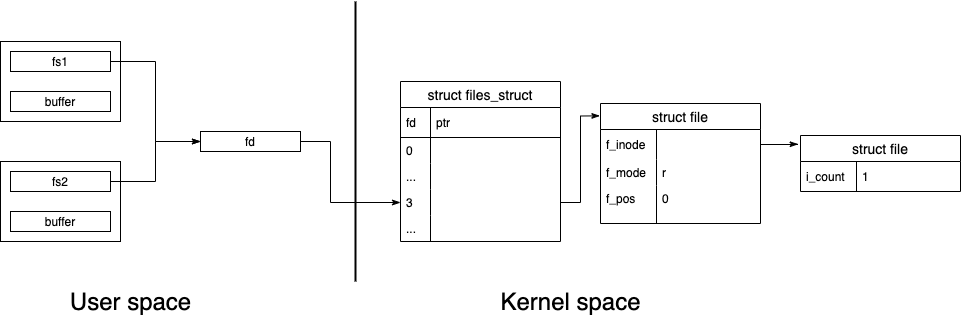
Картинка 1. Результат работы testCIO.c

Анализ работы:

При вызове системного вызова open() создается дескриптор файла в системной таблице файлов, открытых процессом и запись в системной таблице открытых файлов. В этой таблице содержится информация о файле: позиция для чтения/записи в файл, режим открытия файла.

Затем фунцией fdopen создаются два объекта типа FILE, которые ссылаются на созданный файловый дескриптор. С помощью функций setvbuf создаются 20 байтовые буферы и режим полной буферизации для объектов типа FILE. В цикле осуществляется посимвольное чтение из файла при помощи вызова fscanf(fs, "%c”, &c) и затем запись в стандартный поток вывода при помощи fprintf(stdout, “%c”, c). При чтении из файла происходит буферизация, то есть при первом вызове fscanf(fs1, “%c”, &c) первый буфер заполняется первыми 20 символами файла, а при следующей вызове функции второй буфер заполняется оставшимися символами. Причина этому в том, что обе структуры FILE ссылаются на один и тот же дескриптор файла. При первом чтении указатель в файле смещается на максимальное для первого буфера 20 байт. Посимвольное чтение и вывод фактически происходят не из файла, а из буферов.

Cхема связей дескриптора:

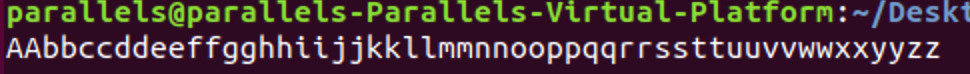


# Программа testKernel

1. #include <stdio.h>
2. #include <fcntl.h>
4. **int** main()
5. {
6. **char** c;
7. **int** fd1 = open("alphabet.txt", O\_RDONLY);
8. **int** fd2 = open("alphabet.txt", O\_RDONLY);
9. **int** flag = 1;
10. **while**(flag)
11. {
12. **if** (read(fd1, &c, 1) == 1)
13. write(1, &c, 1);
14. **else**
15. flag = 0;
16. **if** (read(fd2, &c, 1) == 1)
17. write(1, &c, 1);
18. **else**
19. flag = 0;
20. }
21. **return** 0;
22. }

Листинг 2. Листинг программы testKernel.c

Результат работы:

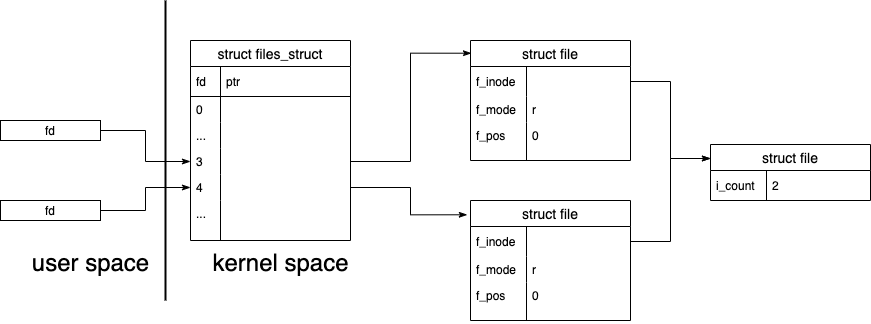


Картинка 2. Результат работы testKernel.c

Анализ работы:

При вызове системного вызова open создаётся 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. Так как были созданы 2 независимых записи и каждая запись имеет. свой указатель позиции в файле, вывод в стандартный пыток происходит дважды. Таким образом, символы из файла продублировались.

Схема связей дескриптора:

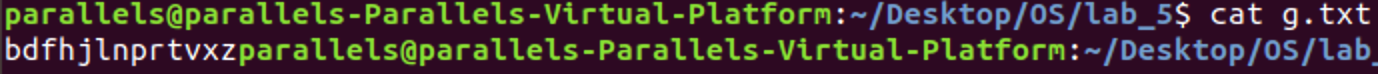


# Программа testFOpen

1. #include <stdio.h>
3. **int** main()
4. {
5. **FILE** \*f1 = fopen("g.txt", "w");
6. **FILE** \*f2 = fopen("g.txt", "w");
7. **for**(**char** c='a'; c<='z'; c++)
8. {
9. **if** (c % 2 == 1)
10. fprintf(f1, "%c", c);
11. **else**
12. fprintf(f2, "%c", c);
13. }
15. fclose(f1);
16. fclose(f2);
17. **return** 0;
18. }

Листинг 3. Листинг программы testFOpen.c

Результат работы:

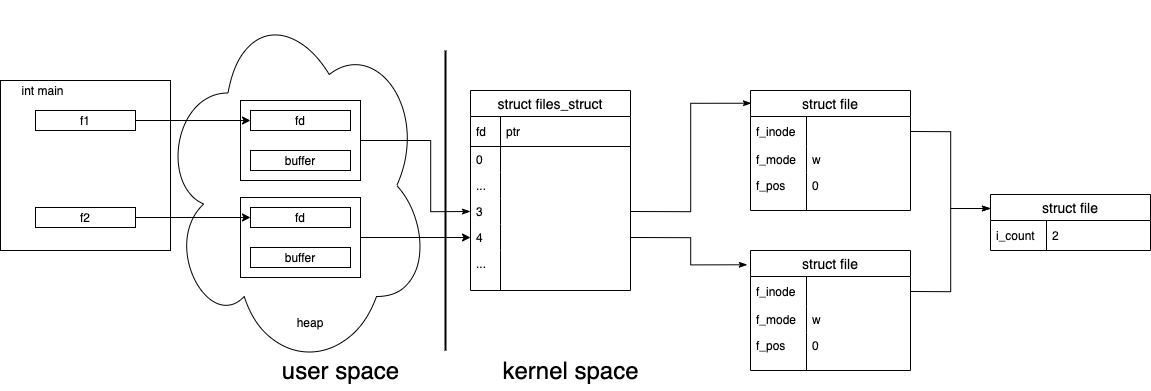


Картинка 3. Содержание файла g.txt.

Анализ работы:

С помощью двух вызовов fopen(“output.txt”, “w”) создаются два дескриптора файла в таблице файлов, открытых процессом, и две записи в системной таблице открытых файлов. Функция fprintf обеспечивает буферизацию и таким образом запись производится в два разных буфера. В цикле от латинского символа “a” до латинского символа “z” происходит запись символов в первый и второй буферы: в первый буфер записываются нечётные символы, во второй – чётные. Запись в файл осуществляется только в случае переполнения буфера, вызове fflush или fclose, внутри которого осуществляется вызов fflush, если поток использовался для вывода данных. При вызове fflush. происходит запись всех данных из буфера в файл. При первом вызове fclose осуществляется запись в файл “g.txt” данных из первого буфера, то есть нечётных символов. Затем при втором вызове fclose осуществляется перезапись файла “g.txt” данными из второго буфера с чётными символами.

Схема связей дескриптора:



Структура FILE

1. **struct** \_IO\_FILE {
2. **int** \_flags; /\* High-order word is \_IO\_MAGIC; rest is flags. \*/
3. #define \_IO\_file\_flags \_flags
4. /\* The following pointers correspond to the C++ streambuf protocol. \*/
5. /\* Note: Tk uses the \_IO\_read\_ptr and \_IO\_read\_end fields directly. \*/
6. **char**\* \_IO\_read\_ptr; /\* Current read pointer \*/
7. **char**\* \_IO\_read\_end; /\* End of get area. \*/
8. **char**\* \_IO\_read\_base; /\* Start of putback+get area. \*/
9. **char**\* \_IO\_write\_base; /\* Start of put area. \*/
10. **char**\* \_IO\_write\_ptr; /\* Current put pointer. \*/
11. **char**\* \_IO\_write\_end; /\* End of put area. \*/
12. **char**\* \_IO\_buf\_base; /\* Start of reserve area. \*/
13. **char**\* \_IO\_buf\_end; /\* End of reserve area. \*/
14. /\* The following fields are used to support backing up and undo. \*/
15. **char** \*\_IO\_save\_base; /\* Pointer to start of non-current get area. \*/
16. **char** \*\_IO\_backup\_base; /\* Pointer to first valid character of backup area \*/
17. **char** \*\_IO\_save\_end; /\* Pointer to end of non-current get area. \*/
19. **struct** \_IO\_marker \*\_markers;
21. **struct** \_IO\_FILE \*\_chain;
23. **int** \_fileno; #**if** 0
24. **int** \_blksize; #**else**
25. **int** \_flags2; #endif
26. \_IO\_off\_t \_old\_offset; /\* This used to be \_offset but it's too small. \*/
28. #define \_\_HAVE\_COLUMN /\* temporary \*/
29. /\* 1+column number of pbase(); 0 is unknown. \*/
30. unsigned **short** \_cur\_column;
31. **signed** **char** \_vtable\_offset;
32. **char** \_shortbuf[1];
33. /\* char\* \_save\_gptr; char\* \_save\_egptr; \*/
34. \_IO\_lock\_t \*\_lock;
35. #ifdef \_IO\_USE\_OLD\_IO\_FILE
36. };