|  |  |
| --- | --- |
| 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 предмету: «Операционные системы»*

**Тема: Буферизованный и не буферизованный ввод-вывод**

Преподаватель: Рязанова Н.Ю.

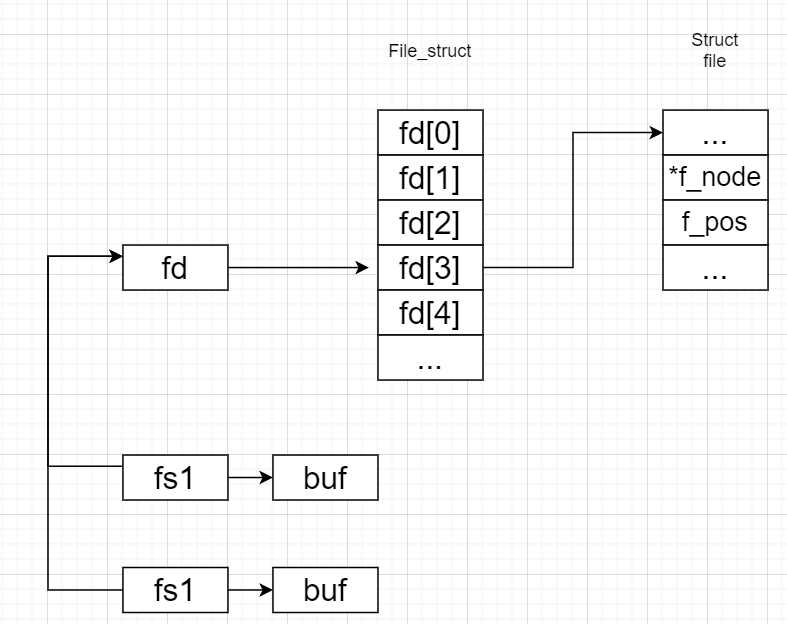
Студент: Мирзоян С.А.

Группа: ИУ7-65Б

Москва, 2020 г.

**Программа 1**

Связь структур:



С помощью системного вызова open() создается дескриптор файла. Файл открывается только на чтение, так как передается флаг O\_RDONLY.

При успешном завершении системного вызова появляется новый дескриптор открытого файла alphabet.txt и запись в системной таблице открытых файлов.

Функция fdopen() создает два объекта типа FILE, которые связываются с открытым файлом, на который ссылается файловый дескриптор fd.

struct FILE {

ssize\_t \_cnt; // число байт в буфере \_cnt

unsigned char \*\_ptr; // указатель на следующий

//символ, который подлежит чтению или записи \_ptr

unsigned char \*\_base; // указатель на буфер \_base

unsigned char \_flag; // флаги состояния потока \_flag

unsigned char \_file; // указатель на файловый дескриптор \_file, с которым //ассоциирован данный поток

...

};

При создании буфера размер для данного потока выбирается системой. Чтобы его изменить используется системный вызов setvbuf(). B данной программе системный вызов setvbuf() изменяет тип буферизации для каждого объекта FILE на полную буферизацию, a также явно задает размер буфера 20 байт.

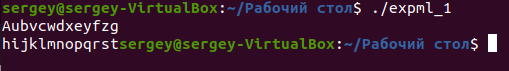
При первом вызове fscanf() буфер структуры FILE заполняется до тех пор, пока он не будет заполнен полностью, либо пока не будет достигнут конец файла. Так как буфер имеет размер 20 байт, а файл содержит 26 байт данных (26 букв латинского алфавита), то после первого вызова fscanf() в буфере первой структуры FILE будут находиться первые 20 байт файла (то есть буквы с A по t). Так как оба объекта FILE связаны с одним и тем же файловым дескриптором, то позиция в файле будет определяться для обоих файловых потоков ввода полем f\_pos структуры struct file, на которую ссылается указанный дескриптор файла. Поэтому после второго вызова fscanf() в буфере второй структуры FILE окажутся последние 6 байт файла ( то есть символы с u по z).

Затем в стандартный поток вывода stdout будет поочередно осуществляться вывод по одному символу из каждого буфера. Когда второй буфер опустеет, из первого буфера продолжат выводится оставшиеся символы.

Листинг:

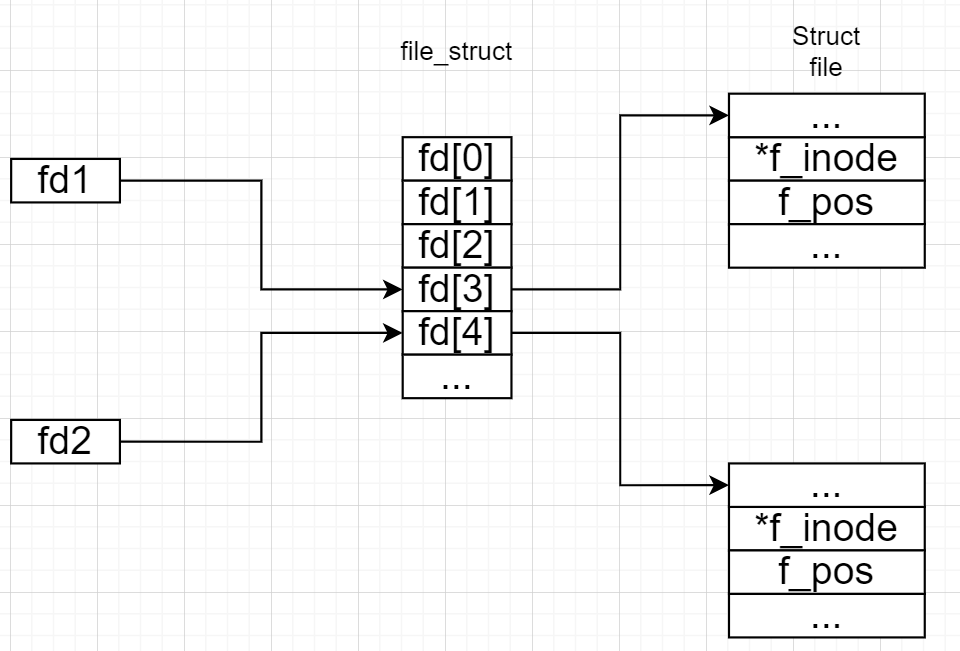
1. #include <stdio.h>
3. #include <fcntl.h>
5. int main()
6. {
7. // have kernel open connection to file alphabet.txt
8. int fd = open("alphabet.txt",O\_RDONLY);
9. // create two a C I/O buffered streams using the above connection
11. FILE \*fs1 = fdopen(fd,"r");
12. char buff1[20];
13. setvbuf(fs1,buff1,\_IOFBF,20);
14. FILE \*fs2 = fdopen(fd,"r");
15. char buff2[20];
16. setvbuf(fs2,buff2,\_IOFBF,20);
18. // read a char & write it alternatingly from fs1 and fs2
19. int flag1 = 1, flag2 = 2;
20. while(flag1 == 1 || flag2 == 1)
21. {
22. char c;
23. flag1 = fscanf(fs1,"%c",&c);
24. if (flag1 == 1)
25. {
26. fprintf(stdout,"%c",c);
27. }
29. flag2 = fscanf(fs2,"%c",&c);
30. if (flag2 == 1)
31. {
32. fprintf(stdout,"%c",c);
33. }
35. return 0;
36. }

Результат работы программы:



Программа 2

Связь структур:



В данной программе файл открывается дважды для чтения с помощью системного вызова ореn(). Создаются две различных структуры struct file, которые описывают открытый файл, которые связаны с одним и тем же физическим файлом. Текущие позиции в файле для каждой структуры будут изменяться независимо друг от друга. Поэтому чтение с использованием одной структуры не затрагивает текущую позицию в другой структуре, и каждый символ из физического файла будет продублирован.

Листинг:

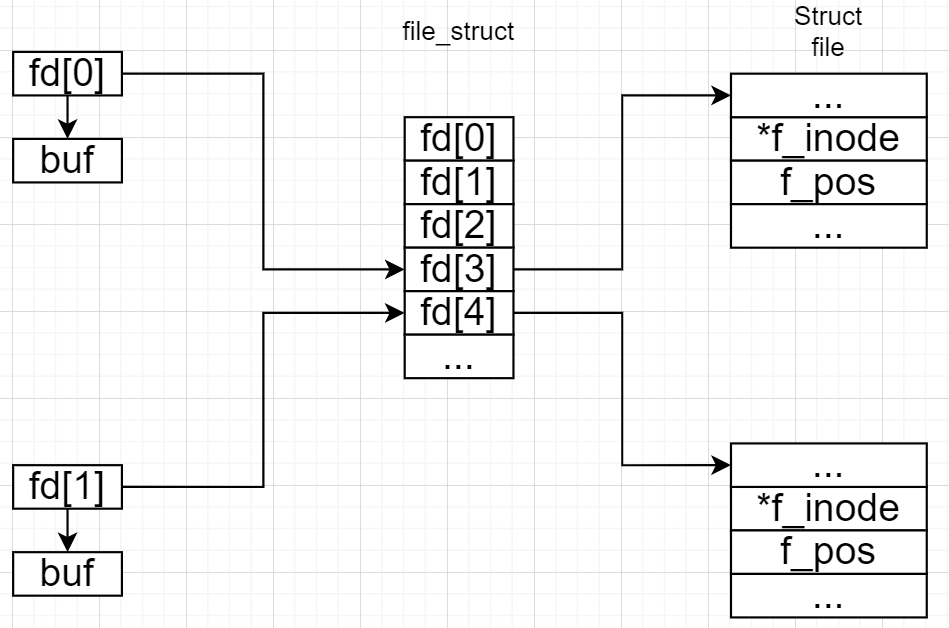
1. #include <fcntl.h>
3. int main()
4. {
5. char c;
7. // have kernel open two connection to file alphabet.txt
8. int fd1 = open("alphabet.txt",O\_RDONLY);
9. int fd2 = open("alphabet.txt",O\_RDONLY);
11. int break\_flag = 1;
12. // read a char & write it alternatingly from connections fs1 & fd2
13. while(break\_flag)
14. {
15. if (read(fd1,&c,1)!= 1)
16. break\_flag = 0;
17. write(1,&c,1);
19. if (read(fd2,&c,1)!= 1)
20. break\_flag = 0;
21. write(1,&c,1);
22. }
23. return 0;
25. }

Результат работы программы:



Программа 3

Связь структур:



В данной программе с помощью функции fopen() два раза открывается на запись файл alphabet.txt, то есть создаются две разных структуры struct file, в которых поля f\_pos при вызове функций ввода/вывода меняются независимо (аналогично предыдущему примеру).

B результате поочередной записи букв латинского алфавита в первый буфер будут записаны нечетные символы, а во второй буфер — четные.

Запись из буфера в файл происходит автоматически в следующих случаях:

1) при заполнении буфера:

2) по завершении процесса:

3) при вызове функций fclose() или fflush( ).

Сначала закрывается первый поток, поэтому изначально в файл in.txt осуществляется запись буфера первого потока (acegikmoqsuwy), при этом данные записываются с начала файл, затем происходит закрытие второго потока и запись его буфера в файл in.txt. При этом так как оба объекта FILE связаны с разными структурами struct file, то значения их полей f\_pos изменяются независимо, и запись второго буфера будет также произведена с начала файла, то есть данные, записанные в файл из буфера первою потока будут перезаписаны

Листинг:

1. #include <stdio.h>
3. **int** main()
4. {
5. FILE \*f1;
6. FILE \*f2;
8. f1 = fopen("alphabet.txt","w");
9. f2 = fopen("alphabet.txt","w");
11. **int** flag;
13. **for** (**char** c = 'a'; c <= 'z'; c++)
14. {
15. **if** (!flag)
16. fprintf(f1, "%c", c);
18. **if** (!flag)
19. fprintf(f2, "%c", c);
20. flag = !flag;
21. }
22. fclose(f1);
23. fclose(f2);
25. **return** 0;
26. }

Результат работы программы:

****

**Вывод**

Необходимо учитывать при буферизированном вводе/выводе факт записи/чтения данных из буфера, так как неправильные действия с данными могут привести к неправильной последовательности данных или даже к их потере. Также необходимо учитывать, что при одновременном открытии одного и того же файла создается дескриптор открытого файла. Каждый дескриптор struct file имеет поле f\_pos, указатель на позицию чтения или записи в логическом файле.