Московский государственный технический университет
имени Н. Э. Баумана

Факультет: Информатика и системы управления
Кафедра: Программное обеспечение ЭВМ и информационные технологии
Дисциплина: Операционные системы

Лабораторная работа №4
Анализ особенностей работы функций ввода-вывода в UNIX/Linux

Выполнила: Щербатюк Дарья
Группа:ИУ7-64

Москва, 2018 г.

1 testCIO.c

Текст программы

```c
//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
  // associates a stream with the existing file descriptor
  FILE *fs1 = fdopen(fd,"r");
  char buff1[20];
  setvbuf(fs1,buff1,_IOFBF,20); // set fully buffering

  FILE *fs2 = fdopen(fd,"r");
  char buff2[20];
  setvbuf(fs2,buff2,_IOFBF,20);

  // read a char and 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;
}
```

Вывод

```
Aubvcwdxeyfzghijklmnopqrst
```

Анализ

С помощью системного вызова $open()$ создается дискриптор файла, файл открывается только на чтение, указатель устанавливается на начало файла. Если системный вызов завершается успешно, возвращенный файловый дескриптор является наименьшим, который еще не открыт процессом. В результате этого вызова появляется новый открытый файл, не разделяемый никакими процессами, и запись в системной таблице открытых файлов.

Далее функция $fdopen()$ связывает два потока с существующим дискриптором файла. Функция $setvbuf()$ изменяет тип буферизации на блочную (полную) размером в 20 байт.

В цикле осуществеляется чтение из потоков и вывод в $stdout$с помощью системных функций $fscanf, fprintf$. Флаги $flag1, flag2$ изменят свое значение с 1 на -1 тогда, когда число прочитанных символов станет равно нулю. Стоит помнить о том, что открытые файлы, для которых используется ввод/вывод потоков, буферизуются. Т.к. размер буфера установлен в 20 байт, по факту в $buff1$ помещается строка $Abcdefghijklmnopqrst$, а в $buff2$ - $uvwxyz$. В результате поочередного вывода из каждого буфера потока получим строку выше.

2 testKernelIO.c

Текст программы

```
1  //testKernelIO.c
2  #include <fcntl.h>
3
4  int main()
5  {
6    // have kernel open two connection to file alphabet.txt
7    int fd1 = open("alphabet.txt",O_RDONLY);
8    int fd2 = open("alphabet.txt",O_RDONLY);
9
10
11   // read a char & write it alternatingly from connections fs1 & fd2
12   while(1)
13   {
14     char c;
15     if (read(fd1,&c,1) != 1) break;
16     write(1,&c,1);
17     if (read(fd2,&c,1) != 1) break;
18     write(1,&c,1);
19   }
20
21   return 0;
22 }
```

Вывод

```
1  AAbbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
```

Анализ

В отличии от первой программы, создаются два файловых дескриптора и две разные записи в системной таблице открытых файлов. Файловые дескрипторы независимы друг от друга, поэтому положения указателей в файле, связанных с данным файловым дескриптором, будут независимы. В результате прохода цикла, где с помощью системных вызовов *read*, *write* в *stdout* выводится по одному символу из файла, увидим строку с дублирующимися буквами.

3 testFOpen.c

```
1  #include <stdio.h>
2  int main() {
3
4    FILE* fd[2];
5    fd[0] = fopen("testFOpen_output.txt","w");
6    fd[1] = fopen("testFOpen_output.txt","w");
7
8    int curr = 0;
9
10   for(char c = 'a'; c <= 'z'; c++, curr = ((curr != 0) ? 0 : 1))
11   {
12       fprintf(fd[curr], "%c", c);
13   }
14   fclose(fd[0]);
15   fclose(fd[1]);
16   return 0;
17 }
```

Вывод

```
1  bdfhjlnprtvxz
```

Анализ

С помощью функций $fopen()$ открываем два потока на запись с начала файла. Они имеют два разных файловых дескриптора и следовательно независимые позиции в файле. Затем в цикле с помощью $fprintf()$ в потоки поочередно записываются буквы от a до z, т.е. нечетные в первый поток, четные во второй. Следует помнить о том, что функция $fprintf()$ обеспечивает буферизацию. Запись непосредственно в сам файл происходит в трех случаях: либо при полном заполнении буфера, либо при вызове функций $fclose()$ и $fflush()$. Функция $fclose()$ отделяет указанный поток от связанного с ним файла или набора функций. Если поток использовался для вывода данных, то все данные, содержащиеся в буфере, сначала записываются с помощью $fflush()$. Функция $fflush()$ принудительно записывает все буферизированные данные в устройство вывода данных. При этом поток остается открытым. Т.к. оба потока открыты в режиме перезаписи в файл, то после выполнения второго $fclose()$ данные в файле, записанные с помощью первого потока, будут переписаны и мы увидим там буквы, чтоящие на нечетных местах.

4 Вывод

Исходя из вышеприведенных рассуждений, можно сделать несколько выводов.

1. Предпочтительней использовать функцию $fopen()$, т.к. $fopen()$ выполняет ввод-вывод с буферизацией, что может оказаться значительно быстрее, чем с использованием $open()$, $FILE*$ дает возможность использовать $fscanf()$ и другие функции $stdio.h$.

2. Следует помнить о буферизации и вовремя использовать $fclose()$ для записи в файл.

3. С острожностью использовать $fflush()$, т.к. она оставляет поток открытым.

4. Необходимо следить за режимом, с котором открывается поток.

5. Созданный новый дескриптор открытого файла изначально не разделяется с любым другим процессом, но разделение может возникнуть через $fork()$.

6. Функции $fscanf, fprintf, fopen, fclose$ являются обертками высшего уровня над системными вызовами $open, close, read, write$.