|  |  |
| --- | --- |
| **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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Буферизованный и не буферизованный ввод-вывод**

|  |  |
| --- | --- |
| **Студент** Жигалкин Д.Р.  **Группа** ИУ7-65Б  **Преподаватель** Рязанова Н. Ю. |  |

Москва.

2021 г.

**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.

Структура FILE

|  |
| --- |
| Typedef struct \_IO\_FILE 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  }; |

**Проанализировать работу приведенных программ и объяснить результаты их работы.**

В файле alphabet.txt находятся следующие символы: Abcdefghijklmnopqrstuvwxyz\n

**Первая программа.**

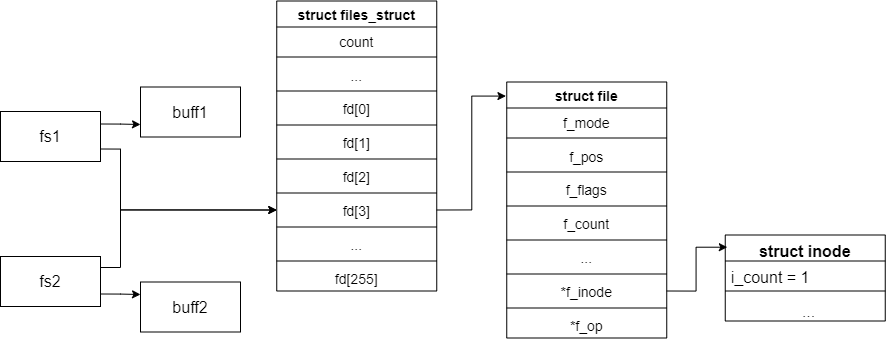
|  |
| --- |
| #include <stdio.h>  #include <fcntl.h>  /\*  On my machine, a buffer size of 20 bytes  translated into a 12-character buffer.  Apparently 8 bytes were used up by the  stdio library for bookkeeping.  \*/  int main()  {  // have kernel open connection to file alphabet.txt  Int fd = open("alphabet.txt", 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;  } |

Листинг 1. Первая программа.

Результат работы программы:

Aubvcwdxeyfzghijklmnopqrst

Схема связей структур



С помощью системного вызова open() создается дескриптор открытого на чтение файла. Системный вызов open() возвращает индекс в массиве fd структуры files\_struct. fdopen() создает структуры типа FILE(fs1 и fs2), которые ссылаются на дескриптор, созданный системным вызовом open.

Создаём буферы buff1 и buff2 размером 20 байт. Для анализа буферов для дескрипторов fs1 и fs2 помощью setbuv задаём соответствующие буферы и задаём тип буферизации \_IOFBF(полная буферизация).

Далее выполняем в цикле  fscanf() поочерёдно для fs1 и  fs2. Так как установлена полная буферизация, то при первом вызове fscanf() буфер будет заполнен полностью либо вплоть до конца файла, а f\_pos установится на следующий за последним записанным в буфер символ.

При первом вызове fscanf(fs1,"%c",&c) в буфер buff1 считаются первые 20 символов (abcdefghijklmnopqrst), в переменную c 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').

Внутри цикла будут поочередно выводится символы из buff1 и buff2 до тех пор, пока символы в одном из буферов не закончатся. Тогда на экран будут последовательно выведены оставшиеся символы из другого буфера.

**Вторая программа.**

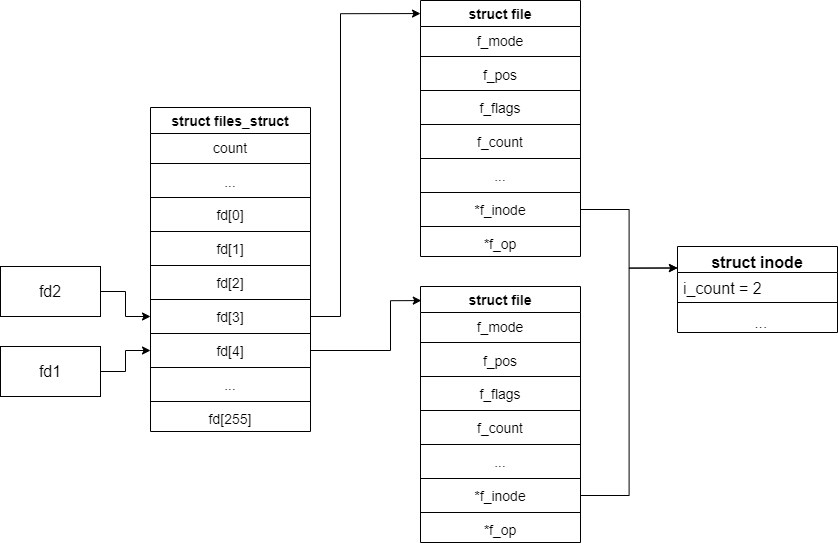
|  |
| --- |
| #include <fcntl.h>  int main()  {  char c;  // have kernel open two connection to file alphabet.txt  int fd1 = open("alphabet.txt", O\_RDONLY);  int fd2 = open("alphabet.txt", O\_RDONLY);  // read a char & write it alternatingly from connections fd1 & fd2  while (read(fd1, &c, 1) == 1 || read(fd2, &c, 1) == 1)  {  write(1, &c, 1);  write(1, &c, 1);  }  return 0;  } |

Листинг 2.1. Вторая программа.

Результат работы программы:

AAbbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz

Схема связей структур



Программа демонстрирует ситуацию, когда один и тот же файл открыт 2 раза для чтения. При вызове системного вызова open() создается дескриптор файла в системной таблице файлов, открытых процессом и запись в системной таблице открытых файлов.

В данном случае файл открывается 2 раза, поэтому в таблице открытых файлов будет 2 дескриптора и каждый такой дескриптор имеет собственный f\_pos. Это позволяет сделать чтение независимым, т.е.  при вызове read() для обоих дескрипторов по очереди, оба указателя проходят по всем позициям файла, и каждый символ считывается и выводится по два раза. Несмотря на то, что существует 2 дескриптора открытого файла, открывается один и тот же файл, т.е. inode один и тот же.

**Третья программа.**

|  |
| --- |
| #include <stdio.h>  #include <sys/stat.h>  int main()  {  struct stat \*statbuf;  FILE \*f1 = fopen("write\_res.txt", "w");  stat("write\_res.txt", statbuf);  printf("fopen file #1inode = %ld, buffsize = %ldblocksize= %ld\n", statbuf->st\_ino, statbuf->st\_size, statbuf->st\_blksize);  FILE \*f2 = fopen("write\_res.txt", "w");  stat("write\_res.txt", statbuf);  printf("fopen file #2inode = %ld, buffsize = %ldblocksize= %ld\n", statbuf->st\_ino, statbuf->st\_size, statbuf->st\_blksize);  for (char c = 'a'; c <= 'z'; c++)  {  if (c % 2)  fprintf(f1, "%c", c);  else  fprintf(f2, "%c", c);  }  fclose(f1);  stat("write\_res.txt", statbuf);  printf("fclose file #1inode = %ld, buffsize = %ldblocksize= %ld\n", statbuf->st\_ino, statbuf->st\_size, statbuf->st\_blksize);  fclose(f2);  stat("write\_res.txt", statbuf);  printf("fclose file #2inode = %ld, buffsize = %ldblocksize= %ld\n", statbuf->st\_ino, statbuf->st\_size, statbuf->st\_blksize);  return 0;  } |

Листинг 3.1. Третья программа.

Результат работы программы

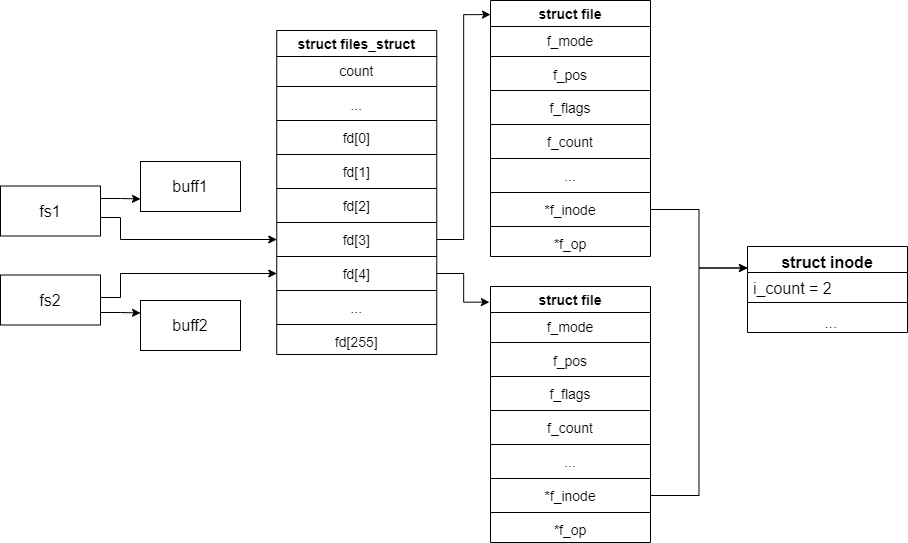
Закрывается сначала первый файл.

bdfhjlnprtvxz

Закрывается сначала второй файл

acegikmoqsuwy

Схема связей структур



Файл открывается 2 раза для записи. Создается два дескриптора открытых файлов, следовательно, два независимых f\_pos, но ссылаются на один inode. fopen и fprintf функции stdio– библиотеки буферизуемого ввода/вывода, поэтому сначала информация пишется в буфер.

Из буфера информация пишется в файл в результате трех действий:

1. Когда буфер заполнен;
2. fflush - принудительная запись содержимого в файл;
3. fclose.

В данном примере запись в файл происходит в результате вызова функции fclose. В файл записываются буквы латинского алфавита поочередно через f1 и f2.

При вызове fclose() для f1 буфер для f1 записывается в файл. При вызове fclose() для f2, все содержимое файла перезаписывается содержимым буфера для f2. В итоге происходит утеря данных, в файле окажется только содержимое одного из буферов.

**Третья программа с использованием потока.**

|  |
| --- |
| #include <stdio.h>  #include <sys/stat.h>  #include <pthread.h>  void \*write\_file(void \*arg)  {  int num = (int)arg;  struct stat statbuf;  FILE \*f = fopen("write\_thread.txt", "w");  stat("write\_thread.txt", &statbuf);  printf("fopen file #%d\ninode = %ld, buffsize = %ld blocksize= %ld\n", num, statbuf.st\_ino, statbuf.st\_size, statbuf.st\_blksize);  for (char c = 'a' + num - 1; c <= 'z'; c += 2)  fprintf(f, "%c", c);  fclose(f);  stat("write\_thread.txt", &statbuf);  printf("fclose file #%d inode = %ld, buffsize = %ld blocksize= %ld\n", num, statbuf.st\_ino, statbuf.st\_size, statbuf.st\_blksize);  return 0;  }  int main()  {  pthread\_t thread;  int code = pthread\_create(&thread, NULL, write\_file, 2);  if (code != 0)  {  printf("can't create thread, code = %d\n", code);  return -1;  }  write\_file(1);  pthread\_join(thread, NULL);  return 0;  } |

Листинг 3.2. Третья программа.

