|  |  |
| --- | --- |
| 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»\_

**Отчёт**

**к лабораторной работе № 5**

**По курсу: «Операционные системы»**

|  |  |
| --- | --- |
| **Студент** Русинова Д. Э.  **Группа** ИУ7-65Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватели 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

stdio.h

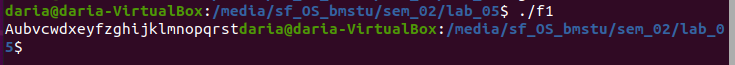
|  |
| --- |
| typedef struct \_\_sFILE {  unsigned char \*\_p; /\* current position in (some) buffer \*/  int \_r; /\* read space left for getc() \*/  int \_w; /\* write space left for putc() \*/  short \_flags; /\* flags, below; this FILE is free if 0 \*/  short \_file; /\* fileno, if Unix descriptor, else -1 \*/  struct \_\_sbuf \_bf; /\* the buffer (at least 1 byte, if !NULL) \*/  int \_lbfsize; /\* 0 or -\_bf.\_size, for inline putc \*/  /\* operations \*/stdio.h  void \*\_cookie; /\* cookie passed to io functions \*/  int (\* \_Nullable \_close)(void \*);  int (\* \_Nullable \_read) (void \*, char \*, int);  fpos\_t (\* \_Nullable \_seek) (void \*, fpos\_t, int);  int (\* \_Nullable \_write)(void \*, const char \*, int);  /\* separate buffer for long sequences of ungetc() \*/  struct \_\_sbuf \_ub; /\* ungetc buffer \*/  struct \_\_sFILEX \*\_extra; /\* additions to FILE to not break ABI \*/  int \_ur; /\* saved \_r when \_r is counting ungetc data \*/  /\* tricks to meet minimum requirements even when malloc() fails \*/  unsigned char \_ubuf[3]; /\* guarantee an ungetc() buffer \*/  unsigned char \_nbuf[1]; /\* guarantee a getc() buffer \*/  /\* separate buffer for fgetln() when line crosses buffer boundary \*/  struct \_\_sbuf \_lb; /\* buffer for fgetln() \*/  /\* Unix stdio files get aligned to block boundaries on fseek() \*/  int \_blksize; /\* stat.st\_blksize (may be != \_bf.\_size) \*/  fpos\_t \_offset; /\* current lseek offset (see WARNING) \*/  } FILE; |

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

**Первая программа:**

|  |
| --- |
| #include <stdio.h>  #include <fcntl.h>  int main()  {      int fd = open("alph.txt", O\_RDONLY);      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);      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;  } |

**Результат работы программы**



**Анализ работы программы**

С помощью системного вызова open() создается дескриптор открытого файла. Системный вызов open() возвращает индекс в массиве fd структуры files\_struct.

Вызов fdopen() создает структуры типа FILE(fs1 и fs2), которые ссылаются на дескриптор открытого файла, созданный системным вызовом open.

Для дескрипторов fs1 и fs2 помощью setbuf задаём соответствующие буферы и тип буферизации \_IOFBF (INPUT OUTPUT FULL).

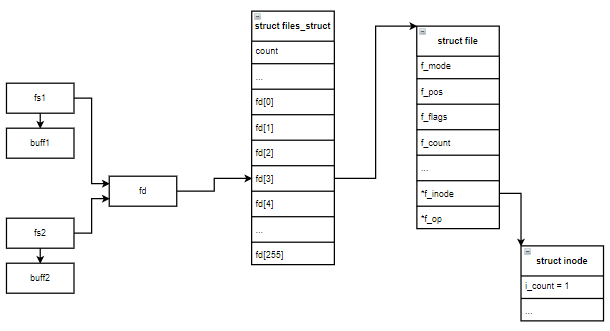
Выполняем в цикле fscanf() для fs1 и fs2 поочередно.

Так как установлен тип буфферизации \_IOFBF (полная), то буфер будет заполнен при первом вызове fscanf(), указатель 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 (с a по t включительно). При первом вызове fscanf(fs2,"%c",&c), в буфер buff2 считываются оставшиеся в файле символы – (c u по z). При дальнейших вызовах

В цикле 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.

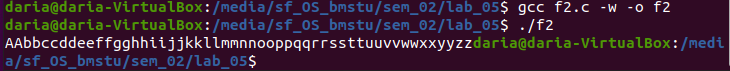
**Схема связей структур**



**Вторая программа (без доп потоков)**

|  |
| --- |
| #include <fcntl.h>  int main()  {        char c;        int fd1 = open("alph.txt", O\_RDONLY);        int fd2 = open("alph.txt", O\_RDONLY);          while (read(fd1, &c, 1) == 1 && read(fd2, &c, 1) == 1)        {              write(1, &c, 1);              write(1, &c, 1);        }        return 0;  } |

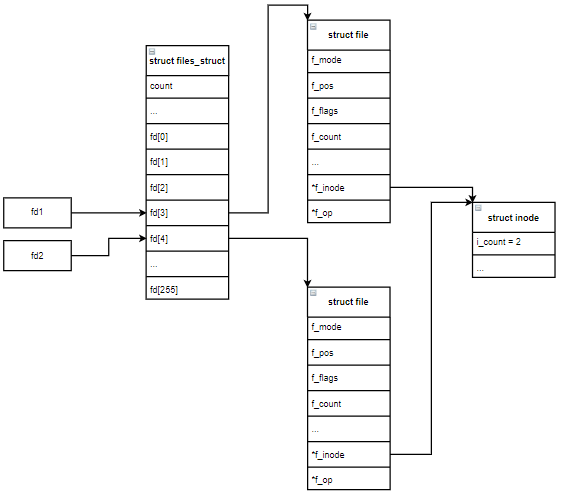
**Результат работы программы:**



**Анализ работы программы:**

Один и тот же файл "alph.txt" открыт 2 раза для чтения. При вызове open() создается дескриптор открытого файла в системной таблице открытых файлов и дескриптор файла в таблице файлов, открытых процессом. В данной программе после вызова open() в таблице открытых файлов будет 2 дескриптора, каждый из которых содержит собственный указатель f\_pos. При вызове read() для дескрипторов их указатели f\_pos сдвигаются по файлу независимо от указателя другого дескриптора, таким образом каждый символ считывается и выводится по два раза. Двум дескрипторам открытого файла соответствует один inode.

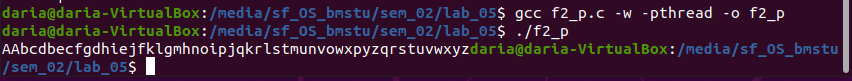
**Cхема связей структур:**



**Вторая программа (с доп потоками)**

|  |
| --- |
| #include <fcntl.h>  #include <pthread.h>  #include <stdio.h>  #include <stdlib.h>  void \*read\_file(int \*fd)  {      char c;      while (read(\*fd, &c, 1) == 1)      {          write(1, &c, 1);      }      sleep(rand() % 2);  }  int main()  {      int fd1 = open("alph.txt", O\_RDONLY);      int fd2 = open("alph.txt", O\_RDONLY);  pthread\_t thread1;      pthread\_t thread2;      int status1 = pthread\_create(&thread1, NULL, read\_file, &fd1);      if (status1 != 0)      {          printf("Error: can't create thread 1\n");          return -1;      }      int status2 = pthread\_create(&thread2, NULL, read\_file, &fd2);      if (status2 != 0)      {          printf("Error: can't create thread 2\n", status2);          return -1;      }      pthread\_join(thread1, NULL);      pthread\_join(thread2, NULL);      return 0;  } |

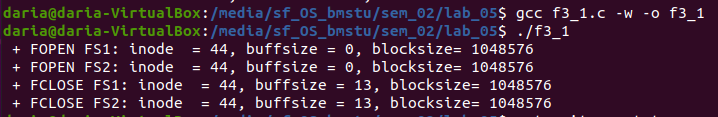
**Результат работы программы:**



**Третья программа (без доп потоков):**

|  |
| --- |
| #include <stdio.h>  #include <sys/stat.h>  #include <errno.h>  int main()  {      struct stat statbuf;      FILE \*fs1 = fopen("writeres.txt", "w");      stat("writeres.txt", &statbuf);      printf(" + FOPEN FS1: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      FILE \*fs2 = fopen("writeres.txt", "w");      stat("writeres.txt", &statbuf);      printf(" + FOPEN FS2: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      for (char c = 'a'; c <= 'z'; c++)      {          if (c % 2)              fprintf(fs1, "%c", c);          else              fprintf(fs2, "%c", c);      }      fclose(fs1);      stat("writeres.txt", &statbuf);      printf(" + FCLOSE FS1: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      fclose(fs2);      stat("writeres.txt", &statbuf);      printf(" + FCLOSE FS2: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      return 0;  } |

**Результат работы программы:**

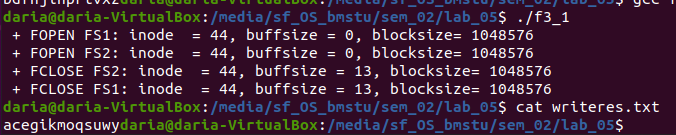




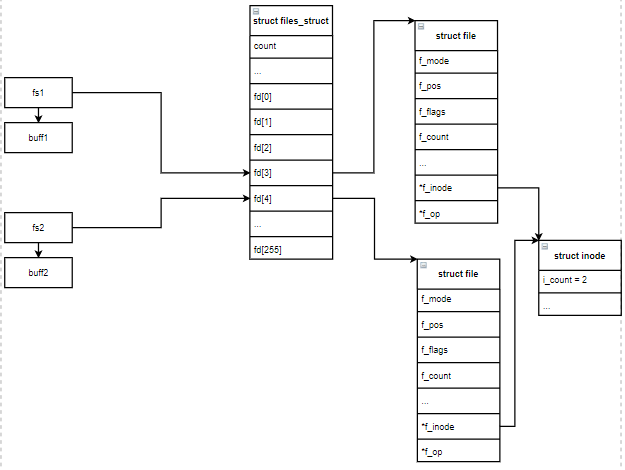
Другой порядок вызовов fclose()

|  |
| --- |
| #include <sys/stat.h>  #include <errno.h>  int main()  {      struct stat statbuf;      FILE \*fs1 = fopen("writeres.txt", "w");      stat("writeres.txt", &statbuf);      printf(" + FOPEN FS1: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      FILE \*fs2 = fopen("writeres.txt", "w");      stat("writeres.txt", &statbuf);      printf(" + FOPEN FS2: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      for (char c = 'a'; c <= 'z'; c++)      {          if (c % 2)              fprintf(fs1, "%c", c); //aceg...          else              fprintf(fs2, "%c", c); //bdfh...      }      fclose(fs2);      stat("writeres.txt", &statbuf);      printf(" + FCLOSE FS2: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      fclose(fs1);      stat("writeres.txt", &statbuf);      printf(" + FCLOSE FS1: inode  = %ld, buffsize = %ld, blocksize= %ld\n",          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      return 0;  } |

**Результат работы программы:**



**Схема связей структур:**



**Анализ работы программы:**

При вызове fopen для записи создается новый файл. Создается два дескриптора открытых файлов, каждый из которых содержит позицию f\_pos, не зависящую от второго дескриптора. Два дескриптора открытых файлов соответствуют одному и тому же inode.

fprintf – функция буферизованного ввода/вывода, т.е. сначала данные записываются в буфер. Данные выводятся из буфера в файл одном из трех случаев:

1. Буфер заполнен.
2. Вызов fflush - принудительная запись содержимого в файл.
3. Вызов fclose.

Если сначала вызывается fclose(fs1), а затем fclose(fs2):

* При вызове fclose() для fs1 буфер для fs1 записывается в файл. При вызове fclose() для fs2, все содержимое файла очищается, а в файл записывается содержимое буфера для fs2. **Произошла утеря данных fs1, в файле окажется только содержимое буфера для fs2.**

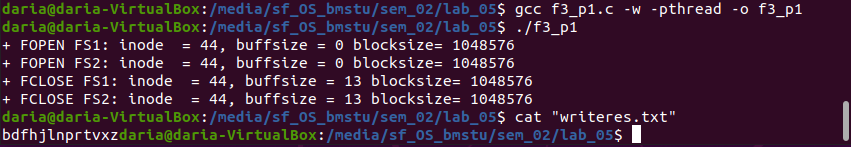
Если сначала вызывается fclose(fs2), а затем fclose(fs1):

* При вызове fclose() для fs2 буфер для fs2 записывается в файл. При вызове fclose() для fs1, все содержимое файла очищается, а в файл записывается содержимое буфера для fs1. **Произошла утеря данных fs2, в файле окажется только содержимое буфера для fs1.**

**Третья программа с использованием потоков**

|  |
| --- |
| #include <stdio.h>  #include <sys/stat.h>  #include <pthread.h>  void \*write\_to\_file(int \*filenum)  {      struct stat statbuf;      FILE \*fs = fopen("writeres.txt", "w");      stat("writeres.txt", &statbuf);      printf("+ FOPEN FS%d: inode  = %ld, buffsize = %ld blocksize= %ld\n",  \*filenum,          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      for (char c = 'a'; c <= 'z'; c++)      {          if (c % 2 && \*filenum == 1) //acegi...              fprintf(fs, "%c", c);          if (!(c % 2) && \*filenum == 2) //bdfh...              fprintf(fs, "%c", c);      }      fclose(fs);      stat("writeres.txt", &statbuf);      printf("+ FCLOSE FS%d: inode  = %ld, buffsize = %ld blocksize= %ld\n",  \*filenum,          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);  }  int main()  {      pthread\_t thread1;      pthread\_t thread2;      int num1 = 1;      int num2 = 2;      if (pthread\_create(&thread1, NULL, write\_to\_file, &num1) != 0)      {          printf("Error: can't create thread 1\n");          return -1;      }        if (pthread\_create(&thread2, NULL, write\_to\_file, &num2) != 0)      {          printf("Error: can't create thread 2\n");          return -1;      }      pthread\_join(thread1, NULL);      pthread\_join(thread2, NULL);      return 0;  } |

**Демонстрация работы программы:**



Сначала создается поток для fs2, затем для fs1.

|  |
| --- |
| #include <stdio.h>  #include <sys/stat.h>  #include <pthread.h>  void \*write\_to\_file(int \*filenum)  {      struct stat statbuf;      FILE \*fs = fopen("writeres.txt", "w");      stat("writeres.txt", &statbuf);      printf("+ FOPEN FS%d: inode  = %ld, buffsize = %ld blocksize= %ld\n", \*filenum,          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);      for (char c = 'a'; c <= 'z'; c++)      {          if (c % 2 && \*filenum == 1) //acegi...              fprintf(fs, "%c", c);          if (!(c % 2) && \*filenum == 2) //bdfh...              fprintf(fs, "%c", c);      }      fclose(fs);      stat("writeres.txt", &statbuf);      printf("+ FCLOSE FS%d: inode  = %ld, buffsize = %ld blocksize= %ld\n", \*filenum,          (long int)statbuf.st\_ino,          (long int)statbuf.st\_size,          (long int)statbuf.st\_blksize);  }  int main()  {      pthread\_t thread1;      pthread\_t thread2;      int num1 = 1;      int num2 = 2;      if (pthread\_create(&thread2, NULL, write\_to\_file, &num2) != 0)      {          printf("Error: can't create thread 2\n");          return -1;      }      if (pthread\_create(&thread1, NULL, write\_to\_file, &num1) != 0)      {          printf("Error: can't create thread 1\n");          return -1;      }      pthread\_join(thread1, NULL);      pthread\_join(thread2, NULL);      return 0;  } |

Результат работы программы:

