|  |  |
| --- | --- |
| 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** Прохорова Л. А.  **Группа** ИУ7-63Б  **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 (возможно не та, но это взято из CLion)

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; |

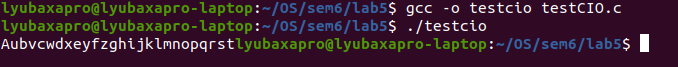
**Проанализировать работу приведенных программ и объяснить результаты их работы.**

**Первая программа:**

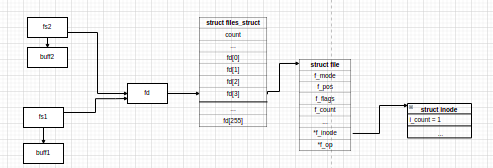
В листинге 1 представлен код первой программы.

|  |
| --- |
| Листинг 1  //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  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;  } |

**Результат работы программы**

****

**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 буфферизация, 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** (переписано без использования break)

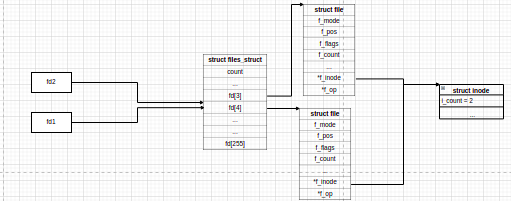
В листинге 2.1 представлен код второй программы.

|  |
| --- |
| Листинг 2.1  //testKernelIO.c  #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 fs1 & fd2  while(read(fd1,&c,1) == 1 && read(fd2,&c,1) == 1)  {  write(1,&c,1);  write(1,&c,1);  }    return 0;  } |

**Результат работы программы:**



**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. Так как в данном случае файл открывается 2 раза, то 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 1 и тот же.

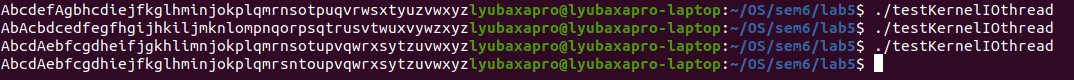
**Вторая программа с использованием двух потоков:**

В листинге 2.2 представлен код второй программы, переписанный с использованием потоков.

|  |
| --- |
| Листинг 2.2  //testKernelThreadIO.c  #include <fcntl.h>  #include <pthread.h>  #include <stdio.h>  struct file\_desc{  int fd;  };  typedef struct file\_desc file\_desc\_t;  void\* read\_file(void \*arg){  char c;  file\_desc\_t\* fd = (file\_desc\_t\*) arg;  while(read(fd->fd, &c, 1) == 1){  write(1, &c, 1);  }  }  int main()  {  pthread\_t thread1;  pthread\_t thread2;  // have kernel open two connection to file alphabet.txt  file\_desc\_t fd1;  fd1.fd = open("alphabet.txt",O\_RDONLY);  file\_desc\_t fd2;  fd2.fd = open("alphabet.txt",O\_RDONLY);    int status1 = pthread\_create(&thread1, NULL, read\_file, &fd1);  if (status1 != 0) {  printf("main error: can't create thread, status = %d\n", status1);  return -1;  }    int status2 = pthread\_create(&thread2, NULL, read\_file, &fd2);  if (status2 != 0) {  printf("main error: can't create thread, status = %d\n", status2);  return -1;  }  pthread\_join(thread1, NULL);  pthread\_join(thread2, NULL);  return 0;  } |

**Результат работы программы:**





Из работы программы видно, что потоки параллельно обрабатывают файл и выводят каждый раз в разном порядке.

**Третья программа:**

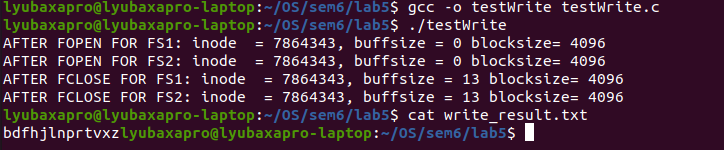
Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции fopen(). Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции fprintf() то первый дескриптор, то – второй.

Результат прокомментировать.

В листингах 3.1 и 3.2 представлен код третьей программы, написанной без использования потоков.

|  |
| --- |
| Листинг 3.1  #include <stdio.h>  #include <sys/stat.h>  #include <errno.h>  int main()  {  struct stat statbuf;  FILE \*fs1 = fopen("write\_result.txt", "w");  stat ("write\_result.txt", &statbuf);  printf("AFTER FOPEN FOR 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("write\_result.txt", "w");  stat ("write\_result.txt", &statbuf);  printf("AFTER FOPEN FOR 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 ("write\_result.txt", &statbuf);  printf("AFTER FCLOSE FOR 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 ("write\_result.txt", &statbuf);  printf("AFTER FCLOSE FOR 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;  } |

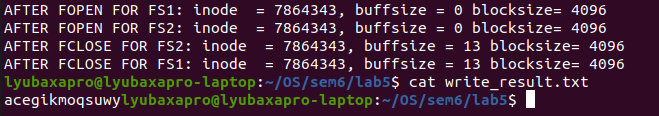
Результат работы программы:



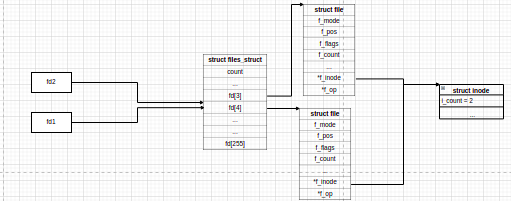
Поменяем порядок вызовов fclose().

|  |
| --- |
| Листинг 3.2  #include <stdio.h>  #include <sys/stat.h>  #include <errno.h>  int main()  {  struct stat statbuf;  FILE \*fs1 = fopen("write\_result.txt", "w");  stat ("write\_result.txt", &statbuf);  printf("AFTER FOPEN FOR 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("write\_result.txt", "w");  stat ("write\_result.txt", &statbuf);  printf("AFTER FOPEN FOR 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(fs2);  stat ("write\_result.txt", &statbuf);  printf("AFTER FCLOSE FOR 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 ("write\_result.txt", &statbuf);  printf("AFTER FCLOSE FOR 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:**

****

Если используем опцию w в fopen, то создаём новый файл. Файл открывается 2 раза для записи.Создается два дескриптора открытых файлов, две независимые позиции, но inode один и тот же. Так как используется fopen и fprintf, то ввод/вывод буферизованный. Буфер создается автоматически. Сначала информация пишется в буфер. Из буфера информация переписывается в результате трех действий:

1. Информация из буфера записывается в файл когда буфер заполнен. В этом случае содержимое буфера автоматически переписывается в файл.
2. Если вызван fflush - принудительная запись содержимого в файл.
3. Если вызван fclose.

В моём случае запись в файл происходит в результате вызова функции fclose.В цикле записываются в файл буквы латинского алфавита поочередно передавая функции fprintf() то первый дескриптор, то – второй.

В случае когда 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. В итоге произошла утеря данных, в файле окажется только содержимое буфера для fs2.

В случае когда вызывается fclose(fs2), а затем fclose(fs1):

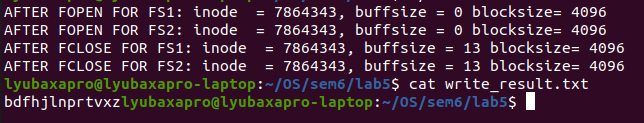
При вызове fclose() для fs2 буфер для fs2 записывается в файл. При вызове fclose() для fs1, все содержимое файла очищается, а в файл записывается содержимое буфера для fs1. В итоге произошла утеря данных, в файле окажется только содержимое буфера для fs1.

**Третья программа с использованием потоков**

Сначала создаём первый поток, потом второй.

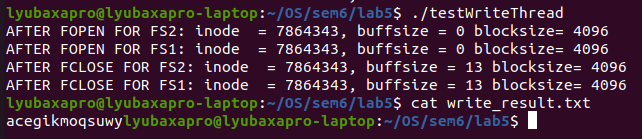
|  |
| --- |
| Листинг 3.1  #include <stdio.h>  #include <sys/stat.h>  #include <pthread.h>  struct file\_s{  int num;  };  typedef struct file\_s file\_num;  void\* write\_to\_file(void \*arg){  struct stat statbuf;  file\_num \*f\_num = (file\_num\*) arg;  FILE\* fs = fopen("write\_result.txt", "w");  stat ("write\_result", &statbuf);  stat ("write\_result.txt", &statbuf);  printf("AFTER FOPEN FOR FS%d: inode = %ld, buffsize = %ld blocksize= %ld\n",f\_num->num, (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 && f\_num->num == 1)  fprintf(fs, "%c", c);  if (!(c % 2) && f\_num->num == 2)  fprintf(fs, "%c", c);  }  fclose(fs);  stat ("write\_result.txt", &statbuf);  printf("AFTER FCLOSE FOR FS%d: inode = %ld, buffsize = %ld blocksize= %ld\n",f\_num->num, (long int)statbuf.st\_ino, (long int)statbuf.st\_size,  (long int)statbuf.st\_blksize);  }  int main()  {  pthread\_t thread1;  pthread\_t thread2;  file\_num f1;  file\_num f2;  f1.num = 1;  f2.num = 2;  int status1 = pthread\_create(&thread1, NULL, write\_to\_file, &f1);  if (status1 != 0) {  printf("main error: can't create thread, status = %d\n", status1);  return -1;  }  int status2 = pthread\_create(&thread2, NULL, write\_to\_file, &f2);  if (status2 != 0) {  printf("main error: can't create thread, status = %d\n", status2);  return -1;  }    pthread\_join(thread1, NULL);  pthread\_join(thread2, NULL);    return 0;  } |

**Демонстрация работы программы:**

****

Сначала создаём второй поток, потом первый.

|  |
| --- |
| Листинг 3.2  #include <stdio.h>  #include <sys/stat.h>  #include <pthread.h>  struct file\_s{  int num;  };  typedef struct file\_s file\_num;  void\* write\_to\_file(void \*arg){  struct stat statbuf;  file\_num \*f\_num = (file\_num\*) arg;  FILE\* fs = fopen("write\_result.txt", "w");  stat ("write\_result", &statbuf);  stat ("write\_result.txt", &statbuf);  printf("AFTER FOPEN FOR FS%d: inode = %ld, buffsize = %ld blocksize= %ld\n",f\_num->num, (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 && f\_num->num == 1)  fprintf(fs, "%c", c);  if (!(c % 2) && f\_num->num == 2)  fprintf(fs, "%c", c);  }  fclose(fs);  stat ("write\_result.txt", &statbuf);  printf("AFTER FCLOSE FOR FS%d: inode = %ld, buffsize = %ld blocksize= %ld\n",f\_num->num, (long int)statbuf.st\_ino, (long int)statbuf.st\_size,  (long int)statbuf.st\_blksize);  }  int main()  {  pthread\_t thread1;  pthread\_t thread2;  file\_num f1;  file\_num f2;  f1.num = 1;  f2.num = 2;  int status2 = pthread\_create(&thread2, NULL, write\_to\_file, &f2);  if (status2 != 0) {  printf("main error: can't create thread, status = %d\n", status2);  return -1;  }  int status1 = pthread\_create(&thread1, NULL, write\_to\_file, &f1);  if (status1 != 0) {  printf("main error: can't create thread, status = %d\n", status1);  return -1;  }    pthread\_join(thread1, NULL);  pthread\_join(thread2, NULL);    return 0;  } |

****

В данных примерах прослеживается ситуация, аналогичная с работой без потоков.