|  |  |
| --- | --- |
| **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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 5 |

**Дисциплина:** Операционные системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-62Б |  |  | В.А. Иванов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Н.Ю. Рязанова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2021

Структура FILE ( \_IO\_FILE)

|  |
| --- |
| struct \_IO\_FILE;  /\* The opaque type of streams. This is the definition used elsewhere. \*/  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;  int \_flags2;  \_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;  \_IO\_off64\_t \_offset;  void \*\_\_pad1;  void \*\_\_pad2;  void \*\_\_pad3;  void \*\_\_pad4;  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)];  }; |

**Анализ структуры**

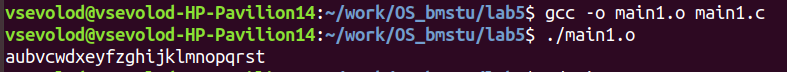
Структура содержит указатели char\* на начало, конец и текущую позицию буфера ввода, вывода и неактивного буфера. Поле \_fileno хранит int дескриптор открытого файла. Также содержатся поля флагов и прав доступа.

**Задание 1**

Листинг программы (однопоточная версия):

|  |
| --- |
| #include <stdio.h>  #include <fcntl.h>  #define FNAME "alph.txt"  int main()  {  int fd = open(FNAME,O\_RDONLY);  if (fd == -1)  {  printf("Open failed\n");  return -1;  }  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 = 1;  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);  }  printf("\n");  return 0;  } |

Результат работы:

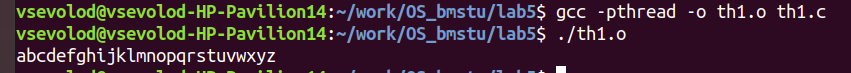


Буфер структуры fs1 при первом вызове fscanf заполняется символами “ab...t“, буфер fs2 заполняется оставшимся содержимым файла: “uv...z“. В каждой итерации цикла с помощью fscanf производится чтение из буфера fs1 и fs2 и запись в stdout с помощью fprintf. Поэтому в начале работы программы символы из двух буферов идут вперемешку. После считывания “z“ из fs2 символы остаются только в fs1, поэтому на всех оставшихся итерациях выводятся симолы только из него.

Листинг программы (многопоточная версия):

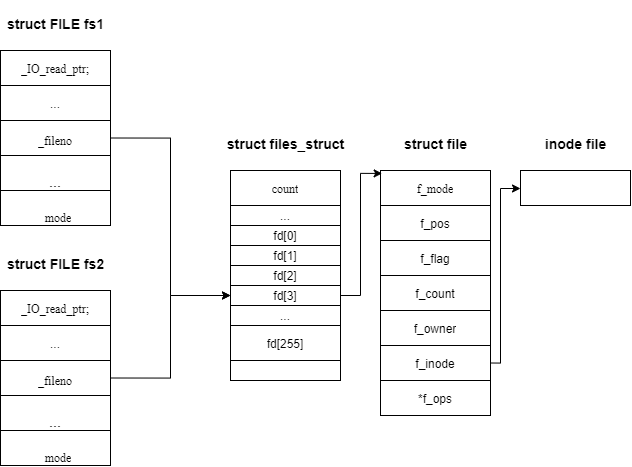
|  |
| --- |
| #include <stdio.h>  #include <fcntl.h>  #include <pthread.h>  #define FNAME "alph.txt"  void\* thread\_f(void \*data)  {  int fd = \*((int\*)data);  FILE \*fs2 = fdopen(fd,"r");  char buff2[20];  setvbuf(fs2,buff2,\_IOFBF,20);  int flag = 1;  while (flag == 1)  {  char c;  flag = fscanf(fs2,"%c",&c);  if (flag == 1)  fprintf(stdout,"%c",c);  }  }  int main()  {  int fd = open(FNAME,O\_RDONLY);  if (fd == -1)  {  printf("Open failed\n");  return -1;  }  pthread\_t tid;  int err = pthread\_create(&tid, NULL, thread\_f, (void\*)(&fd));  if (err)  {  printf("It's imposible to create a thread");  return -1;  }  FILE \*fs1 = fdopen(fd,"r");  char buff1[20];  setvbuf(fs1,buff1,\_IOFBF,20);  int flag1 = 1;  while(flag1 == 1)  {  char c;  flag1 = fscanf(fs1,"%c",&c);  if (flag1 == 1)  fprintf(stdout,"%c",c);  }  err = pthread\_join(tid, NULL);  if (err)  {  printf("It's imposible to join the thread");  return -1;  }  printf("\n");  return 0;  } |

Результат работы:



В отличии от однопоточной версии, здесь содержимые буферов идут один за другим. Сначала считываются символы “ab...t“ из fs1 в главном потоке, после чего “uv...z“ из f2 в другом потоке. Последовательность в работе потоков связана с тем, что один из них был действующим в то время, как другой находился в состоянии сна.

Диаграмма дескрипторов:

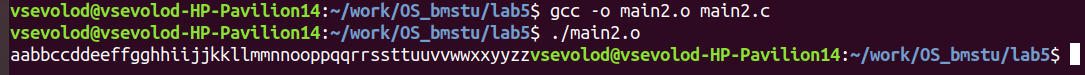


Задание 2

Листинг программы (однопоточная версия):

|  |
| --- |
| #include <fcntl.h>  #include <unistd.h>  #include <stdio.h>  #define FNAME "alph.txt"  int main()  {  char c1, c2;  int fd1 = open(FNAME,O\_RDONLY);  int fd2 = open(FNAME,O\_RDONLY);    if (fd1 == -1 || fd2 == -1)  {  printf("Open failed\n");  return -1;  }  int flag1 = 1, flag2 = 1;  while(flag1 && flag2)  {  flag1 = (read(fd1,&c1,1) == 1);  flag2 = (read(fd2,&c2,1) == 1);  if (flag1) write(1,&c1,1);  if (flag2) write(1,&c2,1);  }  return 0;  } |

Результат работы:

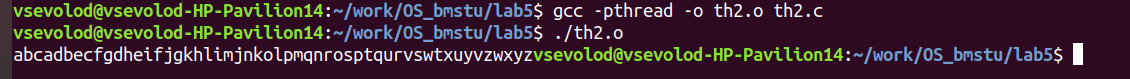


В данном случае с помощью двух вызовов open создаётся два разных дескриптора открытого файла "alph.txt". Поэтому read из fd1 не оказывает влияния на read из fd2, т. к. они изменяют разный f\_pos. Следователно, fd1 и fd2 каждую итерацию считывают один и тот же символ из файла и выводят его с помощью write в fd[1] (т. е. stdout).

Листинг программы (многопоточная версия):

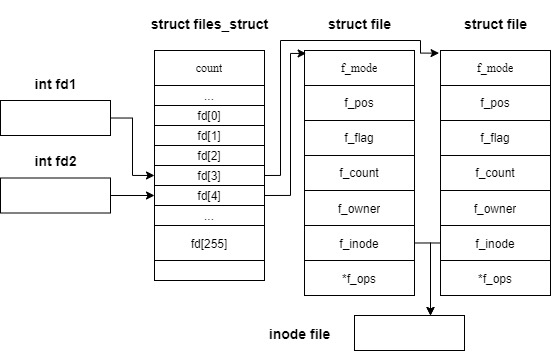
|  |
| --- |
| #include <fcntl.h>  #include <unistd.h>  #include <pthread.h>  #include <stdio.h>  #define FNAME "alph.txt"  void\* thread\_f (void \*data)  {  int fd2 = open(FNAME,O\_RDONLY);  if (fd2 == -1)  return (void\*)-1;  char c2;  int flag2 = 1;  while(flag2)  {  flag2 = (read(fd2,&c2,1) == 1);  if (flag2) write(1,&c2,1);  }  return (void\*)0;  }  int main()  {  char c1;    pthread\_t tid;  int err = pthread\_create(&tid, NULL, thread\_f, NULL);  if (err)  {  printf("It's imposible to create a thread");  return -1;  }  int fd1 = open(FNAME,O\_RDONLY);  if (fd1 == -1)  {  printf("Open failed\n");  return -1;  }  int flag1 = 1;  while(flag1)  {  flag1 = (read(fd1,&c1,1) == 1);  if (flag1) write(1,&c1,1);  }  int thread\_code;  err = pthread\_join(tid, (void\*\*)(&thread\_code));  if (err)  {  printf("It's imposible to join the thread");  return -1;  }  if (thread\_code == -1)  {  printf("Open failed (in thread)\n");  return -1;  }  return 0;  } |

Результат работы:



В данном случае, read из fd1 и fd2 происходит параллельно, поэтому символы, считанные в разных потоках выводятся вперемешку, но не стого по очереди, как в однопоточной реализации. Один из потоков в начале считывает 3 символа до того, как начинает чтение другой, поэтому возникает смещение в считываемых символах. Также существуют моменты, когда один из потоков успевает сделать два чтения до того, как это сделает другой поток.

Диаграмма дескрипторов:

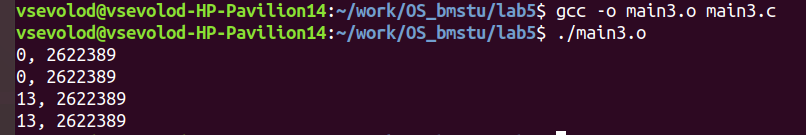


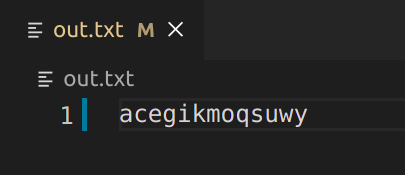
Задание 3

Листинг программы (однопоточная версия):

|  |
| --- |
| #include <fcntl.h>  #include <unistd.h>  #include <stdio.h>  #include <sys/stat.h>  #define FNAME "out.txt"  int main()  {  struct stat statbuf;  FILE\* fd1 = fopen(FNAME, "w");  if (!fd1)  {  printf("Fopen failed\n");  return -1;  }  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  FILE\* fd2 = fopen(FNAME, "w");  if (!fd2)  {  printf("Fopen failed\n");  fclose(fd1);  return -1;  }  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  for (int i=0; i<26; i++)  {  if (i % 2)  fprintf(fd2, "%c", 'a' + i);  else  fprintf(fd1, "%c", 'a' + i);  }  fclose(fd2);  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  fclose(fd1);  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  return 0;  } |

Результат работы:



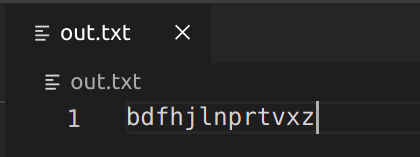
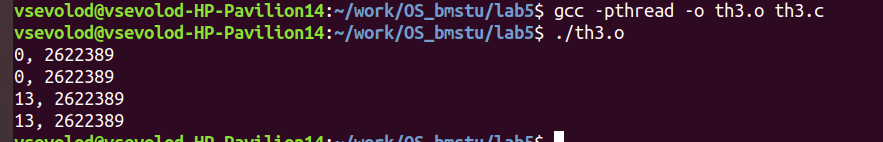


Как видно по содержимому файла, сохранился результат записи только одного из дескрипторов (для которого fclose был вызван последним). При fopen происходит обнуление размера файла, его inode не изменяется. fclose также не меняет размера файла, изменяется только размер (на величину равную количеству символов в буфере).

Листинг программы (многопоточная версия):

|  |
| --- |
| #include <fcntl.h>  #include <unistd.h>  #include <stdio.h>  #include <sys/stat.h>  #include <pthread.h>  #define FNAME "out.txt"  void\* thread\_f(void \*data)  {  struct stat statbuf;  FILE\* fd2 = fopen(FNAME, "w");  if (!fd2)  {  printf("Fopen failed\n");  return (void\*)-1;  }  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  for (int i=1; i<26; i+=2)  fprintf(fd2, "%c", 'a' + i);    fclose(fd2);  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  return (void\*)0;  }  int main()  {  pthread\_t tid;  int err = pthread\_create(&tid, NULL, thread\_f, NULL);  if (err)  {  printf("It's imposible to create a thread");  return -1;  }  ///  struct stat statbuf;  FILE\* fd1 = fopen(FNAME, "w");  if (!fd1)  {  printf("Fopen failed\n");  return -1;  }  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  for (int i=0; i<26; i+=2)  fprintf(fd1, "%c", 'a' + i);    fclose(fd1);  stat(FNAME, &statbuf);  printf("%ld, %lu\n", statbuf.st\_size, statbuf.st\_ino);  ///  int thread\_code;  err = pthread\_join(tid, (void\*\*)(&thread\_code));  if (err)  {  printf("It's imposible to join the thread");  return -1;  }  if (thread\_code == -1)  {  printf("Open failed (in thread)\n");  return -1;  }  return 0;  } |

Результат работы:



В многопоточной версии наблюдается такое же изменение inode и размера файла, как было описано выше. Содержимое файла зависит от того, какой из потоков сделает fclose первым, что неоднозначно, поэтому в результирующем файле оказываются как чётные, так и нечётные символы.

Диаграмма дескрипторов:

