Структура FILE:

|  |
| --- |
| typedef \_IO\_FILE FILE;  struct \_IO\_FILE  {    int \_flags;  /\* High-order word is \_IO\_MAGIC; rest is flags. \*/    /\* The following pointers correspond to the C++ streambuf protocol. \*/    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;    \_\_off\_t \_old\_offset; /\* This used to be \_offset but it's too small.  \*/    /\* 1+column number of pbase(); 0 is unknown. \*/    unsigned short \_cur\_column;    signed char \_vtable\_offset;    char \_shortbuf[1];    \_IO\_lock\_t \*\_lock;  #ifdef \_IO\_USE\_OLD\_IO\_FILE  };  struct \_IO\_FILE\_complete  {    struct \_IO\_FILE \_file;  #endif    \_\_off64\_t \_offset;    /\* Wide character stream stuff.  \*/    struct \_IO\_codecvt \*\_codecvt;    struct \_IO\_wide\_data \*\_wide\_data;    struct \_IO\_FILE \*\_freeres\_list;    void \*\_freeres\_buf;    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)];  }; |

Программа 1.c (попеременное чтение в главном потоке):

|  |
| --- |
| #include <stdio.h>  #include <fcntl.h>  int main()  {      int fd = open("alphabet.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 выполнения:



1. Функция open() своим именем возвращает значение типа int – файловый дескриптор, который является индексом в таблице дескрипторов открытых файлов текущего процесса.
2. Далее в результате вызова функции fdopen() был открыт поток (stream) fs1 в режиме “r” и связан с дескриптором файла fd. Затем в результате вызова функции setvbuf() был изменён тип буферизации потока fs1 на полную (блочную) буферизацию, с использованием буфера пространства пользователя buff1 размером 20 символов. В результате аналогичных действий был открыт поток fs2 и связан с fd, а также был изменён его тип буферизации на полную, с использованием буфера пространства пользователя buff2 размером 20 символов.
3. Таким образом, в результате проделанных действий, при первом считывании одного символа из потока fs1, в буфер buff1 было записано 20 символов. Аналогично в буфер buff2 были записаны оставшиеся 6 символов. В переменную c попеременно считывались символы из buff1 и buff2 и записывались в поток stdout, пока не были считаны все символы из каждого буфера.

В приведённой программе демонстрируется проблема чтения данных не в той последовательности, в какой предполагалось. Эта проблема возникла из-за применения блочной буферизации при попеременном чтении из двух потоков (stream). Для получения алфавита в правильной последовательности можно изменить тип буферизации на \_IONBF.

Программа 1\_th.c – попеременное чтение в двух рабочих потоках:

|  |
| --- |
| #include <pthread.h>  #include <fcntl.h>  #include <stdio.h>  int fd;  void \*th\_work()  {      FILE \*fs = fdopen(fd, "r");      char buf[20];      setvbuf(fs, buf, \_IOFBF, 20);      char c;      while((fscanf(fs, "%c", &c)) == 1)          fprintf(stdout, "%c", c);  }  int main()  {      fd = open("alphabet.txt", O\_RDONLY);      pthread\_t th1\_id, th2\_id;      pthread\_create(&th1\_id, NULL, th\_work, NULL);      pthread\_create(&th2\_id, NULL, th\_work, NULL);      pthread\_join(th1\_id, NULL);      pthread\_join(th2\_id, NULL);  } |

Результат выполнения:



В приведённой модификации программы 1.c происходит попеременное чтение из двух потоков (stream), в двух рабочих потоках (thread). В многопроцессорной системе порядок одновременной записи из двух рабочих потоков в один поток вывода не определён, поэтому алфавит выводится в stdout по-разному для последовательных запусков одной программы.

На рисунке 1 изображена связь дескрипторов со структурами в программе 1:

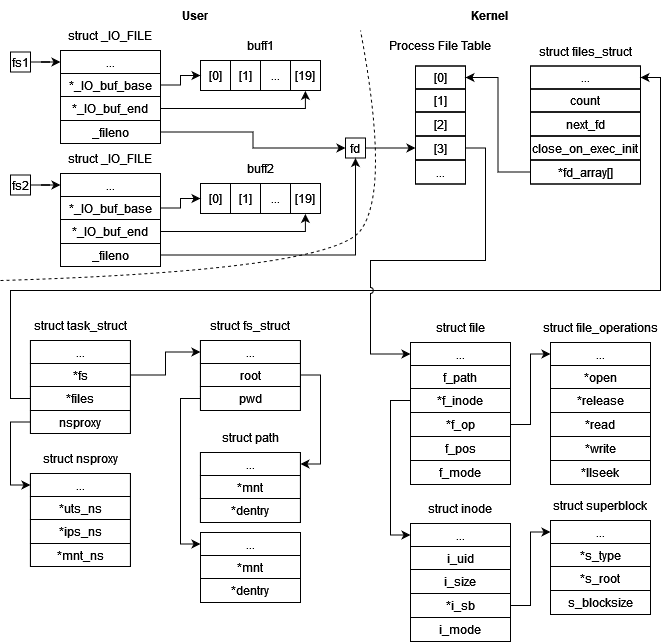


Рисунок – Схема связи файловых дескрипторов со структурами в программе 1

Программа 2.c – Попеременные чтение и запись в главном потоке:

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

Результат выполнения:



В результате двух последовательных системных вызовов open() были созданы два дескриптора открытых файлов в таблице открытых файлов процесса и два дескриптора открытых файлов в системной таблице открытых файлов. Переменные fd1 и fd2 получили значения типа int – дескрипторы файлов для первого и второго вызова open() соответственно.

Каждый дескриптор открытого файла имеет своё значение поля fpos, которое инициализируется 0. Поэтому при попеременном чтении из fd1 и fd2 и записи в stdout в цикле алфавит выводится именно таким образом.

Программа 2\_th.c – Чтение и запись без взаимоисключения в двух потоках:

|  |
| --- |
| #include <fcntl.h>  #include <unistd.h>  #include <pthread.h>  void \*th\_work() {      char c;      int fd = open("alphabet.txt", O\_RDONLY);      while (read(fd, &c, 1) == 1 && write(1, &c, 1) == 1);  }  int main() {      pthread\_t th1\_id, th2\_id;      pthread\_create(&th1\_id, NULL, th\_work, NULL);      pthread\_create(&th2\_id, NULL, th\_work, NULL);      pthread\_join(th1\_id, NULL);      pthread\_join(th2\_id, NULL);      return 0;  } |

Результат выполнения:



В результате системного вызова open() в каждом из рабочих потоков был создан свой дескриптор открытого файла в таблице открытых файлов процесса. Своим именем функция open() возвращает в каждом из рабочих потоков int — файловый дескриптор, который является индексом в таблице дескрипторов открытых файлов текущего процесса. Далее в каждом из рабочих потоков в цикле происходит посимвольное чтение из открытого файла с алфавитом и посимвольная запись в stdout.

Каждый дескриптор открытого файла имеет своё значение поля fpos, которое инициализируется 0. Поэтому в каждом из рабочих потоков происходит последовательное чтение алфавита (от A до z). Порядок одновременной записи из нескольких потоков (thread) в stdout не определён, поэтому результат отличается от случая с одним потоком.

Программа 2\_th\_mutex.c – Чтение и запись с использованием мьютекса в двух потоках:

|  |
| --- |
| #include <fcntl.h>  #include <unistd.h>  #include <pthread.h>  pthread\_mutex\_t mu;  void \*th\_work()  {      char c;      int fd = open("alphabet.txt", O\_RDONLY);      pthread\_mutex\_lock(&mu);      while (read(fd, &c, 1) == 1 && write(1, &c, 1) == 1);      pthread\_mutex\_unlock(&mu);  }  int main() {      pthread\_t th1\_id, th2\_id;      pthread\_create(&th1\_id, NULL, th\_work, NULL);      pthread\_create(&th2\_id, NULL, th\_work, NULL);      pthread\_join(th1\_id, NULL);      pthread\_join(th2\_id, NULL);      return 0;  } |

Результат выполнения:



В качестве средства взаимоисключения в этой модификации программы 2 используется мьютекс. Рабочий поток при входе в критическую секцию вызывает pthread\_mutex\_lock() и производит чтение из файла и запись в stdout, после чего вызывает pthread\_mutex\_unlock(). Только после этого другой рабочий поток начинает чтение из открытого файла и запись в stdout. В результате алфавит выводится два раза в правильном порядке.

На рисунке 2 изображена связь дескрипторов со структурами в программе 2:

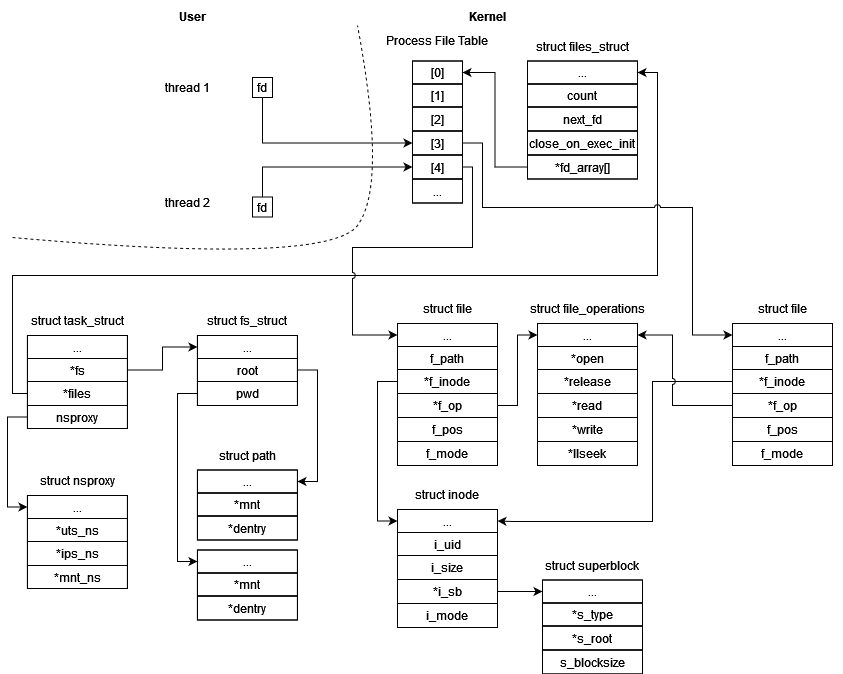


Рисунок 2 – Схема связи файловых дескрипторов со структурами в программе 2

Программа 3\_th.c:

|  |
| --- |
| #include <pthread.h>  #include <stdlib.h>  #include <stdio.h>  #include <sys/stat.h>  struct th\_data {      int symb\_start, symb\_end;  };  void \*th\_work(struct th\_data \*data) {      FILE \*f = fopen("output.txt", "w");      if (f == NULL) {          printf("Error while fopen\n");          exit(1);      }      struct stat s;      int res = stat("output.txt", &s);      if (res == 1)          exit(1);      int pos = ftell(f);      printf("fopen: inode = %ld, size = %ld, pos = %d\n", s.st\_ino, s.st\_size, pos);      for (int i = data->symb\_start; i <= data->symb\_end; i++)          fprintf(f, "%c", i);      pos = ftell(f);      fclose(f);      res = stat("output.txt", &s);      if (res == 1)          exit(1);      printf("fclose: inode = %ld, size = %ld, pos = %d\n", s.st\_ino, s.st\_size, pos);  }  int main() {      pthread\_t th1\_id, th2\_id;      struct th\_data th1\_data = {              .symb\_start = '0',              .symb\_end = '9'      }, th2\_data = {              .symb\_start = 'a',              .symb\_end = 'z'      };      pthread\_create(&th1\_id, NULL, &th\_work, &th1\_data);      pthread\_create(&th2\_id, NULL, &th\_work, &th2\_data);      pthread\_join(th1\_id, NULL);      pthread\_join(th2\_id, NULL);      return 0;  } |

Результат выполнения:

Изображение выглядит как текст

Автоматически созданное описание

В файле output.txt:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



В данной программе каждый из потоков открывает один и тот же файл на запись. Поскольку используется функция fopen(), которая создает экземпляр структуры \_IO\_FILE, речь идет о буферизованном выводе. Для каждого из экземпляров структуры \_IO\_FILE создается свой дескриптора открытого файла, поэтому, для каждого из потоков будет создан свой дескриптор открытого файла в режиме записи. Один поток записывает в файл английский алфавит, другой – цифры от 0 до 9. Итоговый результат зависит от выделения процессором квантов времени потокам. Возможны три ситуации:

1. В файле окажутся записаны только цифры. Такой результат говорит о том, что поток, записывающий буквы успел завершить все операции с файлом до того, как другой поток начал с ним работать, т.е. он вызвал fclose() раньше, чем другой поток вызвал fopen(). В таком случае в результате вызова fopen() записанные буквы будут утеряны, другой поток запишет цифры и закроет файл.

2. В файле начало алфавита окажется затёрто цифрами. Такой результат говорит о том, что оба потока открыли файл на запись до того, как второй вызвал fclose(), и они начали писать в него одновременно.

3. В файле записан только алфавит. Такой результат может быть следствием двух исходов:

* Поток, записывающий цифры, закончил работу с файлом раньше, чем с ним начал работать другой поток.
* Потоки записывали в файл одновременно, но поток, записывающий буквы, вызвал fclose() раньше, чем записывающий цифры, в связи с чем при вызове fclose() была полностью затёрта.

Решить перечисленные выше проблемы можно двумя способами:

1. Открывать файл в режиме O\_APPEND. В таком случае перед каждым вызовом write смещение устанавливается в конец файла.

2. Использовать lseek и мьютексы для обеспечения монопольного доступа к файлам.

Программа 3\_th\_append.c:

|  |
| --- |
| #include <pthread.h>  #include <stdlib.h>  #include <stdio.h>  #include <sys/stat.h>  struct th\_data {      int symb\_start, symb\_end;  };  void \*th\_work(struct th\_data \*data) {      FILE \*f = fopen("output.txt", "a");      if (f == NULL) {          printf("Error while fopen\n");          exit(1);      }      struct stat s;      int res = stat("output.txt", &s);      if (res == 1)          exit(1);      int pos = ftell(f);      printf("fopen: inode = %ld, size = %ld, pos = %d\n", s.st\_ino, s.st\_size, pos);      for (int i = data->symb\_start; i <= data->symb\_end; i++)          fprintf(f, "%c", i);      pos = ftell(f);      fclose(f);      res = stat("output.txt", &s);      if (res == 1)          exit(1);      printf("fclose: inode = %ld, size = %ld, pos = %d\n", s.st\_ino, s.st\_size, pos);  }  int main() {      pthread\_t th1\_id, th2\_id;      struct th\_data th1\_data = {              .symb\_start = '0',              .symb\_end = '9'      }, th2\_data = {              .symb\_start = 'a',              .symb\_end = 'z'      };      pthread\_create(&th1\_id, NULL, &th\_work, &th1\_data);      pthread\_create(&th2\_id, NULL, &th\_work, &th2\_data);      pthread\_join(th1\_id, NULL);      pthread\_join(th2\_id, NULL);      return 0;  } |

Результат выполнения:

Изображение выглядит как текст

Автоматически созданное описание

Файл output.txt:

Изображение выглядит как текст

Автоматически созданное описание

В данной модификации программы 3 файл открывается в режиме O\_APPEND. В результате в output.txt записываются и цифры, и буквы. Хотя потеря данных и не происходит, порядок записи (например, сначала буквы, затем цифры) потоками в файл не определён.

На рисунке 3 изображена связь дескрипторов со структурами в программе 3:

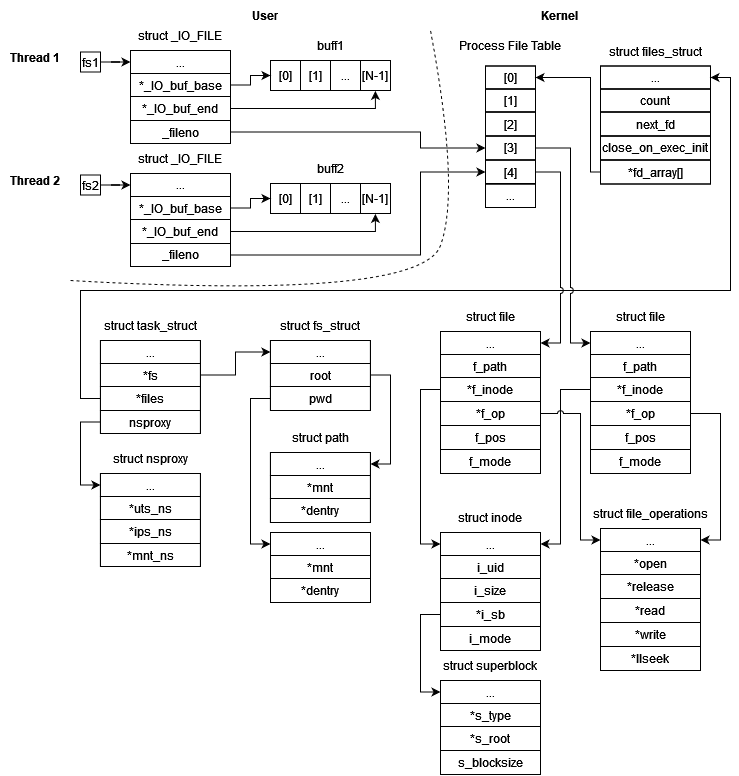


Рисунок 3 – Схема связи файловых дескрипторов со структурами в программе 3