



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»

Отчет по лабораторной работе №6 по курсу "Операционные системы"

Тема Буферизованный и не буферизованный ввод-вывод

Студент Калашков П. А.

Группа ИУ7-66Б

Оценка (баллы) _____

Преподаватели Рязанова Н. Ю.

Структура FILE

Листинг 1 – Структура FILE

```
1     typedef struct _IO_FILE FILE;
2 struct _IO_FILE
3 {
4     int _flags;    /* High-order word is _IO_MAGIC; rest is flags. */
5
6     /* The following pointers correspond to the C++ streambuf
7        protocol. */
7     char *_IO_read_ptr; /* Current read pointer */
8     char *_IO_read_end; /* End of get area. */
9     char *_IO_read_base; /* Start of putback+get area. */
10    char *_IO_write_base; /* Start of put area. */
11    char *_IO_write_ptr; /* Current put pointer. */
12    char *_IO_write_end; /* End of put area. */
13    char *_IO_buf_base; /* Start of reserve area. */
14    char *_IO_buf_end; /* End of reserve area. */
15
16    /* The following fields are used to support backing up and undo.
17       */
17    char *_IO_save_base; /* Pointer to start of non-current get area
18       . */
18    char *_IO_backup_base; /* Pointer to first valid character of
19       backup area */
19    char *_IO_save_end; /* Pointer to end of non-current get area.
20       */
21
21    struct _IO_marker *_markers;
22
23    struct _IO_FILE *_chain;
24
25    int _fileno;
26    int _flags2;
27    __off_t _old_offset; /* This used to be _offset but it's too
28       small. */
```

Листинг 2 – Структура FILE

```
1 /* 1+column number of pbase(); 0 is unknown. */
2 unsigned short _cur_column;
3 signed char _vtable_offset;
4 char _shortbuf[1];
5
6 _IO_lock_t *_lock;
7 #ifdef _IO_USE_OLD_IO_FILE
8 };
```

Программа 1, один поток

Листинг 3 – Программа 1 — один поток— часть 1

```
1 #include <stdio.h>
2 #include <fcntl.h>
3
4 int main()
5 {
6     int fd = open("alphabet.txt", O_RDONLY);
7
8     FILE *fs1 = fdopen(fd, "r");
9     char buff1[20];
10    setvbuf(fs1, buff1, _IOFBF, 20);
11
12    FILE *fs2 = fdopen(fd, "r");
13    char buff2[20];
14    setvbuf(fs2, buff2, _IOFBF, 20);
15
16    int flag1 = 1, flag2 = 2;
17    while(flag1 == 1 || flag2 == 1)
18    {
19        char c;
20        flag1 = fscanf(fs1, "%c", &c);
21        if (flag1 == 1)
22        {
23            fprintf(stdout, "%c", c);
24        }
25        flag2 = fscanf(fs2, "%c", &c);
```

Листинг 4 – Программа 1 — один поток — часть 2

```
1      if (flag2 == 1)
2      {
3          fprintf(stdout, "%c", c);
4      }
5  }
6  printf("\n");
7  return 0;
8  }
```

Результаты работы:



```
paul@paul:~/Desktop/os/open-fopen$ ./prog1.out
aubvcwdxeyfzghijklmnopqrst
```

Рисунок 1 – Результаты работы программы 1 (один поток)

Программа 1, два потока

Листинг 5 – Программа 1 — два потока — часть 1

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4
5 void *thread_routine(void *fd)
6 {
7     int flag = 1;
8     char c;
9
10    FILE *fs = fdopen(*((int *)fd), "r");
11    char buf[20];
12    setvbuf(fs, buf, _IOFBF, 20);
13
14    while (flag == 1)
15    {
16        flag = fscanf(fs, "%c", &c);
17        if (flag == 1)
18        {
19            fprintf(stdout, "%c", c);
20        }
21    }
```

Листинг 6 – Программа 1 — два потока — часть 2

```
1     }
2 }
3
4 int main(void)
5 {
6     int fd = open("alphabet.txt", O_RDONLY);
7
8     FILE *fs = fdopen(fd, "r");
9     char buf[20];
10    setvbuf(fs, buf, _IOFBF, 20);
11
12    pthread_t thr_worker;
13
14    pthread_create(&thr_worker, NULL, thread_routine, &fd);
15
16    int flag = 1;
17    char c;
18    while (flag == 1)
19    {
20        flag = fscanf(fs, "%c", &c);
21        if (flag == 1)
22        {
23            fprintf(stdout, "%c", c);
24        }
25    }
26    pthread_join(thr_worker, NULL);
27    printf("\n");
28    return 0;
29 }
```

Результаты работы:

```
paul@paul:~/Desktop/os/open-fopen$ ./prog1_thread.out
abcdefghijklmnopqrstuvwxy
```

Рисунок 2 – Результаты работы программы 1 (два потока)

Анализ полученного результата

При первом вызове `fscanf(fs1, "%c &c)` в буфер `buff1` будут скопированы первые 20 символов (т.е. `abcdefghijklmnopqrst`), после чего в переменную `c` записывается (и потом выводится) буква `a`.

При первом вызове `fscanf(fs2, "%c &c)` в буфер `buff2` будут скопированы оставшиеся символы (т.е. `vwxyz`), после чего в переменную `c` записывается (и потом выводится) буква `u`.

Внутри цикла будут поочерёдно выводиться символы из буферов `buff1` и `buff2` до тех пор, пока символы в одном из них не закончатся. После этого будут выведены оставшиеся символы из другого буфера.

Системный вызов `open` создаёт дескриптор открытого для чтения файла и возвращает числовой дескриптор, являющийся индексом в массиве `fd` таблицы открытых файлов процесса `struct files_struct`.

Вызов функции `fdopen` создаёт экземпляры структуры `FILE`, которые ссылаются на созданный функцией `open` дескриптор.

Функция `setvbuf` задают буфера для соответствующих дескрипторов `fs1` и `fs2`, а также устанавливают тип буферизации `IOFBF` (полная буферизация).

Связь структур

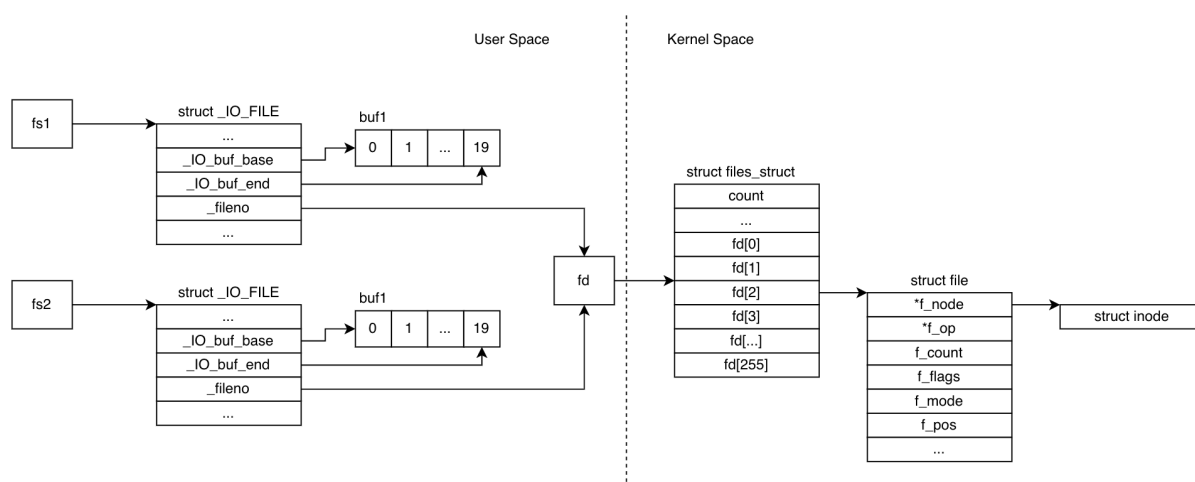


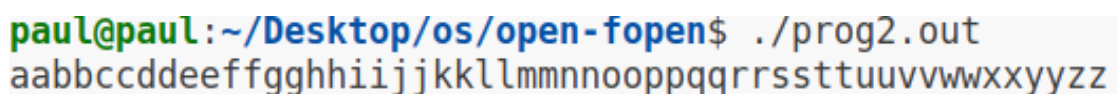
Рисунок 3 – Связь структур

Программа 2, один поток

Листинг 7 – Программа 2 — один поток

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     char c;
8     int fd1 = open("alphabet.txt", O_RDONLY);
9     int fd2 = open("alphabet.txt", O_RDONLY);
10
11     int flag1 = 1, flag2 = 2;
12     while(flag1 == 1 || flag2 == 1)
13     {
14         char c;
15         flag1 = read(fd1, &c, 1);
16         if (flag1 == 1)
17         {
18             write(1, &c, 1);
19         }
20         flag2 = read(fd2, &c, 1);
21         if (flag2 == 1)
22         {
23             write(1, &c, 1);
24         }
25     }
26     printf("\n");
27     return 0;
28 }
```

Результаты работы:



```
paul@paul:~/Desktop/os/open-fopen$ ./prog2.out
aabbbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
```

Рисунок 4 – Результаты работы программы 2 (один поток)

0.0.1 Программа 2, два потока

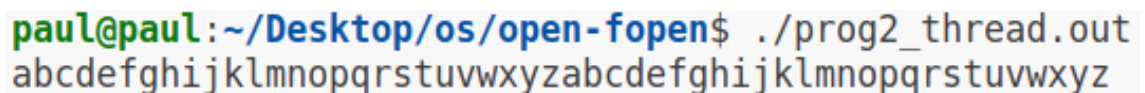
Листинг 8 – Программа 2 — два потока — часть 1

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 pthread_mutex_t mux;
7
8 void *thread_routine(void *arg)
9 {
10     int fd = *((int *)arg);
11
12     int flag = 1;
13     char c;
14
15     pthread_mutex_lock(&mux);
16     while (flag == 1)
17     {
18         flag = read(fd, &c, 1);
19         if (flag == 1)
20             write(1, &c, 1);
21     }
22     pthread_mutex_unlock(&mux);
23 }
24
25 int main()
26 {
27     int fd1 = open("alphabet.txt", O_RDONLY);
28     int fd2 = open("alphabet.txt", O_RDONLY);
29
30     pthread_t thr_worker;
31
32     if (pthread_mutex_init(&mux, NULL) != 0)
33     {
34         printf("can't pthread_mutex_init.\n");
35         return 1;
36     }
37
38     pthread_create(&thr_worker, NULL, thread_routine, &fd1);
```


Листинг 9 – Программа 2 — два потока — часть 2

```
1   int flag = 1;
2   char c;
3
4   pthread_mutex_lock(&mux);
5   while (flag == 1)
6   {
7       flag = read(fd2, &c, 1);
8       if (flag == 1)
9           write(1, &c, 1);
10  }
11  pthread_mutex_unlock(&mux);
12
13  pthread_join(thr_worker, NULL);
14  printf("\n");
15  return 0;
16 }
```

Результаты работы:



```
paul@paul:~/Desktop/os/open-fopen$ ./prog2_thread.out
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
```

Рисунок 5 – Результаты работы программы 2 (два потока)

Анализ результата

При использовании функции *open* для открытия одного и того же файла 2 раза будет создано 2 дескриптора в таблице открытых файлов процесса (причём *inode* у них будет один и тот же), у каждого из них значение поля *f_pos* будет своё,

При чтении, использующем такие дескрипторы, будет поочерёдно изменяться значение *f_pos*, что приведёт к полному прочтению файла 2 раза.

Связь структур

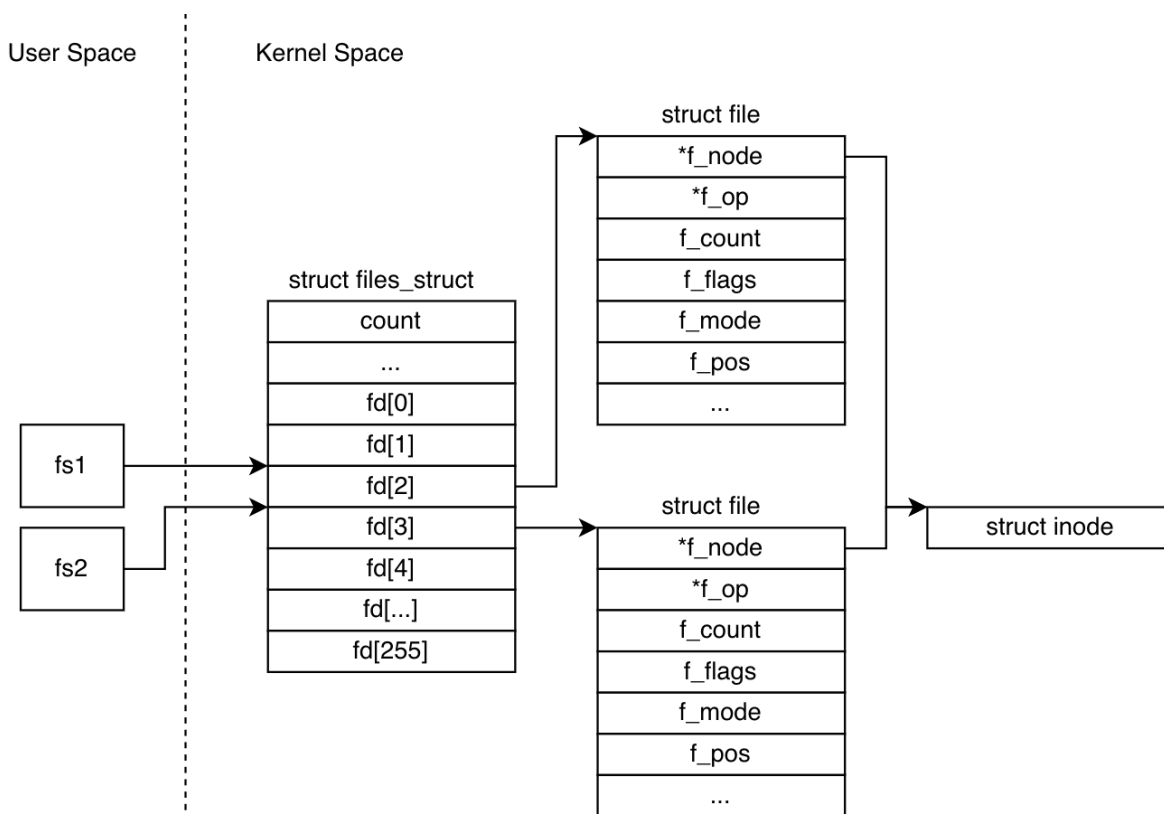


Рисунок 6 – Связь структур

Программа 3, один поток

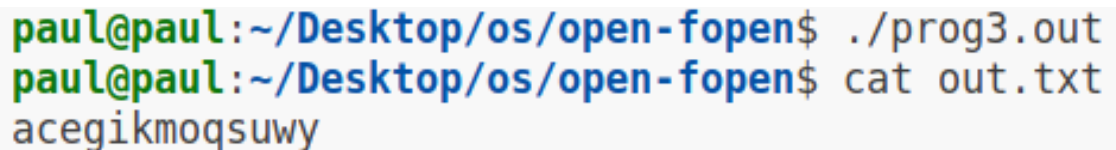
Листинг 10 – Программа 3 — один поток — часть 1

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     FILE *f1 = fopen("out.txt", "w");
8     FILE *f2 = fopen("out.txt", "w");
9
10    for (char letr = 'a'; letr < '{'; letr++)
11    {
12        letr % 2 ? fprintf(f1, "%c", letr) : fprintf(f2, "%c",
13            letr);
14    }
```

Листинг 11 – Программа 3 — один поток — часть 2

```
1   fprintf(f1, "\n");
2   fclose(f2);
3   fclose(f1);
4   return 0;
5 }
```

Результаты работы:



```
paul@paul:~/Desktop/os/open-fopen$ ./prog3.out
paul@paul:~/Desktop/os/open-fopen$ cat out.txt
acegikmoqsuwy
```

Рисунок 7 – Результаты работы программы 3 (один поток)

Программа 3, два потока

Листинг 12 – Программа 3 — два потока — часть 1

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6
7 struct stat statbuf;
8
9 void *thread_routine()
10 {
11     FILE *f2 = fopen("out.txt", "a");
12     stat("out.txt", &statbuf);
13     printf("open for fs2: inode = %ld, buffsize = %ld blocksize=
14           %ld\n",
15           (long int)statbuf.st_ino, (long int)statbuf.st_size,
16           (long int)statbuf.st_blksize);
17
18     for (char letr = 'a'; letr < '{'; letr += 2)
19         fprintf(f2, "%c", letr);
20     fclose(f2);
21     stat("out.txt", &statbuf);
22 }
```

Листинг 13 – Программа 3 — два потока — часть 2

```
1     printf("close for fs2: inode  = %ld , bufsize = %ld blocksize=
      %ld\n",
2         (long int)statbuf.st_ino , (long int)statbuf.st_size ,
3         (long int)statbuf.st_blksize);
4 }
5
6 int main()
7 {
8     FILE *f1 = fopen("out.txt", "a");
9     stat("out.txt", &statbuf);
10    printf("open for fs1: inode  = %ld , bufsize = %ld blocksize=
      %ld\n",
11        (long int)statbuf.st_ino , (long int)statbuf.st_size ,
12        (long int)statbuf.st_blksize);
13    pthread_t thr_worker;
14    pthread_create(&thr_worker, NULL, thread_routine , f1);
15    pthread_join(thr_worker , NULL);
16
17    for (char letr = 'a'; letr < '{'; letr += 2)
18        fprintf(f1, "%c", letr);
19
20    fprintf(f1, "\n");
21    fclose(f1);
22    stat("out.txt", &statbuf);
23    printf("close for fs2: inode  = %ld , bufsize = %ld blocksize=
      %ld\n",
24        (long int)statbuf.st_ino , (long int)statbuf.st_size ,
25        (long int)statbuf.st_blksize);
26
27    return 0;
28 }
```

Результаты работы:

```
paul@paul:~/Desktop/os/open-fopen$ ./prog3_thread.out
open for fs1: inode = 1072988, buffsize = 0 blocksize= 4096
open for fs2: inode = 1072988, buffsize = 0 blocksize= 4096
close for fs2: inode = 1072988, buffsize = 13 blocksize= 4096
close for fs1: inode = 1072988, buffsize = 26 blocksize= 4096
paul@paul:~/Desktop/os/open-fopen$ cat out.txt
acedikmoosuwvacedikmoosuwvpaul@paul:~/Desktop/os/open-fopen$
```

Рисунок 8 – Результаты работы программы 3 (один поток)

Анализ результатов

При использовании функции *fopen* для открытия файла *out.txt* на запись два раза будет создано две структуры *IOFILE*, а также две независимых структуры *struct file* с независимыми полями *f_pos*, но с одной и той же структурой *struct inode*.

Функция *fprintf* создаёт буфер, в который информация помещается перед тем, как быть записана в файл.

Из буфера, созданного *fprintf*, информация копируется в файл при выполнении одного из трёх условий:

1. Буфер заполнен.
2. Вызван *fflush* для принудительной записи в файл.
3. Вызван *fclose*.

Для приведённых программ: запись осуществляется при вызове *fclose* для *f1* (буфер *f1* записывается в файл), при вызове *fclose* для *f2* (буфер *f2* записывается в файл), происходит утеря данных, в файле оказывается только содержимое буфера *f2*.

Данную проблему можно решить, использовав флаг *O_APPEND* при вызове функции *open* — это гарантирует неделимость каждой операции записи,

Связь структур

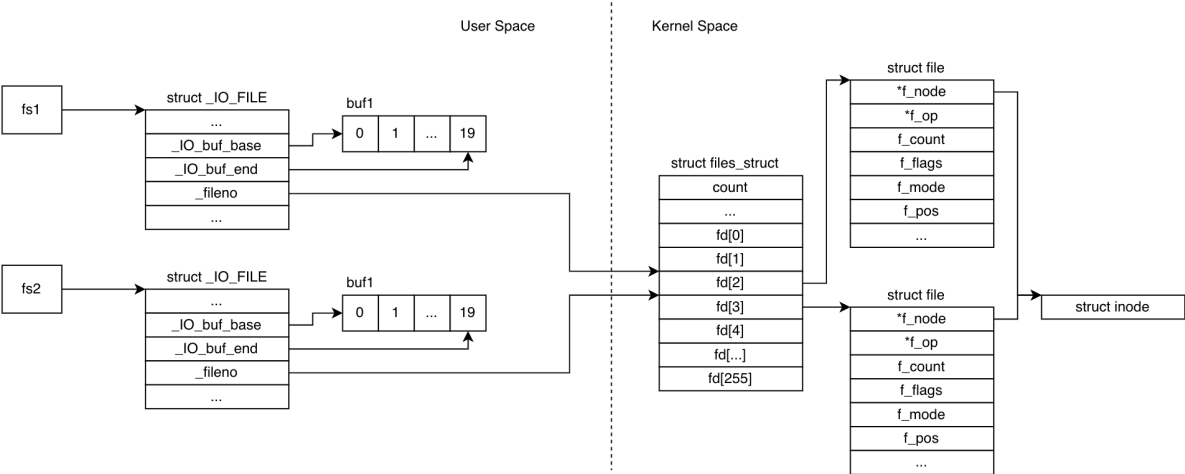


Рисунок 9 – Связь структур