



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 №5 по дисциплине «Операционные системы»

Тема Буферизованный и не буферизованный ввод-вывод.

Студент Варламова Е. А.

Группа ИУ7-61Б

Оценка (баллы) _____

Преподаватель Рязанова Н. Ю.

Задание

В лабораторной работе анализируется результат выполнения трех программ. Программы демонстрируют открытие одного и того же файла несколько раз. Реализация, когда файл открывается в одной программе несколько раз выбрана для простоты. Однако, как правило, такая ситуация возможна в системе, когда один и тот же файл несколько раз открывают разные процессы или потоки одного процесса. При выполнении асинхронных процессов такая ситуация является вероятной и ее надо учитывать, чтобы избежать потери данных, получения неверного результата при выводе данных в файл или чтения данных не в той последовательности, в какой предполагалось, и в результате при обработке этих данных получения неверного результата. Каждую из приведенных программ надо выполнить в многопоточном варианте: в программах создается дополнительный поток, а работа с открываемым файлом выполняется в потоках.

Проанализировать работу приведенных программ и объяснить результаты их работы.

Программа №1

Код однопоточной версии

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 int main() {
4     int fd = open("alphabet.txt", O_RDONLY);
5
6     FILE *fs1 = fdopen(fd, "r");
7     char buff1[20];
8     setvbuf(fs1, buff1, _IOFBF, 20);
9
10    FILE *fs2 = fdopen(fd, "r");
11    char buff2[20];
12    setvbuf(fs2, buff2, _IOFBF, 20);
13
14    int flag1 = 1, flag2 = 2;
15
16    while (flag1 == 1 || flag2 == 1) {
17        char c;
18        flag1 = fscanf(fs1, "%c", &c);
19        if (flag1 == 1) {
20            fprintf(stdout, "%c", c);
21        }
22        flag2 = fscanf(fs2, "%c", &c);
23        if (flag2 == 1) {
24            fprintf(stdout, "%c", c);
25        }
26    }
27    return 0; }
```

Результат работы

```
kate@MacBook-Pro-Ekaterina ~/D/o/l/src (master)> ./app
Aubvcwdxeyfzghijklmnopqrst
```

Рис. 1: Однопоточная версия 1-ой программы

Код многопоточной версии

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <pthread.h>
4 void thread(int fd)
5 {
6     FILE *fs = fdopen(fd, "r");
7     char buff[20];
8     setvbuf(fs, buff, _IOFBF, 20);
9
10    char c;
11    int flag = fscanf(fs, "%c", &c);
12    while (flag == 1) {
13        fprintf(stdout, "%c", c);
14        flag = fscanf(fs, "%c", &c);
15    }
16 }
17 int main() {
18     int fd = open("alphabet.txt", O_RDONLY);
19
20     pthread_t tid[2];
21     for (int i = 0; i < 2; i++) {
22         if (pthread_create(&tid[i], NULL, thread, fd)) {
23             printf("Error: can't create thread\n");
24             return -1; }
25     }
26     pthread_join(tid[0], NULL);
27     pthread_join(tid[1], NULL);
28     return 0;
29 }
```

Результат работы

```
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/01_two_threads (master)> ./app
Abcdefghuvwxyzijklmnopqrst
```

Рис. 2: Многопоточная версия 1-ой программы

Объяснение результатов

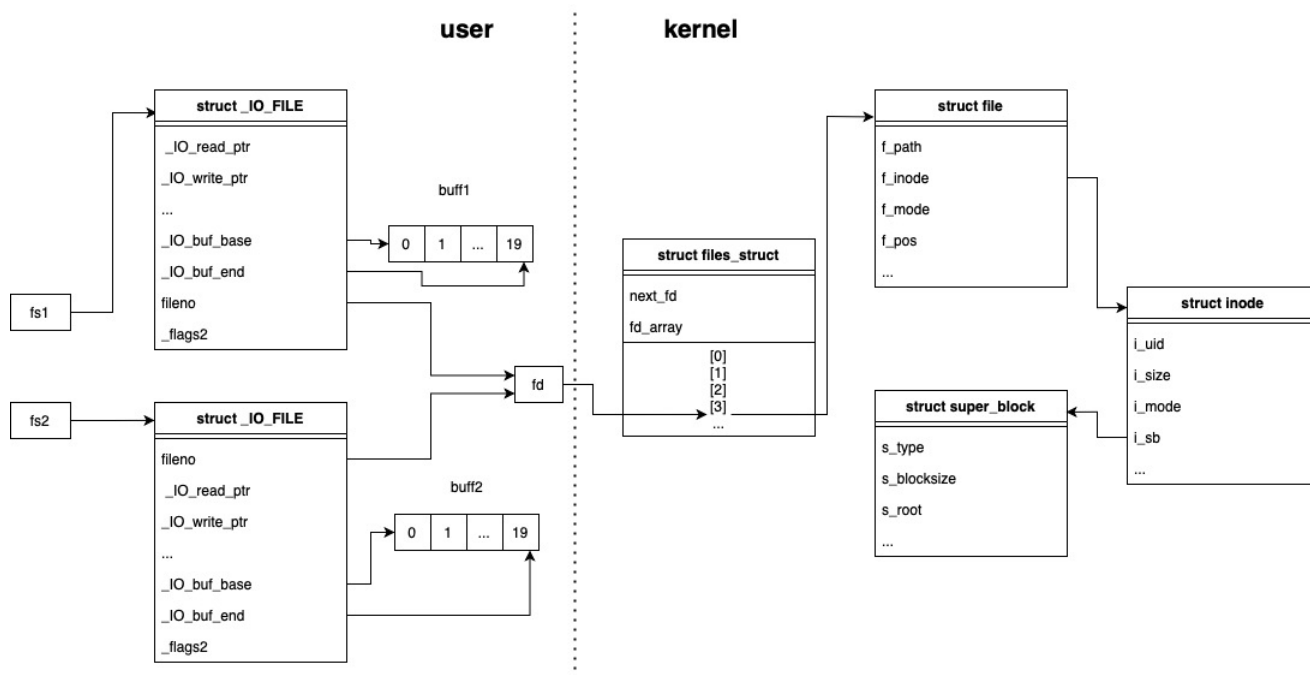


Рис. 3: Связь структур

Системный вызов `open()` создаёт дескриптор открытого файла (описанного структурой `struct file`) и возвращает индекс в массиве `fd_array` структуры `files_struct`.

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

С помощью `setbuf` для структур `FILE` (`fs1`, `fs2`) задаются буферы и тип буферизации `_IOFBF` (Fully buffered).

При первом вызове `fscanf(fs1, ...)` буфер структуры `fs1` полностью заполнится, то есть будут прочитаны первые 20 символов, а поле `f_pos` структуры `struct file` станет равным 20. Затем при вызове `fscanf(fs2, ...)` оставшиеся символы считаются в буфер структуры `fs2`, `f_pos` структуры `struct file` станет равным 20 + количество оставшихся символов. При последующих вызовах `fscanf(fs1, ...)` и `fscanf(fs2, ...)` работа будет производиться с буферами, поэтому в результате будет получен приведённый результат:

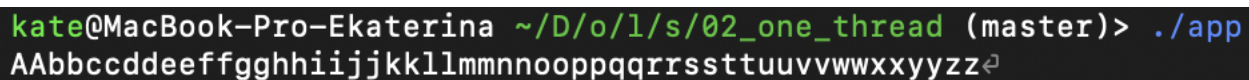
- в однопоточной версии вызовы `fscanf(fs1, ...)` и `fscanf(fs2, ...)` происходят поочерёдно, поэтому символы в результате соответствуют поочерёднему чтению из первого и второго буферов;
- в многопоточной версии порядок вызовов `fscanf(fs, ...)` двух потоков не определён, поэтому символы в результате соответствуют чтению из буферов в произвольном порядке.

Программа №2

Код однопоточной версии

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

Результат работы



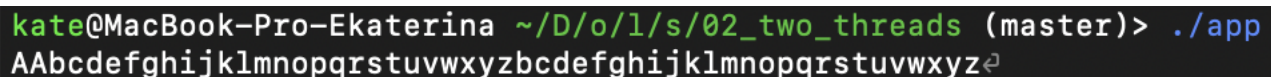
```
kate@MacBook-Pro-Ekaterina ~/D/o/1/s/02_one_thread (master)> ./app
AAbbccddeeffgghhiijjkkllmmnnoopppqrrssttuuvvwwxyyyz
```

Рис. 4: Однопоточная версия 2-ой программы

Код многопоточной версии

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <pthread.h>
5 void thread()
6 {
7     char c;
8     int fd = open("alphabet.txt", O_RDONLY);
9     while (read(fd, &c, 1) == 1)
10     {
11         write(1, &c, 1);
12     }
13 }
14 int main()
15 {
16     pthread_t tid[2];
17     for (int i = 0; i < 2; i++) {
18         if (pthread_create(&tid[i], NULL, thread, NULL)) {
19             printf("Error: can't create thread\n");
20             return -1; }
21     }
22     pthread_join(tid[0], NULL);
23     pthread_join(tid[1], NULL);
24     return 0;
25 }
```

Результат работы



```
kate@MacBook-Pro-Ekaterina ~/D/o/1/s/02_two_threads (master)> ./app
AAbcdefghijklmnopqrstuvwxyzbcdefghijklmnopqrstuvwxyz
```

Рис. 5: Многопоточная версия 2-ой программы

Объяснение результатов

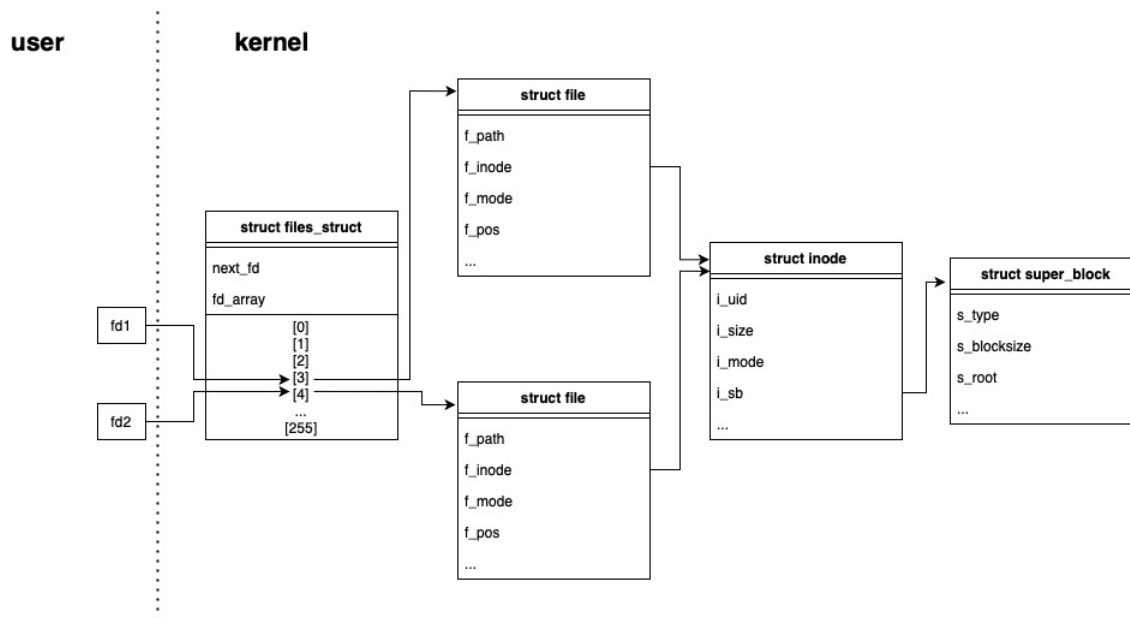


Рис. 6: Связь структур

Системный вызов `open()` создаёт 2 файловых дескриптора открытого файла (описанного структурами `struct file`) и возвращает индекс в массиве `fd_array` структуры `files_struct`. Таким образом, создаётся две структуры открытого файла, каждая из которых имеет собственное поле `f_pos`. Именно поэтому в результате будет получен приведённый результат:

- в однопоточной версии вызовы `read` происходят поочерёдно, поэтому символы в результате соответствуют двум поочерёдным независимым чтениям из файла;
- в многопоточной версии порядок вызовов `read` двух потоков не определён, поэтому символы в результате соответствуют двум независимым чтениям в произвольном порядке.

Программа №3

Код

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3 #include <pthread.h>
4 #define FMT_STR " FS%d: inode = %ld , size = %ld\n"
5 void *thread(int data) {
6     int fid = (int)data;
7     struct stat statbuf;
8     FILE *fs = fopen("out.txt", "w");
9     stat("out.txt", &statbuf);
10 }
```

```

11     printf("FOPEN " FMT_STR, fid , statbuf.st_ino , statbuf.st_size);
12     for (char c = 'a'; c <= 'z'; c++) {
13         if ((c % 2) == (fid == 1))
14             fprintf(fs, "%c", c);
15     }
16     fclose(fs);
17     stat("out.txt", &statbuf);
18     printf("FCLOSE" FMT_STR, fid , statbuf.st_ino , statbuf.st_size);
19 }
20 int main() {
21     pthread_t tid[2]; int fid[2] = {0, 1};
22     for (int i = 0; i < 2; i++) {
23         if (pthread_create(&tid[i], NULL, thread, fid[i])) {
24             printf("Error: can't create thread\n");
25             return -1; }
26     }
27     pthread_join(tid[0], NULL);
28     pthread_join(tid[1], NULL); return 0;
29 }

```

Результат работы

```

kate@MacBook-Pro-Ekaterina ~/D/o/l/s/03_threads (master)> ./app
FOPEN  FS0: inode = 41305511, size = 0
FOPEN  FS1: inode = 41305511, size = 0
FCLOSE FS1: inode = 41305511, size = 13
FCLOSE FS0: inode = 41305511, size = 13
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/03_threads (master)> cat out.txt
acegikmoqsuwy↵
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/03_threads (master)> ./app
FOPEN  FS1: inode = 41305511, size = 0
FOPEN  FS0: inode = 41305511, size = 0
FCLOSE FS1: inode = 41305511, size = 13
FCLOSE FS0: inode = 41305511, size = 13
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/03_threads (master)> cat out.txt
bdfhjlnprtvxz↵
kate@MacBook-Pro-Ekaterina ~/D/o/l/s/03_threads (master)> █

```

Рис. 7: 3-я программа

Объяснение результатов

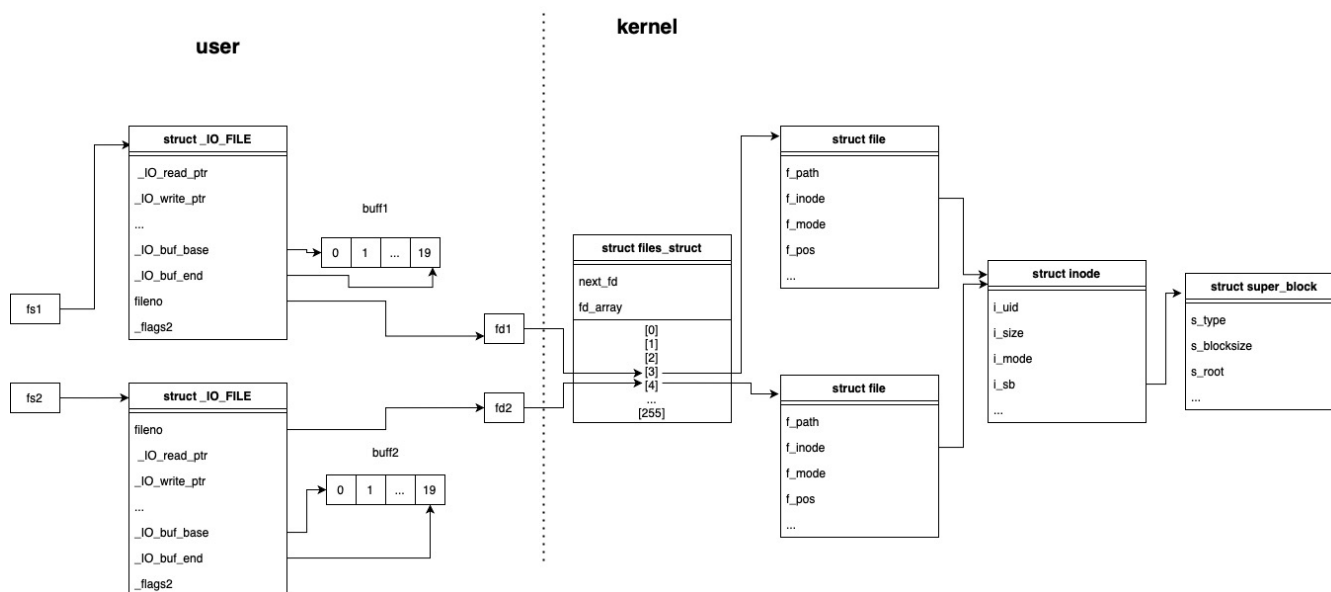


Рис. 8: Связь структур

Вызов `open()` создаёт файловый дескриптор открытого файла (описанного структурой `struct file`) и возвращает индекс в массиве `fd_array` структуры `files_struct`. Кроме того, `open` создаёт структуру типа `FILE`, которая ссылается на дескриптор открытого файла, созданный ранее. Таким образом, создаётся две структуры открытого файла, каждая из которых имеет собственное поле `f_pos`.

Так как запись производится с помощью функции `fprintf`, использующий буферизованный ввод-вывод, то запись в двух потоках будет осуществляться в буфер. Существуют 3 условия записи из буфера в файл:

- переполнение буфера;
- `fflush`;
- закрытие файла.

Так как буфер не переполняется и не вызывается `fflush`, то запись в файл в приведённой программе осуществится лишь при вызове `fclose`. Так как потоки работают с одним и тем же файлом, причём ссылаются на разные структуры `struct file` (то есть поле `f_pos` для каждого потока своё), то результат записи будет зависеть от того, какой поток вызвал `fclose` позже. При этом данные, записанные в файл ранее, будут утеряны.

Решение проблемы

Причина: каждый дескриптор открытого файла имеет своё поле `f_pos`.

Решение №1: необходимо использовать режим `O_APPEND`. В данном режиме перемещение позиции в конец файла и добавление символа происходят атомарно, поэтому данные не будут утеряны.

Решение №2: необходимо использовать мьютекс для перемещения позиции в конец файла и записи символа. Код:

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 #define FMT_STR " FS%d: inode = %ld , size = %ld\n"
7 pthread_mutex_t lock;
8 void *thread(int data) {
9     int fid = (int)data;
10    struct stat statbuf;
11    FILE *fs = fopen("out.txt", "w");
12    stat("out.txt", &statbuf);
13
14    printf("FOPEN " FMT_STR, fid , statbuf.st_ino , statbuf.st_size);
15    for (char c = 'a'; c <= 'z'; c++) {
16        if ((c % 2) == (fid == 1)) {
17            pthread_mutex_lock(&lock);
18            lseek(fileno(fs), NULL, SEEK_END);
19            fprintf(fs, "%c", c);
20            pthread_mutex_unlock(&lock);
21        }
22    }
23    fclose(fs);
24    stat("out.txt", &statbuf);
25    printf("FCLOSE" FMT_STR, fid , statbuf.st_ino , statbuf.st_size);
26 }
27 int main() {
28     if (pthread_mutex_init(&lock , NULL) != 0)
29     {
30         printf("\n mutex init failed\n");
31         return 1;
32     }
33     pthread_t tid[2]; int fid[2] = {0, 1};
34     for (int i = 0; i < 2; i++) {
35         if (pthread_create(&tid[i], NULL, thread , fid[i])) {
36             printf("Error: can't create thread\n");
37             return -1; }
38     }
39     pthread_join(tid[0], NULL);
40     pthread_join(tid[1], NULL); return 0;
41 }
```

struct _IO_FILE

```
1 struct _IO_FILE
2 {
3     int _flags;                /* High-order word is _IO_MAGIC; rest is
4                                flags. */
5
6     /* The following pointers correspond to the C++ streambuf 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    char *_IO_save_base; /* Pointer to start of non-current get area. */
18    char *_IO_backup_base; /* Pointer to first valid character of backup
19                           area */
20    char *_IO_save_end; /* Pointer to end of non-current get area. */
21
22    struct _IO_marker *_markers;
23
24    struct _IO_FILE *_chain;
25
26    int _fileno;
27    int _flags2;
28    __off_t _old_offset; /* This used to be _offset but it's too small. */
29
30    /* 1+column number of pbase(); 0 is unknown. */
31    unsigned short _cur_column;
32    signed char _vtable_offset;
33    char _shortbuf[1];
34
35    _IO_lock_t *_lock;
36 #ifdef _IO_USE_OLD_IO_FILE
37 };
38
```