



**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»

Лабораторная работа № 5

Тема Буферизованный и не буферизованный ввод-вывод

Студент Садулаева Т. Р.

Группа ИУ7-64Б

Оценка (баллы) _____

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

Структура FILE

«stdio.h»:

```
#ifndef __FILE defined
#define __FILE_defined 1
struct __sFILE;
typedef struct __sFILE FILE;
#endif

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 _lbfsz; /* 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;

```

Программа 1

Программа с одним потоком

```

#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;
}

```

Программа с дополнительными потоками

```
#include <stdio.h>
#include <fcntl.h>
#include <pthread.h>
#include <unistd.h>

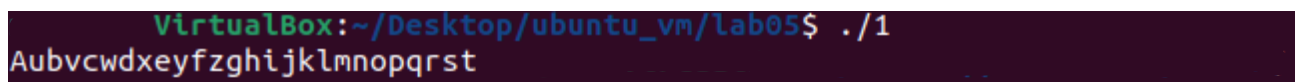
void *reading_func(void* d) {
    int fd = *(int*) d;
    FILE *fs = fdopen(fd, "r");
    char buff[20];
    setvbuf(fs, buff, _IOFBF, 20);
    char c;
    int flag = 1;
    while (flag == 1) {
        flag = fscanf(fs, "%c", &c);
        if (flag == 1) {
            fprintf(stdout, "%c", c);
        }
    }
}

int main() {
    int fd = open("alphabet.txt", O_RDONLY);
    pthread_t th1;
    pthread_t th2;

    if (pthread_create(&th1, NULL, reading_func, (void*) &fd)) {
        printf("Unable to create thread 1\n");
        return -1;
    }


    if (pthread_create(&th2, NULL, reading_func, (void*) &fd)) {
        printf("Unable to create thread 2\n");
        return -1;
    }

    pthread_join(th1, NULL);
    pthread_join(th2, NULL);
    return 0;
}
```



```
VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./1
Aubvcwdxeyfzghijklmnopqrst
```

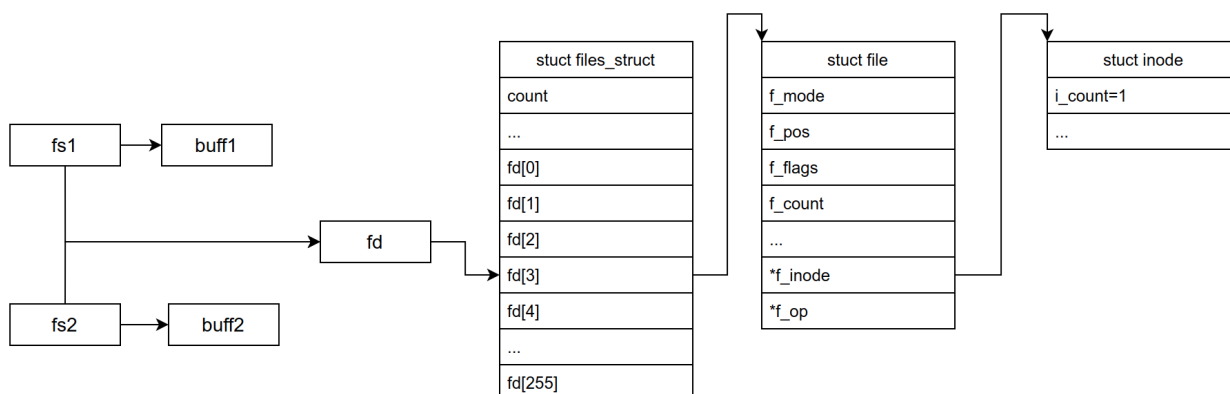
Рисунок 1. Результаты работы программы 1 с одним потоком.



```
VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./1t
Abcdefghijklmnopqrstuvwxyz
```

Рисунок 2. Результаты работы программы 1 с доп. потоками.

Схема связей структур в программе 1



Анализ программы 1

При помощи системного вызова `open()`, который возвращает индекс в массиве `fd` структуры `files_struct`, создается дескриптор файла “alphabet.txt”. Вызов `fdopen()` создает структуры `FILE fs1` и `FILE fs2`, которые связаны с `fd`. При помощи вызова `setvbuf` устанавливаем буферы размером 20 байт с типом буферизации `_IOFBF` – полная буферизация.

В цикле выполняется вызов `fscanf` для `fs1` и `fs2`. Пока не достигнут конец файла, буфер заполняется полностью при каждом вызове `fscanf()`, указатель `f_pos` устанавливается на следующий за последним записанным в буфер символ. В `buff1` будет помещена строка “Abcdefghijklmnopqrst” (первые 20 символов), а в `buff2` – “vwxyz”. Далее в цикле выводится на экран содержимое заполненных буферов (по одному символу из каждого буфера), в результате чего формируется результат “Aubvcwdxeyfzghijklmnopqrst”.

В программе с использованием дополнительных потоков содержимое первого буфера выводится полностью до начала вывода содержимого второго буфера, так как первый из дополнительных потоков инициализируется раньше, чем второй. В результате отображается строка “Abcdefghijklmnopqrstuvwxyz”.

Программа 2

Программа с одним потоком

```
#include <fcntl.h>
#include <unistd.h>

int main() {
    char c;
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

    int flag = 1;
    while(flag) {
        flag = read(fd1, &c, 1) == 1? 1 : 0;
        if (flag == 1) {
            write(1, &c, 1);
        }

        flag = flag == 1 && read(fd2, &c, 1) == 1? 1 : 0;
        if (flag == 1) {
            write(1, &c, 1);
        }
    }
    return 0;
}
```

Программа с дополнительными потоками

```
#include <stdio.h>
#include <fcntl.h>
#include <pthread.h>
#include <unistd.h>

void *reading_func(void * d) {
    char c;
    int fd = open("alphabet.txt", O_RDONLY);
    int flag = 1;

    while(flag) {
        flag = read(fd, &c, 1) == 1? 1 : 0;

        if (flag == 1) {
            write(1, &c, 1);
        }
    }
}
```

```

int main() {
    pthread_t th1;
    pthread_t th2;

    if (pthread_create(&th1, NULL, reading_func, NULL)) {
        printf("Unable to create thread 1\n");
        return -1;
    }

    if (pthread_create(&th2, NULL, reading_func, NULL)) {
        printf("Unable to create thread 2\n");
        return -1;
    }

    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}

```

```

VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./2
AAbbccddeeffgghhiijjkkllmmnnoppqrrssttuuvvwwxxyyzz

```

Рисунок 3. Результаты работы программы 2 с одним потоком.

```

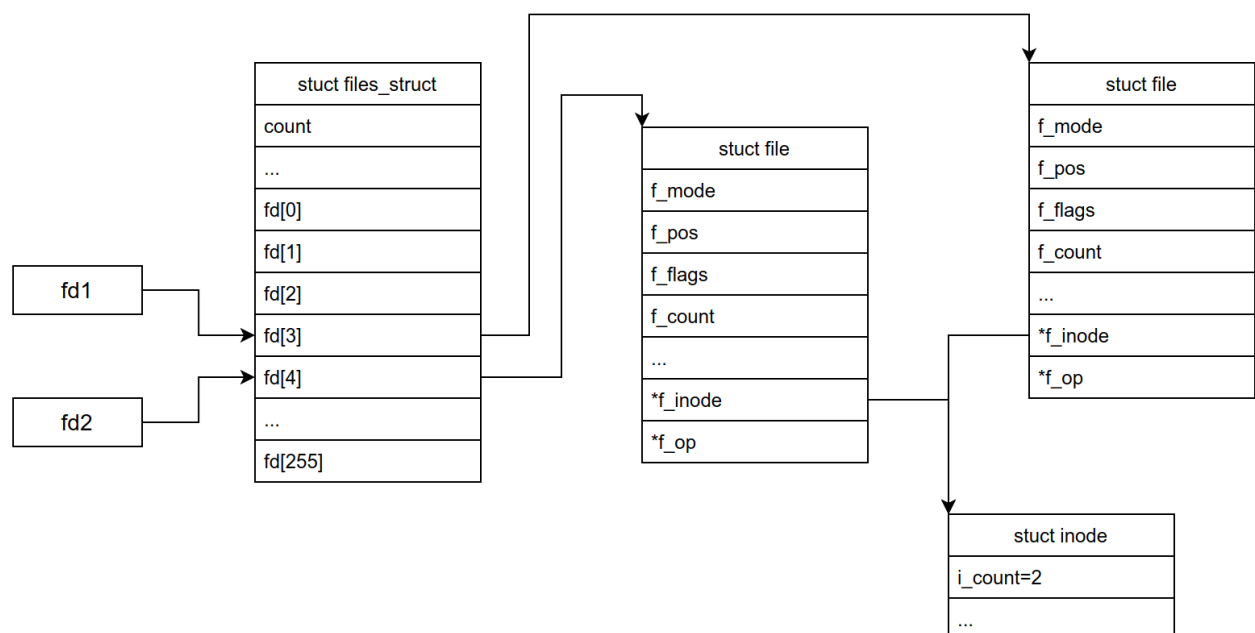
VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./2t
AbcdAefbcdghijklmnoefghijklmnopqrstuvwxyzpqrstuvwxyz
AAbccdefghijklmnopqrstuvwxyzbcdefghijklmnopqrstuvwxyz
AAbbccdefghijklmngopqrstuvwxyzjklmnopqrstuvwxyz
AAbbccdedefghghijklklmmnnoppqrrssttuuvvwwxywxyz

VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./2t
VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./2t
VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./2t
VirtualBox:~/Desktop/ubuntu_vm/lab05$

```

Рисунок 4. Результаты работы программы 2 с доп. потоками.

Схема связей структур в программе 2



Анализ программы 2

При помощи системного вызова `open()`, который возвращает индекс в массиве `fd` структуры `files_struct`, создаются два дескриптора файла “alphabet.txt” и две записи в системной таблице открытых файлов. Каждый дескриптор содержит указатель `f_pos`, который указывает на текущую позицию в файле и изменяется независимо от `f_pos` второго дескриптора. Результатом чтения данных при помощи двух дескрипторов является вывод каждого из символов файла в двух экземплярах: “Aabbccddeeffgghhiijjkkllmmnnnooppqqrrsstt uuvvwxxxyyzz”.

В программе с использованием дополнительных потоков символы выводятся со случайным чередованием (например, “AbAcbcdefghijklmnopqdrsef tuvwxxyzghijklmnopqrstuvwxyz”, так как данные обрабатываются двумя потоками параллельно.

Программа 3

Структура `struct stat`

```
struct stat {
    dev_t      st_dev;          /* ID of device containing file */
    ino_t      st_ino;          /* Inode number */
    mode_t     st_mode;         /* File type and mode */
    nlink_t    st_nlink;        /* Number of hard links */
    uid_t      st_uid;          /* User ID of owner */
    gid_t      st_gid;          /* Group ID of owner */
    dev_t      st_rdev;         /* Device ID (if special file) */
    off_t      st_size;         /* Total size, in bytes */
    blksize_t  st_blksize;      /* Block size for filesystem I/O */
    blkcnt_t   st_blocks;       /* Number of 512B blocks allocated */

    struct timespec st_atim;     /* Time of last access */
    struct timespec st_mtim;     /* Time of last modification */
    struct timespec st_ctim;     /* Time of last status change */
};
```

Программа с одним потоком

```
#include <stdio.h>
#include <sys/stat.h>

#define FNAME "result.txt"

int main() {
    struct stat sbuf;
    FILE *fs1 = fopen(FNAME, "w");
    stat(FNAME, &sbuf);
```



```

printf("<fopen fs1> inode: %ld size: %ld\n",
      (long int) sbuf.st_ino,
      (long int) sbuf.st_size);

FILE *fs2 = fopen(FNAME, "w");
stat(FNAME, &sbuf);
printf("<fopen fs2> inode: %ld size: %ld\n",
      (long int) sbuf.st_ino,
      (long int) sbuf.st_size);

for (char c = 'a'; c <= 'z'; c++) {
    if (c % 2)
        fprintf(fs1, "%c", c);
    else
        fprintf(fs2, "%c", c);
}

fclose(fs1);
stat(FNAME, &sbuf);
printf("<fclose fs1> inode: %ld size: %ld\n",
      (long int) sbuf.st_ino,
      (long int) sbuf.st_size);

fclose(fs2);
stat(FNAME, &sbuf);
printf("<fclose fs2> inode: %ld size: %ld\n",
      (long int) sbuf.st_ino,
      (long int) sbuf.st_size);

return 0;
}

```

Программа с дополнительными потоками

```

#include <stdio.h>
#include <sys/stat.h>
#include <pthread.h>

#define FNAME "result.txt"

void *writing_func(void * d) {
    int tnum = *(int*) d;
    struct stat sbuf;

    FILE *fs = fopen(FNAME, "w");
    stat(FNAME, &sbuf);
    printf("<fopen fs%d> inode: %ld size: %ld\n",
          tnum,
          (long int) sbuf.st_ino,
          (long int) sbuf.st_size);
}

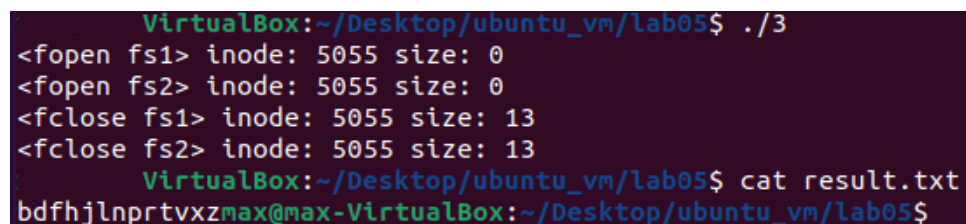
```

```

for (char c = 'a'; c <= 'z'; c++) {
    if (c % 2 && tnum == 1) {
        fprintf(fs, "%c", c);
    }
    else if (!(c % 2) && tnum == 2) {
        fprintf(fs, "%c", c);
    }
}
fclose(fs);
stat(FNAME, &sbuf);
printf("<fclose fs%d> inode: %ld size: %ld\n",
        tnum, (long int) sbuf.st_ino, (long int) sbuf.st_size);
pthread_exit(0);
}
int main() {
    pthread_t th1;
    pthread_t th2;
    int tnum1 = 1;
    int tnum2 = 2;
    if (pthread_create(&th1, NULL, writing_func, (void*) &tnum1)){
        printf("Unable to create thread 1\n");
        return -1;
    }

    if (pthread_create(&th2, NULL, writing_func, (void*) &tnum2)){
        printf("Unable to create thread 2\n");
        return -1;
    }
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);
    return 0;
}

```

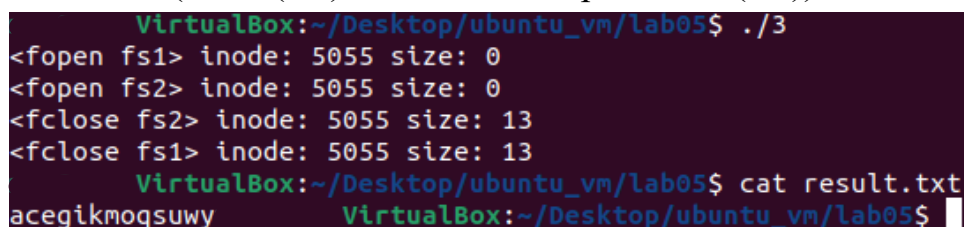


```

VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./3
<fopen fs1> inode: 5055 size: 0
<fopen fs2> inode: 5055 size: 0
<fclose fs1> inode: 5055 size: 13
<fclose fs2> inode: 5055 size: 13
VirtualBox:~/Desktop/ubuntu_vm/lab05$ cat result.txt
bdfhjlnprtvmx@max-VirtualBox:~/Desktop/ubuntu_vm/lab05$

```

Рисунок 5. Результаты работы программы 3 с одним потоком (fclose(fs1) вызывается перед fclose(fs2)).



```

VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./3
<fopen fs1> inode: 5055 size: 0
<fopen fs2> inode: 5055 size: 0
<fclose fs2> inode: 5055 size: 13
<fclose fs1> inode: 5055 size: 13
VirtualBox:~/Desktop/ubuntu_vm/lab05$ cat result.txt
acegikmoqsuwy
VirtualBox:~/Desktop/ubuntu_vm/lab05$

```

Рисунок 6. Результаты работы программы 3 с одним потоком (fclose(fs2) вызывается перед fclose(fs1)).

```

VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./3t
<fopen fs1> inode: 5055 size: 0
<fopen fs2> inode: 5055 size: 0
<fclose fs1> inode: 5055 size: 13
<fclose fs2> inode: 5055 size: 13
VirtualBox:~/Desktop/ubuntu_vm/lab05$ cat result.txt
bdfhjlnprtvxz
VirtualBox:~/Desktop/ubuntu_vm/lab05$

```

Рисунок

7. Результаты работы программы 3 с доп. потоками (fclose(fs1) выполняется раньше, чем fclose(fs2)).

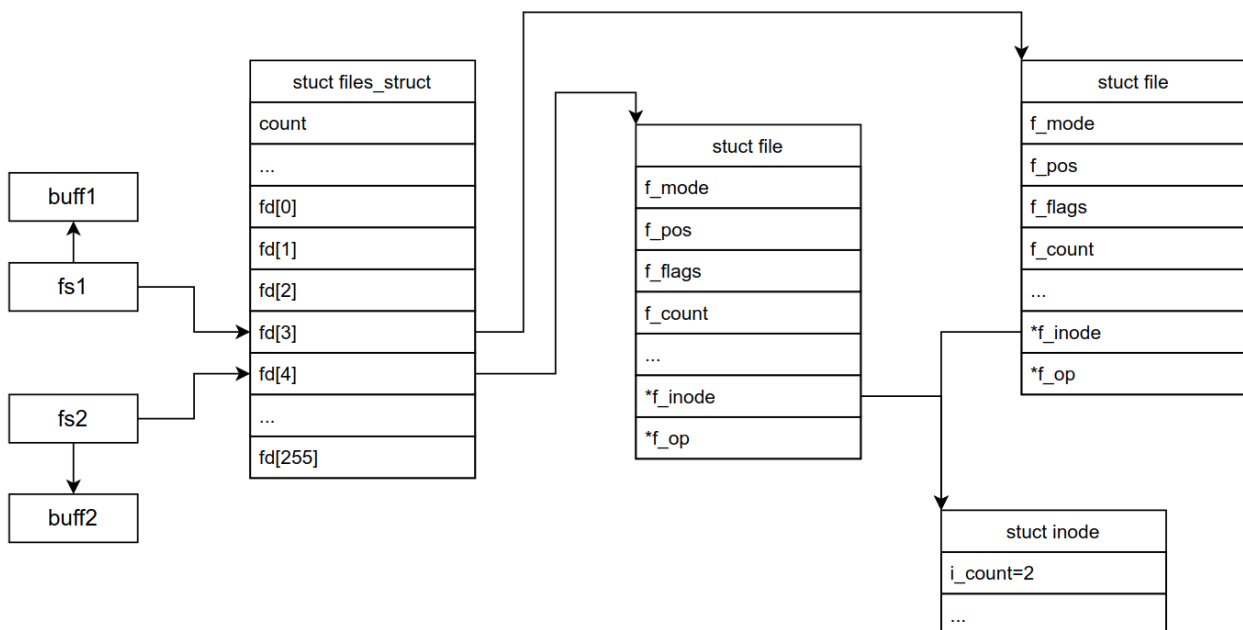
```

VirtualBox:~/Desktop/ubuntu_vm/lab05$ ./3t
<fopen fs2> inode: 5055 size: 0
<fopen fs1> inode: 5055 size: 0
<fclose fs2> inode: 5055 size: 13
<fclose fs1> inode: 5055 size: 13
VirtualBox:~/Desktop/ubuntu_vm/lab05$ cat result.txt
acegikmoqsuwy
VirtualBox:~/Desktop/ubuntu_vm/lab05$

```

Рисунок 8. Результаты работы программы 3 с доп. потоками (fclose(fs2) выполняется раньше, чем fclose(fs1)).

Схема связей структур в программе 3



Анализ программы 3

Для записи результата при помощи вызова `fopen` создаются два дескриптора файла “result.txt”. Каждый дескриптор содержит указатель `f_pos`, который указывает на текущую позицию в файле и изменяется независимо от `f_pos` второго дескриптора. Обоим дескрипторам соответствует единственный `inode`, что демонстрирует поле `struct.st_ino`.

При вызове `fprintf` осуществляется буферизованный вывод данных. Данные выводятся из буфера, если буфер заполнен, либо вызвана функция `fflush`, либо вызвана функция `fclose`.

В цикле осуществляется запись букв алфавита: буквы с нечётными кодами записываются в `fs1`, с чётными – в `fs2`.

В зависимости от того, какой файл получил квант, файл перезапишется данными другого (содержать только данные из буфера) и часть данных будет утеряна.

В программе с использованием дополнительных потоков вызов `fclose(fs1)` может произойти как до, так и после вызова `fclose(fs2)`, поэтому в файле могут оказаться различные данные.

Для того, чтобы избежать потери данных, необходимо открывать файл на дозапись – то есть вызывать `open` с флагом `O_APPEND`.