



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»

Отчёт по лабораторной работе №5 по курсу «Операционные системы»

Тема _____ Буферизованный и небуферизованный ввод-вывод

Студент _____ Якуба Д.В.

Группа _____ ИУ7-63Б

Оценка (баллы) _____

Преподаватель _____ Рязанова Н.Ю.

Москва, 2021 г

Структура FILE

/usr/include/x86_64-linux-gnu/bits/types/

```
#ifndef __FILE_defined
#define __FILE_defined 1

struct _IO_FILE;

/* The opaque type of streams. This is the definition used elsewhere. */
typedef struct _IO_FILE FILE;

#endif
```

/usr/include/x86_64-linux-gnu/bits/types/

```
/* The tag name of this struct is _IO_FILE to preserve historic
   C++ mangled names for functions taking FILE* arguments.
   That name should not be used in new code. */
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
};
```

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.

Программа 1.

Код программы:

```
#include <fcntl.h>
#include <stdio.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:

```
(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]  
$ ./fProg.exe  
Aubvcwdxeyfzghijklmnopqrst
```

Код программы с потоками:

```
#include <fcntl.h>  
#include <stdio.h>  
#include <pthread.h>  
  
void *threadFunction(void *gotFs)  
{  
    int flag = 1;  
    char c;  
    while (flag == 1)  
    {  
        flag = fscanf(gotFs, "%c", &c);  
  
        if (flag == 1)  
            fprintf(stdout, "\033[34m[additive: %c]\033[0m ", c);  
    }  
}  
  
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);  
  
    pthread_t additiveThread;  
    pthread_create(&additiveThread, NULL, threadFunction, fs1);  
  
    int flag = 1;  
    char c;  
    while (flag == 1)  
    {  
        flag = fscanf(fs2, "%c", &c);  
  
        if (flag == 1)  
            fprintf(stdout, "\033[32m[main: %c]\033[0m ", c);  
    }  
}
```

```

    }

    pthread_join(additiveThread, NULL);

    return 0;
}

```

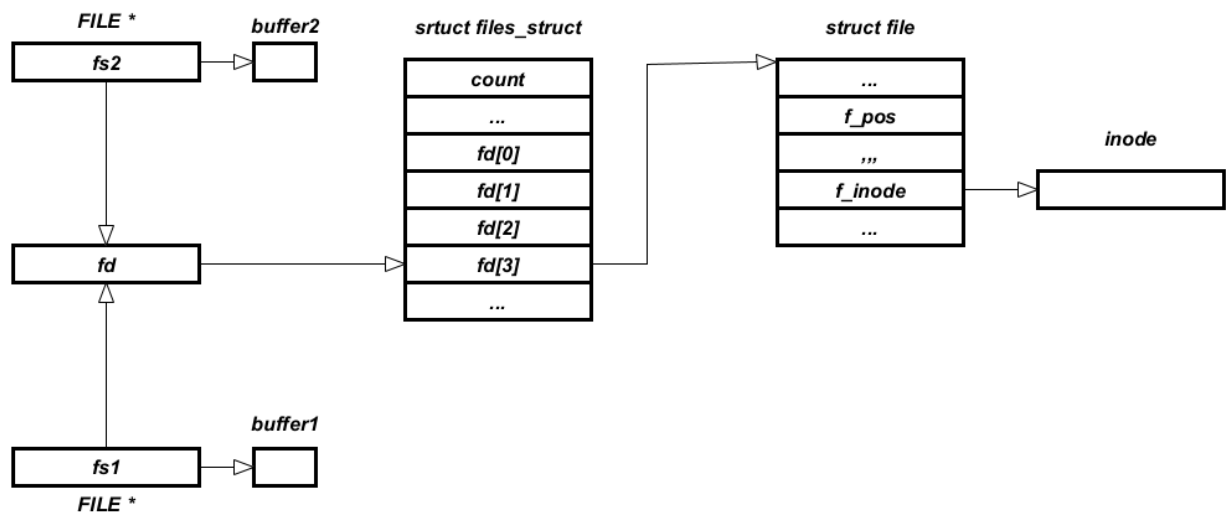
Результат выполнения:

```

(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ ./fProgT.exe
[main: A] [additive: u] [main: b] [additive: v] [main: c] [main: d] [additive: w] [main: e] [additive:
x] [main: f] [main: g] [additive: y] [main: h] [additive: z] [main: i] [main: j] [main: k] [main: l]
[main: m] [main: n] [main: o] [main: p] [main: q] [main: r] [main: s] [main: t]

```

Созданные дескрипторы и связь между ними



Анализ

Системный вызов `open()` создаёт и возвращает дескриптор открытого файла “alphabet.txt”. Также данный вызов создаёт запись в таблице открытых файлов, в которой указаны флаги и смещение в файле. Данный файл открыт только на чтение — `O_RDONLY`. Данному файловому дескриптору присваивается значение 3 — наименьший дескриптор, который ещё не открыт процессом.

В дальнейшем с использованием вызова `fdopen()` будет создано 2 структуры `FILE`, которые будут связаны с `fd`.

Далее с использованием функции `setbuf()` создаётся два буфера размером в 20 байт для созданных структур `FILE`. Также задаётся режим использования буфера — полная буферизация (Допустимые значения для `mode` — это `_IOFRF` (буферизация потока производится построчно - это значит, что очистка буфера производится каждый раз, когда в буфер для потока вывода записывается символ «новая строка»), `_IONBF` (полная буферизация) и `_IOLBF` (поток не имеет буферов, независимо от значения `buf`)).

При первом вызове `fscanf()` для `fs1` в структуре `struct_file` (для `fd`) значение `f_pos` будет увеличено на 20 единиц, так как для заполнения буфера `fs1` потребуется 20 байт – считается 20 символов. Далее остальные символы будут считаны в буфер `fs2`.

Для последующих вызовов `fscanf()` символы будут уже изыматься из заполненных буферов, пока те не станут пустыми.

Программа 2.

Код программы:

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

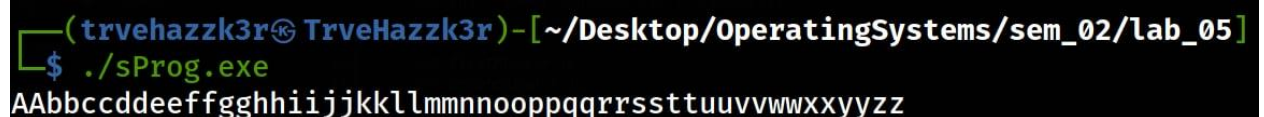
int main()
{
    int fd1 = open("alphabet.txt", O_RDONLY);
    int fd2 = open("alphabet.txt", O_RDONLY);

    int firstRead = 1;
    int secondRead = 1;
    char c;
    while (firstRead == 1 || secondRead == 1)
    {
        firstRead = read(fd1, &c, 1);
        if (firstRead == 1)
            write(1, &c, 1);

        secondRead = read(fd2, &c, 1);
        if (secondRead == 1)
            write(1, &c, 1);
    }

    return 0;
}
```

Результат выполнения:



```
(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ ./sProg.exe
AAbbccddeeffgghhiijjkkllmmnnooppqrrssttuuvvwwxxyyzz
```

Код программы с потоками:

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

void *threadFunction(void *gotFd)
```

```

{
    int fd = *((int *)gotFd);

    int gotRead = 1;
    char c;
    while (gotRead == 1)
    {
        gotRead = read(fd, &c, 1);
        if (gotRead == 1)
            write(1, &c, 1);
    }
}

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

    pthread_t thread;
    pthread_create(&thread, NULL, threadFunction, &fd1);

    int gotRead = 1;
    char c;
    while (gotRead == 1)
    {
        gotRead = read(fd2, &c, 1);
        if (gotRead == 1)
            write(1, &c, 1);
    }

    pthread_join(thread, NULL);

    return 0;
}

```

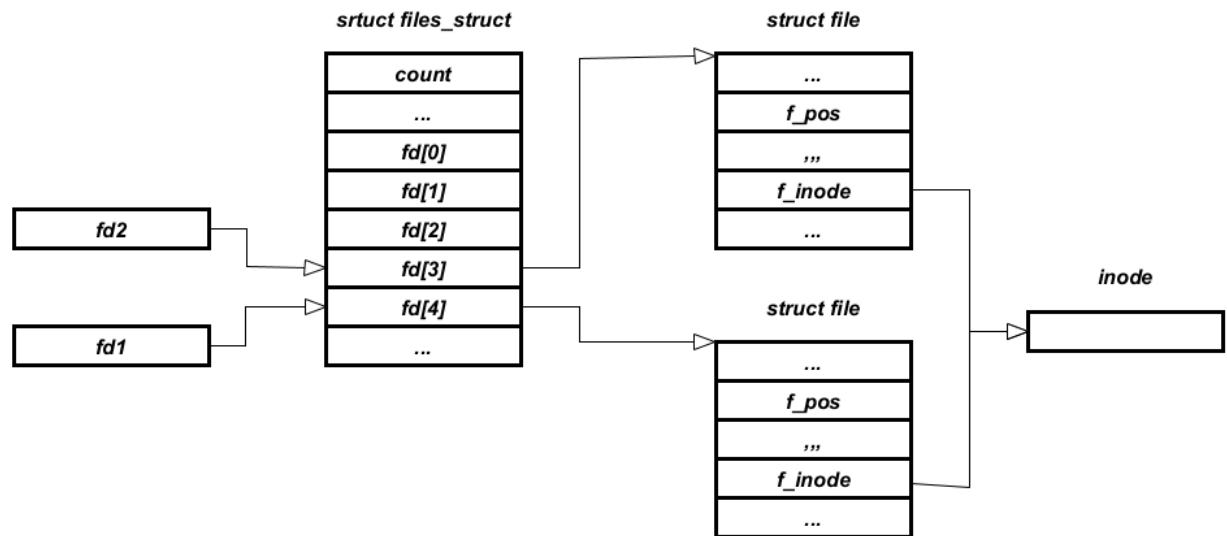
Результат выполнения:

```

(trvehazzk3r® TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ ./sProgT.exe
AbcdefghijkAlbmcndoe pfqgrhsitjukv lwxnyozpqrstuvwxyz

```

Созданные дескрипторы и связь между ними



Анализ

С использованием системного вызова `open()` создаётся два файловых дескриптора для одного текстового файла. Данный файл открыт только на чтение – `O_RDONLY`. Таким образом, в программе присутствует два различных `struct file`, которые ссылаются на один `struct inode`.

Так как в программе присутствуют две различные `struct file` – в реализации с единственным потоком каждый символ будет выведен с повторением. В реализации с двумя потоками символы будут идти в разном порядке, но будет прослеживаться «алфавитная последовательность», полученная от каждого потока.

Программа 3.

Написать программу, которая открывает один и тот же файл два раза с использованием библиотечной функции `fopen()`. Для этого объявляются два файловых дескриптора. В цикле записать в файл буквы латинского алфавита поочередно передавая функции `fprintf()` то первый дескриптор, то – второй.

Результат прокомментировать.

Код программы:

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

int main()
{
    FILE *f1 = fopen("out.txt", "w");
    FILE *f2 = fopen("out.txt", "w");
```



```

    for (char curLetter = 'a'; curLetter < '{'; curLetter++)
        curLetter % 2 ? fprintf(f1, "%c", curLetter) : fprintf(f2, "%c", curLetter);

    fclose(f2);
    fclose(f1);

    return 0;
}

```

Результат работы программы:

```

(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ make tProg
gcc tProg.c -o tProg.exe

(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ ./tProg.exe

(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ cat out.txt
acegikmoqsuwy

```

Код программы с потоками:

```

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

void *threadFunction(void *file)
{
    for (char curLetter = 'a'; curLetter < '{'; curLetter += 2)
        fprintf(file, "%c", curLetter);
}

int main()
{
    FILE *f1 = fopen("out.txt", "w");
    FILE *f2 = fopen("out.txt", "w");

    pthread_t thread;
    pthread_create(&thread, NULL, threadFunction, f1);

    for (char curLetter = 'b'; curLetter < '{'; curLetter += 2)
        fprintf(f2, "%c", curLetter);

    pthread_join(thread, NULL);

    fclose(f1);
}

```

```

    fclose(f2);

    return 0;
}

```

Результат работы программы:

```

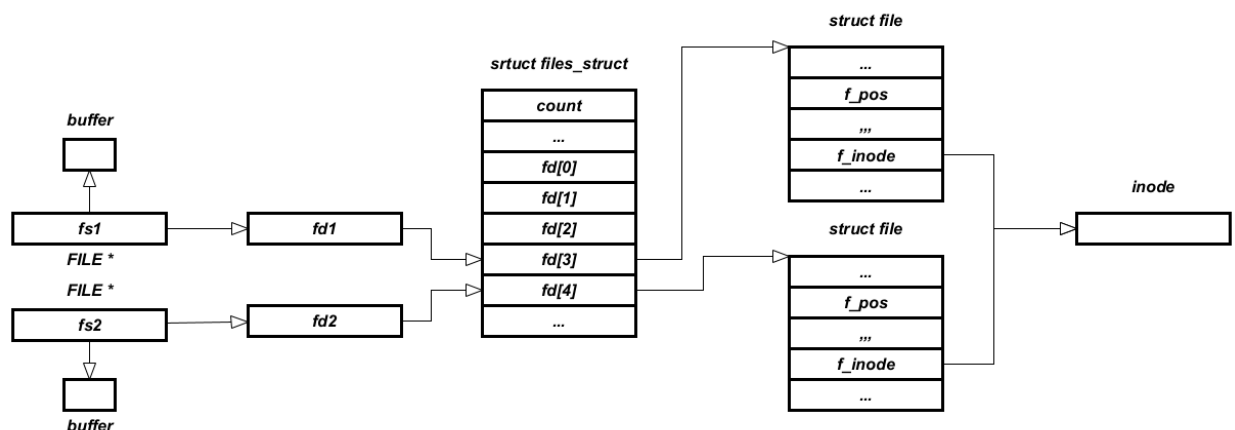
(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ make tProgT
gcc tProgThreading.c -pthread -o tProgT.exe

(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ ./tProgT.exe

(trvehazzk3r@TrveHazzk3r)-[~/Desktop/OperatingSystems/sem_02/lab_05]
$ cat out.txt
bdfhjlnprtvxz

```

Созданные дескрипторы и связь между ними



Анализ

С использованием функции `fopen()` файл `out.txt` «открывается» для записи 2 раза. Функция `fprintf()` (функция записи в файл) самостоятельно создаёт буфер, в который заносимая в файл информация первоначально и помещается. Информация будет записана в файл в трёх случаях: если буфер **полон** или если вызваны функции `fflush()` (приводит к физической записи содержимого буфера в файл) или `fclose()` (сохраняет в файл данные, находящиеся в буфере, и выполняет операцию системного уровня по закрытию файла). В анализируемой программе содержимое буферов будет записано при вызове функций `fclose()`. Так как в программе существует два различных дескриптора и их `f_pos` независимы и с самого начала установлены в начало файла, то и запись в файл с каждый `fclose()` будет происходить с начала файла. Именно поэтому информация, записанная при вызове `fclose(f1)` будет утеряна при

вызове `fclose(f2)` в первой реализации. Так как во второй реализации мною намеренно была изменена последовательность этих вызовов, можно убедиться в том, что, если вторым вызовом будет стоять `fclose(f2)`, то в таком случае будут потеряны изменения, внесённые вызовом `fclose(f1)`.