



Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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 по дисциплине «Операционные системы»

Тема Буферизованный и небуферизованный ввод-вывод

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-62Б

Преподаватель Рязанов Н.Ю.

Москва, 2025

1 Структура FILE

Листинг 1.1 — Описание FILE

```
// /usr/include/bits/types/FILE.h
typedef struct _IO_FILE FILE;

// /usr/include/bits/types/struct_FILE.h
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 _flags2:24;
    /* Fallback buffer to use when malloc fails to allocate one. */
    char _short_backupbuf[1];
    __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;
#ifdef
    __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;
    struct _IO_FILE **_prevchain;
    int _mode;
    /* Make sure we don't get into trouble again. */
    char _unused2[15 * sizeof (int) - 5 * sizeof (void *)];
};

```

2 Программа №1

2.1 Код

Листинг 2.1 — Код первой программы

```
#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 = 1;
    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;
}
```

2.2 Результат работы программы

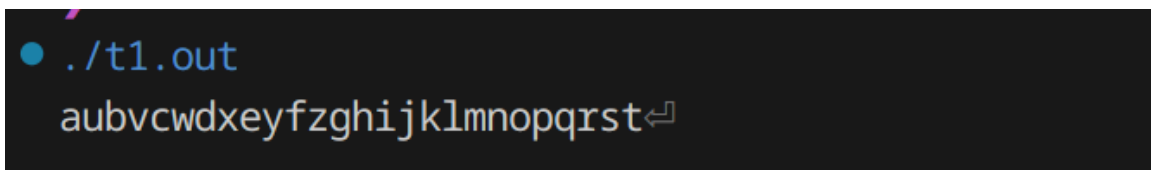


Рисунок 2.1 — Результат работы программы 1

2.3 Анализ работы программы

Системный вызов `open` в программе 1 открывает файл `«alphabet.txt»` в режиме «только для чтения» (`O_RDONLY`), создаёт дескриптор открытого файла в таблице открытых процессом файлом (поле `fdtab` структуры `files`) и возвращает индекс в этой таблице. Первые 3 индекса заняты файлами стандартных потоков ввода/вывода (`stdin`, `stdout`, `stderr`), поэтому открытый файл кладётся по индексом 3, который и возвращает системный вызов.

Функция `fdopen` инициализирует структуру `FILE` со значением поля `_fileno` равным дескриптору открытого файла `fd`. Функция выполняется два раза, возвращая указатели на проинициализированные структуры (`fs1`, `fs2`).

Функция `setvbuf` устанавливает буфер и его размер в 20 байт в структуре `FILE`.

В цикле вызывается функция `fscanf` для `fs1` и `fs2`. При этом на первой итерации цикла функция `fscanf` записывает первые 20 символов («a»–«t») из файла в буфер структуры `fs1` и смещает указатель в структуре открытого файла `file` на 20. Далее функция запишет «a» в переменную `s` и сместит указатель в структуре `fs` на 1. Так как `fs2` ссылается на тот же открытый файл, что и `fs1`, а его указатель уже был смещён на 20 байт, то первый вызов `fscanf` для структуры `fs2` запишет оставшиеся символы («u» – «z») из файла в буфер структуры `fs2`. При этом в переменную `s` запишется символ «u».

Далее поочерёдно будут выводиться символы из буферов `fs1` и `fs2`, при этом при достижении конца одного из буферов, будут выводиться символы из оставшегося, до конца последнего.

2.4 Схема связи структур

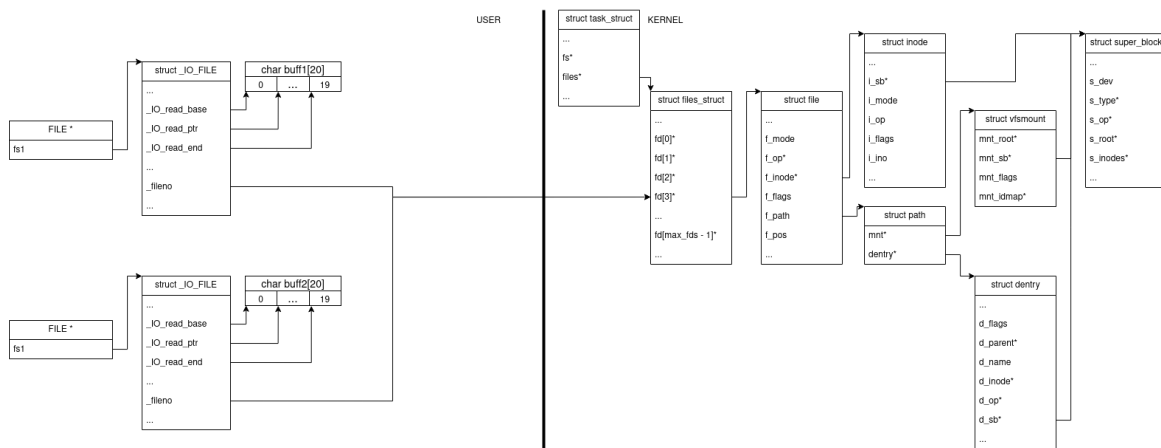


Рисунок 2.2 — Схема связи структур для программы 1

3 Программа №2, вариант 1

3.1 Однопоточная версия

3.1.1 Код

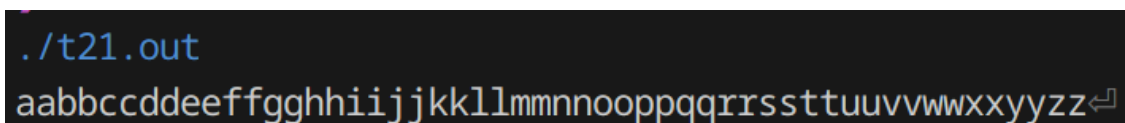
Листинг 3.1 — Код второй программы, вариант 1, однопоточная версия

```
#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 fl1 = 1, fl2 = 1;
    while(fl1 || fl2)
    {
        fl1 = read(fd1, &c, 1);
        if (fl1 == 1) write(1, &c, 1);
        fl2 = read(fd2, &c, 1);
        if (fl2 == 1) write(1, &c, 1);
    }
    return 0;
}
```

3.1.2 Результат работы программы



```
./t21.out
aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz↵
```

Рисунок 3.1 — Результат работы однопоточной версии программы 2, вариант 1

3.1.3 Анализ работы программы

Во второй программе двумя системными вызовами `open()` будут созданы два дескриптора открытых файлов в режиме «только для чтения» (`O_RDONLY`), которые ссылаются на один файл «`alphabet.txt`», а значит на один и тот же `inode`. Однако в таблице открытых файлов

инициализируются две структуры `struct file` с индексами 3 и 4 (`fd1` и `fd2`), которые имеют независимые поля `f_pos`.

Далее программа считывает первый символ («а») из открытого файла с дескриптором `fd1`, смещая указатель `f_pos` в соответствующей структуре `file` на 1, и выводит его. Затем также считывается символ из открытого файла с дескриптором `fd2`, так как структура `file` с дескриптором `fd2` не связан с структурой `file` с дескриптором `fd1`, то значение её поля `f_pos` по прежнему равно нулю и соответственно также считывается первый символ.

Поочерёдное чтение и вывод символов продолжаютсся, пока об открытых файла не дойдут до конца, в результате чего каждая буква алфавита будет выведена на экран дважды.

3.2 Версия с одним `detached` потоком

3.2.1 Код

Листинг 3.2 — Код второй программы, вариант 1, версия с одним `detached` потоком

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

void *thread_func(void *arg)
{
    int fd = *(int *)arg;
    char c;
    int fl = 1;
    while(fl)
    {
        fl = read(fd, &c, 1);
        if (fl == 1) write(1, &c, 1);
    }
    return NULL;
}

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

    pthread_attr_t attr;
    pthread_attr_init(&attr);
```



```

pthread_attr_t attr;
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

pthread_t t;
if (pthread_create(&t, &attr, thread_func, &fd1)) {
    perror("pthread_create");
    exit(1);
}

char c;
int fl = 1;
while(fl)
{
    fl = read(fd2, &c, 1);
    if (fl == 1) write(1, &c, 1);
}
return 0;
}

```

3.2.2 Результат работы программы

```

• ./t21main.out
abcdefghijklmnopqrstuvwxyzpqr↵

~/.Projects/Operating-System/6thsem/lab7 main*
>
• ./t21main.out
abcdefghijklmnopghicjkdelfmgnhoipjqkrslmtnuovpwqxrysztuv↵

~/.Projects/Operating-System/6thsem/lab7 main*
>
• ./t21main.out
abcdeafbgchdiejfkglhminjokplqmnrnsotpuqvrwsxytyuzvwxy↵

```

Рисунок 3.2 — Результат работы версии с одним detached потоком программы 2, вариант 1

3.2.3 Анализ работы программы

В данной версии программы также инициализируются две структуры `file` системными вызовами `open`. Один из дескрипторов использует главный поток, а второй передаётся допол-

нительному потоку как аргумент.

Главный поток создаёт дополнительный в detached режиме и начинает посимвольно читать открытый файл с дескриптором fd1. Так как на создание потока требуется время, то главный поток успевает прочесть и вывести часть символов. Далее главный и дополнительный потоки поочерёдно читают и выводят символы, при этом когда главный поток заканчивает чтение, процесс завершается и дополнительный поток не успевает дочитать до конца файла.

3.3 Версия с двумя detached потоками

3.3.1 Код

Листинг 3.3 — Код второй программы, вариант 1, версия с двумя detached потоками

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

void *thread_func(void *arg)
{
    int fd = *(int *)arg;
    char c;
    int fl = 1;
    while(fl)
    {
        fl = read(fd, &c, 1);
        if (fl == 1) write(1, &c, 1);
    }
    return NULL;
}

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

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    pthread_t t1, t2;
```

```

    if (pthread_create(&t1, &attr, thread_func, &fd1)) {
        perror("pthread_create");
        exit(1);
    }
    if (pthread_create(&t2, &attr, thread_func, &fd2)) {
        perror("pthread_create");
        exit(1);
    }

    return 0;
}

```

3.3.2 Результат работы программы

```

• ./t21sub.out
  abcdefghi↵

~/Projects/Operating-System/6thsem/lab7 main*
>
• ./t21sub.out
  abcdefgh↵

~/Projects/Operating-System/6thsem/lab7 main*
>
• ./t21sub.out
  abcdefgh↵

~/Projects/Operating-System/6thsem/lab7 main*
>
• ./t21sub.out
  a↵

```

Рисунок 3.3 — Результат работы версии с двумя detached потоками программы 2, вариант 1

3.3.3 Анализ работы программы

В данной версии главный поток завершается как только создаёт два дополнительных потока, не дожидаясь их окончания, при этом один из дополнительных потоков или оба могут успеть прочитать часть символов из файла, в зависимости от порядка планирования и выполне-

3.4 Версия с двумя потоком с join

3.4.1 Код

Листинг 3.4 — Код второй программы, вариант 1, версия с двумя потоками с join

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

void *thread_func(void *arg)
{
    int fd = *(int *)arg;
    char c;
    int fl = 1;
    while(fl)
    {
        fl = read(fd, &c, 1);
        if (fl == 1) write(1, &c, 1);
    }
    return NULL;
}

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

    pthread_t t1, t2;
    if (pthread_create(&t1, NULL, thread_func, &fd1)) {
        perror("pthread_create");
        exit(1);
    }
    if (pthread_create(&t2, NULL, thread_func, &fd2)) {
        perror("pthread_create");
        exit(1);
    }
}
```

```

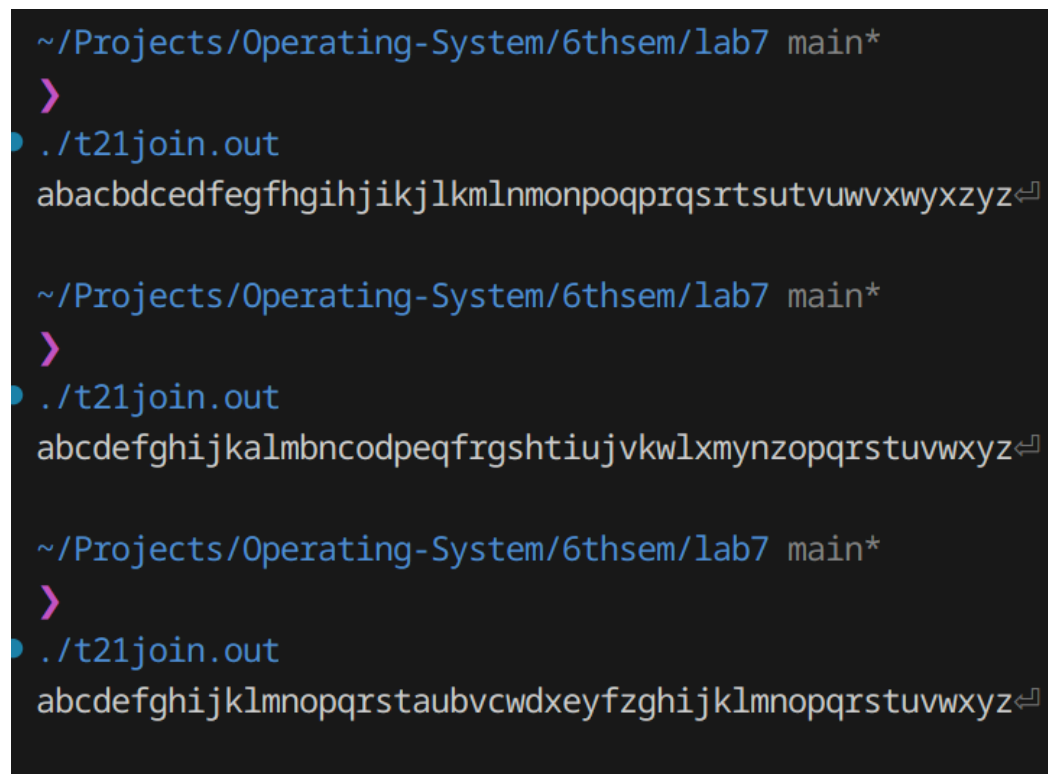
err = pthread_join(t1, NULL);
if (err != 0) {
    perror("pthread_join");
    exit(1);
}

err = pthread_join(t2, NULL);
if (err != 0) {
    perror("pthread_join");
    exit(1);
}

return 0;
}

```

3.4.2 Результат работы программы



```

~/Projects/Operating-System/6thsem/lab7 main*
>
./t21join.out
abacbdcedfegfhgihjikjklmnlmonpoqprqsrtsutvuwvwxwxyz↵

~/Projects/Operating-System/6thsem/lab7 main*
>
./t21join.out
abcdefghijklmbncodpeqfrgshtiujvkwlxmynzopqrstuvwxyz↵

~/Projects/Operating-System/6thsem/lab7 main*
>
./t21join.out
abcdefghijklmnopqrstaubvcwdxeyfzghijklmnopqrstuvwxyz↵

```

Рисунок 3.4 — Результат работы версии с двумя потоками с join программы 2, вариант 1

3.4.3 Анализ работы программы

В данной версии программы также создаются два дополнительных потока но в состоянии `attached` и главный поток «подсоединяет» их к себе (`pthread_join()`), тем самым дожидаясь их окончания. Дополнительные потоки будут поочерёдно читать символы из файла и выводить их на экран, при этом порядок чтения зависит от порядка выполнения потоков в системе.

4 Программа №2, вариант 2

4.1 Однопоточная версия

4.1.1 Код

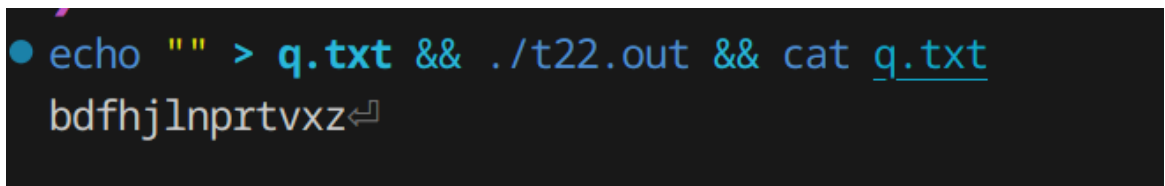
Листинг 4.1 — Код второй программы, вариант 2, однопоточная версия

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

int main()
{
    int fd1 = open("q.txt", O_RDWR);
    int fd2 = open("q.txt", O_RDWR);

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c%2){
            write(fd1, &c, 1);
        }
        else{
            write(fd2, &c, 1);
        }
    }
    close(fd1);
    close(fd2);
    return 0;
}
```

4.1.2 Результат работы программы



```
• echo "" > q.txt && ./t22.out && cat q.txt
bdfhjlnprtvxz
```

Рисунок 4.1 — Результат работы однопоточной версии программы 2, вариант 2

4.1.3 Анализ работы программы

В результате двух системных вызовов `open()` будут инициализированы две структуры `file` в таблице открытых файлов процесса с индексами 3 и 4 (`fd1` и `fd2`), которые ссылаются на один и тот же `inode` (файла «q.txt»), имеют режим на чтение и запись (`O_RDWR`), но будут иметь независимые поля `f_pos`.

При первом вызове `write` в открытый файл с индексом `fd2` (так как «а» = 97, $97 \% 2 = 1$) будет записан символ «а» и указатель `f_pos` будет смещён на 1. Вторым вызовом в открытый файл с индексом `fd1` будет записан символ «b», но так как его указатель `f_pos` все ещё равен 0, то он перезапишет символ «а». Таким образом в файл будут записаны только символы, стоящие на чётных позициях в алфавите.

Чтобы избежать потерю данных, можно открыть файл с флагом `O_APPEND` (`O_RDWR | O_APPEND`), который указывает на то, что нужно всегда писать в конец файла. Таким образом «а» запишется позицию 0, «b» на последнюю позицию на тот момент, то есть 1 и так далее.

4.2 Версия с одним detached потоком

4.2.1 Код

Листинг 4.2 — Код второй программы, вариант 2, версия с одним detached потоком

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

struct thread_arg {
    int fd;
    int i;
};

void *thread_func(void *arg)
{
    struct thread_arg *args = (struct thread_arg *)arg;
    int fd = args->fd;
    int i = args->i;

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2 == i) {
            write(fd, &c, 1);
        }
    }
}
```



```

        }
    }
    return NULL;
}

int main()
{
    int fd1 = open("q.txt",O_RDWR);
    int fd2 = open("q.txt",O_RDWR);

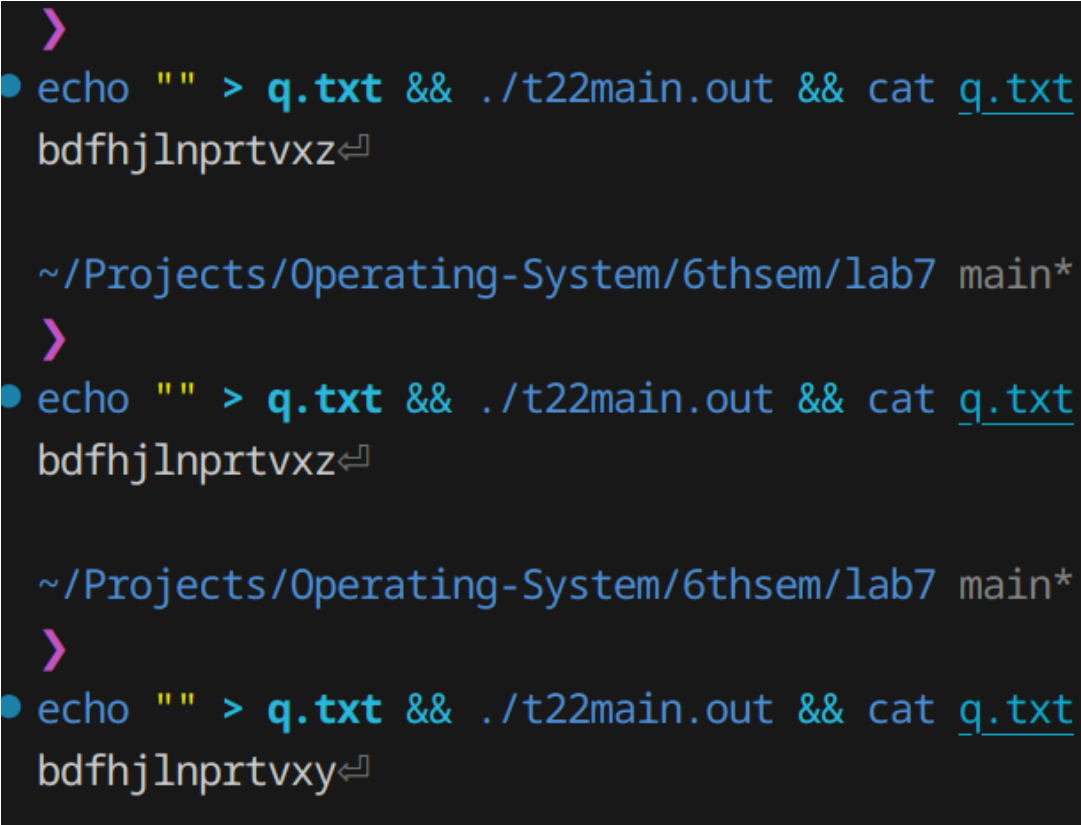
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    struct thread_arg args;
    args.fd = fd1;
    args.i = 0;
    pthread_t t1;
    if (pthread_create(&t1, &attr, thread_func,&args)) {
        perror("pthread_create");
        exit(1);
    }

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c%2 == 1){
            write(fd2, &c, 1);
        }
    }
    close(fd1);
    close(fd2);
    return 0;
}

```

4.2.2 Результат работы программы



```
>  
• echo "" > q.txt && ./t22main.out && cat q.txt  
bdfhjlnprtvxz  
  
~/Projects/Operating-System/6thsem/lab7 main*  
>  
• echo "" > q.txt && ./t22main.out && cat q.txt  
bdfhjlnprtvxz  
  
~/Projects/Operating-System/6thsem/lab7 main*  
>  
• echo "" > q.txt && ./t22main.out && cat q.txt  
bdfhjlnprtvxy
```

Рисунок 4.2 — Результат работы версии с одним detached потоком программы 2, вариант 2

4.2.3 Анализ работы программы

В данной версии программы также инициализируются две структуры `file` системными вызовами `open()` с режимом для чтения и записи. Один из дескрипторов использует главный поток, а второй передаётся дополнительному как аргумент.

Главный поток создаёт дополнительный в режиме `detached`, после чего начинает поочередно записывать символы алфавита, стоящие на чётных позициях. Так как на создание потока требуется время, то главный поток успевает записать часть символов в файл. Далее потоки поочередно записывают символы в файл, при этом так как потоки используют разные индексы открытых файлов, то дополнительный поток перезаписывает символы главного потока. Как только главный поток завершает цикл, процесс завершается, из-за чего дополнительный поток может не закончить записывать свои символы и в итоге в файле будут часть символов на чётных позициях алфавита, а часть – нечётных.

4.3 Версия с двумя detached потоками

4.3.1 Код

```

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

struct thread_arg {
    int fd;
    int i;
};

void *thread_func(void *arg)
{
    struct thread_arg *args = (struct thread_arg *)arg;
    int fd = args->fd;
    int i = args->i;

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2 == i) {
            write(fd, &c, 1);
        }
    }
    return NULL;
}

int main()
{
    int fd1 = open("q.txt", O_RDWR);
    int fd2 = open("q.txt", O_RDWR);

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    struct thread_arg args1;
    args1.fd = fd1;
    args1.i = 0;
    pthread_t t1, t2;
    if (pthread_create(&t1, &attr, thread_func, &args1)) {

```

```

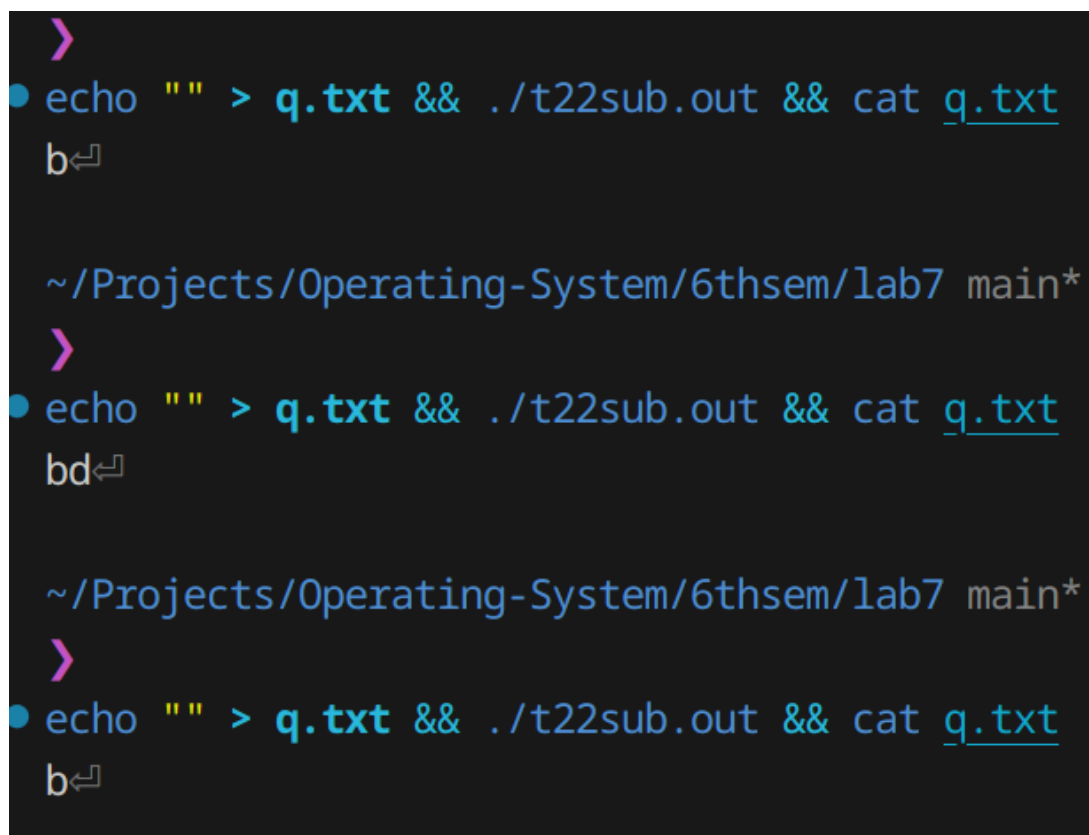
    perror("pthread_create");
    exit(1);
}

struct thread_arg args2;
args2.fd = fd2;
args2.i = 1;
if (pthread_create(&t2, &attr, thread_func, &args2)) {
    perror("pthread_create");
    exit(1);
}

close(fd1);
close(fd2);
return 0;
}

```

4.3.2 Результат работы программы



```

>
• echo "" > q.txt && ./t22sub.out && cat q.txt
b

~/Projects/Operating-System/6thsem/lab7 main*
>
• echo "" > q.txt && ./t22sub.out && cat q.txt
bd

~/Projects/Operating-System/6thsem/lab7 main*
>
• echo "" > q.txt && ./t22sub.out && cat q.txt
b

```

Рисунок 4.3 — Результат работы версии с двумя detached потоками программы 2, вариант 2

4.3.3 Анализ работы программы

В данной версии главный поток создаёт два дополнительных потока в `detached` режиме, то есть не дожидаясь их завершения. Поэтому оба потока могут успеть что-то записать в файл или не успеть, до завершения главного потока.

4.4 Версия с двумя потоком с `join`

4.4.1 Код

Листинг 4.4 — Код второй программы, вариант 2, версия с двумя потоками с `join`

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

struct thread_arg {
    int fd;
    int i;
};

void *thread_func(void *arg)
{
    struct thread_arg *args = (struct thread_arg *)arg;
    int fd = args->fd;
    int i = args->i;

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c % 2 == i) {
            write(fd, &c, 1);
        }
    }
    return NULL;
}

int main()
{
    int fd1 = open("q.txt", O_RDWR);
    int fd2 = open("q.txt", O_RDWR);
```

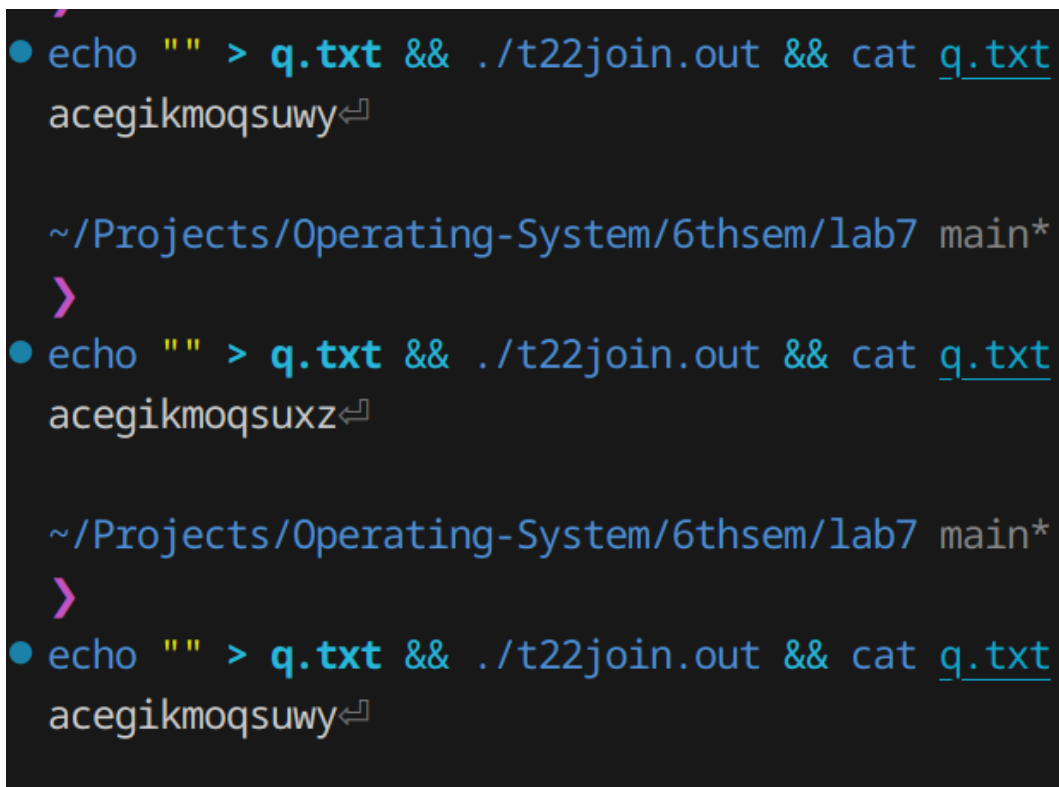
```
struct thread_arg args1;
args1.fd = fd1;
args1.i = 0;
pthread_t t1, t2;
if (pthread_create(&t1, NULL, thread_func, &args1)) {
    perror("pthread_create");
    exit(1);
}

struct thread_arg args2;
args2.fd = fd2;
args2.i = 1;
if (pthread_create(&t2, NULL, thread_func, &args2)) {
    perror("pthread_create");
    exit(1);
}

pthread_join(t1, NULL);
pthread_join(t2, NULL);

close(fd1);
close(fd2);
return 0;
}
```

4.4.2 Результат работы программы



```
● echo "" > q.txt && ./t22join.out && cat q.txt
acegikmoqsuwy

~/Projects/Operating-System/6thsem/lab7 main*
>
● echo "" > q.txt && ./t22join.out && cat q.txt
acegikmoqsuxz

~/Projects/Operating-System/6thsem/lab7 main*
>
● echo "" > q.txt && ./t22join.out && cat q.txt
acegikmoqsuwy
```

Рисунок 4.4 — Результат работы версии с двумя потоками с join программы 2, вариант 2

4.4.3 Анализ работы программы

В данной версии программы два потока создаются в attached режиме, то есть главный поток дожидается их завершения «подсоединив» (pthread_join) к себе. Оба потока поочерёдно пишут в файл, при этом они перезаписывают друг друга и в файле остаётся то значение, который записал последний дошедший до этой позиции поток. Порядок записи определяется порядком выполнения потоков системой.

4.5 Схема связи структур

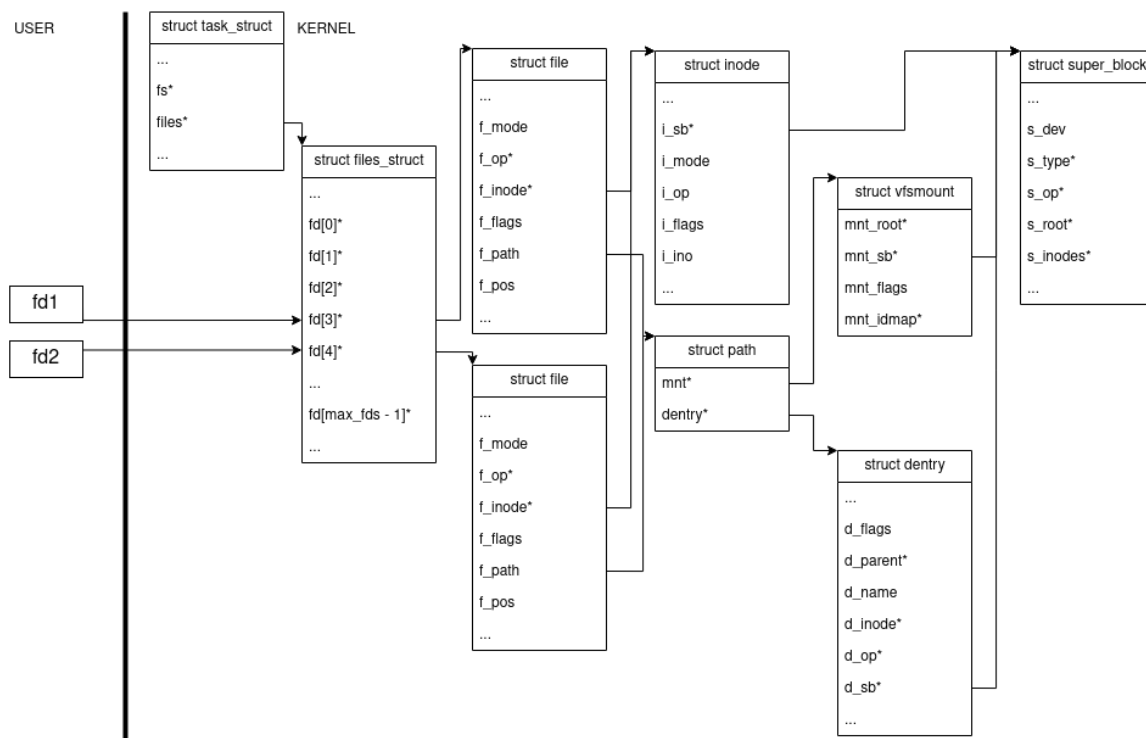


Рисунок 4.5 — Схема связи структур для программы 2

5 Программа №3

5.1 Код

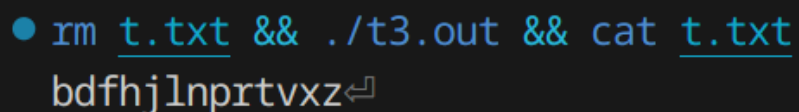
Листинг 5.1 — Код третьей программы

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

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

    for(char c = 'a'; c <= 'z'; c++)
    {
        if (c%2){
            fprintf(f1, "%c", c);
        }
        else{
            fprintf(f2, "%c", c);
        }
    }
    fclose(f1);
    fclose(f2);
    return 0;
}
```

5.2 Результат работы программы



```
● rm t.txt && ./t3.out && cat t.txt
bdfhjlnprtvxz
```

Рисунок 5.1 — Результат работы программы 3

5.3 Анализ работы программы

Файл t.txt дважды открывается функцией `open`, которая инициализирует две структуры `file` в таблице открытых файлов процесса, при этом обе этих структуры ссылаются на один и тот же `inode`, а также инициализирует две структуры `FILE` (`f1` и `f2`) с буферами символов для записи.

Так как используется буферизация, то содержимое, записанное в одну из структур `FILE` будет записано в файл либо при переполнении буфера, либо при вызове `fflush()`, либо при вызове `fclose()`.

В данном случае буферов по умолчанию (размеров в страницу – 4096 байт), хватает для записи в обе структуры `FILE` без переполнения. Поэтому содержимое файла определяется порядком вызова `fclose()`. Если первым вызвать `fclose` для `f1`, то при вызове `fclose()` для `f2` содержимое файла будет перезаписано содержимым буфера `f2` и наоборот, так как структуры `file` открытых файлов отличны для `f1` и `f2`.

Использование режима открытия файла «а», позволяет избежать потери данных, так как в таком случае запись каждый раз будет производиться в реальный конец файла.

5.4 Схема связи структур

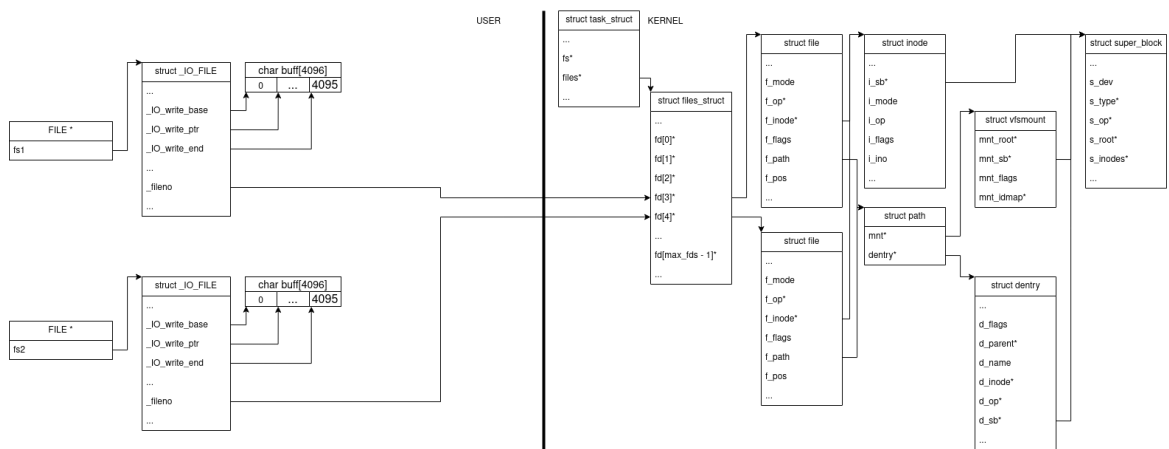


Рисунок 5.2 — Схема связи структур для программы 3