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

Название «Буферизованный и не буферизованный ввод-вывод»

Дисциплина «Операционные системы»

Студент ИУ7-65Б _____ Клименко А.К.
(подпись, дата) (Фамилия И.О.)

Преподаватель _____ Рязанова Н.Ю.
(подпись, дата) (Фамилия И.О.)

Москва, 2022

1 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.

Структура _IO_FILE

```
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;
```

```
21      struct _IO_FILE *_chain;
22
23      int _fileno;
24      int _flags2;
25      __off_t _old_offset; /* This used to be _offset but it's too small.  */
26
27      /* 1+column number of pbase(); 0 is unknown. */
28      unsigned short _cur_column;
29      signed char _vtable_offset;
30      char _shortbuf[1];
31
32      _IO_lock_t *_lock;
33  #ifdef _IO_USE_OLD_IO_FILE
34  };
35
36  struct _IO_FILE_complete
37  {
38      struct _IO_FILE _file;
39  #endif
40      __off64_t _offset;
41      /* Wide character stream stuff.  */
42      struct _IO_codecvt *_codecvt;
43      struct _IO_wide_data *_wide_data;
44      struct _IO_FILE *_freeres_list;
45      void *_freeres_buf;
46      size_t __pad5;
47      int _mode;
48      /* Make sure we don't get into trouble again.  */
49      char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
50  };
```

2 Первая программа

```c
/* testCIO.c */
#define _XOPEN_SOURCE 600
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <pthread.h>

void* task(void *fd)
{
    FILE *fp = fdopen( *(int *)fd, "r");

    char buffer[20];
    setvbuf(fp, buffer, _IOFBF, sizeof(buffer));

    for (char tmp; fscanf(fp, "%c", &tmp) == 1;)
        printf("%c", tmp);
    printf("\n");

    return NULL;
}

int main(void)
{
    int fd = open("alphabet.txt", O_RDONLY);
    pthread_t tid[2];

    if (pthread_create(&tid[0], NULL, task, (void *)&fd) != 0 ||
        pthread_create(&tid[1], NULL, task, (void *)&fd) != 0)
    {
        perror("pthread_create");
        return -1;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);

    close(fd);
    return 0;
}
```

2.1 Результат работы

```
$ ./testCIO
Abcdeklmnopqrstuvwxyfzghij
```

2.2 Обьяснение

Системный вызов `open` создает новый файловый дескриптор для файла, открытого только на чтение. Функция `open` своим именем возвращает индекс созданного дескриптора в таблице открытых файлов процесса (индекс в массиве `fd_array` структуры `files_struct`).

Функция `fdopen` создает структуру `_IO_FILE`, ссылающуюся на созданный ранее дескриптор открытого файла. Таким образом в программе создается 2 структуры `_IO_FILE`, ссылающихся на один дескриптор открытого файла.

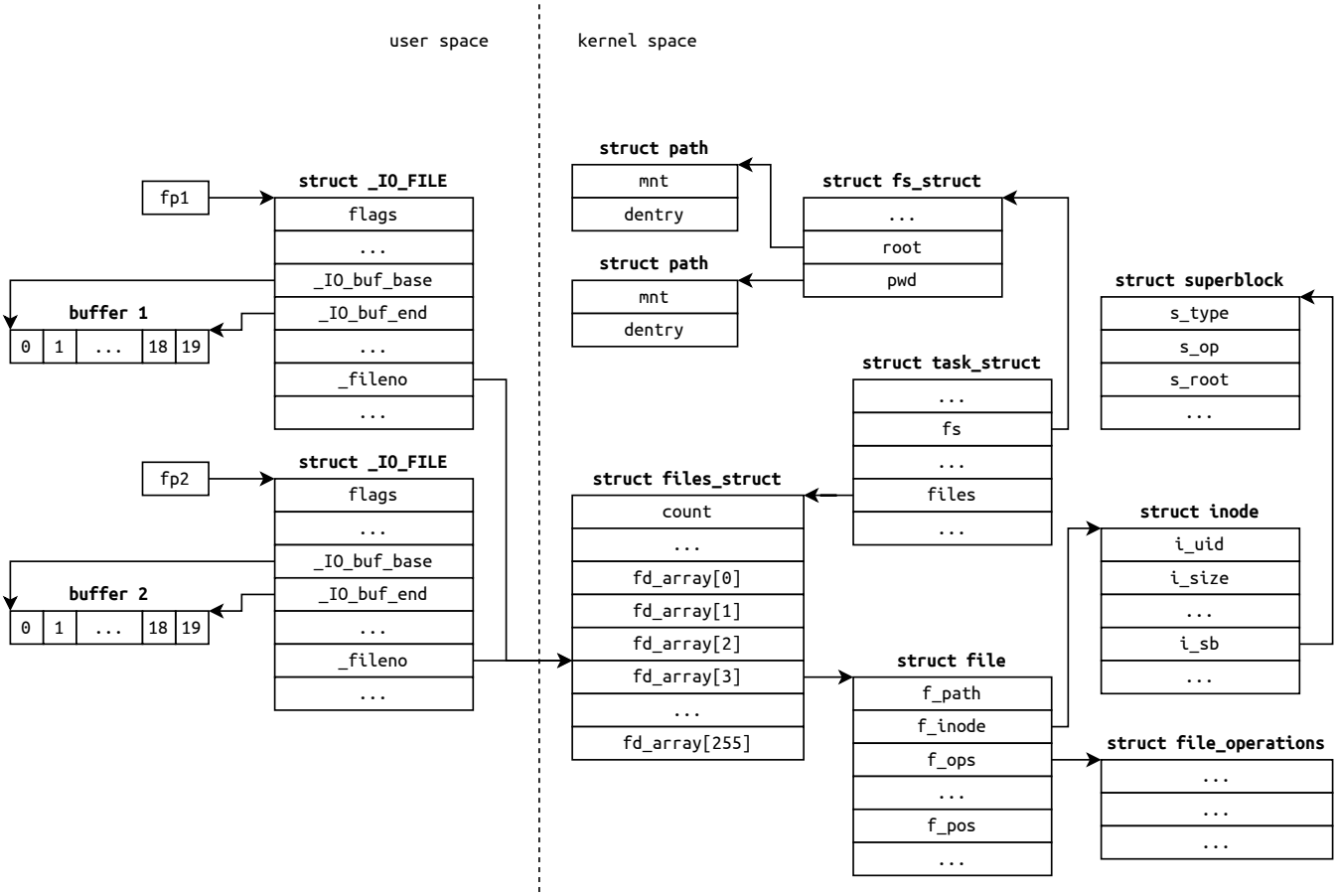
`setbuf` позволяет задать буферы и тип буферизации для структур `_IO_FILE`. Таким образом для каждого из потоков задаем буфер, состоящий из 20 элементов, и буферизацию `_IOFBF` (fully buffered).

Пусть первый поток первым начал чтение из файла с помощью библиотечной функции `fscanf(fp, ...)`. Поскольку `fscanf` – функция буферизованного ввода, буфер заполнится первыми 20 символами файла, указатель `f_pos` установится на следующий за последним считанным символом символ, после этого начнется считывание символов из буфера.

Если в этот момент первый поток потеряет квант и второй поток начнет выполнение, он также вызовет `fscanf(fp, ...)`, его буфер заполнится оставшимися в файле символами, `f_pos` будет указывать на конец файла.

Таким образом, если процессор будет поочередно выделять квант времени каждому из потоков, алфавит будет выведен в произвольном порядке с учетом того, что первые 20 символов читает первый поток, последующие – второй.

Связь структур



3 Вторая программа

```c
/* testKernelIO.c */
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <pthread.h>

void* task(void *data)
{
    int fd = open("alphabet.txt", O_RDONLY);

    for (char c; read(fd, &c, 1) == 1;)
        write(1, &c, 1);

    close(fd);
    return NULL;
}

int main(void)
{
    pthread_t tid1, tid2;
    if (pthread_create(&tid1, NULL, task, NULL) != 0 ||
        pthread_create(&tid2, NULL, task, NULL) != 0)
    {
        perror("pthread_create");
        return -1;
    }

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    write(1, "\n", 1);
    return 0;
}
```

3.1 Результат работы

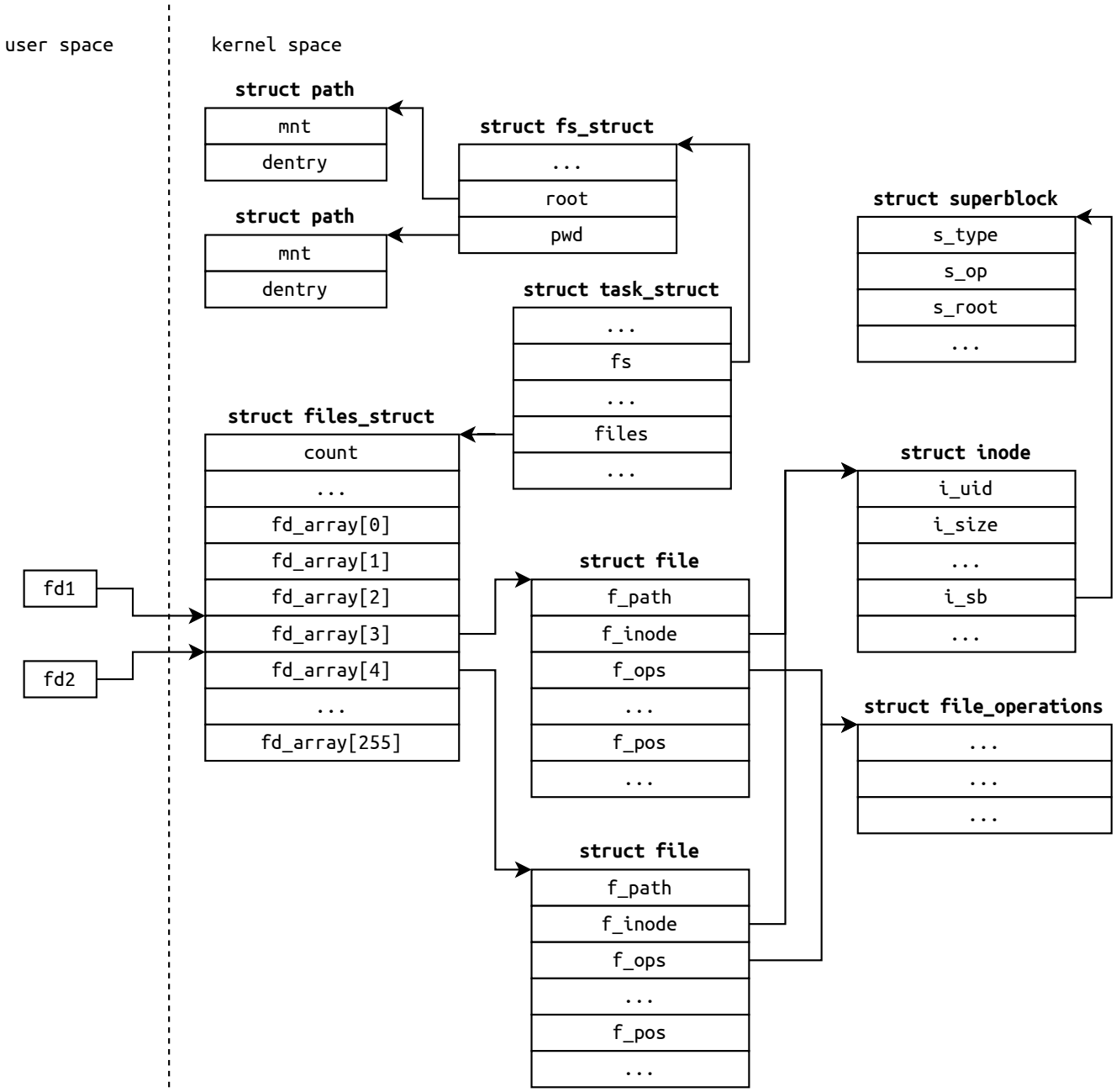
```
$ ./testKernelIO
AAbcbdcedfegfhgihjikjlkmlnmonpoqprqsrtsutvuwvxwyxzyz
```

3.2 Объяснение

Данная программа использует системные вызовы `read()` и `write()`, которые не буферизуют ввод/вывод. При каждом вызове `open()` создается новый дескриптор в таблице открытых файлов процесса, таким образом в каждом из потоков создастся свой дескриптор открытого файла, в связи с чем чтение будет происходить независимо (в каждом из потоков будет изменяться смещение в файле для отдельного дескриптора открытого файла).

Таким образом, каждый из потоков напечатает по порядку все символы из файла, полученный результат будет зависеть от того, как процессор выделит квант времени потокам.

Связь структур



4 Третья программа

```c
#define _XOPEN_SOURCE 600
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/stat.h>

void describe(FILE* fp, const char* prefix)
{
    int pos = ftell(fp);
    struct stat s;
    if (stat("output.txt", &s) != 0)
    {
        perror("stat");
        exit(-1);
    }
    printf("%s: inode = %ld, size = %ld, pos = %d\n", prefix, s.st_ino, s.st_size, pos);
}

void write_syms(FILE* fp, char begin, char end)
{
    while (begin != end)
        fprintf(fp, "%c", begin++);
}

void* task(void* data)
{
    FILE* fp = fopen("output.txt", "wt");
    describe(fp, "fopen");
    char buffer[10];
    setvbuf(fp, buffer, _IOFBF, sizeof(buffer));
    if (data == NULL)
        write_syms(fp, 'a', 'z' + 1);
    else
        write_syms(fp, '0', '9' + 1);
    describe(fp, "fclose");
    fclose(fp);
    return NULL;
}
```

```
39 int main(void)
40 {
41     pthread_t tid1, tid2;
42     if (pthread_create(&tid1, NULL, task, NULL) != 0 ||
43         pthread_create(&tid2, NULL, task, (void*)1) != 0)
44     {
45         perror("pthread_create");
46         return -1;
47     }
48
49     pthread_join(tid1, NULL);
50     pthread_join(tid2, NULL);
51     return 0;
52 }
```

4.1 Результат работы

```
1 $ ./testFFIO
2 fopen: inode = 28709680, size = 0, pos = 0
3 fopen: inode = 28709680, size = 20, pos = 0
4 fclose: inode = 28709680, size = 20, pos = 10
5 fclose: inode = 28709680, size = 20, pos = 26
6 $ cat output.txt
7 0123456789klmnopqrstuvwxyz
```

4.2 Объяснение

В данной программе каждый из потоков открывает один и тот же файл на запись. Поскольку используется функция `fopen()`, которая создает экземпляр структуры `_IO_FILE`, речь идет о буферизованном выводе. Для каждого из потоков создается свой дескриптор открытого файла в режиме записи. Один поток записывает в файл английский алфавит, другой – цифры от 0 до 9. Итоговый результат зависит от выделения процессором квантов времени потокам.

Возможны три ситуации:

1) В файле окажутся записаны только цифры. Такой результат говорит о том, что соответствующему потоку был выделен больший квант времени и он успел завершить все операции с файлом до того, как другой поток начал с ним работать, то есть первый вызвал `fclose()` раньше, чем второй вызвал `fopen()`.

В таком случае вызов функции `fopen()` вторым потоком сотрет всю записанную главным потоком информацию, она будет полностью утеряна, второй поток запишет свою информацию, закроет файл, данные запишутся из буфера в файл.

2) В файле начало алфавита окажется затерто цифрами. Такой результат говорит о том, что оба потока открыли файл на запись до того, как второй вызвал `fclose()`, и они начали писать в него одновременно. Так как часть символов алфавита затерта цифрами, первый поток вызвал `fclose()` раньше второго, записав всю информацию из буфера в файл. После этого `fclose()` вызвал второй поток, записал всю информацию из буфера в файл, чем затер записанную первым потоком информацию.

3) В файле записан только алфавит. Такой результат может быть следствием двух исходов:

a) Второй поток закончил работу с файлом раньше, чем с ним начал работать первый поток.

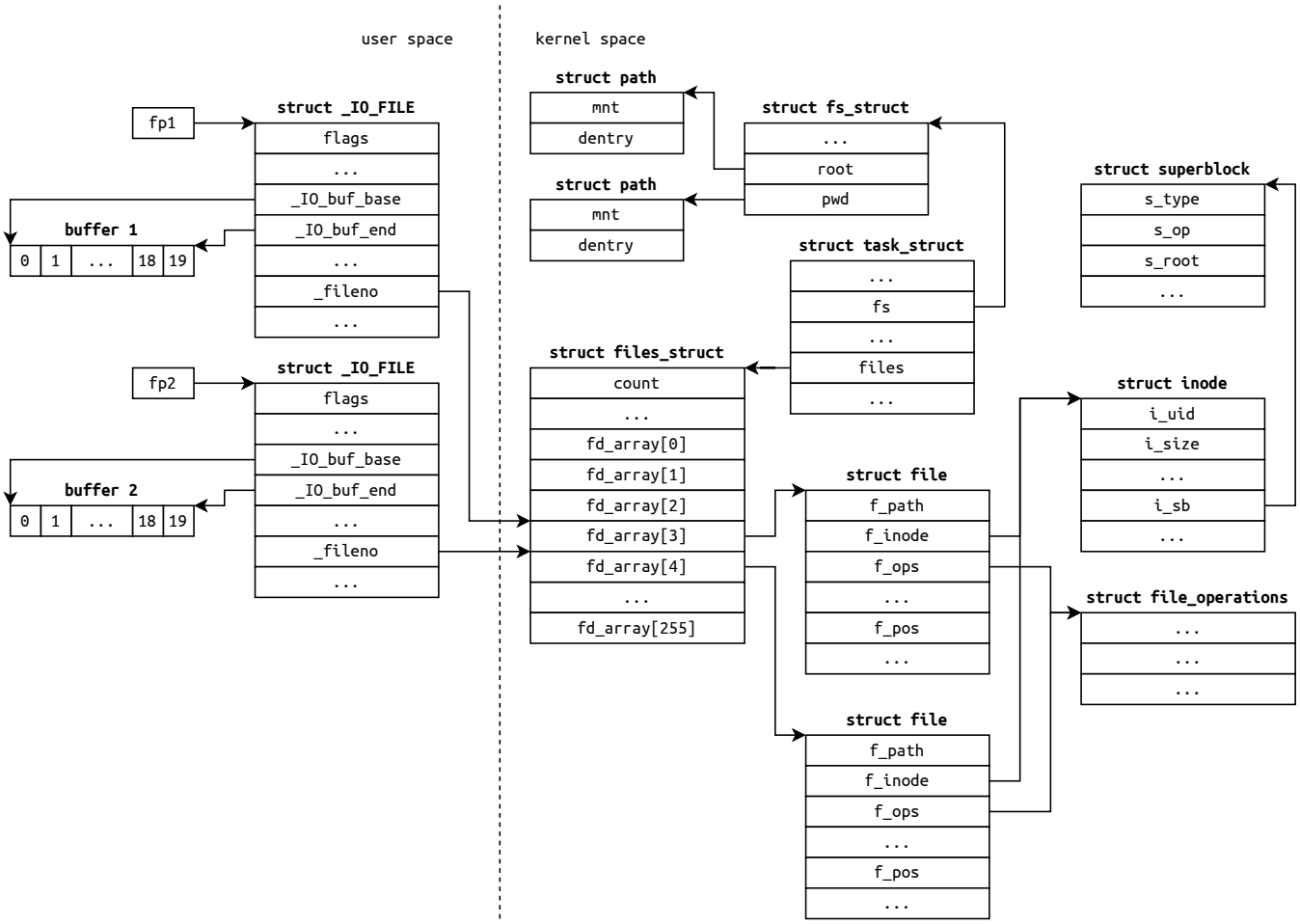
b) Второй поток писал в файл одновременно с первым потоком, но вызвал `fclose()` раньше, чем первый поток, в связи с чем при вызове `fclose()` первым потоком информация была полностью затерта.

Решить перечисленные выше проблемы можно двумя способами:

1) Открывать файл в режиме `O_APPEND`. В таком случае каждой операции гарантируется неделимость, перед каждым вызовом `write()` смещение устанавливается в конец файла.

2) Использовать `lseek()` и мьютексы для обеспечения монопольного доступа к файлам.

Связь структур



С использованием блокировок

```
1  #define _XOPEN_SOURCE 600
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <pthread.h>
5  #include <sys/stat.h>
6
7  pthread_mutex_t mutex;
8
9  void describe(FILE* fp, const char* prefix)
10 {
11     int pos = ftell(fp);
12     struct stat s;
13     if (stat("output.txt", &s) != 0)
14     {
15         perror("stat");
16         exit(-1);
17     }
```

```
18      printf("%s: inode = %ld, size = %ld, pos = %d\n", prefix, s.st_ino, s.st_size, pos);
19  }
20
21  void write_syms(FILE* fp, char begin, char end)
22  {
23      pthread_mutex_lock(&mutex);
24      fseek(fp, 0, SEEK_END);
25      while (begin != end)
26          fprintf(fp, "%c", begin++);
27      pthread_mutex_unlock(&mutex);
28  }
29
30  void* task(void* data)
31  {
32      FILE* fp = fopen("output.txt", "at");
33      describe(fp, "fopen");
34
35      char buffer[10];
36      setvbuf(fp, buffer, _IOFBF, sizeof(buffer));
37
38      if (data == NULL)
39          write_syms(fp, 'a', 'z' + 1);
40      else
41          write_syms(fp, '0', '9' + 1);
42
43      describe(fp, "fclose");
44      fclose(fp);
45      return NULL;
46  }
47
48  int main(void)
49  {
50      pthread_mutex_init(&mutex, NULL);
51      pthread_t tid1, tid2;
52      if (pthread_create(&tid1, NULL, task, NULL) != 0 ||
53          pthread_create(&tid2, NULL, task, (void*)1) != 0)
54      {
55          perror("pthread_create");
56          return -1;
57      }
58
59      pthread_join(tid1, NULL);
60      pthread_join(tid2, NULL);
61      return 0;
62  }
```

Результат работы

```
$ ./testFFIO && cat output.txt
fopen: inode = 28709680, size = 0, pos = 0
fopen: inode = 28709680, size = 0, pos = 0
fclose: inode = 28709680, size = 0, pos = 10
fclose: inode = 28709680, size = 30, pos = 36
0123456789abcdefghijklmnopqrstuvwxyz
```