Государственное образовательное учреждение высшего
профессионального образования
“Московский государственный технический университет имени
Н.Э.Баумана”

ОТЧЕТ

По лабораторной работе № 5

Студент: Мхитарян Виктория
Группа: ИУ7-64
Преподаватель: Рязанова Н. Ю.

2019 г.

Задача 1.

Код 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 = 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;
}
```

Анализ.

В данной части лабораторной работы рассматриваются функции буферизованного ввода/вывода стандартной библиотеки stdio.h. Системный вызов open() создает дескриптор файла, открываемый на чтение, при этом указатель устанавливается на начало файла. Внутри системного вызова вызываются другие функции, среди которых функция get_unused_fd(), ищущая пустой слот в таблице дескриптором файлов процесса. В случае, если таковых нет, возвращается ошибка, если же пустой слот найден, то возвращается наименьший файловый дескриптор из этой таблицы. Затем вызывается функция filp_open(const char* name, int flags, int mode), заполняющая структуру struct file для данного открытого файла. После создания дескриптора, функция fdopen() связывает два потока для чтения с данным дескриптором. Функция setvbuf() изменяет тип буферизации на блочную размером 20 байт. В цикле осуществляется чтение из двух потоков в стандартный поток вывода stdout. Так как размер буфера установлен 20 байт, то в поток fs1 считано первые 20 букв

алфавита, а в потом fs2 - оставшиеся. Таким образом, при поочередном чтении из обоих потоков, получаем следующую строку: Aubvcwdxeyfzghijklmnopqrst.

Задача 2.

Код программы:

```
#include <fcntl.h>
int main()
{
   int fd1 = open("alphabet.txt",O_RDONLY);
   int fd2 = open("alphabet.txt",O_RDONLY);
   while(1)
   {
         char c;
         if (read(fd1,&c,1) != 1) break;
         write(1,&c,1);
         if (read(fd2,&c,1) != 1) break;
         write(1,&c,1);
   }

   return 0;
}
```

Анализ.

В данной части лабораторной работы рассматриваются функции не буферизованного ввода/вывода с помощью системных вызовов read() и write(). Здесь, с помощью системного вызова open(), описанного выше, создается два файловых дескриптора одного и того же файла, открытого на чтение. Соответственно, в таблице открытых файлов процесса создается две разные записи. Следовательно, положения указателей в файле не зависят друг от друга и в начале цикла указывают на начало файла. Поэтому в ходе выполнения цикла, с помощью системных вызовов read() и write() из файла считывается и записывается в стандартный поток вывода соответственно два раза один и тот же символ. Результат: AAbbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz

Задача 3.

Код программы:

```
#include <stdio.h>
int main()
{
  FILE* fd[2];
  fd[0] = fopen("testFopen.txt","w");
  fd[1] = fopen("testFopen.txt","w");
  int curr = 0;
  for(char c = 'a'; c <= 'z'; c++)
  {
        fprintf(fd[curr], "%c", c);
        if (curr)
        curr = 0;
        else
        curr = 1;
  }
  fclose(fd[0]);
  fclose(fd[1]);
  return 0;
}
```

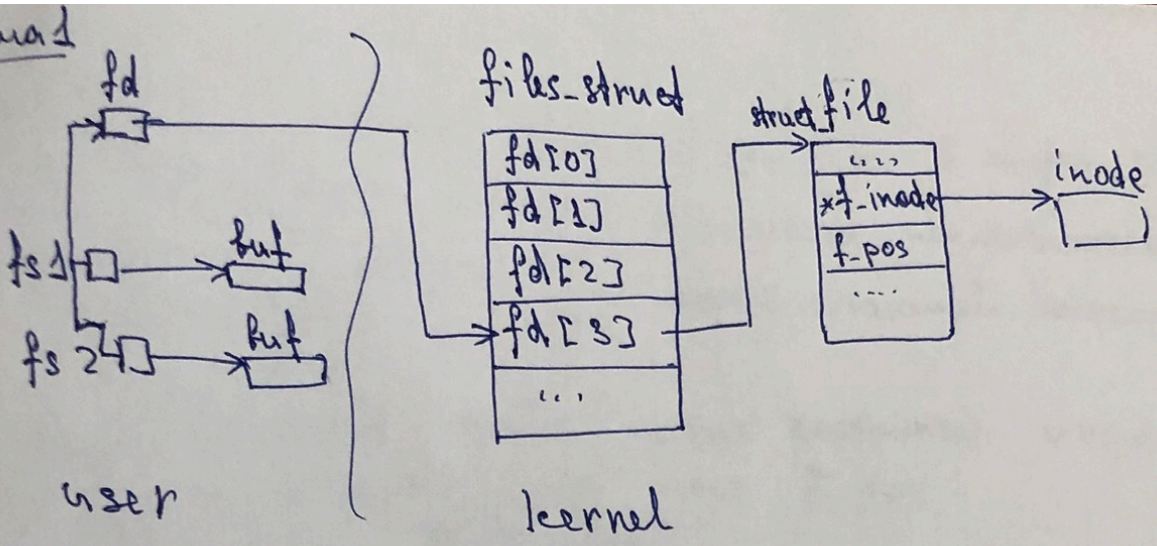
Анализ.

Функция fopen() - стандартная функция библиотеки stdio.h, выполняющая ввод/вывод с буферизацией. С ее помощью мы создаем два независимых потока для ввода - они имеют два разных файловых дескриптора и, следовательно, положения их указателей в файле независимы.
После этого четные буквы алфавита записываются в первый поток, а нечетные во второй поток с помощью функции буферизованного вывода fprintf(). Все значения сохраняются в буфера, запись непосредственно в файл происходит при вызове fclose() и fflush(). Так как поток fd[0] использовался для вывода данных, после вызова fclose() все данные записываются из буфера в файл с помощью fflush(). При этом поток остается открытым. Так как оба файла были открыты на запись, после второго вызова fclose() данные, записанные первым вызовом будут перезаписаны. Соответственно, результатом будет последовательность букв алфавита, стоящих на четных позициях. Результат: bdfhjlnprtvxz.

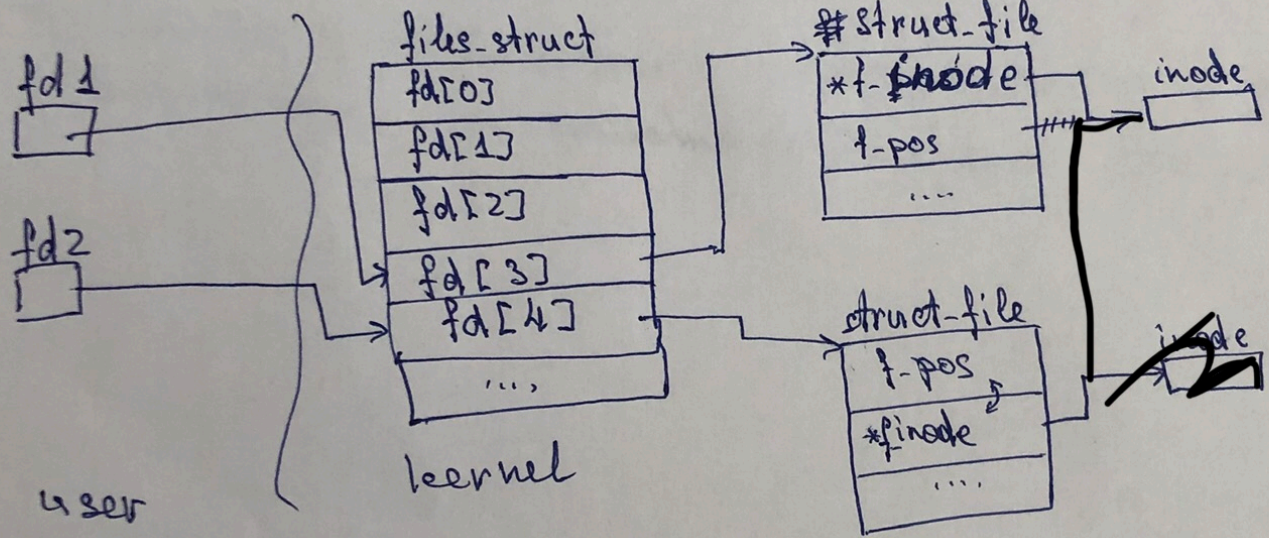
```
struct _IO_FILE {
  int _flags;       /* High-order word is _IO_MAGIC; остальные флаги. */
#define _IO_file_flags _flags

  /* Следующие указатели соответствуют протоколу C++ streambuf. */
  /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
  char* _IO_read_ptr;   /* Текущий указатель чтения */
  char* _IO_read_end;   /* Конец области get. */
  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;
#if 0
  int _blksize;
#else
  int _flags2;
#endif
  _IO_off_t _old_offset; /* This used to be _offset but it's too small.  */

#define __HAVE_COLUMN /* temporary */
  /* 1+column number of pbase(); 0 is unknown. */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];

  /*  char* _save_gptr;  char* _save_egptr; */
  _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

```
struct _IO_FILE_complete {
  struct _IO_FILE _file;
#endif
#if defined _G_IO_IO_FILE_VERSION && _G_IO_IO_FILE_VERSION == 0x20001
  _IO_off64_t _offset;
# if defined _LIBC || defined _GLIBCPP_USE_WCHAR_T
  /* Wide character stream stuff.  */
  struct _IO_codecvt *_codecvt;
  struct _IO_wide_data *_wide_data;
  struct _IO_FILE *_freeres_list;
  void *_freeres_buf;
  size_t _freeres_size;
# else
  void *__pad1;
  void *__pad2;
  void *__pad3;
  void *__pad4;
  size_t __pad5;
# endif
  int _mode;
  /* Make sure we don't get into trouble again.  */
  char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
#endif
};
#ifndef __cplusplus
typedef struct _IO_FILE _IO_FILE;
#endif
typedef struct _IO_FILE FILE;
```

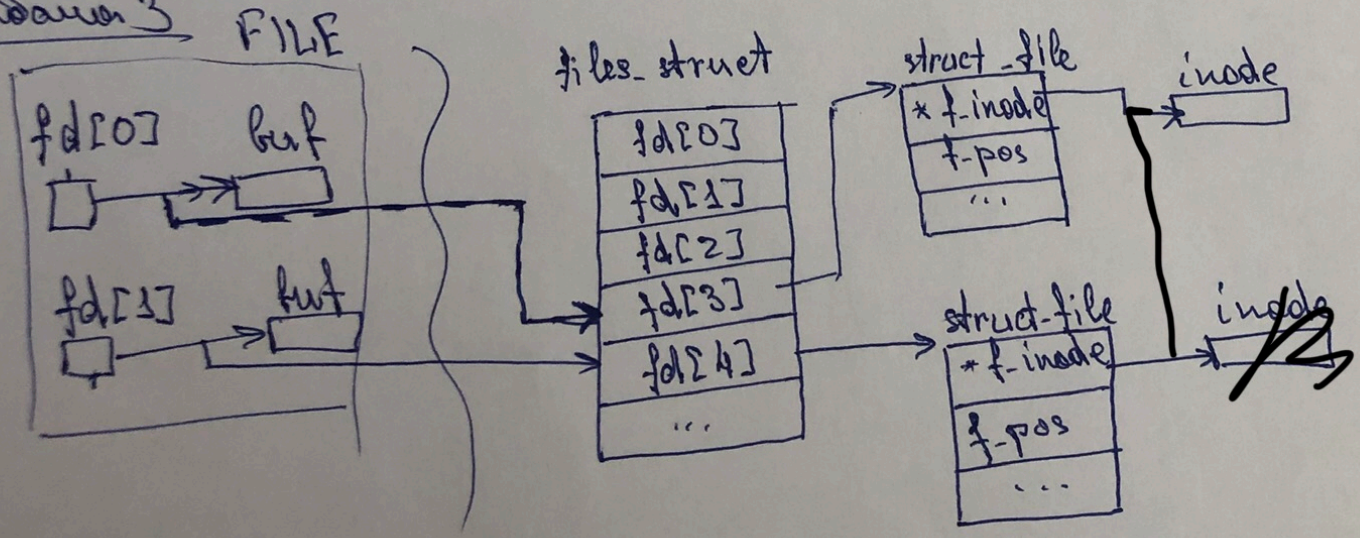
Zadanie 1



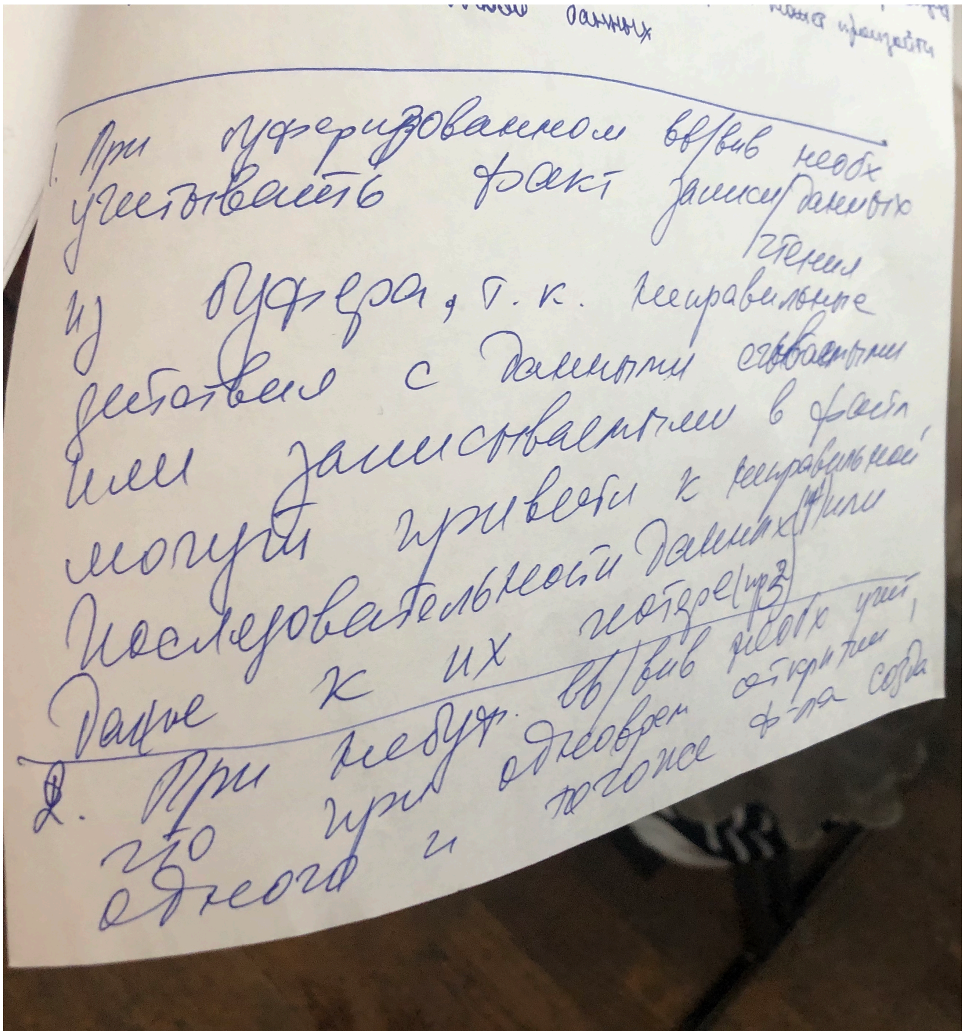
Zadanie 2



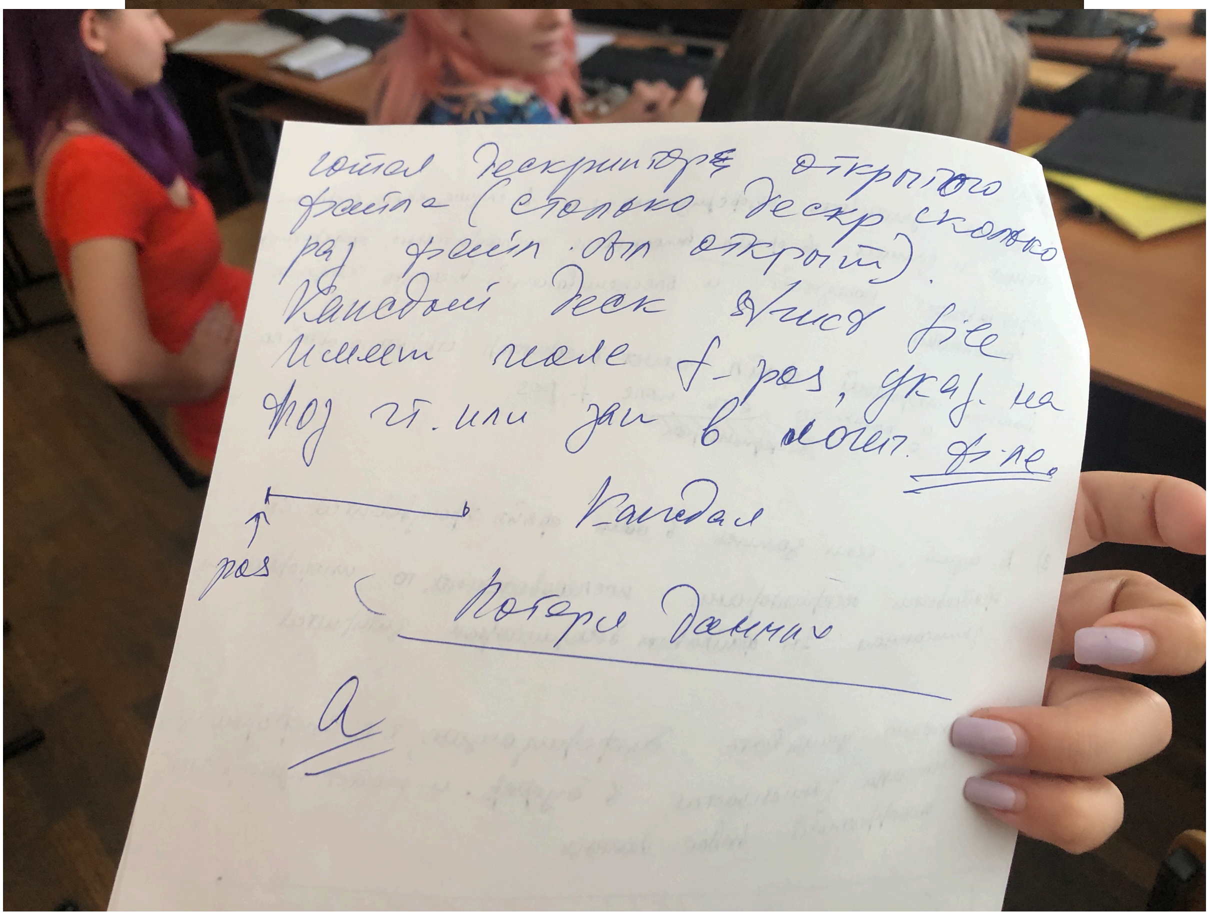
Zadanie 3



Вывод



1. При буферизованном вв/выв необх учитывать факт записи данных и чтения из буфера, т.к. неправильные действия с данными считанными или записываемыми в файл могут привести к неправильной последовательности данных (или чтения) или даже к их потере (пр3)
2. При небуф. вв/выв необх учит, что при одновремен открытии одного и того же ф-ла созд



столько дескрипторов открытого файла (столько дескр сколько раз файл был открыт). Каждый деск д/ися file имеет поле f_pos, указ. на поз чт. или зап в соотв. ф-ле.

pos — Каждая

Потеря данных

а

попросила написать 3ю задачу без использования буфера

```
#include <fcntl.h>
int main()
{
  int fd1 = open("q.txt",O_RDWR);
  int fd2 = open("q.txt",O_RDWR);
  int curr = 0;
  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;
}
```