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»

Отчет по лабораторной работе по дисциплине «Операционные системы»

Тема Буферизированный и небуферизированный ввод-вывод

Студент Ляпина Н.В.

Группа ИУ7-62Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва — 2023 г.

Описание структур

Листинг 1: Структура FILE

```
typedef struct _IO_FILE 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;

  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
};
```

Программа 1

Листинг 2: Программа 1

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

```
3
4  int main() {
5    // have kernel open connection to file alphabet.txt
6    int fd = open("alphabet.txt", O_RDONLY);
7
8    // create two a C I/O buffered streams using the above connection
9    FILE *fs1 = fdopen(fd, "r");
10   char buff1[20];
11   setvbuf(fs1, buff1, _IOFBF, 20);
12
13   FILE *fs2 = fdopen(fd, "r");
14   char buff2[20];
15   setvbuf(fs2, buff2, _IOFBF, 20);
16
17   // read a char & write it alternatingly from fs1 and fs2
18   int flag1 = 1, flag2 = 2;
19   while (flag1 == 1 || flag2 == 1) {
20     char c;
21
22     flag1 = fscanf(fs1, "%c", &c);
23     if (flag1 == 1) fprintf(stdout, "%c", c);
24
25     flag2 = fscanf(fs2, "%c", &c);
26     if (flag2 == 1) fprintf(stdout, "%c", c);
27   }
28
29   return 0;
30 }
```

Результат работы программы: aubvcwdxeyfzghijklmnopqrst

Листинг 3: Программа №1 (многопоточная)

```
1 #include <fcntl.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #define BUF_SIZE 20
5 #define FILENAME "alphabet.txt"
6
7 void *t1_run(void *args) {
8   int *fd = (int *)args;
9   FILE *fs1 = fdopen(*fd, "r");
10  char buff1[BUF_SIZE];
11  setvbuf(fs1, buff1, _IOFBF, BUF_SIZE);
12  int flag = 1;
13  char c;
14  while ((flag = fscanf(fs1, "%c", &c)) == 1)
15  fprintf(stdout, "%c", c);
16  return NULL;
17 }
18
```

```
19 void *t2_run(void *args) {
20   int *fd = (int *)args;
21   FILE *fs2 = fdopen(*fd, "r");
22   char buff2[BUF_SIZE];
23   setvbuf(fs2, buff2, _IOFBF, BUF_SIZE);
24   int flag = 1;
25   char c;
26   while ((flag = fscanf(fs2, "%c", &c)) == 1)
27   fprintf(stdout, "%c", c);
28   return NULL;
29 }
30
31 int main() {
32   pthread_t t1, t2;
33   int fd = open(FILENAME, O_RDONLY);
34
35   pthread_create(&t1, NULL, t1_run, &fd);
36   pthread_create(&t2, NULL, t2_run, &fd);
37
38   pthread_join(t1, NULL);
39   pthread_join(t2, NULL);
40   return 0;
41 }
```

Результаты работы (несколько запусков программы):

- abcdefghijklmnopqrstuvwxyz
- abuvwxyzcdefghijklmnopqrst
- aubvcwdxeyfzghijklmnopqrst

В программе файл открывается *один* раз системным вызовом `open`. Системный вызов `open` возвращает дескриптор файла типа `int`. Возвращенный номер `fd` – индекс дескриптора открытого файла в таблице дискрипторов файлов процесса.

Функция `fdopen()` создаёт указатель на структуру `FILE`. Поле `_fileno` в `struct _IO_FILE` содержит номер дескриптора, который вернула функция `open()`.

Функция `setvbuf()` явно задает размер буффера в 20 байт и меняет тип буферизации (для `fs1` и `fs2`) на полный.

При первом вызове функции `fscanf()` в цикле (для `fs1`) `buff1` будет заполнен полностью – первыми 20 символами (буквами алфавита). `f_pos` в структуре `struct_file` открытого файла увеличится на 20.

При втором вызове `fscanf()` в цикле (для `fs2`) буффер `buff2` будет заполнен оставшимися 6 символами (начиная с `f_pos`).

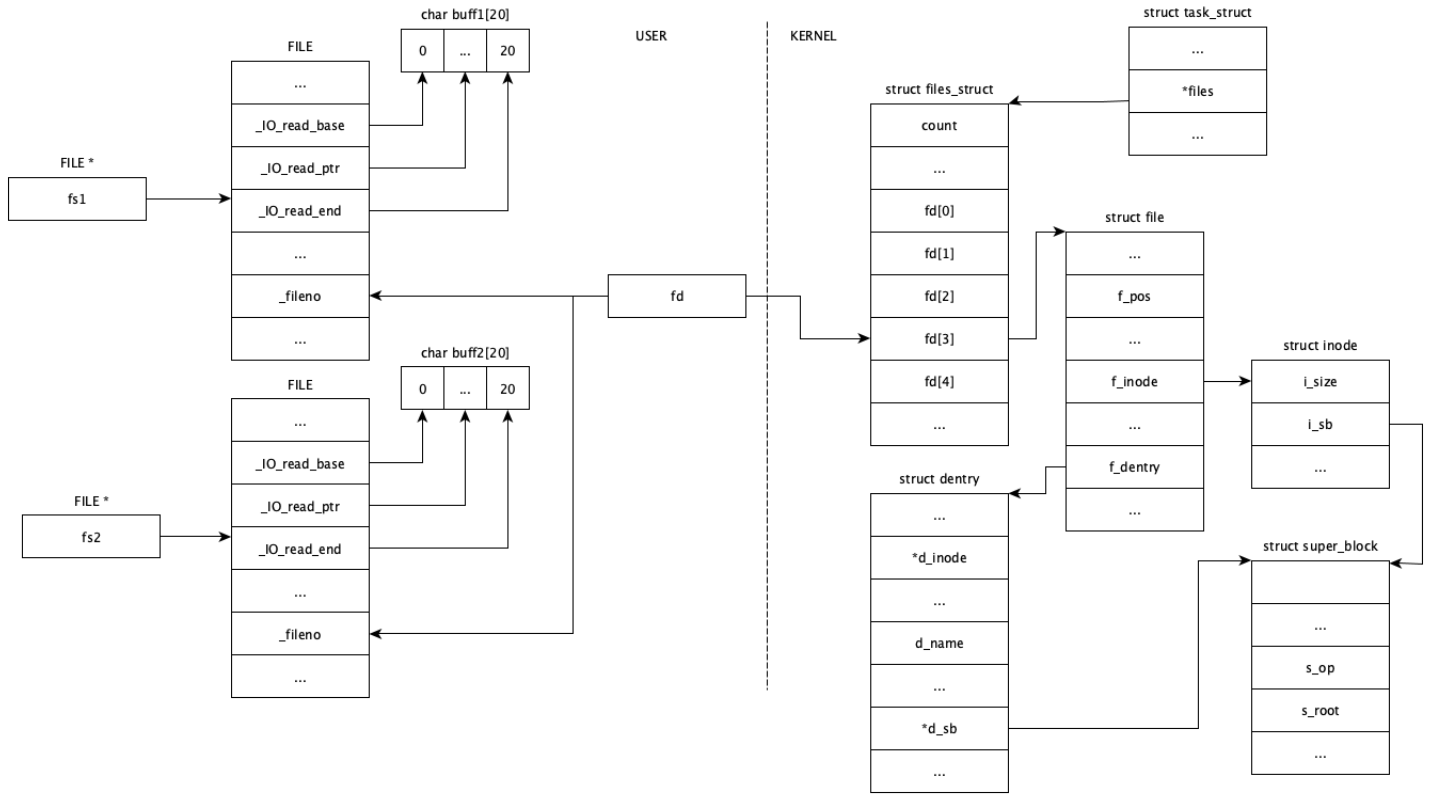

Рис. 1: Используемые структуры. Программа 1

Программа 2

Листинг 4: Программа 2

```c
#include <fcntl.h>
#include <unistd.h>
int main() {
  int fd1 = open("alphabet.txt", O_RDONLY);
  int fd2 = open("alphabet.txt", O_RDONLY);
  char c;
  while (read(fd1, &c, 1) == 1 && read(fd2, &c, 1) == 1) {
    write(1, &c, 1);
    write(1, &c, 1);
  }
  return 0;
}
```

Результат работы: aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz

Листинг 5: Программа 2 (многопоточная)

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

```

```c
#define FILENAME "alphabet.txt"

void *t_run(void *args) {
    int fd = open(FILENAME, O_RDONLY);
    int flag = 1;
    char c;
    while ((flag = read(fd, &c, 1)) == 1) {
        write(1, &c, 1);
    }
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, t_run, NULL);
    pthread_create(&t2, NULL, t_run, NULL);

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

    return 0;
}
```

Результаты работы (несколько запусков программы):

- aabcdbefcgdheifjgkhlimjnkolpmqnrosptqurvswtxuyvzwxyz
- aabbcdcedfefgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
- aabbccddeeffgghihjijkkllmmnnopoqprqsrtsutvuwvxwyxzyz

Функция `open()` создает файловый дескриптор, два раза для одного и того же файла, поэтому в программе существует два дескриптора открытого файла `struct file`, но ссылающиеся на один и тот же `struct inode`.

Из-за того что структуры разные (у каждой структуры свое поле `f_pos`), посимвольная печать дважды выведет содержимое файла в формате «`aabbcc...`» (в случае однопоточной реализации).

В случае многопоточной реализации, потоки выполняются с разной скоростью и символы перемешаются.

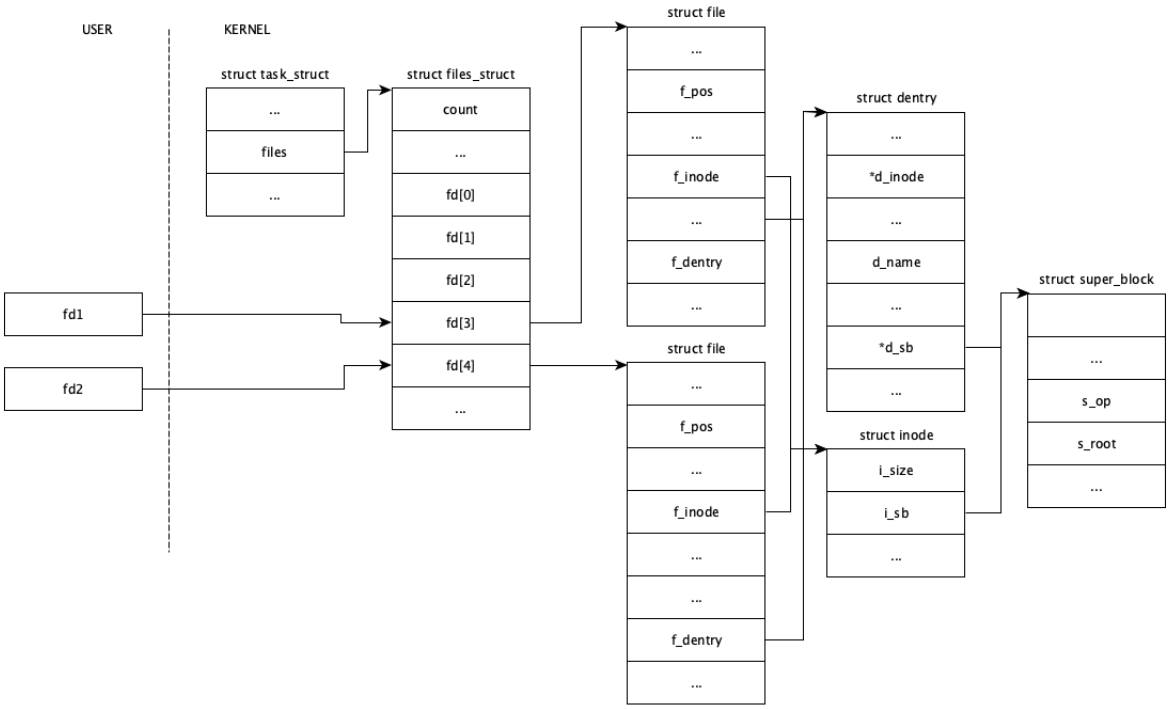

Рис. 2: Используемые структуры. Программа 2

Программа 3

Листинг 6: Программа 3

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

#define FILENAME "out.txt"

int main() {
  FILE *f1 = fopen(FILENAME, "w");
  FILE *f2 = fopen(FILENAME, "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;
}
```

Результат работы: bdfhjlnprtvxz

Листинг 7: Программа 3 (многопоточная)

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

#define FILENAME "out.txt"

void *t1_run(void *args) {
  FILE *f = fopen(FILENAME, "w");

  for (char c = 'a'; c <= 'z'; c += 2) {
    fprintf(f, "%c", c);
  }

  fclose(f);

  return NULL;
}

void *t2_run(void *args) {
  FILE *f = fopen(FILENAME, "w");

  for (char c = 'b'; c <= 'z'; c += 2) {
    fprintf(f, "%c", c);
  }

  fclose(f);

  return NULL;
}

int main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, t1_run, NULL);
  pthread_create(&t2, NULL, t2_run, NULL);

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

```

```
40    return 0;
41 }
```

Результаты работы (несколько запусков программы):

- acegikmoqsuwy
- bdfhjlnprtvxz

Файл открывается на запись два раза функцией `fopen()`.

Из-за того `f_pos` независимы для каждого дескриптора файла, запись в файл будет производится с нулевой позиции.

Функция `fprintf()` предоставляет буферизованный вывод.

Изначально информация пишется в буфер, а из буфера в файл если произошло одно из событий:

1. буффер заполнен
2. вызвана функция `fclose()`
3. вызвана функция `fflush()`

В случае нашей программы, информация в файл запишется в результате вызова функции `fclose()`. При вызове fclose() для fs1 буфер для fs1 записывается в файл. При вызове fclose() для fs2, все содержимое файла затирается, а в файл записывается содержимое буфера для fs2. В итоге произошла потеря данных, в файле окажется только содержимое буфера для fs2. Чтобы этого избеать, необходимо использовать флаг O_APPEND. Если этот флаг установлен, первая запись в файл не теряется.

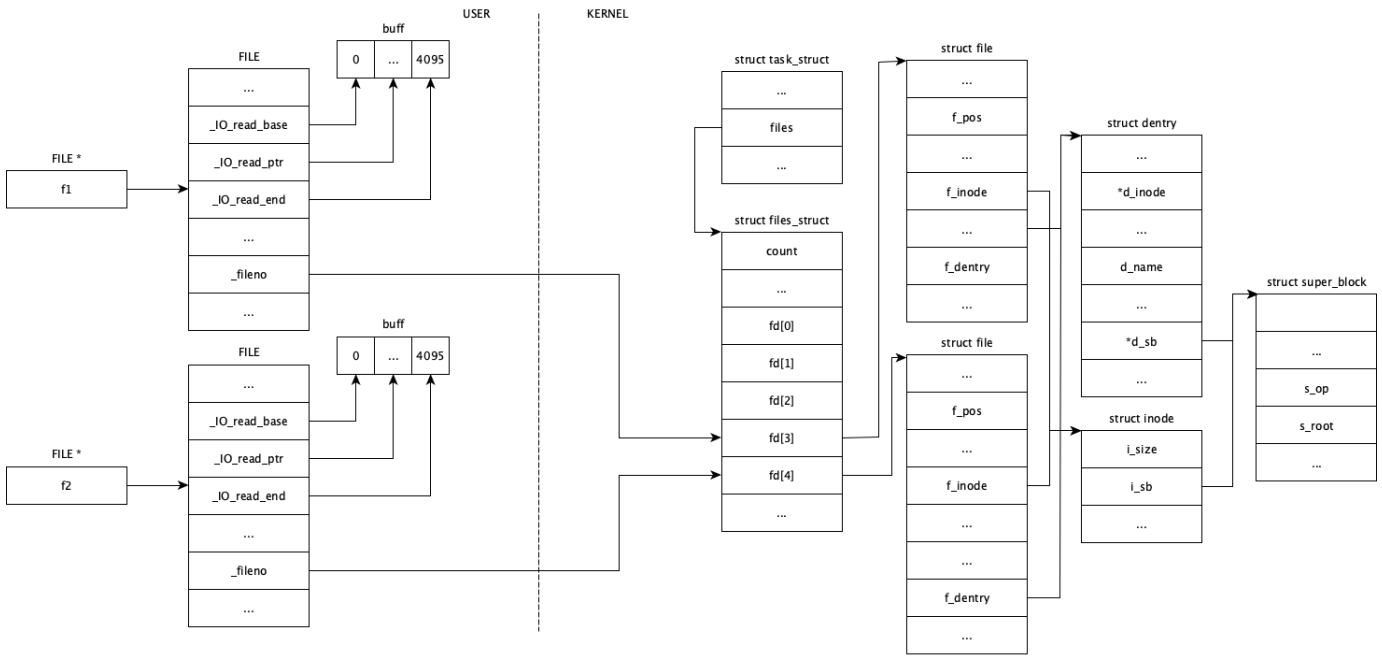

Рис. 3: Используемые структуры. Программа 3

Листинг 8: Программа 3 (с open, read, write)

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

#define FILENAME "out.txt"

int main() {
  int f1 = open(FILENAME, O_WRONLY | O_CREAT);
  int f2 = open(FILENAME, O_WRONLY | O_CREAT);

  for (char c = 'a'; c <= 'z'; c++) {
    if (c % 2) {
      write(f1, &c, 1);
    } else {
      write(f2, &c, 1);
    }
  }

  close(f1);
  close(f2);
  return 0;
}
```

Результат работы: bdfhjlnprtvxz

Листинг 9: Программа 3 (с open, read, write) (многопоточная)

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

#define FILENAME "out.txt"

void *t1_run(void *args) {
  int f = open(FILENAME, O_WRONLY | O_CREAT);

  for (char c = 'a'; c <= 'z'; c += 2) {
    write(f, &c, 1);
  }

  close(f);

  return NULL;
}

void *t2_run(void *args) {
  int f = open(FILENAME, O_WRONLY | O_CREAT);

  for (char c = 'b'; c <= 'z'; c += 2) {
```

```
24      write(f, &c, 1);
25    }
26
27    close(f);
28
29    return NULL;
30  }
31
32  int main() {
33    pthread_t t1, t2;
34    pthread_create(&t1, NULL, t1_run, NULL);
35    pthread_create(&t2, NULL, t2_run, NULL);
36    pthread_join(t1, NULL);
37    pthread_join(t2, NULL);
38    return 0;
39  }
```

Результаты работы (несколько запусков программы):

- acegikmoqsuwy
- bdfhjlnprtvxz

Системный вызов `write` записывает данные в файл сразу после вызова, а отличие от библиотечной функции `fprintf()`, которая сначала записывает данные в буфер, и только после вызова функции `fclose()` – в файл.

Программа, представленная в листинге 8, работает следующим образом. Системный вызов `open` возвращает индекс в массиве открытых файлов процесса, причем каждый раз возвращается новый индекс, даже если открывается один и тот же файл. Затем при системном вызове `write` для f1 данные сразу записываются в файл, но из-за того, что при системной вызове `open` указатель устанавливается в начало файла, следующий системный вызов `write` для f2 затирает данные предыдущего вызова.

Для предотвращения затирания данных нужно использовать флаг O_APPEND в качестве аргумента системного вызова `open`. Этот флаг устанавливает указатель на конец файла перед каждым вызовом `write`.