Семестр 4 (2021), занятие 1. Низкоуровневый ввод-вывод

Низкоуровневый ввод-вывод реализуется четырьмя функциями `open`, `write`, `read`, `close`, которые по существу являются обертками над соответствующими системными вызовами. Их прототипы и связанные с ними числовые константы определены в стандартных заголовочных файлах `unistd.h` и `fcntl.h`.

Функция `open`

Функция `open` используется для открытия файла и имеет два варианта прототипа.

```
int open(const char *name, int mode, int perms);
int open(const char *name, int mode);
```

Первый входной параметр `name` задает маршрутное имя файла, второй параметр `mode` задает режим работы с файлом, третий параметр `perms` задает права доступа. В случае успеха функция возвращает неотрицательное целое число, называемое *файловым дескриптором*. В случае ошибки возвращается `-1`.

Параметр `mode` представляет собой целое число, отдельные биты которого задают особенности предстоящей работы с файлом. Рекомендуется формировать значение этого параметра как результат выполнения операции побитового «или» над предопределенными константами, к числу которых относятся:

- `O_RDONLY` – только чтение;
- `O_WRONLY` – только запись;
- `O_CREAT` – создание файла вслучае его отсутствия;
- `O_TRUNC` – удаление содержимого для существующего файла.

Параметр `perms` указывется только, если в выражении, задающем значение парамета `mode`, присутствует константа `O_CREAT`. Рекомендуется также формировать значение этого параметра как результат выполнения операции побитового «или» над предопределенными константами.

Приведем примеры использования функции `open`.

Для открытия файла `a.dat` на запись можно использовать следуюший вариант вызова. При этом, при создании файла его владелец получает права на чтение и запись файла.

```
int fd = open("a.dat",
              O_CREAT | O_WRONLY | O_TRUNC,
              S_IRUSR | S_IWUSR);
```

Этот вызов может быть переписан в виде, где явно задаются значения числовых параметров.

```
int fd = open("a.dat", 577, 384);
```

Для открытия файла `a.dat` на чтение можно использовать следуюший вариант вызова.

```
int fd = open("a.dat", O_RDONLY);
```

Функция `write`

Функция `write` используется для записи данных в файл и имеет следующий прототип.

```
ssize_t write(int fd, const void *b, size_t l);
```

Первый входной параметр `fd` задает файловый дескриптор. Второй входной параметр `b` указывает на область памяти, из которой нужно взять данные для записи. Третий входной параметр `l` задает размер (в байтах) этих данных.

- положительное целое число, не превосходящее значение `l` – реальное количество записанных байтов;
- `-1` – ошибка чтения.

Пример использования.

```
int x = 12345;
c = write(fd, &x, sizeof x);
...
const char *s = "abcde";
c = write(fd, s, 6);
...
```

После каждого вызова функции `write` необходимо проверять возвращаемое ею значение, которое в данном примере сохраняется в переменную `c`.

В рамках первого вызова осуществляется попытка записи последовательности байтов, кодирующей целое число `12345`. В рамках первого вызова осуществляется попытка записи символов строки `abcde` вместе с завершающим нулевым байтом.

Функция `read`

Функция `read` используется для чтения данных из файла и имеет следующий прототип.

```
ssize_t read(int fd, void *b, size_t l);
```

Первый входной параметр `fd` задает файловый дескриптор. Второй входной параметр `b` указывает на область памяти, в которую следует сохранить прочитанные данные. Третий входной параметр `l` задает размер (в байтах) этих данных.

Функция может вернуть следующие значения:

- положительное целое число, не превосходящее значение `l` – реальное количество прочитанных байтов;
- `0` – ситуация конца файла;
- `-1` – ошибка чтения.

Пример использования.

```
int x;
c = read(fd, &x, sizeof x);
...
char s[6];
c = read(fd, s, 6);
...
```

В этом примере осуществляется попытка чтения данных, которые были записаны в рамках предыдущего примера.

Функция `close`

Функция `close` используется для закрытия файла и имеет следующий прототип.

```
int close(int fd);
```

Входной параметр `fd` задает дескриптор закрываемого файла. В случае успеха функция возвращает `0`, а в случае ошибки возвращает `-1`. В любом случае происходит закрытие файла (освобождение файлового дескриптора).

Вспомогательные программы

С помощью программы `od` можно распечатать содержимое бинарного файла в виде последовательности отдельных байтов. Указание ключа `-c` приводит к печати в символьном виде, а указание ключа `-t u1` приводит к печати в числовом виде.

```
$ echo -n "abcde01234" > a.dat

$ od -c a.dat
0000000   a   b   c   d   e   0   1   2   3   4

$ od -t u1 a.dat
0000000  97  98  99 100 101  48  49  50  51  52
```

Программа `truncate` позволяет уменьшить размер файла. В следующем примере вначале с помощью команды `ls -l` узнается текущий размер файла `a.dat`. После этого с помощью команды `truncate` размер файла уменьшается на один байт.

```
$ ls -l a.dat
-rw-rw-r-- 1 student student 10 feb 01 09:00 a.dat

$ truncate -s 9 a.dat

$ ls -l a.dat
-rw-rw-r-- 1 student student 9 feb 01 09:00 a.dat
```

Задание

Требуется разработать две программы `w` и `r`. Программа `w` через аргументы командной строки получает имя файла и последовательность строк. В этот файл в бинарном виде последовательно записываются переданные строки в виде кодирующих последовательностей байтов. Кодирующая последовательность строки имеет следующий формат. Сначала записываются 4 байта, кодирующие целое число равное длине строки. Затем записываются символы строки.

Программа `r` через аргументы командной строки получает имя файла, хранящего последовательность строк в указанном выше формате, и распечатывает эти строки.

```
$ ./w a.dat a ab

$ ./r a.dat
2     a
3     ab
```

С помощью программы `truncate` нужно показать, как программа `r` обрабатывает некорректные данные.

```
$ od -t u1 a.dat
0000000   2   0   0   0  97   0   3   0   0   0  97  98   0


$ truncate -s 12 a.dat

$ ./r a.dat
2     a
Can't read string bytes.
```