UNIX

Лекция 3

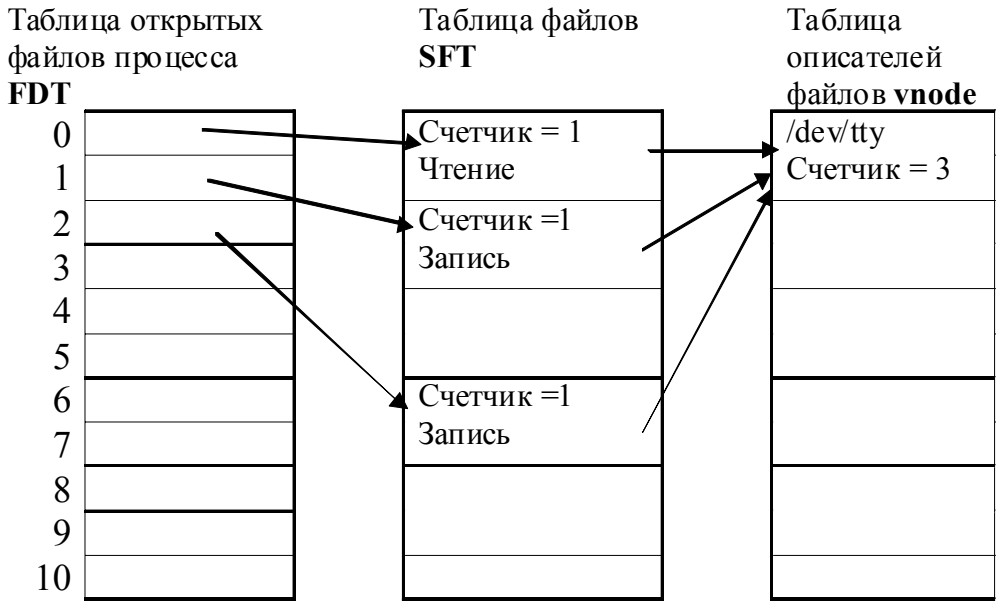
Еще раз о стандартных потоках

Соглашение: первые три пользовательских дескриптора (0, 1 и 2) именуются дескрипторами файлов *стандартного ввода, стандартного вывода и стандартного файла ошибок*.

read(0, &c, 1); // Читаем из стандартного входного потока
// (считывание) с клавиатуры.

write(1, “unix”, 4); // Стандартный выходной поток. По умолчанию –
// вывод на консоль

write(2, “error”, 5); // Выводим в стандартный поток ошибок



Программа копирования

```
1.  #include <fcntl.h>
2.  char buffer[2048];
3.  main(int argc, char *argv[])
4.  {       int fdold, fdnew;
5.          int count;
6.          if (argc != 3)
7.          {       printf("need 2 arguments for copy file\n");
8.                  exit(1);
9.          }
10.         fdold = open(argv[1],O_RDONLY); /* открыть исходный файл только для чтения */
11.         if (fdold == -1)
12.         {       printf("cannot open file %s\n",argv[1]); // perror
13.                 exit(1);
14.         }
15.         /* создать новый файл с разрешением чтения и записи для всех
16.         пользователей */
17.         fdnew = open(argv[2],O_CREAT|O_TRUNC|O_WRONLY,0666);
18.         if (fdnew == -1)
19.         {       printf("cannot create file %s\n",argv[2]);
20.                 exit(1);
21.         }
22.         while ((count = read(fdold,buffer,sizeof(buffer))) > 0)
23.                 write(fdnew,buffer,count);
24.         close(fdnew);
25.         close(fdold);
26.         exit(0);
27. }
```

U			G			O			
r	w	x	r	w	x	r	w	x	
r	w	-	r	w	-	r	w	-	Text
1	1	0	1	1	0	1	1	0	Binary
6			6			6			Octal

Примеры использования

copy copy.c newcopy.c

copy copy newcopy

Обычно считается, что существующий файл также может быть и каталогом.

copy . Dircontents

Но лучше пользоваться *opendir* и *readdir*

copy /dev/tty terminalread

copy /dev/tty /dev/tty

Программирование операций ввода-вывода

UNLINK. Удаление файла
int unlink(const char *path)

OPEN. Открыть файл (получение доступа к файлу):
int open(char *pathname, int flags, mode_t mode);
flags определяет режим открытия файла:
O_CREAT – создать файл, если его не существует;
O_TRUNC – если файл существует, перезаписать его;
O_RDONLY – файл открывается только для чтения;
O_WRONLY – файл открывается только для записи;
O_RDWR – файл открывается для чтения и записи.
и т.д. Эти параметры можно комбинировать, используя операцию ИЛИ ('|').
mode задает права доступа к создаваемому файлу.
Если нет возможности открыть файл, open возвращает -1.

CLOSE. Закрыть файл. Функция уничтожает связь между пользовательским дескриптором файла и самим файлом:
void close(int fd);
Параметр fd - дескриптор файла, возвращенный вызовом open.

STAT, LSTAT и FSTAT

int stat(char *path, struct stat *statbuf);

int lstat(char *path, struct stat *statbuf); // выдает информацию о самом файле символической связи, а не о файле, на который он ссылается.

int fstat(int fd, struct stat *statbuf);

Обычно описаны эти функции в sys/stat.h и unistd.h.

```
struct stat
{   dev_t st_dev;          /* номер устройства */
    ino_t st_ino;                  /* номер inode */
    ushort st_mode;                /* режим доступа и тип файла */
    short st_nlink;        /* счетчик числа ссылок на файл */
    ushort st_uid;         /* идентификатор его владельца */
    ushort st_gid;         /* идентификатор группы */
    dev_t st_rdev;         /* тип устройства */
    off_t st_size;                 /* размер файла в байтах */
    time_t st_atime;               /* дата последнего доступа */
    time_t st_mtime;               /* дата последней модификации */
    time_t st_ctime;               /* дата модификации метаданных */
}
```

st_mode

```
#define   S_IFMT   00170000          /* маска выделения типа файла */
#define   S_IFDIR  0040000           /* каталог */
#define   S_IFCHR 0020000            /*байт-ориентированный спец.файл */
#define   S_IFBLK  0060000           /*блок-ориентированный спец.файл */
#define   S_IFREG 0100000            /* обычный файл */
#define   S_IFIFO  0010000           /* файл FIFO */
#define   S_IFSOCK          0140000              /* сокет */
#define   S_IFLNK 0120000            /* сиволическая связь - symbolic link */
#define   S_ISUID  04000             /* идентификатор владельца - set UID bit */
#define   S_ISGID  02000             /* идентификатор группы - set GID bit */
#define   S_ISVTX 01000              /* сохранить свопируемый текст – sticky bit*/
```

```
#include <sys/stat.h>
#include <unistd.h>
struct stat stbuf;
char *filename = ”myfile”;
 . . . . . .
stat(filename, &stbuf);
if ((stbuf.st_mode & S_IFMT) == S_IFDIR)
printf("%s является каталогом", filename);
```

Утилита STAT

stat filename [filenames ...]

Пример использования: *stat calc.x:*

File: "calc.x"

Size: 115 Filetype: Regular File

Mode: (0664/-rw-rw-r--) Uid: (0/ root) Gid: (0/ root)

Device: 3,4 Inode: 6543 Links: 1

Access: Fri Oct 1 09:44:27 1999(00000.09:27:48)

Modify: Thu Jul 22 17:07:05 1999(00071.02:05:10)

Change: Fri Oct 1 09:43:22 1999(00000.09:28:53)

Прочие функции

READ. Осуществляет чтение из открытого файла указанного количества символов в буфер:

int read(int fd, void *buffer, unsigned count)

Возвращает количество реально прочитанных байт или отрицательный код ошибки.

WRITE. Осуществляет запись в открытый файл указанного количества символов из буфера:

int write(int fd, void *buffer, unsigned count)

Возвращает количество реально записанных байт или отрицательный код ошибки.

LSEEK. Перемещает указатель файла с пользовательским дескриптором fd на offset байт:

long lseek(int fd, long offset, int fromwhere)

Параметр fromwhere определяет положение указателя файла перед началом перемещения: SEEK_SET - от начала файла;

SEEK_CUR - от текущей позиции указателя;

SEEK_END - от конца файла.

LSEEK. Создание файлов

Создание файлов заданного размера. Достаточно переместить указатель на нужную позицию.

Для создания файла размером N байт указатель перемещается в позицию N-1, а затем делается запись write(f,"",1). Без этой записи ничего не выйдет. Это - "резервирование" места. Характерно то, что при всем этом новые блоки хранения данных не выделяются.

Блоки хранения данных выделяются только при записи в них информации.

```
// Программа, создающая файл заданного размера
main()
{   int n = 10240, char c = 0;
    f = open(“file”, O_CREAT|O_WRONLY|O_TRUNC, 0777);
    lseek(f, n-1, SEEK_SET);
    write(f, &c, 1);
    close(f);
}
```

Системные вызовы для работы с каталогами

- *int mkdir(const char *path, mode_t mode)*
- *int rmdir(const char *path)*
- *DIR *opendir(const char *name)*
- *struct dirent *readdir(DIR *fd)*
- *int readdir(unsigned int fd, struct dirent *dirp, unsigned int count)*
- *int rewinddir(DIR *fd)*
- *int closedir(DIR *dir)*

```
struct dirent
{   long d_ino;    // номер inode
    off_t d_off;    // смещение от начала директории
                                // до текущей записи
    unsigned short d_reclen;        // длина имени d_name, не считая
                                    // нулевой символ-терминатор
    char d_name [NAME_MAX+1];           // имя файла
                                        // (заканчивается нулем)
}
```

Работа со специальными файлами устройств и каналами

mknod. Создание специального файла устройства и канала

- *int mknod(const char *path, mode_t mode, int device_id)*

Пример :

mknod("dev", S_IFBLK|S_IRWXU|S_IRWXG|S_IRWXO, (15<<8) | 3)

- Создание специального файла устройства с именем «devX», со старшим номером 15 и младшим номером 3. Это – блок-ориентированный файл (флаг S_IFBLK).

mkfifo. Создание канала – файла FIFO

- *int mkfifo(const char *path, mode_t mode, int device_id)*

Пример:

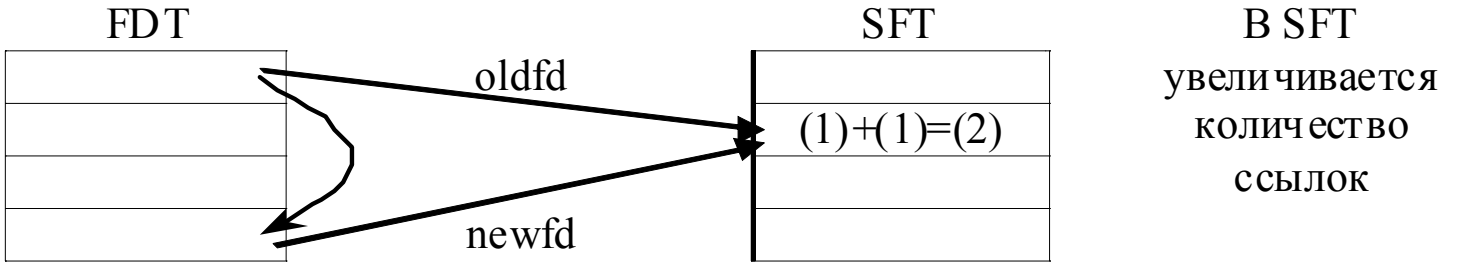
- *mkfifo("fifo", S_IFIFO |S_IRWXG|S_IRWXO)*
- Далее с этим файлом можно работать как с самым обычным. При этом, если указать флаг O_NONBLOCK, то процесс, обращающийся к этому файлу, не будет блокироваться ни при каких условиях.

Работа с файловыми дескрипторами

dup и **dup2.** Дублирование пользовательского дескриптора файла:

- *int dup(int handle)*
- *int dup2(int oldhandle, int newhandle)*
- *Если newhandle до этого указывал на открытый файл, этот файл в результате вызова dup2 будет закрыт.*

Вызов: *dup2: fd = dup2 (oldfd, newfd)*

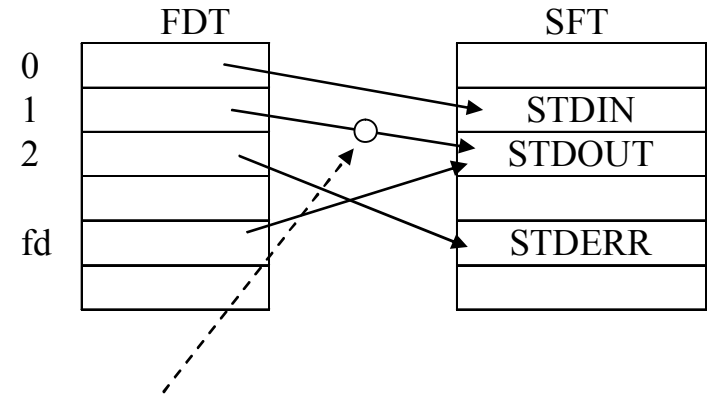


Примеры

fd=dup(1);
write(1,…); или write(fd,…)

Общая схема дублирования файловых дескрипторов:

1. *save = dup(1)*
2. *new = open(…)*
3. *dup2(new,1)*
4. *close(new)*
5. *write(1,…)*
6. *dup2(save,1)*
7. *close(save)*



Пример 2. Дублирование дескриптора файла

```
1.  /* Фрагмент программы DUP.C – перенаправление стандартного вывода в файл */
2.  #include <unistd.h>
3.  #include <stdio.h>
4.  #include <string.h>
5.  #include <fcntl.h>
6.  void main(void)
7.  {       int outf, std_out;
8.          char *str1 = "Вывод строки в файл", *str2 = "Вывод строки на экран";
9.          std_out = dup(1);
10.         /* закрытие стандартного вывода */
11.         close(1);
12.         outf = open(“1.dat”, O_WRONLY|O_CREAT|O_TRUNC,0666);
13.         // скорее всего, outf будет равен 1, т.к.
14.         // open ищет первый свободный элемент в FDT
15.         puts(str1);
16.         write(std_out,str2,strlen(str2));
17.         /* восстановление предыдущих значений */
18.         close(outf);
19.         outf = open(“\dev\tty”, O_WRONLY);
20.         close(std_out);
21.         exit(0);
22. }
```

Пример 3

```
1.  /* Утилита msg [outfile]: выводит в файл (outfile) или на экран сообщения */
2.  #include <fcntl.h>
3.  main (int argc, char *argv[])
4.  {       char *msg = “Dup example”;
5.  int f;               /* номер нового файлового дескриптора */
6.  int save;            /* здесь мы сохраним исходное значение */
7.  if (argc>1)
8.  {       f = open (argv[1], O_CREAT|O_TRUNC|O_WRONLY, 0666);
9.          save = dup(1);
10.         dup2(f,1);
11.         close(f);
12. }
13. write (1, msg, strlen(msg));
14. /*1 – стандартный поток вывода */
15. if (argc>1) // восстанавливаем
16. {
17.         dup2(save, 1);
18.         close(save);
19. }
20. }
```