Разработка программного обеспечения ОС UNIX

Файловая система:

Интерфейсы доступа

Потоковый ввод/вывод: библиотека языка Си

```
#include   <stdio.h>

FILE *fopen(const char *filename, const char *mode);
int fclose(FILE *stream);

size_t fread(void *ptr,           size_t size,  size_t nitems,   FILE *stream);
size_t fwrite(const void *ptr, size_t size,  size_t nitems,   FILE *stream);

int fseek(   FILE *stream,   long offset,   int whence  );
int fseeko( FILE *stream,   off_t offset,   int whence  );

long ftell(   FILE *stream );
off_t ftello( FILE *stream );

int fflush(  FILE *stream  );
```

Потоковый ввод/вывод: библиотека языка Си

```
int feof(FILE *stream);
int ferror(FILE *stream);

void clearerr(FILE *stream);
                                                    // Буферизация
void setbuf( FILE *stream, char *buf);
int setvbuf(  FILE *stream, char *buf,   int type,   size_t size);
                        //тип: _IOFBF,   _IOLBF,   _IONBF.

int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);     //  ???
     -- альтернатива
char *fgets(char *s, int n, FILE *stream);
    #include <stdlib.h>
double strtod(const char *nptr, char **endptr);
long strtol(const char *str, char **endptr, int base);
```

Системные вызовы и дескрипторы файлов

Дескриптор открытого файла – целое число

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);        // Открыть дескриптор
```

Флаги открытия:

O_RDONLY	O_RDWR	O_WRONLY
O_APPEND	O_CREAT	O_EXCL
O_TRUNC	O_NONBLOCK	O_CLOEXEC

-- Пример:

```
pfd = open(  filename, O_CREAT|O_EXCL,
                       S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
if (pfd == -1 ) Ошибка!

#include <unistd.h>          // Закрыть дескриптор
int close(int fildes);

ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset);
```

Позиционирование в файле

```
#include <unistd.h>

off_t lseek(  int fildes, off_t offset,  int whence  ); Возвращает новую позицию
```

Параметр whence:

SEEK_SET	От начала
SEEK_CUR	От текущей позиции
SEEK_END	От конца

Установка заданного размера

```
#include <unistd.h>

int truncate(  const char *path,  off_t length  );
int ftruncate(  int fildes,              off_t length  );
```

Внимание: можно получить файл с «дырами» (gaps) – разреженный файл.

Связь с потоковым вводом/выводом. Управление флагами файла

```
#include <stdio.h>
int fileno(FILE *stream);      //Отображает указатель на дескриптор

FILE *fdopen(int fildes, const char *mode);   //Связывает поток с дескриптором
     mode как у fopen:  r, rb, w, wb, a, ab, r+, w+, a+, …

-- Управление состоянием файла
#include <fcntl.h>
int fcntl(int fildes, int cmd, ...);
F_GETFD
F_SETFD  // флаги. 3-й аргумент - FD_CLOEXEC или 0
      // Пример:
            flags = fcntl(fd, F_GETFD);
            if (flags == -1)        /* Ошибка */;
            flags |= FD_CLOEXEC;
            if ( fcntl( fd, F_SETFD, flags ) == -1) /* Ошибка */;"
F_GETFL / F_SETFL         // флаги состояния и режим доступа
```

Дублирование и связывание описателей

F_DUPFD, F_DUPFD_CLOEXEC

Возвращает наименьший описатель не меньше указанного в 3 аргументе, связанный с заданным файлом

```
#include <unistd.h>
int dup(int fildes);                // дублирует описатель fildes
    реализация:
        return fcntl( fildes, F_DUPFD, 0 );

int dup2(int fildes, int fildes2);     //связывает fildes2 с файлом fildes
    реализация:
    {       close( fildes2 );
        return fcntl( fildes, F_DUPFD, fildes2 );
    }
```

Примеры связывания описателей

Переназначает стандартный поток ошибок на стандартный вывод

```
dup2( STDOUT_FILENO, STDERR_FILENO );
        можно проще:
dup2(1, 2);
```

Переназначение ввода из некоторого файла
и стандартного вывода в желаемый файл:

```
int fd1;
int fd2;
fd1 = open( “somefile.txt”, O_RDONLY );
fd2 = open( “desiredfile.txt”, O_WRONLY|O_CREAT|O_TRUNC, 0644 );
if ( fd1 == -1 ) … Ошибка!
if ( fd2 == -1 ) … Ошибка!
if ( dup2( fd1, STDIN_FILENO ) == -1 ) … Ошибка!
if ( dup2( fd2, STDOUT_FILENO ) == -1 ) … Ошибка!
```

Блокировка участков файла

Блокировки:

- на чтение (разделяемая) / на запись (эксклюзивная)
- обязательная (mandatory) / рекомендуемая (advisory)

Обязательная – если установлен бит setGID, а файл не отмечен как выполняемый

Команды fcntl: int fcntl(int fildes, int cmd, struct flock *fl);

F_GETLK F_SETLK F_SETLKW (с ожиданием).

Используется структура flock.

```
struct flock {
    short  l_type;   //Тип блокировки; F_RDLCK, F_WRLCK, F_UNLCK.
    short  l_whence;          //Откуда считать позицию.
    off_t  l_start;           // Начальное смещение
    off_t  l_len;             // Размер участка, если 0 – до конца файла.
    pid_t  l_pid      //идентификатор заблокировавшего процесса,
                      //возвращается вызовом с флагом F_GETLK.
};
```

Пример установки блокировки на запись

```
-- Пример:
    int fd;
    struct flock fl;
    fd = open("testfile", O_RDWR);    if (fd == -1) …Ошибка!
    fl.l_type = F_WRLCK;                    // Блокировка на запись с позиции 100 на 10
    байт
    fl.l_whence = SEEK_SET;
    fl.l_start = 100;  fl.l_len = 10;
    if (  fcntl(fd, F_SETLK, &fl ) == -1 ) {
        if ( errno == EACCES || errno == EAGAIN) printf("Already locked by another
    process\n");
     } else {
            // Здесь записываем данные, а потом освободим занятый участок файла
        fl.l_type = F_UNLCK;
         fl.l_whence = SEEK_SET;
         fl.l_start = 100;        fl.l_len = 10;
         if ( fcntl( fd, F_SETLK, &fl ) == -1) Ошибка!
     }
// Можно одновременно освобождать смежные участки, занятые разными
    запросами!
```

Характеристики файла

```
#include <sys/stat.h>

int fstat(int fildes, struct stat *buf);
int lstat(const char *path, struct stat *buf);
int stat(const char *path, struct stat *buf);

    //Если путь относительный, то относительно открытого дескриптора каталога
int fstatat(int fd, const char *path, struct stat *buf, int flag);   //AT_SYMLINK_NOFOLLOW

struct stat {
    dev_t             st_dev;    Device ID of device containing file.
    ino_t             st_ino;      File serial number.
    mode_t            st_mode;    Mode of file (see below).
    nlink_t                      st_nlink;              Number of hard links to the file.
    uid_t             st_uid;      User ID of file.
    gid_t             st_gid;      Group ID of file.
    dev_t             st_rdev;    Device ID (if file is character or block special).
    off_t             st_size;     For regular files, the file size in bytes.
    struct timespec   st_atim;   Last data access timestamp.
    struct timespec   st_mtim;   Last data modification timestamp.
    struct timespec   st_ctim;   Last file status change timestamp.
    blksize_t         st_blksize;  A file system-specific preferred I/O block size
    blkcnt_t          st_blocks;   Number of blocks allocated for this object.
};
```

Проверка поля st_mode

S_IFMT	Проверка типа: if ((S_IFMT & st_mode)==S_IFDIR) …
S_IFBLK	Block special.
S_IFCHR	Character special.
S_IFIFO	FIFO special.
S_IFREG	Regular.
S_IFDIR	Directory.
S_IFLNK	Symbolic link.
S_IFSOCK	Socket
S_ISBLK(m)	Test for a block special file.
S_ISCHR(m)	Test for a character special file.
S_ISDIR(m)	Test for a directory.
S_ISFIFO(m)	Test for a pipe or FIFO special file.
S_ISREG(m)	Test for a regular file.
S_ISLNK(m)	Test for a symbolic link.
S_ISSOCK(m)	Test for a socket.
S_TYPEISTMO(buf)	Test macro for a typed memory object.
S_TYPEISMQ(buf)	Test for a message queue.
S_TYPEISSEM(buf)	Test for a semaphore.
S_TYPEISSHM(buf)	Test for a shared memory object.

Права доступа в st_mode

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
S_ISVTX	01000	On directories, restricted deletion flag.

Информация о файловой системе

```
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *restrict path, struct statvfs *restrict buf);
struct statvfs {
    unsigned long   f_bsize    Размер блока
    unsigned long   f_frsize   Размер базового блока
    fsblkcnt_t      f_blocks   Всего блоков
    fsblkcnt_t      f_bfree    Свободных блоков
    fsblkcnt_t      f_bavail   Свободных для пользовательского процесса.
    fsfilcnt_t      f_files    Сколько индексных дескрипторов
    fsfilcnt_t      f_ffree    Из них свободных.
    fsfilcnt_t      f_favail   Для пользователя.
    unsigned long   f_fsid     Идентификатор файловой системы
    unsigned long   f_flag     Флаги системы
    unsigned long   f_namemax  Макс. длина имени файла.
};
```

Временные файлы

```
#include <stdlib.h>

char *mkdtemp(char *template);      для каталога
int mkstemp(char *template);        для файла
// template – шаблон, вместо X цифры

Пример:
char template[] = "/tmp/fileXXXXXX";
int fd;
fd = mkstemp(template);

#include <stdio.h>
FILE *tmpfile(void);
char *tempnam(const char *dir, const char *pfx);        Имя временного файла
```

Знакомые по командной строке …

```
#include <unistd.h>

int chdir(const char *path);
int fchdir(int fildes);

int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fildes, uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner, gid_t group);
int fchownat(int fd, const char *path, uid_t owner, gid_t group, int flag);

#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
int fchmodat(int fd, const char *path, mode_t mode, int flag);

mode_t umask(mode_t cmask);
```

Ссылки на файл. Удаление файла. Каталоги

```
#include <unistd.h>

int link(const char *path1, const char *path2);
int linkat(int fd1, const char *path1, int fd2, const char *path2, int flag);
int symlink(const char *path1, const char *path2);
int symlinkat(const char *path1, int fd, const char *path2);
ssize_t readlink(const char *path, char *buf,  size_t bufsize);
ssize_t readlinkat(int fd, const char *path, char *buf, size_t bufsize);

int unlink(const char *path);
int unlinkat(int fd, const char *path, int flag);

#include <stdio.h>
int rename(const char *old, const char *new);

#include <sys/stat.h>
int mkdir(const char *path, mode_t mode);
int mkdirat(int fd, const char *path, mode_t mode);
int rmdir(const char *path);                    //Пустой!
```

Работа с содержимым каталога

```
#include <dirent.h>

DIR *fdopendir( int fd );
DIR *opendir( const char *dirname );

struct dirent *readdir( DIR *dirp );
int readdir_r( DIR *dirp, struct dirent *entry, struct dirent **result );
void rewinddir( DIR *dirp );
long telldir(DIR *dirp);
void seekdir(DIR *dirp, long loc);
int closedir( DIR *dirp );

struct dirent {
    ino_t  d_ino;              индексный дескриптор файла.
    char   d_name[];           Имя файла (не более NAME_MAX).
};
```

Векторные чтение и запись

```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);

struct iovec {
    void   *iov_base;
    size_t  iov_len;
};
-- Пример:
ssize_t bytes_written;        int fd;
char *buf0 = “1\n";           char *buf1 = “12\n";          char *buf2 = “1234\n";
int iovcnt;                   struct iovec iov[3];
iov[0].iov_base = buf0;               iov[0].iov_len = strlen(buf0);
iov[1].iov_base = buf1;               iov[1].iov_len = strlen(buf1);
iov[2].iov_base = buf2;               iov[2].iov_len = strlen(buf2);
bytes_written = writev(fd, iov, 3);
```

Синхронизация памяти и файла

```
#include <unistd.h>
void sync(void);

#include <unistd.h>
int fsync(int fildes);
```

Интересные функции:

```
#include <stdlib.h>
char *realpath(const char *file_name,   char *resolved_name);
```

Самостоятельно: Обход дерева каталогов

```
#include <ftw.h>
int ftw( const char *path, int (*fn)(const char *, const struct stat *ptr, int flag),
            int ndirs);
```