АКОС

Семинар 3

Файловая система

open

int open(const char *pathname, int flags /*, mode_t mode */);
int openat(int dirfd, const char *pathname, int flags /*, mode_t mode */);

Flags:
Должен быть один из **access_mode**: O_RDONLY, O_WRONLY, or O_RDWR.
Могут быть (через |) **creation flags**:
O_CLOEXEC, **O_CREAT**, O_DIRECTORY, O_EXCL, O_NOCTTY, O_NOFOLLOW, O_TMPFILE, **O_TRUNC**
Mode:
Если есть флаг O_CREAT или O_TMPFILE, то должен быть mode (z.B. 0642)
Return value:
fd или **-1** и выставляется errno

z.B.:
open(“main.c”, O_RDWR | O_CREAT, S_IRWXU | S_IRWXG);

int creat(const char *pathname, mode_t mode);
creat(…) == **open**(…) with flags equal to O_CREAT|O_WRONLY|O_TRUNC

openat ~ **open**, но если путь относительный, то он указывается относительно директории с дескриптором dirfd, а не от текущей.

read/write

```
#include <unistd.h>
```

ssize_t read(int fd, void *buf, size_t count);

- Пытается прочитать count байтов из файла с дескриптором fd и записать в буфер buf.
- Возвращает количество прочитанных байт. Может прочитать меньше чем хотелось бы, например, из-за EOF, нет символов в pipe или сигнала, **НО** это не является ошибкой.
- При ошибке возвращается -1, выставляется значение errno.

ssize_t write(int fd, const void *buf, size_t count);

- Пытается записать count байтов в файл с дескриптором fd из буфера buf.
- Возвращает количество записанных байт. Аналогично, может записать меньше из-за нехватки памяти или сигнала.
- При ошибке возвращается -1, выставляется значение errno.

pread/pwrite

```
#include <unistd.h>

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

То же самое, но с указанием offset

readv/writev

#include <sys/uio.h>

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

writev(int fd, const struct iovec *iov, int iovcnt);

То же самое, но с записью из/в несколько буферов

```
struct iovec {
  void  *iov_base;     /* Starting address */
  size_t iov_len;      /* Number of bytes to transfer */
};
```

lseek

```
#include <unistd.h>
```

off_t lseek(int fd**, off_t** offset**, int** whence**);**

Перемещает текущую позицию в файле на offset с учётом whence.

whence:

- **SEEK_SET** - позиция от начала файла
- **SEEK_CUR** - от текущего места в файле
- **SEEK_END** - от конца файла
- **SEEK_DATA** - находит ближайший участок с данными (после смещения offset)
- **SEEK_HOLE** - находит ближайшую дырку (после смещения offset)

opendir / mkdir

```
#include <dirent.h>

DIR *opendir(const char *name);
```

Открыть директорию

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

int mkdir(const char *pathname, mode_t mode);
```

Создать директорию

links

- **ln file_name hard_link_name** - создать жесткую ссылку (hard link)
- **ln -s file_name soft_link_name** - создать мягкую ссылку (soft link)

stat

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

int stat(const char *pathname, struct stat *statbuf); int
fstat(int fd, struct stat *statbuf);
int lstat(const char *pathname, struct stat *statbuf);
int fstatat(int dirfd, const char *pathname, struct stat *statbuf, int flags);

Возвращает информацию о файле.

lstat ~ **stat**, но если **lstat** указывает на ссылку, то он возвращает информацию о самой ссылке, а не о файле, на который указывает ссылка.

fstat ~ **stat**, но использует fd.

fstatat ~ stat, но если путь относительный, то он отчитывается от директории с дескриптором fd, а не от текущей.

stat64 ~ stat, но поля имеют больший размер

struct stat

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    nlink_t   st_nlink;       /* Number of hard links */
    uid_t     st_uid;         /* User ID of owner */
    gid_t     st_gid;         /* Group ID of owner */
    dev_t     st_rdev;        /* Device ID (if special file) */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

    struct timespec st_atim;  /* Time of last access */
    struct timespec st_mtim;  /* Time of last modification */
    struct timespec st_ctim;  /* Time of last status change */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

time

```
#include <time.h>

time_t time(time_t *tloc);
```

Возвращает количество секунд с начала **Epoch: 1970-01-01 00:00:00 +0000(UTC).**
Если **tloc** - не NULL - записывает значение и туда (на самом деле устарел, должен быть NULL)

```
struct timespec {
  time_t  tv_sec;  /* Seconds */
  long    tv_nsec; /* Nanoseconds */
};
```