OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Files and Directories – System calls

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Properties of a File

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

int stat(const char *restrict pathname, struct stat *restrict buf);

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

int lstat(const char *restrict pathname, struct stat *restrict buf);

int fstatat(int fd, const char *restrict pathname,
            struct stat *restrict buf, int flag);
```

All four return: 0 if OK, –1 on error

- **stat** function returns a structure of information about the named file
- The **fstat** function obtains information about the file that is already open on the descriptor fd.
- The **lstat** function returns information about the symbolic link, not the file referenced by the symbolic link
- The **fstatat** function returns the file statistics for a pathname relative to an open directory represented by the fd argument.

Structure that represents properties of a File

```
struct stat {
  mode_t          st_mode;    /* file type & mode (permissions) */
  ino_t           st_ino;     /* i-node number (serial number) */
  dev_t           st_dev;     /* device number (file system) */
  dev_t           st_rdev;    /* device number for special files */
  nlink_t         st_nlink;   /* number of links */
  uid_t           st_uid;     /* user ID of owner */
  gid_t           st_gid;     /* group ID of owner */
  off_t           st_size;    /* size in bytes, for regular files */
  struct timespec st_atim;    /* time of last access */
  struct timespec st_mtim;    /* time of last modification */
  struct timespec st_ctim;    /* time of last file status change */
  blksize_t       st_blksize; /* best I/O block size */
  blkcnt_t        st_blocks;  /* number of disk blocks allocated */
};
```

The timespec structure type defines time in terms of seconds and nanoseconds.

It includes at least the following fields:

time_t tv_sec;

long tv_nsec;

OPERATING SYSTEMS

File type macros

File type macros in <sys/stat.h>

Macro	Type of file
S_ISREG()	regular file
S_ISDIR()	directory file
S_ISCHR()	character special file
S_ISBLK()	block special file
S_ISFIFO()	pipe or FIFO
S_ISLNK()	symbolic link
S_ISSOCK()	socket

- Regular file - The most common type of file, which contains data of some form.
- Directory file - A file that contains the names of other files and pointers to information on these files
- Block special file - A type of file providing buffered I/O access in fixed-size units to devices such as disk drives
- Character special file - A type of file providing unbuffered I/O access in variable-sized units to devices.
- FIFO - A type of file used for communication between processes. It's sometimes called a named pipe
- Socket - A type of file used for network communication between processes
- Symbolic link - A type of file that points to another file

OPERATING SYSTEMS

File operations - creat, open and close

```
#include <fcntl.h>
int creat(const char *path, mode_t mode);
                    Returns: file descriptor opened for write-only if OK, –1 on error
```

- A new file can be created by calling the **creat** function.
- creat function is equivalent to **open** (path, O_WRONLY | O_CREAT | O_TRUNC, mode);

```
#include <fcntl.h>
int open(const char *path, int oflag, ... /* mode_t mode */ );
int openat(int fd, const char *path, int oflag, ... /* mode_t mode */ );
                    Both return: file descriptor if OK, –1 on error
```

openat function

```
#include <unistd.h>
int close(int fd);
                    Returns: 0 if OK, –1 on error
```

OPERATING SYSTEMS

File operations - lseek

```
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
                                    Returns: new file offset if OK, –1 on error
```

- Every open file has an associated ‘‘current file offset,’’ normally a non-negative integer that measures the number of bytes from the beginning of the file
- Read and write operations normally start at the current file offset and cause the offset to be incremented by the number of bytes read or written. By default, this offset is initialized to 0 when a file is opened, unless the O_APPEND option is specified.
- If whence is SEEK_SET, the file’s offset is set to offset bytes from the beginning of the file.
- If whence is SEEK_CUR, the file’s offset is set to its current value plus the offset. The offset can be positive or negative.
- If whence is SEEK_END, the file’s offset is set to the size of the file plus the offset. The offset can be positive or negative

OPERATING SYSTEMS

File operations - read

```
#include <unistd.h>

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

                                Returns: number of bytes read, 0 if end of file, –1 on error
```

- Data is read from an open file with the read function
- If the read is successful, the number of bytes read is returned. If the end of file is encountered, 0 is returned.
- The read operation starts at the file's current offset. Before a successful return, the offset is incremented by the number of bytes actually read.

OPERATING SYSTEMS

File operations - write

```
#include <unistd.h>

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

                                    Returns: number of bytes written if OK, –1 on error
```

- Data is written to an open file with the write function.
- The return value is usually equal to the nbytes argument; otherwise, an error has occurred
- A common cause for a write error is either filling up a disk or exceeding the file size limit for a given process
- For a regular file, the write operation starts at the file's current offset.
- If the O_APPEND option was specified when the file was opened, the file's offset is set to the current end of file before each write operation.
- After a successful write, the file's offset is incremented by the number of bytes actually written

OPERATING SYSTEMS

Directories

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

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

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

Both return: 0 if OK, –1 on error

```
#include <unistd.h>

int rmdir(const char *pathname);
```

Returns: 0 if OK, –1 on error

- Directories are created with the mkdir and mkdirat functions, and deleted with the rmdir function.
- The specified file access permissions, **mode**, are modified by the file mode creation mask of the process

OPERATING SYSTEMS

Reading Directories

```
#include <dirent.h>
DIR *opendir(const char *pathname);
DIR *fdopendir(int fd);
                                    Both return: pointer if OK, NULL on error
struct dirent *readdir(DIR *dp);
                          Returns: pointer if OK, NULL at end of directory or error
void rewinddir(DIR *dp);
int closedir(DIR *dp);
                                                Returns: 0 if OK, –1 on error
long telldir(DIR *dp);
                          Returns: current location in directory associated with dp
void seekdir(DIR *dp, long loc);
```

- Directories can be read by anyone who has access permission to read the directory. But only the kernel can write to a directory, to preserve file system sanity
- **fdopendir** provides a way to convert an open file descriptor into a DIR structure for use by the directory handling functions.
- The **dirent** structure defined in is implementation dependent.

```
ino_t  d_ino;          /* i-node number */
char   d_name[];       /* null-terminated filename */
```

Hard links

```
#include <unistd.h>

int link(const char *existingpath, const char *newpath);

int linkat(int efd, const char *existingpath, int nfd, const char *newpath,
           int flag);

                                                    Both return: 0 if OK, –1 on error
```

- **link** function or the **linkat** function can be used to create a link to an existing file.
- These functions create a new directory entry, newpath, that references the existing file existingpath.
- If the newpath already exists, an error is returned. Only the last component of the newpath is created. The rest of the path must already exist

```
#include <unistd.h>

int unlink(const char *pathname);

int unlinkat(int fd, const char *pathname, int flag);

                                                    Both return: 0 if OK, –1 on error
```

the file, the data in the file is still accessible through the other links

Symbolic links

```
#include <unistd.h>
int symlink(const char *actualpath, const char *sympath);
int symlinkat(const char *actualpath, int fd, const char *sympath);
                                    Both return: 0 if OK, –1 on error
```

- A symbolic link is created with either the symlink or symlinkat function
- A new directory entry, sympath, is created that points to actualpath. It is not required that actualpath exist when the symbolic link is created
- The symlinkat function is similar to symlink, but the sympath argument is evaluated relative to the directory referenced by the open file descriptor for that directory (specified by the fd argument)

THANK YOU

Suresh Jamadagni
Department of Computer Science and Engineering

sureshjamadagni@pes.edu