OPERATING SYSTEMS

Storage Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Storage Management – System calls

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all 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

OPERATING SYSTEMS

File access permission

- There are nine permission bits for each file, divided into three categories
- The term user in the first three rows refers to the owner of the file
- The first rule is that to open any type of file by name, user must have execute permission in each directory mentioned in the name, including the current directory.
- The execute permission bit for a directory is often called the search bit
- For example, to open the file /usr/include/stdio.h, user would need execute permission in the directory /, execute permission in the directory /usr, and execute permission in the directory /usr/include and appropriate permission for the file stdio.h

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

OPERATING SYSTEMS

File access permission

- The read permission for a file determines whether user can open an existing file for reading: the O_RDONLY and O_RDWR flags for the open function.
- The write permission for a file determines whether user can open an existing file for writing: the O_WRONLY and O_RDWR flags for the open function.
- User must have write permission for a file to specify the O_TRUNC flag in the open function.
- User cannot create a new file in a directory unless they have write permission and execute permission in the directory.
- To delete an existing file, user needs write permission and execute permission in the directory containing the file.
- Users do not need read permission or write permission for the file itself

Set-User-ID and Set-Group-ID

real user ID real group ID	who we really are
effective user ID effective group ID supplementary group IDs	used for file access permission checks
saved set-user-ID saved set-group-ID	saved by `exec` functions

- Every process has six or more IDs associated with it
- The real user ID and real group ID identify who the user really is. These two fields are taken from an entry in the password file when the user logs in
- The effective user ID, effective group ID, and supplementary group IDs determine file access permissions for the user
- The saved set-user-ID and saved set-group-ID contain copies of the effective user ID and the effective group ID, respectively, when a program is executed.

OPERATING SYSTEMS

Set-User-ID and Set-Group-ID

```
#include <unistd.h>
int setuid(uid_t uid);
int setgid(gid_t gid);
```

Both return: 0 if OK, −1 on error

- We can set the real user ID and effective user ID with the setuid function.
- We can set the real group ID and the effective group ID with the setgid function.
- If the process has superuser privileges, the setuid function sets the real user ID, effective user ID, and saved set-user-ID to uid.
- If the process does not have superuser privileges, but uid equals either the real user ID or the saved set-user-ID, setuid sets only the effective user ID to uid. The real user ID and the saved set-user-ID are not changed.
- If neither of these two conditions is true, errno is set to EPERM and −1 is returned

OPERATING SYSTEMS

access and faccessat

```
#include <unistd.h>
int access(const char *pathname, int mode);
int faccessat(int fd, const char *pathname, int mode, int flag);
                                        Both return: 0 if OK, –1 on error
```

mode	Description
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission

- When a user tries to open a file, the kernel performs access tests based on the effective user and group IDs
- Sometimes, however, a process wants to test accessibility based on the real user and group IDs. This is useful when a process is running as someone else, using either the set-user-ID or the set-group-ID feature.
- The **access** and **faccessat** functions base their tests on the real user and group IDs.

OPERATING SYSTEMS

umask

```
#include <sys/stat.h>
mode_t umask(mode_t cmask);
                                    Returns: previous file mode creation mask
```

- The umask function sets the file mode creation mask for the process and returns the previous value
- The cmask argument is formed as the bitwise OR of any of the nine constants
- The file mode creation mask is used whenever the process creates a new file or a new directory.

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

OPERATING SYSTEMS

chmod, fchmod, and fchmodat

```
#include <sys/stat.h>
int chmod(const char *pathname, mode_t mode);
int fchmod(int fd, mode_t mode);
int fchmodat(int fd, const char *pathname, mode_t mode, int flag);
                                    All three return: 0 if OK, –1 on error
```

- The chmod, fchmod, and fchmodat functions allow us to change the file access permissions for an existing file
- The chmod function operates on the specified file, whereas the fchmod function operates on a file that has already been opened.
- The fchmodat function behaves like chmod when the pathname argument is absolute or when the fd argument has the value AT_FDCWD and the pathname argument is relative
- To change the permission bits of a file, the effective user ID of the process must be equal to the owner ID of the file, or the process must have superuser permissions

OPERATING SYSTEMS

chown, fchown, fchownat, and lchown

```
#include <unistd.h>

int chown(const char *pathname, uid_t owner, gid_t group);

int fchown(int fd, uid_t owner, gid_t group);

int fchownat(int fd, const char *pathname, uid_t owner, gid_t group,
             int flag);

int lchown(const char *pathname, uid_t owner, gid_t group);
```

All four return: 0 if OK, –1 on error

- The chown functions allow users to change a file's user ID and group ID, but if either of the arguments owner or group is −1, the corresponding ID is left unchanged
- The fchown function changes the ownership of the open file referenced by the fd argument.
- lchown and fchownat (with the AT_SYMLINK_NOFOLLOW flag set) change the owners of the symbolic link itself, not the file pointed to by the symbolic link

OPERATING SYSTEMS

File truncation

```
#include <unistd.h>

int truncate(const char *pathname, off_t length);

int ftruncate(int fd, off_t length);
```

Both return: 0 if OK, –1 on error

- **truncate** and **ftruncate** functions truncate an existing file to *length* bytes.
- If the previous size of the file was greater than *length*, the data beyond *length* is no longer accessible.
- If the previous size was less than *length*, the file size will increase and the data between the old end of file and the new end of file will read as 0 (i.e., a hole is probably created in the file)

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu