Unix System Programming

Files and Directories

Chandravva Hebbi
Department of Computer Science and Engineering
chandravvahebbi@pes.edu

Unix System Programming

Files and Directories

Chandravva Hebbi

Department of Computer Science and Engineering

UNIX SYSTEM PROGRAMMING

Topics to be Covered

- File access permissions
- Ownership of New Files and directories
- access Function
- Link
- Unlink
- Symbolic Links
- Reading directories

File access permissions

- The st_mode value also encodes the access permission bits for the file.
- Nine access permissions

st_mode mask	Meaning

File access permissions

- The chmod(1) command used change the permissions.
- u for user (owner), g for group, and o for other are used to change the permissions of the users accordingly.
- Permissions required to open a file by name.
- For example, to open the file **/usr/include/stdio.h**
- Permissions to execute your program ./a.out
- If the flags are O_RDONLY and O_RDWR for the open function. Then read permission is required

File access permissions

- If the flags are O_WRONLY and O_RDWR for the open function.
- What permission is required to specifiy the O_TRUNC flag in the open function.
- To create new file in a directory, write permission required.
- To delete an existing file, write permission and execute permission
- Execute permission: to execute any file using exec()

Ownership of New Files and Directories

- The user ID and group ID of the new file whenever we execute creat,open calls
- The user ID of a new file is set to the effective user ID of the process.

The group ID of a new file are decided based on,

- The group ID of a new file can be the effective group ID of the process.

- The group ID of a new file can be the group ID of the directory in which the file is being created.

- FreeBSD 5.2.1 and Mac OS X 10.3 always uses the group ID of the directory as the group ID of the new file.
- Linux 2.4.22

UNIX SYSTEM PROGRAMMING

access Function

- When a file is opened, kernel performs access test.
- Tests based on the effective user and group Ids.
- Some times the test are done based on the real user and group IDs.
- The access function bases its tests on the real user and group Ids.

int access(const char *pathname, int mode);

Returns: 0 if OK, -1 on error

R_OK	Test for read permission
W_OK	Test for write permission

link Function

- Any file can have multiple directory entries pointing to its i-node.
- The way we create a link to an existing file is with the link function.

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

Returns: 0 if OK, 1 on -error

- This function creates a new directory entry, newpath, that references the existing file existing path.
- If the newpath already exists, an error is returned.

Unlink Functions

To remove an existing directory entry, the unlink function is used.

int unlink(const char *pathname);

Returns 0 if OK, -1 on error

write permission and execute permission in the directory containing the directory entry to unlink a file.

if the sticky bit is set in this directory we must have write permission for the directory and one of the following:

Own the file

Own the directory

Have superuser privileges

Symbolic Links

A symbolic link is an indirect pointer to a file, which pointed directly to the i-node of the file. Symbolic links were introduced to get around the limitations of hard links.

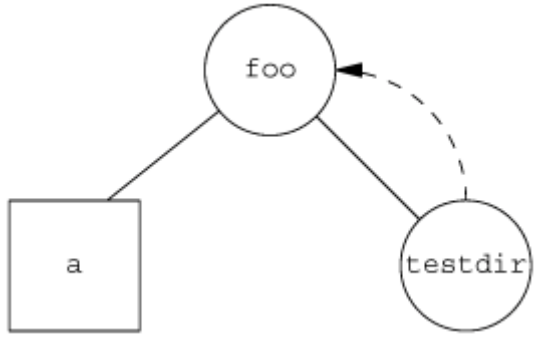
- Hard links normally require that the link and the file reside in the same file system

- Only the superuser can create a hard link to a directory.
- Symbolic links are typically used to move a file or an entire directory hierarchy to another location on a system.

Symbolic links

Symbolic links creates a loop

```
int symlink(const char *actualpath, const char
 *sympath);
```



mkdir and rmdir Functions


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

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

Returns 0 on success , -1 on error

Reading Directories


```
#include <dirent.h>

DIR *opendir(const char *pathname);
                Returns: 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);
```

THANK YOU

Chandravva Hebbi
Department of Computer Science and Engineering

chandravvahebbi@pes.edu