Embedded Operating System

Agenda

- **File Management**
  1. Directory
  2. Links
  3. File System Architecture
  4. File IO syscalls
  5. Disk allocation & Free space management
  6. Linux Ext2/3 FileSystems
  7. Journaling
  8. Disk scheduling algorithms

❖**Reading**

1. Galvin slides (File System & IO subsystem)
2. Professional Linux Kernel Architecture (Virtual File System, Extended File System)
3. Beginning Linux Programming (File SysCalls Programming)

File Management

- **File = Data + Metadata**
  - Data --> Data blocks
  - Metadata --> Inode (FCB)
- **File System = Boot block + Super block + Inode list + Data blocks**

- **Types of Files**
  - User perspective
    - Text files
    - Archive files
    - Media files
    - Document files
    - Executable files
    - etc.

- **Kernel perspective**
  - Regular files (-) (All user perspective file are regular file)
  - Special files
    - Directory files (d)
    - Link files (l)
    - Pipe files (p)
    - Socket files (s)
    - Device files
      - Char device files (c)
      - Block device files (b)

- **Directory**
  - From end user perspective, directory is a container which contains sub-directories and files.
  - However, OS treats directory as a special file.
  - The directory file contains one entry for each subdirectory or file in it.
  - Each directory entry contains i-node number and name of sub-dir / file.
  - terminal> ls -a -i -1 /home/sunbeam.

- **Directory Listing**
  - terminal> ls dirpath
  - Directory access library functions (man section 3)
    - opendir()
    - readdir()
    - closedir()

- opendir()
  - Open the directory file for reading.
  - DIR *dp = opendir("dir-path");
    - arg1: dir path to be opened
    - returns: DIR pointer if dir opened successfully, otherwise NULL

- closedir()
  - Close the directory file.
  - closedir(dp);
    - arg1: DIR pointer.

- **readdir()**
  - Read the next dirent from the directory file.
  - struct dirent *ent = readdir(dp);
    - arg1: DIR pointer.
    - returns: Pointer to struct dirent, if next entry is available.
      - Returns NULL if end of dir file is reached.
  - struct dirent
    - d_name --> name of file or sub-directory.
    - d_ino --> inode number of file or sub-directory

- Symbolic Link
  - A symbolic link, also known as a symlink or soft link, is a special type of file that points to another file or directory.
  - terminal> ln -s /path/of/target/file linkpath
  - Internally use
    - symlink() syscall.
    - man symlink
    - int symlink(const char *target_path, const char *link_path)
- symlink()
  - syscall A new link file is created (new inode and new data block is allocated), which contains info about the target file (absolute or relative path).
  - Link count is not incremented.
  - If target file is deleted, the link becomes useless.
  - Can create symlinks for directories also

Hard Link

- A hard link to a file points to the inode of the file instead of pointing to the file itself.
- This way the hard link gets all the attributes of the original file and points to the same data block as the original file.
- terminal> ln targetfilepath

- Internally use link() syscall.
  - man link
  - int link(const char *target_path, const char *link_path);

- link() syscall
  - A new directory entry is created, which has a new name and same inode number.
  - No new file (inode and data blocks) is created.
  - Link count in the inode of the file is incremented.
  - If directory entry of target file is deleted (rm command), file can be still accessed by link directory entry.
  - Cannot create hard link for directories, because it may lead to infinite recursion (while traversing directories recursively e.g. ls -R)

- rm command
  - The 'rm' means remove.
  - This command is used to remove a file.
  - The rm command in Linux, internally calls unlink() system call.
  - int unlink(const char *filepath);

- unlink() syscall
  - It deletes directory entry of the file.
  - It decrements link count in the inode by 1.
  - If link count = 0, the inode is considered to be deleted/free (updated into super-block).
  - It can be reused for any new file.
  - When inode is marked free, data blocks are also made free, so that they can also be reused for some new file

- **Directory**
  - Directory permissions/mode
    - r -- can read from dir data block -- list directory contents.
    - w -- can write into dir data block -- create new files & sub-directories, remove file/sub-directory, rename file.
    - x -- enable browsing the directory -- "cd" command

- File System Architecture
  - Virtual File System:
    - This layer redirect file system request to the appropriate file system manager.
  - File system manager:
    - File system manager enables access to repective file system on the disk.
    - OS can see all partitions whose file system managers are installed in that OS.
  - IO subsystem:
    - Implement buffer cache and other mechanisms to speed up disk IO.

- **Windows vs Linux**
  - Linux have FS mgrs for ext3/4, reiserfs, xfs, fat, ntfs, cdfs, etc.
  - Hence Linux support many FS.
  - Windows have FS mgrs for FAT, NTFS, CDFS.
  - Hence Windows do not support Linux FS.
  - However, third-party FS managers can be added into Windows to support Linux FS e.g. ext2fsd.

- **File IO syscalls**
  - open() syscall
    - fd = open("/home/nilesh/abc.txt", O_RDONLY);
    - step 1. Convert given file path into its inode number. This is called as path name translation and is done by a kernel ine file from the disk into inode table in memory. Inodes of all recently accessed files are kept in this table.
    - step 3. A file position is initialized to 0 and is stored in the open file table. It also stores mode in which file is opened and pointer to the in-memory inode. Infomation of all files opened in the system, is maintained in this table.
    - step 4. Each process is associated with a open file descriptor table. It keeps info of all files opened by that process. This entry stores pointer to the OFT entry.
    - step 5. Finally index to file desc table entry is returned, which is called as "file descriptor". All further read(), write(), lseek(), close() operations will be using this file desc.

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com