Embedded Operating Systems

Agenda

- Directory
- Links
- File IO System Calls
  - open()
  - close()
  - read()
  - write()
  - lseek()
- Disk allocation mechanisms
- Free space management mechanisms

Current Directory

- Each process has a current directory.
- The relative paths in the program are processed w.r.t. the current directory of that process.
- In Linux, struct task_struct (PCB) has a member struct fs_struct.

```
struct fs_struct {
    struct dentry *pwd;
    struct dentry *root;
    // ...
};
```

- By default current directory is inherited from the parent process.

chdir(dirpath)

- Change the current directory of the current process to the given directory path.
- arg1: dir path
- returns: 0 on success.

Directory permissions

- r (read): Can read directory entries.
  - ls command can list the directory.
  - readdir() can get directory entry.
- w (write): Can add new directory entry, modify existing directory entry, or delete directory entry.
  - new directory entry: when create new sub-directory (e.g. mkdir) or file (e.g. touch, ...).
  - modify existing directory entry: rename file or sub-directory.
  - delete directory entry: delete file or sub-directory (e.g. rm, rmdir, ...)
- x (execute): Can browse the directory.

- cd command (chdir() syscall) will work

Links

- Links are shortcuts to access deeply located files quickly.
- There are two types of links in Linux/UNIX.
  1. Symbolic Link
  2. Hard Link

Symbolic Link

- terminal> ln -s /path/of/target/file linkpath

```
cd ~
mkdir links
cd links
echo "file one" > one.txt
cat one.txt
ls -l -i
ln -s one.txt two.txt
echo "link created" >> two.txt
cat two.txt
cat one.txt
ln -s /home/sunbeam/links/one.txt three.txt
cat three.txt
ls -l -i
rm one.txt
ls -l -i
cat two.txt
rm two.txt three.txt
```

- 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

- terminal> ln targetfilepath

```
pwd
echo "file one." > one.txt
cat one.txt
ls -l -i
ln one.txt two.txt
echo "hard link created" >> two.txt
cat two.txt
cat one.txt
ls -l -i
ln one.txt three.txt
ls -l -i
rm one.txt
ls -l -i
cat two.txt
```

- 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 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.

File IO syscalls

open() syscall

- fd = open("file-path", flags, mode);

    - arg1: path of file to be opened
    - arg2: flags - how you want to open the file
        - O_RDONLY, O_WRONLY, O_RDWR -- read-write flags
        - O_TRUNC -- delete the contents of file while opening
        - O_APPEND -- write at the end of file
        - O_CREAT -- create a new file (if not present) -- must give arg3
    - arg3: mode - permissions for new file - octal number
    - returns file descriptor on success, and -1 on failure.
        - file descriptor is int that uniquely identifies the file in that process.
        - fd is used in other file io syscalls e.g. close(), read(), write(), lseek().
- MCQ questions
    - fopen "w" mode is equivalent to O_WRONLY | O_TRUNC | O_CREAT
    - fopen "r" mode is equivalent to ...
    - fopen "a" mode is equivalent to ...
    - fopen "w+" mode is equivalent to O_RDWR | O_TRUNC | O_CREAT
    - fopen "r+" mode is equivalent to ...
    - fopen "a+" mode is equivalent to ...
- 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 function namei().
    - step 2. Load the inode of the 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 flags 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.

VFS Structures

struct inode

- unsigned long i_ino; // inode number
- loff_t i_size; // file size
- unsigned int i_nlink; // number of hard links
- umode_t i_mode; // file mode (permissions)
- atomic_t i_count; // reference count
- struct list_head i_list; // inode cache
- Device driver related
    - struct list_head i_devices;
    - dev_t i_rdev;
    - union { struct pipe_inode_info *i_pipe; struct block_device *i_bdev; struct cdev *i_cdev; };
    - struct file_operations *i_fop;

struct file

- unsigned int f_flags; // open() arg2
- loff_t f_pos; // current file position
- struct path f_path; // pointer to dentry
- #define f_dentry f_path.dentry
- struct list_head fu_list; // open file table
- atomic_t f_count; // reference count
- Device driver related
    - struct file_operations *f_op;

struct dentry

- struct qstr d_name; // name of file/sub-directory
- struct inode *d_inode; // pointer to the inode
- struct list_head d_lru; // dentry cache
- atomic_t d_count; // reference count

struct fs_struct

- struct dentry * root; // stores "root directory" of the process --> used for absolute path
- struct dentry * pwd; // stores "current directory" of the process --> used for relative path
- int umask; // user file mode mask -- while creating new file this mask is used.

struct files_struct

- struct file * fd_array[NR_OPEN_DEFAULT];

struct task_struct

- struct fs_struct *fs; // current & root directory
- struct files_struct *files; // open file desc tables

Reference counting

- Used to manage life-time of any object (in complex systems e.g. Linux kernel, ...).
- Object has a member called as "refernce count". The count is incremented everytime new pointer points to the same object and decremented everytime the pointer to the object is no more used/required.
- At any moment, reference count is number of pointers referring to the object.
- When reference count become zero, it means no pointer it referring to the object and the object can be deleted safely.

File IO syscalls

close() syscall

- Decrement ref count in open file table entry (struct file).
- If ref count drops to zero, OFT entry is deleted (from OFT).

read() syscall

- count = read(fd, buf, length); -- syscall api
  - sys_read(fd, buffer, length) -- syscall implementation
  - vfs_read(file, buffer, length, inode) -- Virtual file system
    - Logical FS considers file as sequential set of bytes and set of blocks.
    - Example: block size = 4096 and file size = 20000 bytes, then number of blocks = 5 (0 to 4).
      - If current file position = 10000, then reading file block = 2
  - ext3_read(file, inode, file_block) -- File system manager
    - Refers inode and file disk block corresponding to the file block.
    - check buffer cache -- if disk block found.
      - if found, return it;
      - otherwise call disk driver to read that disk block from disk
  - disk_read(disk_device, disk_block) -- device driver
    - Read appropriate sectors and made it available into buffer cache.
    - The current process is blocked/sleep while disk read operation is in progress.