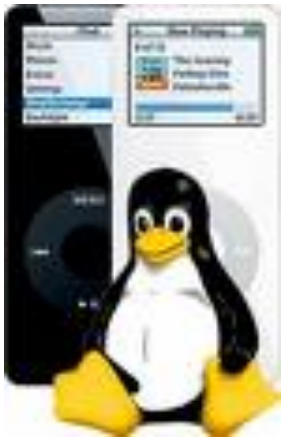
Systems Calls - Files

Alain NINANE – RSSI UCL

1er Mars 2016



File Management

- File names: links, symlinks, unlink
- File status: stat
- File renaming: rename
- File truncation: truncate
- File locking:

File Name Aliases (links)

- #include <unistd.h>
- int link(const char *name1, const char *name2)
  - name2 becomes another name for the exist. object name1
  - name2 is just another directory entry
  - name1 and name2 shares the same inode (ls -i)
  - nobody knows who is the original name
- Notes
  - links cannot cross file system boundaries
  - directories cannot be linked
    - (except by root on some unix systems)
- Demo
  - ls -l (see link count)
  - ls -i (see same inode)

unlink() system call

- #include <unistd.h>
- int unlink(const char *name)
  - Remove name from its directory
  - If this was the **last** reference to the object, the object is also ***removed***
- Note:
  - write permission is required on the **parent** directory

creat() and unlink() -> locks

- creat() and unlink() can be used to implement locks
  - while(creat("lockfile",0) < 0) sleep(...);
    - /*** Mode 0 ---> no access file !! ***/
    - /*** CRITICAL SECTION ***/
  - unlink("lockfile")
- Potential problem: orphan lockfile !!!!
- Demo: test_lock1
  - % ./test_lock1 & sleep 1 ; ./test_lock1

symlink() system call

- #include <unistd.h>
- int symlink(const char *name1, const char *name2)
  - name2 becomes a symbolic link to the object referenced by name1
- returns 0 if success; -1 if failure
- Notes:
  - Similar to hard links without limitations
    - file system boundaries, directories
    - are similar to Windows Shortcuts or MacOS Aliases
  - **symlink** are a special type file
    - file includes names of another file
  - **open()** open the **target** file
- Demo
  - ls -l (lrwx.....)

readlink() system call

- int readlink(path,buf,size)
  - const char *path;
  - void *buf; /* where to put result */
  - size_t bufsize; /* buffer size */
- Read value of a symbolic link
- Result
  - used length of buf
  - -1 in case of failure

user level commands

- link()
  - ln command
  - ln -s
    - symbolic links
  - ln [-s] target alias_name
- unlink()
  - rm
  - unlink

symlink and shell behavior

- symlinks are like “traps”
- Problem ...
  - “cd ..”
- Implementation
  - C Shell
    - cd .. --> go to upper (physical) directory of target
  - Korn Shell (&Bash)
    - cd .. --> go to previous (logical directory) of target
- Demo
  - cd applis (under bash or csh)
  - cd ..
  - pwd

symlink and other OS'es

- symlink are
  - like Windows Short Cuts
  - like Mac OSX Aliases
- Windows Short Cuts
- Mac OSX Aliases
  - are "special" objects on their own file systems
  - Demo: unix commands on MacOSX Aliases

stat() system call (I)

- #include <sys/types.h>
- #include <sys/stat.h>

- int **stat**(const char *path, struct stat *buf)
- int **lstat**(const char *path, struct stat *buf)
- int **fstat**(int fd, struct stat *buf)

stat() system call (II)

- stat() family system calls returns information about the specified object (file, directory, special device, ...)
  - stat: returns info. about file (target file if symlink)
  - lstat: returns info. about symlink itself
  - fstat: returns info. about object pointed to by fd
- buf is a pointer to a **stat** structure info
- Demo: See /usr/include/sys/stat.h
  - description of stat structure

rename() system call

- #include <stdio.h>
- int rename(const char *old, const char *new)
- Change the name of “old” file to “new”
- Notes:
  - **Atomic** equivalent of:
    - link(old,new);
    - unlink(old);

File and data locking

- Problems occurs when multiple processes access the same data simultaneously
- Data can be protected by locks
- Many ways to lock files and data structures.
  - flock, fcntl, lockf, ...
- Locks must be atomic
- Locks in UNIX are **advisory** !
- Locks in UNIX are **not mandatory** !
  - They are in windows !!
  - Exception: mount -o mand on some linux

flock() system call

- #include <sys/file.h>
- #define LOCK_SH 1 /* shared lock */
- #define LOCK_EX 2 /* exclusive lock */
- #define LOCK_NB 4 /* don't block when locking */
- #define LOCK_UN 8 /* unlock */
- int flock(int fd, int operation);
  - advisory lock on file described by fd
  - shared lock - can be used by more than one process
    - e.g. a file reader
  - exclusive lock - can be used by one and only one process
    - e.g. a file writer
  - LOCK_NB (returns -1 in case of file locked)

lockf() library function

- #include <unistd.h>
- int lockf(int fd, int cmd, off_t len);
  - fd: file descriptor
  - cmd:
    - F_LOCK: set the lock (or wait)
    - F_TLOCK: set the lock (and returns 0 or -1)
    - F_ULOCK: clear the lock
    - F_TEST: test the lock
  - len: length of locked area

lockf() library function

- Apply, test and remove a POSIX lock **on a file area**
- Area:
  - starts at the current file position (lseek)
  - stops: len bytes further
- Demo:
  - % test_lock2 start end
  - % ./test_lock2 10 20 & sleep 1 ; ./test_lock2 10 20

locking consistency

Bad example of locking unconsistency (locks.txt)

Because sendmail was unable to use the old flock() emulation, many sendmail installations use fcntl() instead of flock(). This is true of Slackware 3.0 for example. This gave rise to some other subtle problems if sendmail was configured to rebuild the alias file. **Sendmail tried to lock the aliases.dir file with fcntl()** at the same time as the **GDBM routines tried to lock this file with flock(). With pre 1.3.96 kernels this could result in deadlocks that, over time, or under a very heavy mail load, would eventually cause the kernel to lock solid with deadlocked processes.**