OS & Syscall Basics 3: Filesystems and Memory

1. Operating Systems
2. System calls
3. **Filesystems and Memory**
   - **filesystems**
   - **filesystem syscalls**
   - **file locking**
   - **memory management**
   - **program address space**
   - **memory management syscalls**
4. Processes, IPC, Signals

Filesystems

The method for storing files on harddrives and other devices are supplied by the OS, and are one of the most obvious aspects of the OS for users.

The format used to store files is referred to as a **filesystem**.

Linux supports a large set of filesystems, including "native" filesystems, various UNIX filesystems, and Windows filesystems.

The current standard native Linux filesystems are **ext3** and **ext4**.

Both are **journaling filesystems**, meaning they write a **journal** of disk changes so can quickly recover from power failures, etc.

Linux can also access Windows **FAT** and **NTFS** filesystems.

Filesystem Syscalls

There are a fairly large number of filesystem-related syscalls; here are some of the more important ones, grouped by basic purpose:

Syscalls for using files in programs:

open – open a file (return **file descriptor**)

close – close file descriptor (and so file)

read – read from file descriptor

write – write to file descriptor

lseek – reposition read/write offset for file descriptor

pread – read from file descriptor at a given offset

pwrite – write to file descriptor at a given offset

ftruncate – change file descriptor (file) length (size)

Filesystem Syscalls (contd.)

Syscalls for file and filesystem management:

creat – create a file

truncate – change file length (size)

link – create a new link/name for a file

symlink – create a symlink for a file

unlink – remove a link/name and possibly delete file

rename – rename a file, possibly moving it

chmod – change file permissions

lchown – change file ownership

stat – retrieve file/inode information

utime – set file/inode mtime, atime

mount, umount – mount, unmount a filesystem

Filesystem Syscalls (contd.)

Syscalls for directory management:

mkdir – create a new directory

rmdir – delete a (empty) directory

readdir – read one directory entry

In addition to the above kernel syscalls, there are a number of directory-related library wrapper "system calls" including:

opendir – open a directory (for use with `readdir`)

closedir – close a directory

seekdir – set position in directory

File Locking

File locking refers to preventing multiple processes from accessing and/or modifying a file at the same time.

Some system calls for synchronization and locking:

flock – apply/remove an advisory lock on open file

lockf – apply/remove an advisory lock on open file

fcntl – open file locking (plus more)

futex – fast mutex (locking)

(There are various issues with the file locking syscalls that can make their use unreliable.)

OS Memory Management

Another key function of an OS is **memory management**.

Some key elements of OS memory management are:

- allowing multiple programs to run "concurrently"
- allowing programs with larger address spaces than the amount of RAM to run
- preventing programs from interfering with each other
- supporting the use of **shared libraries** (DLLs)

OS Memory Management (contd.)

As with most moden OSs, Linux supports a **virtual memory** model and **paging** mechanism:

- each process has its own **address space**, which can be as large as is allowed by the architecture's address word size (e.g., 32 bits allows 0 through $2^{32} - 1$ or 4GB)
- object code addresses are **virtual addresses** that must be mapped to physical memory addresses at runtime
- the address space is broken up into equal size contiguous chunks called **pages**
- individual pages can be mapped into physical memory **page frames** as needed and in any order
- pages may also be **swapped/paged out** to a harddrive

OS Memory Management (contd.)

- the entire address space of a process need not be in physical memory at once to execute
- if a virtual address whose page is not in memory is accessed, a **page fault** is generated, causing the OS to **swap/page in** the needed page
- the process' **page table** keeps track of whether each page is in memory, its page frame or disk location, and other info
- a page may be accessible to multiple processes (for **shared libraries** and **memory-mapped files**)
- **page replacement algorithms** determine what pages should be in memory and which swapped out to avoid page faults as much as possible.

Process Address Spaces

The address space of any Linux process consists of several different **segments**:

text – machine code/instructions, generally read-only

data – *initialized* global and `static` variables

bss (or **BSS**) – *uninitialized* global/`static` variables (automatically zeroed/`NULL`'d at startup)

heap – dynamically allocated memory

memory mapping – shared libraries and shared memory IPC

thread stacks – stacks for addition Pthreads of process

stack – the program/execution stack (userspace)

These segments are placed in the address space in the order given: text segment at lowest addresses and stack segment at highest addresses.

Process Address Spaces (contd.)

The stack *bottom* is at the *top* of the (user's) virtual address range and *grows down* in address as stack frames are pushed on.

The heap sits at the *top* of the data segment and *grows up* in address if it must be expanded (e.g., by C runtime).

The top of the heap (end of program plus variables/data) is called the **program break**.

The virtual address range between the break and the stack top is used for **dynamic libraries**, **shared memory**, and **thread stacks**.

The text, data, and bss segments come directly from the contents of the **executable file** being run.

Memory Management Syscalls

System calls for allocating program memory were considered too architecture specific to be standardized in POSIX/SUS.

When writing portable code, the standard C language memory management functions are to be used: `malloc()`, `free()`, etc.

Of course, these functions may need to make kernel syscall(s) to *expand heap memory* (by expanding the data segment).

The Linux kernel syscall for this is named **brk** because the program break is adjusted.

Though not formally standardized, this syscall name is common among UNIXes (along with **sbrk**).

Memory Management Syscalls (contd.)

There are a few other memory management-related kernel syscalls, many dealing with mapping files into a process' address space:

mmap – map file or device into memory

munmap – unmap file or device into memory

msync – synchronize a file with a memory map

mremap – re-map a virtual memory address

remap_file_pages – create a non-linear file mapping

mprotect – set protection on address space pages

mlock, mlockall – lock part or all of process' virtual address space into RAM

munlock, munlockall – unlock process' virtual address space