Linux Essential Concepts

system calls

- system calls (short: *syscalls*)
  - function invocations made from user spaceinto kernel, to request services/resources from the OS
  - exmples: `read()`, `write()`, `get_thread_area()` (~ 380 syscalls)

glibc

- implements standard C library
- provides wrappers for system calls (and threading support + basic application facilities)

API and ABI

- Application Programming Interface (*API*)
  - defines interfaces by which one piece of software communicates with another at the *source level*
- Application Binary Interface (*ABI*)
  - defines binary interface between pieces of software on a particular architecture (e.g. x86-64)
  - concerned with e.g.
    - calling conventions, byte ordering, register use, system call invocation, *(cont'd)*
    - linking, library behavior, and the binary object format
- enforced by the toolchain (compiler, linker)

standards

- *POSIX* - Portable Operating System Interface
- *SUS* - Single UNIX Specification
- POSIX and SUS document e.g. the C API for a Unix-like operating system interface
- SUS subsumes/includes POSIX
- *LSB* Linux Standard Base; LSB Wikipedia[1]
  - ... standardize the software system structure, including the Filesystem Hierarchy Standard used in the Linux kernel. The LSB is based on the POSIX specification, the Single UNIX Specification (SUS) and several other open standards, but extends them in certain areas. ...

files / filesystem

- *everything is a file* philosophy
- to be accessed, a file must be opened; can be open for reading or writing or both
- **file *descriptor*** - reference to open file, maps metadata of open file to specific file
  - of C type `int`
  - *fd* abbreviated
  - shared with user space
  - a file can be opened more than once, each open instance has their own unique fd
  - concurrent file access - must be coordinated by user-space programs themselves
- **regular files**
  - bytes of data, organized in linear array called *byte stream*

[1] https://en.wikipedia.org/wiki/Linux_Standard_Base

- *file offset* or *file position* = location within the file
- *offset* max value = size of the C type used to store it (64 bits usually on modern systems)
- writing bytes to file positions beyond the end of file will cause intervening bytes padded with zero

```
#include <fcntl.h> // for open
#include <unistd.h> // for write
int main() {
  int fd; // file descriptor
  // O_RDWR open read+write, O_CREAT create if not exist
  fd = open("file.bin", O_RDWR|O_CREAT);
  char onebyte = 1;
  write(fd, &onebyte, sizeof(char));
  // 'p'=position; last param is  off_t offset
  pwrite(fd, &onebyte, sizeof(char), 10);
  close(fd);
}
```

```
$ hexdump file.bin
0000000 01 00 00 00 00 00 00 00 00 00 01
```

- **inodes**
  - a file is referenced by an *inode* (information node)
  - inode number (or *i-number* or *ino* = integer value unique to filesystem
  - *inode* stores file *metadata* e.g.
    - e.g. modification timestamp, owner, type, length, and the location
    - metadata does *not* include filename
  - both physical obj on disk; and data structure `struct inode` / in-meory representation in kernel
    - see kernel doc at The Inode Object[2]
    - They live either on the disc (for block device filesystems) or in the memory (for pseudo filesystems)
  - `ls -i` lists inode numbers e.g. `ls -i /tmp/file.bin`, output: `2671722 /tmp/file.bin`
  - *directories* - mapping of human-readable names to inode numbers
    - like any normal file, with difference that it contains only mappings of names to inodes
  - *link* = (file-)name and inode pair(-mapping)
    - . . . The physical on-disk form of this mapping—for example, a simple table or a hash—is implemented and managed by the kernel code that supports a given filesystem. . .
  - *dentry* = directory entry
  - *directory or pathname resolution* - kernel's walk though dentries to find a specific inode
  - *fully qualified / absolute pathname* - starting at `root` "/" directory
  - unlike normal files, kernel does not allow directories to be opened/manipulated like regular files
    - can only be manipulated via specific syscalls
- **hard links** and **symbolic links**
  - *hard link* multiple links map different (file-)names to the same inode
    - deleting a file *unlinks* it from directory (removing name-inode pair from directory)
    - each inode contains a *link count*

[2] https://www.kernel.org/doc/html/latest/filesystems/vfs.html

    * *symlinks* - has its own inode and data chunk, containing the complete pathname of linked-to file
    * Example: *note string length "hardlink.txt" = 12*

```
inode   Permissions   Links Size  Name
2674927 .rw-r--r--      2    0  hardlink.txt
2674927 .rw-r--r--      2    0  hardlink2.txt
2674944 lrwxr-xr-x      1   12  softlink.txt -> hardlink.txt
```

special files

- types of special files (four)
  - block device files
  - character device files
  - named pipes (FIFO)
  - Unix domain sockets
- **device files** - two groups 1. *character devices* and 2. *block devices*
  - character device - linear queue of bytes
  - block device - (accessed as) an array of bytes
- **names pipes** (or **FIFO**)
  - Named pipes are an interprocess communication (IPC)
  - communication channel over a fd (file descriptor), accessed via a special file
- **sockets** (Unix domain sockets)
  - advanced form of IPC (interprocess communication), multiple varieties
  - communication between two different processes; on same of different machines
  - *Unix domain socket* = form of socket used for communication within the local machine

filesystem and namespaces

- *unified namespace* in Linux/Unix (e.g. in Windows drives have a separate namespace such as `A:\`
- *filesystem* = collection of files and directories in a hierarchy
- *mount* / *unmount* - adding/removing a filesystem to global namespace
- filesystems...
  - *physical* - stored on disk
  - *virtual* - only exist in memory
  - *network* - exist on machines across network
- *sector* - smallest addressable unit on block device; pysical attribute of device
- *block* - smallest logically addressable unit on a filesystem
  - usually a power-of-two multiple of the sector size
  - generally larger than sector
- see Kernel doc 'Queue sysfs files'[3]...
  - ... and man (2) page for syscall 'stat, fstat, lstat, fstatat - get file status'
  - **hw_sector_size (RO)** - hardware sector size of the device, in bytes.

```
$ cat /sys/block/dm-0/queue/hw_sector_size
512
# block size
$ sudo blockdev --getbsz /dev/dm-0
4096
$ stat -f .
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 3724026    Free: 2246906    Available: 2052807
Inodes: Total: 950272     Free: 794298
```

[3] https://www.kernel.org/doc/html/latest/block/queue-sysfs.html

- *per-process namespaces* by default
  - each process inherits the namespace of parent
  - a process may create its own namespace with own set of mount points and a unique root directory.

processes

- *processes* are object code in executing / active, running programs, consisting of
  - object code
  - data
  - resources
  - state
  - a virtualized computer
- **ELF** *Excecutable and Linkable Format*
  - machine-runnable code in executable format kernel understands, contains:
  - metadata
  - multiple *sections* of code and data
- ELF **sections** - linear chunks of obj code, all bytes in a section are treated same (e.g. permission)
  - *text section* - executable code and read-only data (e.g. constants); read-only
  - *data sections* - initialized data e.g. C variables with defined values; read-write
  - *bss section* - uninitialized global data to be initialized (optimization) by *zero page*
  - *absolute section* - nonrelocatable symbols
  - *undefined section* - (catchall)
- process **resources**
  - managed by kernel
  - resource manipulation through system calls
  - examples: timers, pending signals, open files, network connections, IPC, ..
  - **process descriptor** - inside kernel structure for process resources, data, statistics, ..
  - process = **virtualization abstraction**
    - kernel supports both *preemptive multitasking* and *virtual memory*
    - each process has a single linear address space, as if it were in control of all of system memory
  - **threads**
    - each process consists of one or more *threads* ('thread of execution')
    - *thread* = the unit of activity within a process
    - a *thread* consists of
      - a stack (stores local variables)
      - processor status
      - current location of object code (usually processor's *instruction pointer*)
    - shared with process:
      - address space; thread share same virtual memory abstraction
    - kernel internal - views thread as normal processes to share some resources
  - **process hierarchy**
    - each process is identified by unique positive integer *process ID*
    - processes form a strict hierarcy, the *process tree*
    - PID = 1 first process or *init process*
    - new processes created via the `fork()` syscall
      - creates a duplicate of the calling (=parent) process
      - original process = *parent*, new process = *child*
      - child processes inherit the uids of their parents.
      - *reparenting* - if parent terminates before child, kernel will *reparent* child to init process
    - a process is not immediately removed...

- instead kernel keeps part in memory to allow parent to inquiry about staus WAIT(2):
- ...In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state ... "

users and groups

- *authorization* in Linux is provided by *users* and *groups*
- *user ID* (uid) - unique positive int associated with user
  - each process in turn associated with one uid; called process *real uid*
- uid 0 = *root*
- each user has a *primary* or *login group* (in `/etc/passwd`); *supplemental groups* are in `/etc/groups`

permissions

- each file is associated with
  - an owning user
  - an owning group
  - three sets of permission bits

signals

- *signals* - mechanism for one-way asynchronous notification
- typically alert a process about some event; about 30 signals implemented in kernel
- may be sent from:
  - kernel to process
  - process to another process
  - process to itself
- each signal is represented by a numeric constanct and a textual name e.g.

Signal	Value	Comment
SIGHUP	1	Hangup detected on controlling terminal
SIGINT	2	Interrupt from keyboard
SIGKILL	9	Kill signal

- signals interrupt a process...
  - causing it to stop whatever it is doing + immediately perform a predetermined action
  - processes may control what happens when receiving a signal ('predetermined action')
  - exceptions `SIGKILL` (always terminates) and `SIGSTOP` (always stops) processes

interprocess communication

- IPC mechanisms supported by Linux include:
  - pipes
  - named pipes
  - semaphores
  - message queues
  - shared memory
  - futexes