

Endianness

Value: 0xA0B0C0D0

index	0	1	2	3
• little	0xD0	0xC0	0xB0	0xA0
• big	0xA0	0xB0	0xC0	0xD0

* Little Endian puts the least significant (littlest) stuff first

• x86 is little endian, MIPS is big endian

• networking is done in big endian

IPC

Pipes (FIFO)

• requires copying memory from sender process to kernel memory to recipient process

• can pass large quantities of data

• explicit communication channel

• used just like a regular file descriptor

• Unnamed Pipe

* create using `pipe()`

* can only be used between processes with some parent-child relationship (or grandchild, sibling, etc)

* can override other fds with

- `dup(oldfd)` (next available fd) or

- `dup(oldfd, newfd)` (`newfd` is closed before being overwritten)

• Named Pipe

* created on filesystem using `mkfifo()`

* can be used by completely unrelated processes

Message Passing

Shared Memory

• two processes directly map the same region of physical memory

• is setup using system calls, but then all access is completely in userspace (for better speed)

Semaphore Sets

• shared integer that is enforced to be ≥ 0

• `P()` decrements

* blocks if the value is 0 (wait until non-zero and then immediately decrement)

• `V()` increments

* will never block

• binary semaphore: either 0 or 1

* increment on 1 has no effect and does not block

Signals

• `SIGINT`: keyboard interrupt

• `SIGSTP`: Ctrl-Z

• `sigwait()`: wait until one of a specific set of signals is caught

• signals are not queued, they are masked.
if the kernel delivers two duplicate signals to a process while it isn't scheduled, it will receive exactly one

Network

• client side:

* `getaddrinfo()` to get ip address from human readable address

* `socket()` to make a socket (returns fd)

* `connect()` to connect that socket to the address

• server side:

* `getaddrinfo()` with the port to get the address

* `socket()` to make socket

* `bind()` to bind your process to that socket

* `listen()` to listen on that socket (with a request backlog size)

* `accept()` to get a new connection to a client (returns a new fd)

• you can read/write from network sockets just like regular files

• `/etc/services` contains info about what services are running on what ports

• TCP/IP has 2^{32} possible addresses, and the protocol stack is in the kernel

• routers are comprised of layers: physical, data link, and network

• `netstat` (shell command) gives info about connections between this computer and remote servers/clients

• datagram socket: UDP broadcast?

Files

• descriptor table (DT):

* per each process, indexed by file descriptor

* points to entry in file table

* in user memory, but can't be directly modified

• file table (FT):

* shared by all processes

* contains:

- current cursor position

- reference count (# of descriptors)

- pointer to v-node entry

* in kernel memory

• v-node table (VT):

* one entry per file

* contains stat structure

* in kernel memory

• when `fork()`ing, the entire DT is copied and the relevant FT refcounts are increased

• unix IO vs stdlib IO:

* stdlib may introduce a layer of buffering to be more efficient

* stdlib works on non-unix OSes (portability)

* unix IO is most low level and high performance

* unix IO allows for accessing file metadata

Filesystem

• hard link shares an inode with the other file being linked (because there's really only one file)

* really, it's just when two directory entries point to the same file inode

* in this scenario, the reference count will be > 1

• soft link (symlink) is just another file that contains (in text format) the path to it's target

• superblock: contains info about entire filesystem. Size depends on version of unix/linux

• inode

* all are same size

* contained in inode table (an array of inodes)

* file metadata (permissions, timestamp, etc)

- also reference count: number of hard links

* 12 direct pointers to data blocks (enough for 48KB file)

* one single indirect pointer (+4MB)

* one double indirect pointer (+4GB)

* one triple indirect pointer (+4TB)

• data area

* files can be ≥ 1 block(s) (may not be smaller than 1 block)