Environment

- setting environment for program execution

MY_VAR='myVar'
export MY_VAR

- There are positional parameters passed in through argc and argv.
- There are also environment variables and special parameters that you can pass in by name instead of by position.
- getenv.c (listed in the class code part of content) uses both of these. You pass in the name of the environment variable through argv, and it prints it out with getenv().
- > HELLO=WORLD ./getenv HELLO
  value for HELLO is WORLD
- > ./getenv HELLO
  No value for HELLO
- ./getenv PATH
  value for PATH is
  a/bunch/of/different:paths/separted/by/colons:this/will/vary/from/computer/to:computer/because:PATH/is/an/environment/variable/already/in/the/shell
- In these examples, I'm pretty sure HELLO would be called a positional parameter, HELLO=WORLD is a special parameter, and PATH is an environment variable.

- retrieving environment (e.g., from within C)

https://www.systutorials.com/241139/how-to-set-and-get-an-environment-variable-in-c-on-linux/
https://www.tecmint.com/set-unset-environment-variables-in-linux/

File descriptors

- stdin, stdout, stderr (0, 1, and 2, respectively)
- redirection / duplication within a program
- pipe concepts

A pipeline is a sequence of one or more commands separated by one of the control operators | or |&. The format for a pipeline is:

[time [-p]] [!] command [[|â",|&] command2 ...]

The standard output of command is connected via a pipe to the standard input of command2. This connection is performed before any redirections specified by the command

- dup2() concepts

int dup2(int oldfd, int newfd);

Overwrites the oldfd (file descriptor) with the newfd.

Shell usage

- file descriptor redirection / duplication
- shell pipelines
- jobs / job control
  - background / foreground / suspend

- basic usage of echo, cat, grep

Processes

Definition: A process is an instance of a running program.

One of the most profound ideas in computer science

Not the same as “program” or “processor”

- fork

int fork(void)

Returns 0 to the child process, child’s PID to parent process

Child is almost identical to parent:

Child get an identical (but separate) copy of the parent’s virtual address space.

Child gets identical copies of the parent’s open file descriptors

Child has a different PID than the parent

fork is interesting (and often confusing) because it is called once but returns twice

- concurrency, concurrent execution

Two processes run concurrently (are concurrent) if their flows overlap in time

- synchronization with wait() / waitpid()
- exec
- orphaned processes ("daemons")
- zombies/reaping
- process groups, setpgid(), how a shell handles groups with signals, etc.

Stop and kill signals are sent to groups. The shell needs to split off children into a different group so it doesn’t get killed/stopped when it passes on the kill/stop signals

- retrieving status from exited child processes with macros: WIFEXITED/WIFSTOPPED/WIFSIGNALED/etc.

Exceptions

- asynchronous vs. synchronous

Caused by events external to the processor

Indicated by setting the processor’s interrupt pin

Handler returns to “next” instruction

Examples:

Timer interrupt

Every few ms, an external timer chip triggers an interrupt

Used by the kernel to take back control from user programs

I/O interrupt from external device

Hitting Ctrl-C at the keyboard

Arrival of a packet from a network

Arrival of data from a disk

Synchronous: Caused by events that occur as a result of executing an instruction:

- traps, faults, aborts

Traps

Intentional

Examples: system calls, breakpoint traps, special instructions
Returns control to “next” instruction
Faults
Unintentional but possibly recoverable
Examples: page faults (recoverable), protection faults (unrecoverable), floating point exceptions
Either re-executes faulting (“current”) instruction or aborts
Aborts
Unintentional and unrecoverable
Examples: illegal instruction, parity error, machine check
Aborts current program

- system calls
  Each x86-64 system call has a unique ID number
  Example:
  User calls: open(filename, options)
  Calls __open function, which invokes system call instruction syscall

Signals

- Signal blocking
  There are 2 bit arrays (Received, and blocked) that handle signals
- Sending vs. receiving a signal
  Use kill command to send signals
  When a signal is received the appropriate bit is set to 1
- SIGCHLD, SIGINT, SIGTSTP, SIGCONT, SIGTERM, SIGKILL
  SIGCHLD: A child has terminated
  SIGINT: A ‘would you please terminate’ signal (ctrl-c keyboard interrupt, can be caught)
  SIGTSTP: Program is to stop running
  SIGCONT: Program is set to running
  SIGTERM: ‘I really really want you to terminate’ signal (can be caught/ignored)
  SIGKILL: You dead son (CANNOT BE CAUGHT/IGNORED)
- Default actions, overriding with handlers or with SIG_DFL (default) or SIG_IGN (ignore).
- Signal blocking with sigprocmask()
  Sets the Blocked bitvector to a given bitvector. 1’s are blocked, 0’s are not blocked

Virtual Memory

- Virtual addressing and pages
- Page table / Page table entries (PTE)
- Identifying VPN and VPO given a VA

- Identifying PPN and PPO given a VA and a PTE
- Identifying a PA given a PPN and PPO
- TLB - access using TLB index, TLB tag
- Find a byte of data given a virtual address