CSSE2310/7231 — 4.1

OS + shell

OS

Thief of Time

What?

Write down what you think an operating system *does*.

Abstraction

- OS provides an abstraction of hardware (including CPU) to "user mode" code
- Reduces detail programs need to know
- Smooths over unimportant detail
  - Does the program need to know what sort of device F: is?
- "You are alone on the CPU"

OS — hardware help

OS needs hardware assistance to:

- Keep "user mode" code from doing "kernel" things.
  - Interrupts
  - Exceptions
  - System calls
- DMA
  - Why? Hierarchy of speed
- Memory things (later)

OS — CPU is forgetful

- OS relies on the CPU to not have long term (internal) memory (see TOT).
- Key idea:
  - Saving and restoring registers is enough for a program to think the cpu didn't do anything else.

Putting on the hat

Whenever the CPU switches to kernel mode[1]:

1. Jump to a predefined location in memory
2. Save registers
3. Do kernel things

To return to a process[2]:

1. restore previously saved registers
2. switch to user mode
3. jump to program code

[1] aka "supervisor mode" etc

[2] ie a running program

Getting into kernel mode

- Interrupts
  - Hardware $\rightarrow$ CPU (eg network interface $\rightarrow$)
  - Not directly visible to user programs
- Exceptions
  - Result from program actions (eg illegal floating point operation)
  - Visible because the user code wakes up in a handler
  - Not related to language based software execptions (although one might trigger the other).
- System calls
  - Deliberately triggered by user code via special op codes
  - Visible because the user program asked for them (often presented as a function call).
  - System call functions may using caching / buffering to avoid paying the high cost of getting the kernel involved.

Shells

Shells

Shells:

- Are unpriviledged program
- Are an interface between users and the kernel
- Provide scripting capabilities
- Are often (but not required to be) text based

Examples

Many

- Thompson Shell (sh) – the original UNIX shell
- Bourne Shell (sh) – from UNIX v7 (1977)
- Bourne-Again shell (bash) – superset of sh
- What you're probably using on moss
- Korn Shell (ksh)
- Z shell (zsh)
- C shell (csh)
- TENEX C shell (tcsh)
- Scheme shell (scsh)

In this course we will be assuming `bash`[3].

[3] Some detail and syntax varies with shell

Startup

1. Read startup files (eg .bashrc, /etc/bashrc)
2. Read commands
   - From standard in (for interactive use)
   - From a script file (when running as a script interpreter)

To be able to run text files as scripts, they need to have both `rx` permissions.

- Need read so the shell can read the script

Internal / External commands

- Some commands are built into the shell
  - eg `cd`, `alias`, `type`, `which`
- other commands are executable programs on the filesystem
  - eg: ls, gcc, vim
- Use `type` to see whether something is a builtin
- To find where an external command is located use `which`
- `echo $PATH` to see what directories are searched for commands in what order.

Variables

Variables

- Default to being strings
- Are either local
  - Not passed on to programs started by the shell (child processes)
- Or environment variables
  - Passed on to child processes

Shell variables can be promoted to the environment with `export`.
eg: `export COURSES`

Syntax

- Variables do not need to be declared (they are created by assigning something to them:
  `BOB=7`
  There is now a variable called BOB.
- Do not put cosmetic whitespace around operators.
- To read the value out of a variable, use `$`
  eg: `NPATH=$PATH:~/bin`
  `NPATH` now stores the contents of `PATH` followed by `:~/bin`
- Variable names are case sensitive.

Important variables

- `PATH` — directories to search for commands
- `LD_LIBRARY_PATH` — directories to search for dynamic libraries[4]
- `UID` — current user's (numeric) userid
- `USER` — current user's login name
- `HOME` — path to current user's home directory
- `$?` — exit status of the most recent command
- `$#` — argc - 1
- `$0` — argv[0]
- `$1` — argv[1]

`env` will give dump of current environment variables.

[4] Works on MacOS but they also have `DYLD_LIBRARY_PATH`

Special characters

- Wildcards for filenames:
  - `*` — Stands for zero or more characters
  - `?` — Stands for exactly one character
- `#` — Comment to the end of the line
- `&` — run a command in the background

  ```
  sleep 10 &
  ```
- `;` — run commands in sequence

  sleep 10 ; ls

Quote / escape characters!

```
# comment to end of line
\ escape the next character
$ z="A B C" # treat everything as a single item
      # (escapes spaces, does not escape $)
$ z='A B C' # like " but escapes more things
$ z=`A B C` # evaluates to the output of
      # the command in quotes
```

eg:

```
files=`ls`
```

would make `files` store all of the filenames in the current directory.

Redirecting and piping

- `cmd < in` — stdin uses file "in".
- `cmd > out` — stdout uses file "out"
- `cmd 2> out` — stderr uses file "out"
- `cmd1 | cmd2` — stdout of `cmd1` feeds into `stdin` of `cmd2`

Note

The shell commands covered in the linux tute are examinable.