Topics: process control / writing a shell

Approach: Figure out how to write a shell

Today's System Calls: execvp(), fork(), wait()

Outline

I. A Process is a Program in Action

II. Using `ps` to study processes and their attributes
pid, uid, size, time, nice, status, ...

III. What does a shell do?
allows a user to run programs, control i/o, script the action

```
$ grep fred /etc/passwd               # sending args to a cmd
$ TERM=vt100 ; export TERM ; vi       # setting global vars for cmd
$ grep fred /etc/passwd > fredlist    # send stdout to a file
$ ls /bin | wc -w                     # set up pipeline
$ if grep $TERM /etc/termcap          # control flow
> then
>    vi
> fi
```

IV. The Main Loop of a shell
while (get_a_command)
parse_into_args
do_the_command
wait for exit

Question 1: How does a program run a program?
answer: a process exec()'s the program
demo: psh1.c - shows use of execvp()
problem: exec() replaces the process with the new program

Question 2: How can we create a new process to exec the program
answer: use fork() to clone the current process
question: how does the child tell itself from the parent?
answer: the return value from fork() is 0 in the child
demo: forkdemo.c - shows how fork works

Question 3: What does the original process do during the exec()?
answer: it waits() if it wants to
demo: psh2.c - shows use of fork() and wait()

Question 4: What happens when ^C is typed while the program is running ?
answer: SIGINT from tty goes to all processes attached to the tty
question: what does this mean to the shell ?

future: program, not the user, should split command into args

V. Reflection: execvp() and exit() are like call and return
A C function can call a function and pass arguments to it
the called function returns a value to its caller.
A C *program* can exec() a C program and passes arguments to it
the exec()'ed program exit()'s a value to its parent
C functions can also pass values through global variables
C *programs* can pass values through the 'environment'