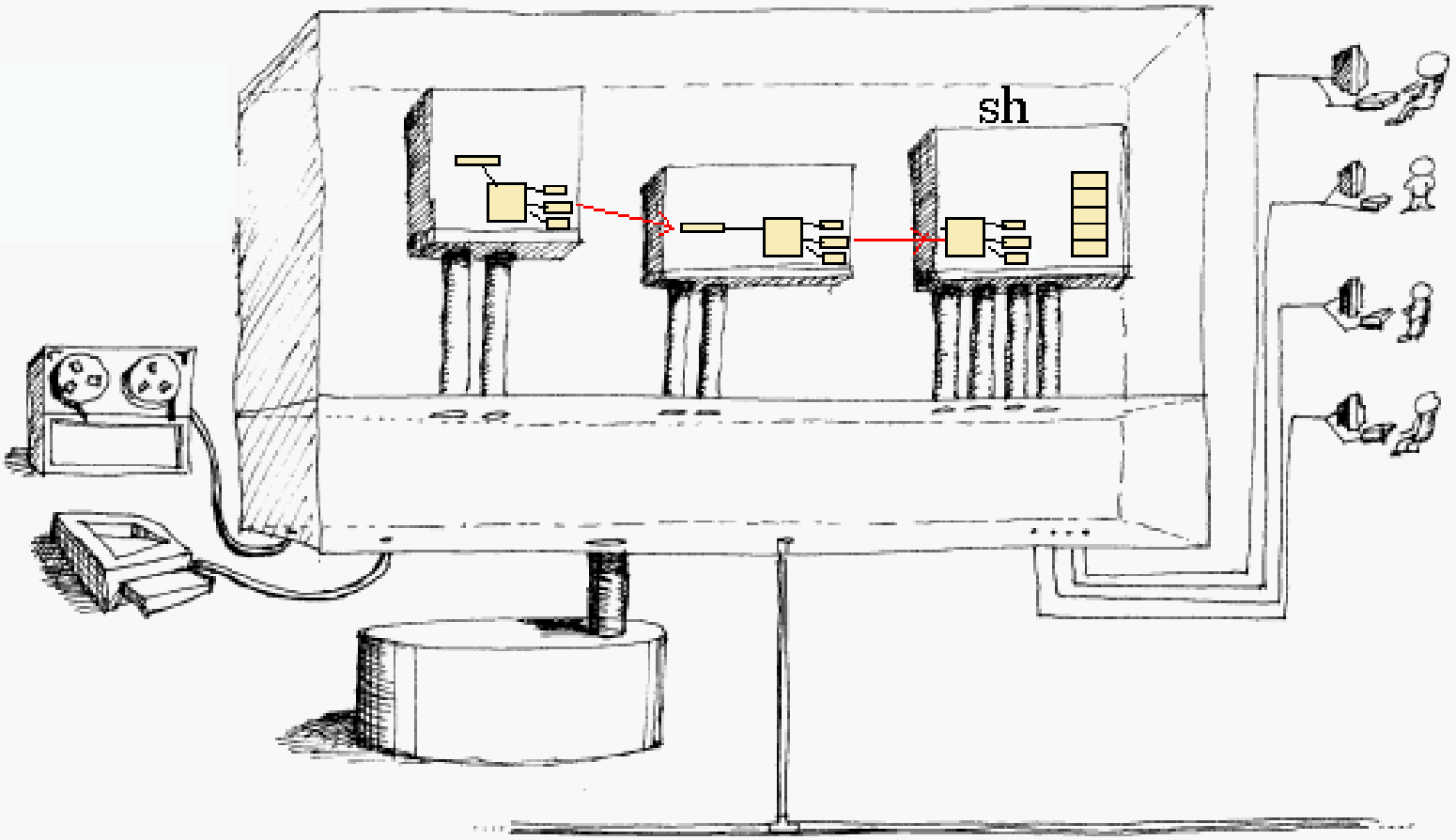
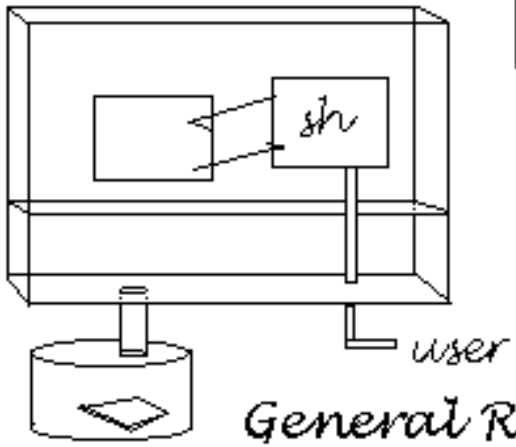
Lecture 9: The environment and shell variables



Class 9: Shell Programming & the Environment

(1) A unix shell runs programs AND is a programming language. Last time, we saw how a shell runs a program. This time, we look at shell programming.



Featuring:

- Shell scripts: what + why
- shell variables: what + why
- exit value, what + why
- Environment: what + why

General Remarks

a. A complex task can be solved by combining several separate programs.
b. The shell provides a language to control execution and communication of programs.
c. The result is a programming environment

Goal: Understand shell scripts
Understand how the shell processes scripts

Method: a. Write and study some shell scripts
b. Identify the underlying operations
c. Design datastructures and logic

(2) Shell scripts: what and why

script1

```
# comment here
# runs some commands
ls
echo the current data/time is
date
echo I am
who am i
```

to run it

```
$ sh script1   or   $ chmod +x script1   <- set executable bit
                    $ script1            <- now, run it
                    $ script1            <- over and over
```

scripts can be more than a batch of commands

script2

```
#!/bin/sh
# a simple phonebook searcher
BOOK=$HOME/phonebook.data
echo Find what name in phonebook
read NAME
if grep $NAME $BOOK > /tmp/pb.tmp
then
    echo Entries for $NAME
    cat /tmp/pb.tmp
else
    echo No entries for $NAME
fi
rm /tmp/pb.tmp
```

notes:

a. variables
b. user input ...
c. control structures (if, while, case..)
d. runs commands

scripts can pass values to commands by setting `environment variables'

script3

```
#!/bin/sh
# world clock program
printf "Time in Boston"
  TZ=EST5EDT ; export TZ ; date
printf "Time in Chicago"
  TZ=CST6CDT ; date
printf "Time in LA"
  TZ=PST8PDT ; date
```

notes:

a. `export' marks a variable as global
b. these environment variables are passed' to programs the shell calls
c. These programs can use environment variables as appropriate
d. e.g. $HOME, $PATH, $USER, $MAIL

Adding these Features to Our Shell

The story so far

In our last episode, we saw how to write a shell that used fork(), execvp(), and wait() to run commands.

It was pretty lame, though. The user had to type each argument on a separate line. It would be nice for the shell to parse a line into arguments.

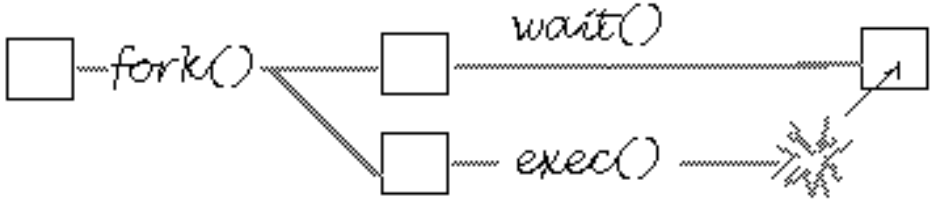
smsh1.c : a shell that parses

smsh1.c is the first really decent version. The user can type ls -l /bin /usr/tmp and the shell will split that into four arguments and run the command.

what next?

We shall gradually enhance this shell during the class and for homework.

2 Review of fork(), exec(), wait() : focus on wait



a. user types in a command

b. shell splits the line into arguments

c. shell uses fork() to create a new process

d. In the child process, the shell uses exec()
   to run a new program (same process)

e. Parent process waits for child to finish

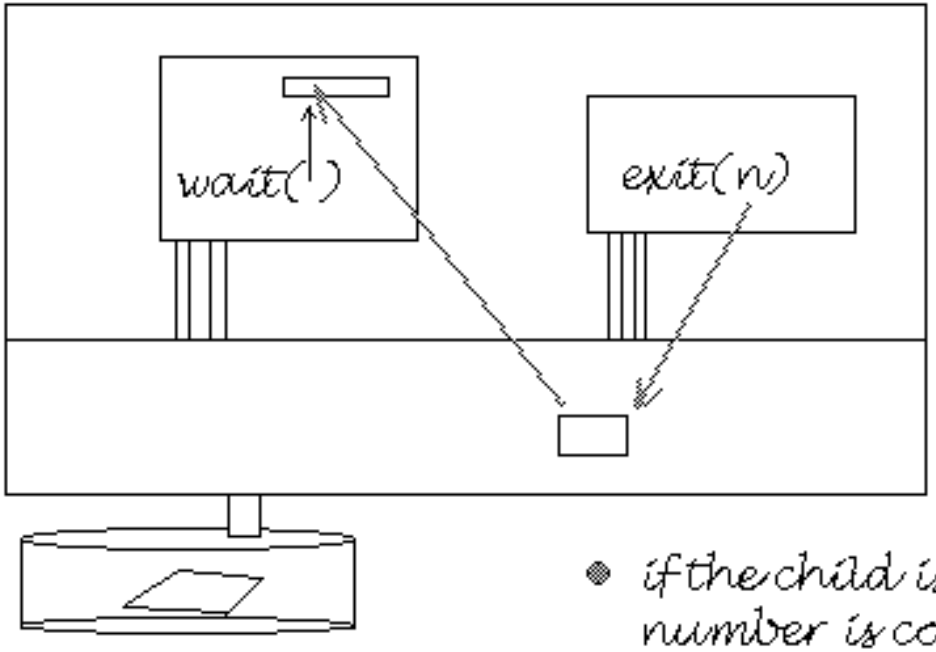
f. Child finishes by
   i. calls exit(n) OR
   ii. dies by receipt of signal (SIGINT, SIGALRM, ...)

g. Parent learns HOW the child finished

   example: forkquiz1.c shows how (g) works

The argument to wait() is a pointer to an int

the kernel stores in that int details of how the child finished



- exit() sends a value to the kernel. The kernel copies that value to a variable in the waiting parent

- if the child is killed, the signal number is copied to the variable in the parent.

if the child dumps core, a bit is set in parent's integer

(smsh2.c puts this to work)

3 Control Flow in the Shell : why and how

We shall now begin our work of enhancing our shell from a program launcher into a programming language.

First, we look at the if command

why? useful for stuff like:

```
if date | grep Fri
then
  echo time for backup
fi
```

how?

a. shell runs command after "if"
b. shell examines exit() value
c. exit(0) => OK,
   non-zero means NOT OK
d. Unix commands adhere to this convention
   see manpage for grep, diff, du, stty, ...

our shell add it for homework

(shell has while loops)

4 Variables in the Shell:

example

```
$ year=1630
$ read name

$ echo $year+$year
$ echo hello, $name, how are you
$ vi $name.c
```

why? its a prgramming language, silly

technical details

assignment:	var=value	(no spaces!)
user input:	read var	(or read v1 v2 ...)
reference:	$var	
list all vars:	set	

varnames: A-Za-z0-9_ no leading digit

values are all strings, no numerical values

Adding Variables to Our Shell

Our shell needs to have variables. We need a system for storing these name=value pairs, and we need a set of functions to

a. store values in variables
b. retrieve the value stored in a variable
c. list variables

the abstract model is

name	fred
book	phonebook.dat
HOME	/user/lib215
TERM	vt102

just a bunch of pairs of strings

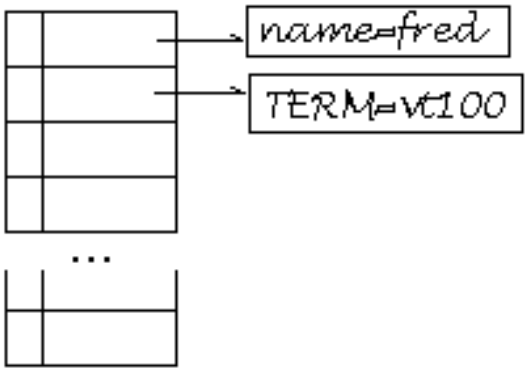
The implementation

Could be a linked list, a hashtable, a tree, almost anything. The varlib.c file has a simple, useful data structure and methods.

Implementation of the Variable Table in varlib.c

The set of shell variables is stored in an array of structs. Each struct has two members

- a pointer to a string of the form var=val
- a flag that marks a variable as global

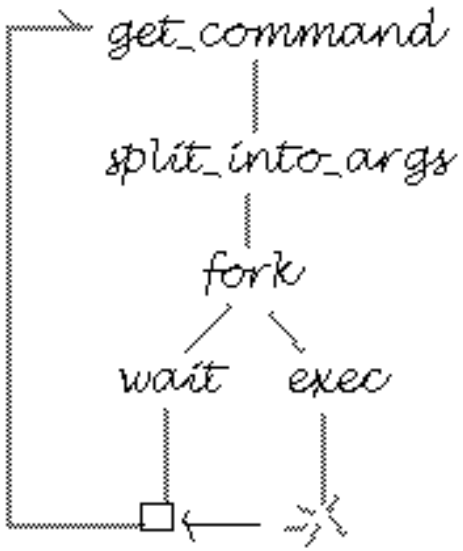


Interface:

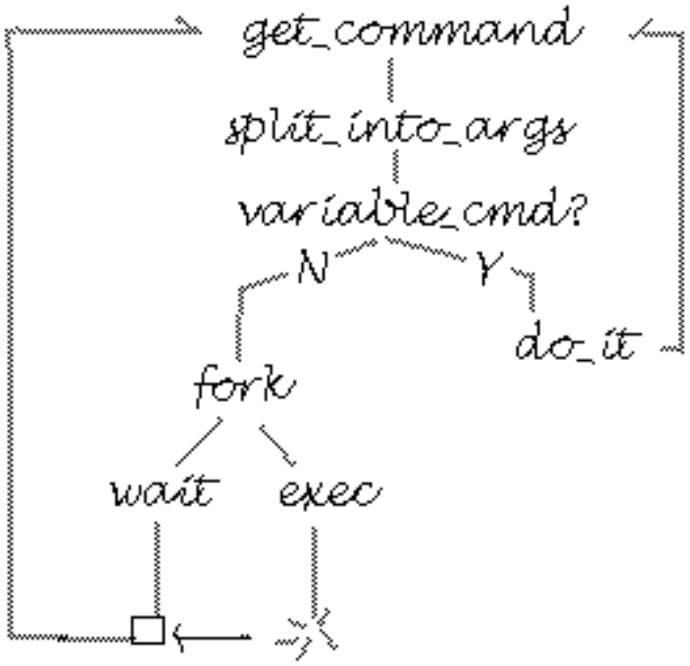
VLstore(char *var, char *val)	adds/updates item
VLlookup(char *var)	retrieves variable
VLset()	lists table to stdout

Adding Variables Requires Changing the Logic

Before



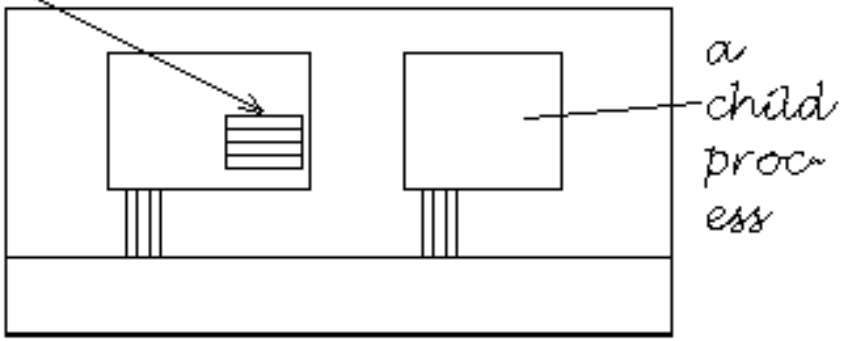
After



a. The path from fork() down is called 'execute()'
b. smsh3.c uses varlib.c and supports = and set
c. Actually replacing $var with value is for homework

We're Making Fine Progress

- We know how the if statement works, and we shall implement if for homework.
- We have a system for storing shell variables, and we have added an assignment statement and 'set' command to our shell. NOTE: these variables are all local to the shell, they are stored in the data space of our process.



Our final project is to study and implement global variables: settings that can be passed to other programs.

5 Global Variables in the Shell: the Environment

Why?

Some settings define a login session. Things like logname, terminaltype, timezone, path, editor, language, affect many programs. It would be nice to set those variables and make them available to all programs.

The Environment Does Just That

The environment is a list of name=value pairs that is automatically passed from one program to another via exec().

Commands to work with the environment

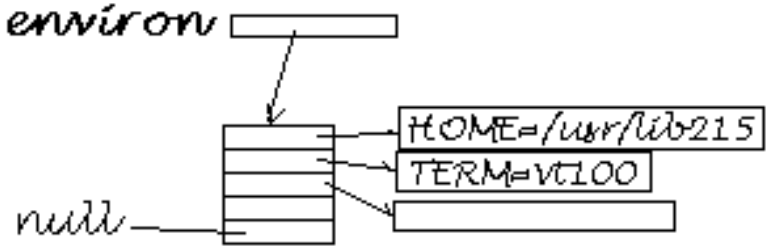
env	lists current settings
var=val	adds a variable
export var	makes var part of the environment

Access to the Enviroment from C Programs

A C program may call getenv(char *var) to fetch the value of environment variable 'var'. See the nice manual page.

What *is* the Environment? How's it work?

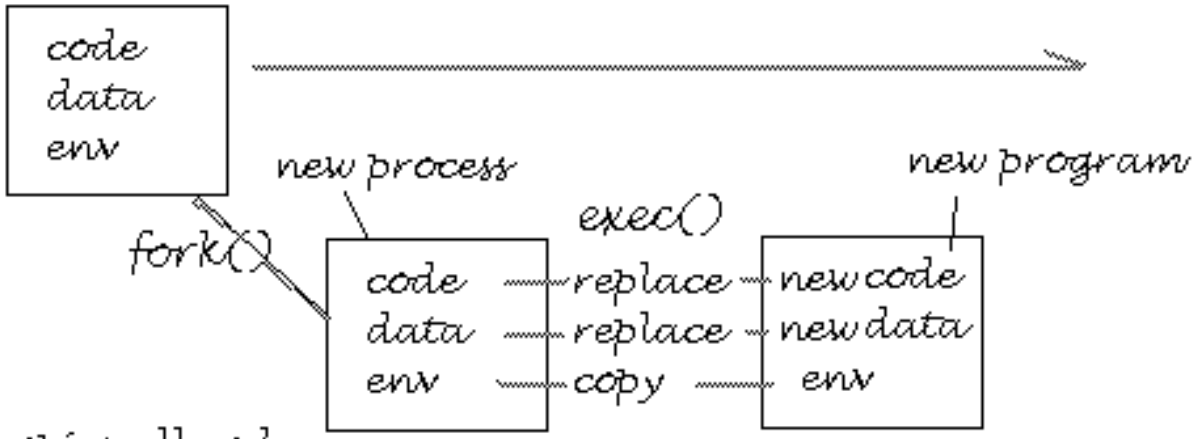
The environment is an array of strings, just like argv[]. Each string is of the form var=value. The address of the array is stored in the global variable



environ is a char **, it points to a list of char *'s
See showenv.c and envchange.c for demos

How is this list of strings passed to other programs?

Ans: The kernel copies the table when it performs the exec system call. The kernel looks for the variable called environ and copies what it points to INTO the data space of the NEW PROGRAM.



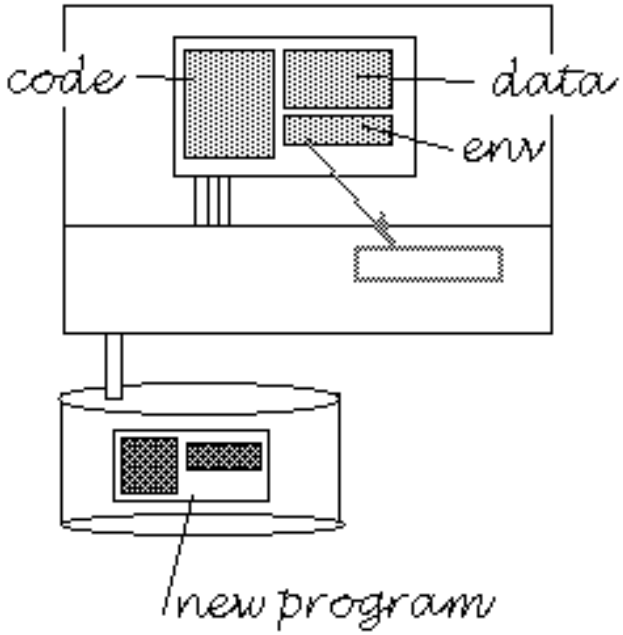
fork() copies all code, data, and environment from parent to create child process

exec() tosses out code and data of calling program, replaces them with code and data of new program but COPIES old environ

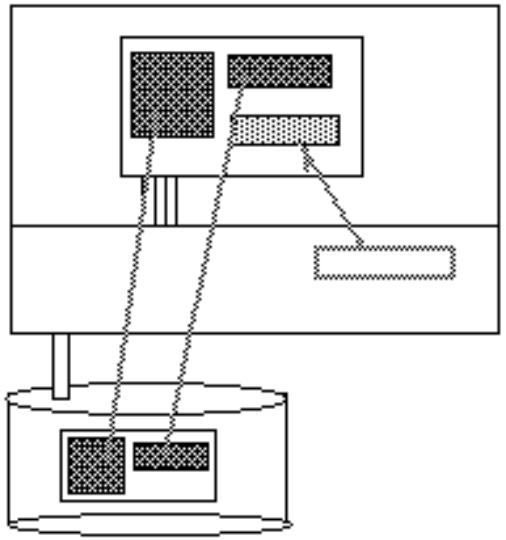
Copying the Environment across exec()

Remember: exec is a brain transplant! It removes all the code and data from the calling program. It makes an exception, though, and copies the environ into the process after the transplant is done.

Before exec()



After exec()



Adding the Environment to Our Variable Table

The shell copies the environment variables into its list of variables when the shell starts up. Before the shell exec()s a new program, it copies the 'global variables' back into the environment.

Our shell can do that by using varlib.c functions. See the code for details.

We also add the export command to our shell.

smsh4.c status report

feature	supports	needs
commands	runs programs	
variables	=, set	$var replacement read
if		if,then,else,fi
environ	all	
exit		exit
cd		cd

The Outline of smsh4.c