Environment Variables

COMP 321

Dave Johnson

1

What Is an Environment Variable?

Each process has a collection of "environment variables"

- Maintained and stored almost entirely by user library functions
  - With just a little help from the kernel for one part
- The environment list is a collection of character strings, each defining one environment variable, each of the form

"NAME=value"

Examples

- "HOME=/storage-home/d/dbj"
- "PATH=/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin"
- "PWD=/tmp"
- "SHELL=/bin/tcsh"

2

Seeing Them From the Shell

Command to print all of your shell's environment variables

```
$ printenv
HOME=/storage-home/d/dbj
PATH=/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
PWD=/tmp
SHELL=/bin/tcsh
etc…
```

Command to print any one of your shell's environment variables

```
$ echo $HOME
/storage-home/d/dbj
```


Where Do Environment Variables Come From?

	Using bash shell	Using csh shell
Set by the mechanism through which you accessed the system (e.g., sshd)	SSH_CLIENT=168.5.18.34 54321 22 SSH_CONNECTION=168.5.18.34 54321 128.42.124.179 22 SSH_TTY=/dev/pts/0 etc.	
Set by the login program	HOME, SHELL, PWD, USER, LOGNAME, etc.	
Set by your shell's system-wide initialization file	e.g., /etc/profile	e.g., /etc/csh.cshrc or /etc/csh.login
Set by your own user initialization file for your shell	e.g., ~/.bash_profile or ~/.bashrc	e.g., ~/.login or ~/.cshrc
Set by you on the command line	export NAME=value	setenv NAME VALUE

The User Library Function getenv()

```
char *getenv(const char *name);
```

Returns (just) the value string for the environment variable "name"

- Searches the environment variable strings for "name=value"
- ***Example***

```
char *my_home = getenv("HOME");
printf("My home is: %s\n", my_home);
```

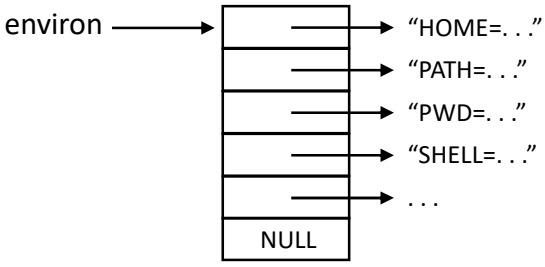
prints My home is: /storage-home/d/dbj

- If the environment variable "name" is not found, getenv() returns NULL

Where Are These Strings Stored?

```
extern char **environ;
```

A global pointer to an array of **char *** ***string pointers***



- The array is terminated by a NULL pointer
- ***All just stored as part of the data inside the user process's address space***

Environment Variables in a New Child Process

The child gets a copy of all of its parent's environment variables

- And it just happens ***automatically*** as part of the fork!

A fork copies all of the parent's address space to create the child

- The "environ" variable itself is in the parent's address space
- That array of char * string pointers is in the parent's address space
- All of the environment variable strings are in the parent's address space
- And fork copies the entire address space, so it's all there in the new child's address space, exactly where it is in the parent's address space
- (Note: modern implementations of fork are more efficient but still have the effect of ***copying*** the entire address space from the parent to the child)

Environment Variables After an execve()

Doing an exec replaces the entire user process's address space

- Everything in the address space is thrown away and replaced by loading the new program from the file in that same process to create new address space
- So the environment variable strings are thrown away too, as part of the old address space contents!

But the kernel helps by the execve kernel call saving these strings explicitly

```
int  execve(const char *pathname,  char *const  argv[], char *const envp[]);
```

- Copied into the kernel's memory, then copied into the new address space (the same as for the command line argv argument strings)
- All other "flavors" of exec (e.g., execl and exev) are user library calls that pass the pointer ***environ*** as the third argument to the execve() ***kernel*** call

More on C Runtime "Wrapper" Code for main()

Reminder: a small piece of assembly language code (traditionally called crt0)

- The real entry point for any program (the first code to execute)
- Different systems are a bit different, but in general this code does
  - Packages the command line arguments in argv[ ] format
  - ***Packages the environment variable strings in environ[] format***
  - ***Saves that array address in the global variable environ***
  - Calls status = main(argc, argv, ***environ***);
  - Calls exit(status); /* library exit(), which ultimately calls kernel _exit() */
- (Nobody really ever uses the third argument to main, and the POSIX standard doesn't even define it, but the argument still gets passed on most systems)