

Recitation Week #3

Corrections, Project 1 Hints, and the Shell Interface

Alejandro Cabrera
Operating Systems
COP4610 / CGS5765

Today's Recitation

- Corrections: fgets
- Shell Interface
 - What is a Shell?
 - Variants of the Shell
 - Shell built-ins vs. system binaries
 - System PATH and path resolution
 - Process basics

Fgets Return Value

- fgets does not return the number of characters it has read.
 - It returns the first argument passed to the function.
 - If unsuccessful, namely, when fgets encounters an error in the input or an End-Of-File (EOF), it returns NULL.

```
char *fgets(char *buffer, int buffer_size,  
            FILE *stream);
```

Shell: What is it?

- The Shell is an interpreter for a simple programming language.
- If you type a command it recognizes, it performs that command.
 - How does a Shell recognize a command?
- A Shell may also be run via a file.
 - If the Shell can understand this file, *a Shell script*, it will execute all the commands in that file.

Shells You May Have Seen Before

- sh – The first Shell produced. Came with the first Unix.
- csh – The C-shell,
- ksh – The Korn shell
- tcsh – The Tenex C-shell. (used on linprog)
- bash – The Bourne Again Shell (most Linux)
- DOS/cmd – The Windows Shell.

- From here on, the presentation assumes the bash Shell.

Shell Built-ins vs. System Executables

- A Shell provides certain commands built-in because they cannot be or are less convenient to provide as a separate executable.
 - `cd` – This changes the present working directory of the Shell.
 - `exec` – This executes a command for the Shell.
 - `pwd` – Prints the present working directory of the Shell.
- This means that the Shell performs these tasks for you, rather than searching the `PATH` to find an executable matching a command name.

The Environment

- When the Shell is loaded, it reads the environment variables:
 - PATH – Where to look for executables.
 - HOME – What to set the PWD to on loading.
- Type 'env' to see all variables defined!
 - `$> env`
 - `TERM=xterm`
 - `SHELL=/bin/bash`
 - `USER=cabrera`
 - `SHLVL=1`
 - ...

Observing the State of Environment Variables

- If you wanted to see the state of the environment variable, you can type:
 - `$> echo $(PATH)`
 - `/usr/local/sbin:/usr/local/bin: ...`
- This applies for any variable defined as listed by the 'env' command.

Setting Environment Variables

- Defining a new environment variable or changing the value of an old one can be accomplished by using the 'export' command:
 - `$> export GCC='/usr/bin/gcc-4.4.1'`
 - `$GCC procinfo.c -o procinfo`

The PATH (Variable)

- Usually looks like:
 - /usr/local/sbin:/usr/local/bin:/usr/bin/~/bin
- A colon-separated list of directory names.
- When the Shell is issued a command, it looks in each directory for a match.
- Example:
 - \$> echo
 - Trying '/usr/local/sbin/echo' ... failed.
 - Tryng '/usr/local/bin/echo' ... failed.
 - ...

Absolutes and Relatives

- A relative path looks like...
 - `$> ../proj1/procinfo -p`
- An absolute path looks like...
 - `$>/home/my_name/cop4610/proj1/procinfo -p`
- Absolute paths start from the root '/' of the file system and indicate how to reach a file exactly.
- Relative paths start from the PWD and use the special directories '..' and '.' to reach a file.
 - '..' - The parent directory
 - '.' - The PWD.
 - '~' - The HOME directory.

Path Resolution

- Path resolution is the process by which a Shell converts a given path name into an absolute path name.
 - `$~/cop4610/> cd ..`
 - `PWD` → `/home/name/cop4610/`
 - `..` → `/home/name/`
 - `PWD = ..`
 - `$~/>`

Processes

- A process is a runnable instance of a program.
- This means it has memory associated with it as well as a stack, a heap, a code segment, ...
- The Shell is a process that facilitates process creation.
 - \$> ls -l
 - Creates a process to execute 'ls -l'.
 - The Shell waits...
 - Once the process completes, control returns to the Shell.
 - \$>

Processes & Background Execution

- When a process is executed with the '&' included in the command list, it executes in the background.
- This means that the Shell does not wait- it regains control immediately:
 - \$> emacs &
 - [1] 8328
 - \$>
 - [Emacs open, you close it]
 - \$> <enter>
 - [1]+ Done emacs

Why Background Execution?

- If a process is spawned from the Shell in the foreground, the Shell cannot respond to user-input.
 - \$> nano
 -
 - ...
- In this case, you would have to exit nano in order to 'talk' to the Shell again.
 - Running make
 - Running ls
- Background execution allows a process to run while the Shell processes user input.

Input/Output Redirection

- The Shell also has the power to create files from running processes or to pass files to the standard input of another process.
 - Output:
 - `$> ls > my_ls.log`
 - Input:
 - `$> grep 'fgets' < /usr/include/stdio.h`
 - Error Output:
 - `$> g++ complicated_template_code.cpp &> error.log`

Input Redirection vs. File Input

- What's the difference between...:
 - `grep 'fgets' < /usr/include/stdio.h`
 - `grep 'fgets' /usr/include/stdio.h`
- The first is the same as calling `grep 'fgets'` and typing the entire contents of `stdio.h` to standard input, then pressing enter.
- The second has `grep` opening a file name `stdio.h` and reading from that file.

Pipes

- Pipes provide the highest level of convenience for having a set of processes communicate using only input and output re-direction.
 - `$> ls | more`
- ... is equivalent to...
 - `$> ls > tmp_file`
 - `$> more < tmp_file`
 - `$> rm tmp_file`

Advice

- Tinker around with the bash Shell over the next week.
 - Practice input/output re-direction
 - Experiment with background processes
 - Observe what happens if you type in a command that can't be found.
 - Observe what happens if a command is found.
 - How about input/out re-direction from a background process?

Next Time:

- Process Creation
 - How it really works, down to the system call level
- A deeper look at the Shell
 - How does it understand commands, input/out redirection, background processes...
- Parent and child processes
 - How does the parent know when the child is done?
- Zombies
 - What happens when a parent process terminates, but the child lives on?
- Project 2

Any Questions?