Systems Programming

updated : 28 sept 2019

08 - Shells

Overview

Last Time

- Stdio
- Chapter 13 – I/O buffering
- Shells:
  - Basics: read command into doubly linked list
  - Shell variables, set command
  - background,
  - Substitutions: variable substitutions, pseudo filename completion, history substitution,

Readings for today

- Review: stdio, I/O buffering
- Shells:
  - Basics: read command into doubly linked list
  - Shell variables, set command
  - background,
  - Substitutions: variable substitutions, pseudo filename completion, history substitution,
  - Simple I/O redirection
- Loose Topics: umask; Set-uid bit, sticky-bit

Rehits from last time

- Stdio
- buffering

Bash Reference Manual

- Reference Documentation for Bash Edition 4.4, for Bash Version 4.4. September 2016

- 178 page pdf
  - http://www.gnu.org/software/bash/manual/bash.pdf

Unix's Command Language Interpreter: the Shell

- Print prompt
- Read command
- Perform substitutions (command, arithmetic)
- Save in history
- Execute
  - Fork/vfork
  - Remap I/O if necessary with dup
  - Execute the command in the child
  - Parent usually waits on child to get exit status
- What happened when a child process exits without a parent waiting for the child process?

Shell basics

- History
  - Thompson, Bourne Sh(V7)
  - csh, tcsh, Korne
  - Bash
- Substitutions
  - History (last time) (!str:0)
  - Filename expanison (ls *.c)
  - Filename completion
    - str<TAB>
  - Variable expansion (ls $c5)
  - Alias, …
- Standard Bash variables
  - PATH, HOME etc
- Builtin functions
  - cd

Chapter 3 Basic Shell Features

- 3.1 Shell Syntax
- 3.1.1 Shell Operation
- 3.1.2 Quoting
- 3.1.2.1 Escape Character
- 3.1.2.2 Single Quotes
- 3.1.2.3 Double Quotes
- 3.1.2.4 ANSI-C Quoting
- 3.1.3 Comments
- 3.2 Shell Commands
- 3.2.1 Simple Commands
- 3.2.2 Pipelines
- 3.2.3 Lists of Commands
  - Separators- ; & && ||
- 3.2.4 Compound Commands
- Scripts

Searching the PATH

- PATH=dir1:dir2: … dirn
- Then when the shell goes to execute a comand say “ls” it search the directories in order looking for an executable named “ls”
- It executes the first “ls” found
- ‘.’ in the path means the current directory
- Actually no search hash table of commands
- Related command:
  - which

CD - Why must it be built-in?

- cd path
- cd
- cd p1 p2 ??

- Directory stack
  - pushd path
  - popd
  - `dirs` – dump stack
- CDPATH
  - Like PATH except for cd
- Related commands: pwd, chdir(2)

Bash functionality

Substitutions (in order)

- Brace expansion
- Tilde expansion
- Parameter and variable expansion
- Arithmetic expansion & command substitution
  - These are at the same level done left to right
- Word splitting
- Filename expansion

Brace Expansion

- “Brace expansion is a mechanism by which arbitrary strings may be generated”
- Examples:

hermes> echo a{d,c,b}e

ade ace abe

Used for shorthand as in

- mkdir /usr/local/src/bash/{old,new,dist,bugs}
- or
- chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}

http://www.gnu.org/software/bash/manual/bash.pdf

Tilde Expansion

- **Tilde-prefix**
  - from ~ upto the first slash
  - Treated as possible login name
- **~ -- the tilde is replaced by the shell variable HOME**
- **~user -- is replaced by HOME of user**
- **~+ -- is replaced by PWD**
- **~- -- is replaced by OLDPWD**
- **~+N -- is replaced by ‘dirs +N’**

http://www.gnu.org/software/bash/manual/bash.pdf

Shell Parameter Expansion

- '$' character introduces parameter expansion, command substitution, or arithmetic expansion
- parameter name or symbol to be expanded may be enclosed in braces
- Examples :

```
hermes> echo $rx

hermes> echo ${rx:=xxx}

xxx

hermes> echo $rx

xxx

hermes> echo ${rx:=yyy}

xxx
```

http://www.gnu.org/software/bash/manual/bash.pdf

More options than you can handle

- ${parameter: -word}
- ${parameter:=word}
- ${parameter:?word}
- ${parameter:+word}
- ${parameter:offset}
- ${parameter:offset:length}
- ${!prefix*}
- ${!prefix@}
- ${!name[@]}
- ${!name[*]}
- ${#parameter}
- ${parameter#word}
- ${parameter##word}
- ${parameter%word}
- ${parameter%%word}
- ${parameter/pattern/string}
- ${parameter^pattern}
- ${parameter^^pattern}
- ${parameter,pattern}
- ${parameter,,pattern}

http://www.gnu.org/software/bash/manual/bash.pdf

Shell Parameter Implementation

- Table
  - Name
  - Value

- Base Shell Variable Substitution

1. Locate word in command that is $name or $(name)
2. Look in the table for name and get value
3. Replace $(name) with the value from the table

- Variations, defaults
- Commands to manipulate table: set, name=value

http://www.gnu.org/software/bash/manual/bash.pdf

Standard Shell Variables

- PATH
- CDPATH
- HOME=/acct/f1/matthews
- HOSTNAME=hermes
- PS1='hermes> '
- PS2='> '
- PS4='+ '
- PWD=/class/csce510-001/Examples
- SHELL=/bin/bash
- SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
- SHLVL=1
- DIRSTACK=()
- HISTFILE=/acct/f1/matthews/.bash_history
- HISTFILESIZE=500
- HISTSIZE=500
- TERM=vt100
- …

http://www.gnu.org/software/bash/manual/bash.pdf

Command Substitution

- $(command) or `command`
- Bash performs the expansion by
  - executing command and
  - replacing the command substitution with the standard output of the command
- savedir=`pwd` -- not really necessary just for illustration
- Maybe nested with quotes ‘\’

http://www.gnu.org/software/bash/manual/bash.pdf

Alias Substitution

- **Essentially the same except**
  - Separate table name/value pairs
  - Only applies to first word of command
  - No $ needed; check every first word against the table
- **Alias command**
  - Alias lsl="ls –lrt"
- **Alias expansion**
  - $lsl *.c

http://www.gnu.org/software/bash/manual/bash.pdf

3.5.5 Arithmetic Expansion

- $((expression)) primarily used in scripts

http://www.gnu.org/software/bash/manual/bash.pdf

3.5.8 Filename Expansion

- Bash scans each word for the characters ‘*’, ‘?’, and ‘[’.
- ‘*’ - wildcard matches any string (delimited by /)
  - ls mlib*c
  - ls a*/*.c
- ‘?’ – Matches any single character
- ‘[’ … ‘]’- matches any character in the class
  - ls [a-z].c -- single character named c files

- Note: looks like regular expressions, match somewhat like reg expr, but definitely not regular expressions

http://www.gnu.org/software/bash/manual/bash.pdf

Others Subs that I have never used

- 3.5.6 Process Substitution
- 3.5.7 Word splitting

http://www.gnu.org/software/bash/manual/bash.pdf

Towards regular expressions

- ?(pattern-list)
  - Matches zero or one occurrence of the given patterns.
- *(pattern-list)
  - Matches zero or more occurrences of the given patterns.
- +(pattern-list)
  - Matches one or more occurrences of the given patterns.
- @(pattern-list)
  - Matches one of the given patterns.
- !(pattern-list)
  - Matches anything except one of the given patterns.

http://www.gnu.org/software/bash/manual/bash.pdf

3.6 I/O Redirections

- Output redirection implementation
- ls -l > listing
- Fork before you start messing with the per process open file table (_iob[])

http://www.gnu.org/software/bash/manual/bash.pdf

Variations of redirection

- **[n]< word**
- [n]> word
- [n]>> word

- [n]>[|]word
  - Noclobber option

http://www.gnu.org/software/bash/manual/bash.pdf

3.6.4 Redirecting Standard Output and Standard Error

- two formats for redirecting standard output and standard error:
  - &>word
  - >&word
- Of the two forms, the first is preferred. This is semantically equivalent to
  - >word 2>&1

- Appending also works

http://www.gnu.org/software/bash/manual/bash.pdf

3.6.6 Here documents, etc

- << *word*

  Here-document

  Another stuff

  word

- **3.6.7 Here Strings -- <<< $word**
- **3.6.8 Duplicating File Descriptors**
- **3.6.9 Moving File Descriptors**

http://www.gnu.org/software/bash/manual/bash.pdf

Redirections with pipes

- `ls –l | grep “^d” | wc`
  -- count the subdirectories

- **Named pipes - fifos (`mkfifo`)**
  - live in hierarchy ‘p’ as file type first character in ls -l

http://www.gnu.org/software/bash/manual/bash.pdf

PIPE(2) Linux Programmer's Manual PIPE(2)

NAME

pipe, pipe2 - create pipe

SYNOPSIS

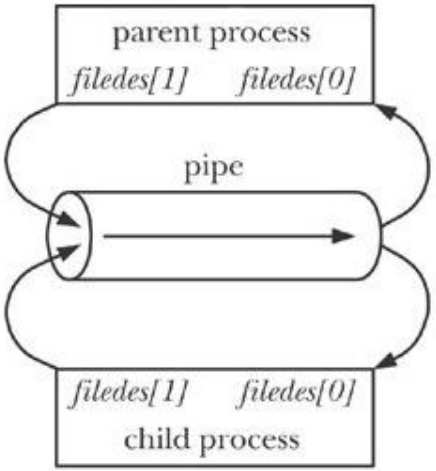
```
#include <unistd.h>
int pipe(int pipefd[2]);

#define _GNU_SOURCE
#include <unistd.h>
int pipe2(int pipefd[2], int flags);
```

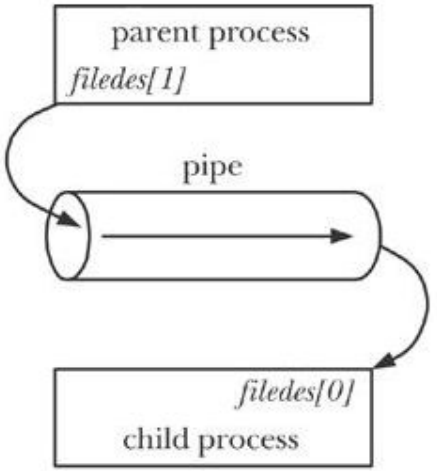
DESCRIPTION - pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

For further details, see pipe(7).

Figure 44-3 Setting up a pipe



a) After *fork()*



b) After closing unused descriptors

- ls -l | wc
- Sh forks; waits
- Child forks (2nd fork)
- Child (ls)
  - Closes read side pfd[0]
- Child of Child (CofC)
  - Closes write side pdf[1]
- Note also requires redirecting stdio

TLPI/pipes

- **hermes> cd pipes**
- **hermes> ls**

simple_pipe.c

pipe_ls_wc.c

popen_glob.c

change_case.c

fifo_seqnum_server.c

fifo_seqnum_client.c

pipe_sync.c

Pipes/pipe_ls_wc.c (sh executes)

Code Should do Overview

- Bash forks – parent waits
- Child (ls)
- Pipe syscall
- Fork - C and CofC

- Child (ls)
  - Close pfd[0]
  - Remap stdout to pdf[1]
    - Close(1)
    - Dup(pfd[1])
    - Close(pdf[1])
  - Execve “ls”
- ChildofChild
  - Close pfd[1]
  - Remap stdin to pdf[0]
    - Close(0)
    - Dup(pfd[0])
    - Close(pdf[0])
  - Execve “wc”

Details - Child Code

```
switch (fork()) {
  case -1:     errExit("fork");
  case 0:              /* First child: exec 'ls' to write to pipe */
      if (close(pfd[0]) == -1)            errExit("close 1");

      /* Duplicate stdout on write end of pipe; close it */
      if (pfd[1] != STDOUT_FILENO) {        /* Defensive check */
          if (dup2(pfd[1], STDOUT_FILENO) == -1)
                  errExit("dup2 1");
          if (close(pfd[1]) == -1)
                  errExit("close 2");
      }
      execlp("ls", "ls", (char *) NULL);       /* Writes to pipe */
      errExit("execlp ls");
  default:           /* Parent falls through to create next child */
      break;
  }
```

Details - Child of Child Code

```
 switch (fork()) {
    case -1:    errExit("fork");
    case 0:    /* Second child: exec 'wc' to read from pipe */
         if (close(pfd[1]) == -1)     errExit("close 3");

       /* Duplicate stdin on read end of pipe; close duplicated fd */
            if (pfd[0] != STDIN_FILENO) {      /* Defensive check */
                    if (dup2(pfd[0], STDIN_FILENO) == -1)
                        errExit("dup2 2");
                    if (close(pfd[0]) == -1)  errExit("close 4");
            }

            execlp("wc", "wc", "-l", (char *) NULL);
    errExit("execlp wc");
default: /* Parent falls through */
        break;
}
```

Examples/pipe.c

```
/* * A simple pipe
example to illustrate ls
| wc */

#include <sys/types.h>
main(){
int pfd[2];
pid_t p;
if(pipe(pfd) < 0)
   fatal("Pipe failed");
if((p = fork()) < 0)
   fatal("Fork failed");

if(p == 0){
close(0);
dup(pfd[0]);
close(pfd[1]);
close(pfd[0]);
execlp("wc","wc",(char*)0);

} else{
close(1);
dup(pfd[1]);
close(pfd[0]);
close(pfd[1]);
execlp("ls","ls",(char*)0);

}
```

Programming assignment 2 -Shell

- Simple I/O redirection; single pipes
- Builtin functions: set, cd, exit
- Startup file ~/.mybashrc
- Filename expansion
- Shell variable(parameter) expansion
  - Standard ones
    - PATH, CDPATH
- Environ passed

QA

Pipeorder ls | grep | wc