Structure of a command shell

--writing our own, homemade but full-fledged

David Morgan

Capabilitites of the “real” shell

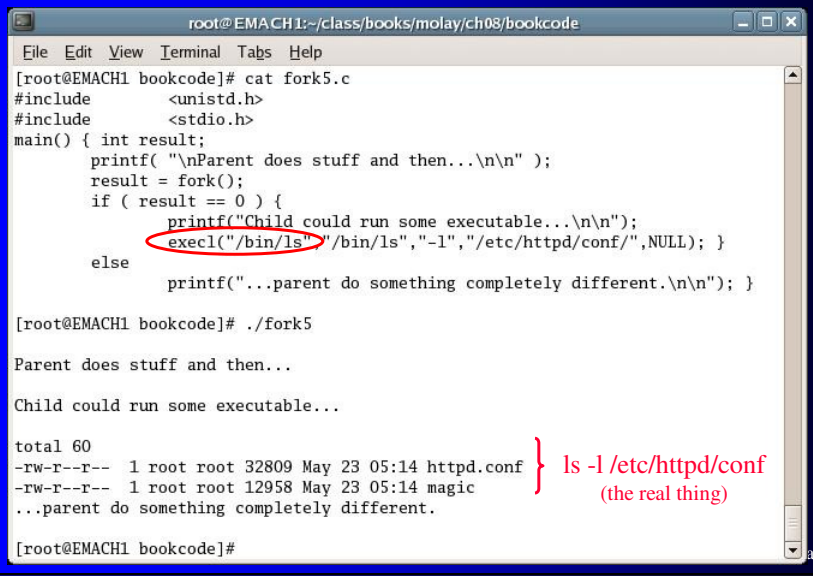
- Command processing
  - parse
  - expand
  - execute
- I/O redirection
- Piping
- Environment control
- Background processing
- Shell scripts

The dispensible ones (⊠)

- Command processing
  - pa⊠e
  - ex⊠and
  - execute ← the single essential, for a *"command"* shell
- I/O ⊠redirection
- Pi⊠ing
- En⊠ironment control
- Ba⊠kground processing
- Sh⊠l scripts

Review: process spawns process

the fork/exec forumla (fork5 spawns ls)



To make a “shell” out of it…

- make it orderly — timing discipline
- make it interactive — user involvement
- make it repetitive — as long as he wants

Some system function calls

- fork - creates a child process that differs from the parent process only in its PID and PPID
- exec - replaces the current process image with a new process image
- wait - suspends execution of the current process until its child has exited
- exit - causes normal program termination and a return value sent to the parent

Make it orderly

insert wait (parent) & exit (child)

root@EMACH1:~/class/books/molay/ch08/bookcode - Shell - Konsole

Session Edit View Bookmarks Settings Help

```
[root@EMACH1 bookcode]# cat fork6.c
#include        <unistd.h>
#include        <stdio.h>
#include        <stdlib.h>
main() { int result;
        printf( "\nParent does stuff and then...\n\n" );
        result = fork();
        if ( result == 0 )
                {
                printf("Child could run some time-consuming executable...\n\n");
                execl("timewaster","timewaster",NULL);
                exit(0);
                }
        else
                printf("...while parent might plunge ahead in parallel...\n");
                wait(NULL);
                printf("...and/or wait for child termination to continue.\n\n");
 }

[root@EMACH1 bookcode]# ./fork6

Parent does stuff and then...

Child could run some time-consuming executable...

...while parent might plunge ahead in parallel...
...and/or wait for child termination to continue.

[root@EMACH1 bookcode]#
```

Shell

stall till child is done

prog that wastes 10 sec

tell parent when done

here, 10 sec passes

05-18

Make it interactive

insert scanf( ) for program name

root@EMACH1:~/class/books/molay/ch08/bookcode - Shell - Konsole

Session Edit View Bookmarks Settings Help

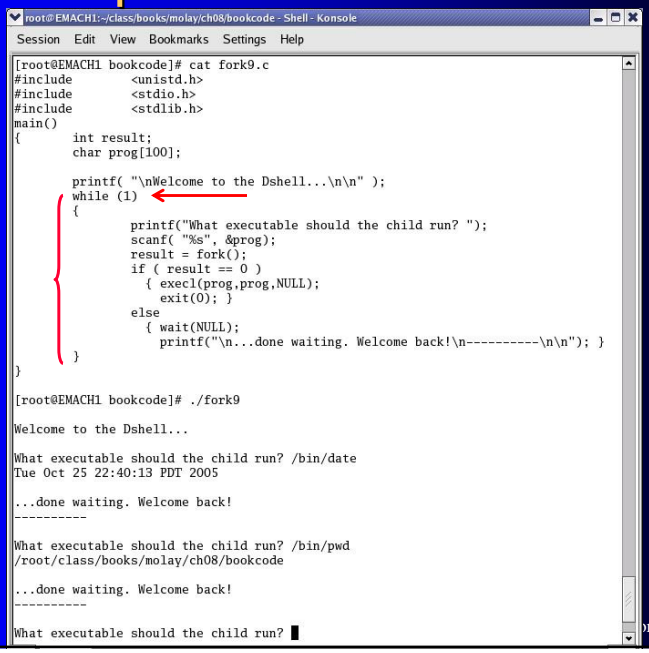
```
[root@EMACH1 bookcode]# cat fork8.c
#include        <unistd.h>
#include        <stdio.h>
#include        <stdlib.h>
main() { int result;
        char prog[100];
        printf( "\nParent does stuff and then...\n\n" );
        result = fork();
        if ( result == 0 )
                {
                printf("Child could run a user-specified executable:\n");
                printf("What executable should the child run? ");
                scanf( "%s", &prog);
                execl(prog,prog,NULL);
                exit(0);
                }
        else
                wait(NULL);
                printf("\n...parent continues now, only when child is done.\n\n"
); }
[root@EMACH1 bookcode]# ./fork8

Parent does stuff and then...

Child could run a user-specified executable:
What executable should the child run? /bin/date
Tue Oct 25 22:32:25 PDT 2005

...parent continues now, only when child is done.

[root@EMACH1 bookcode]#
```

Shell

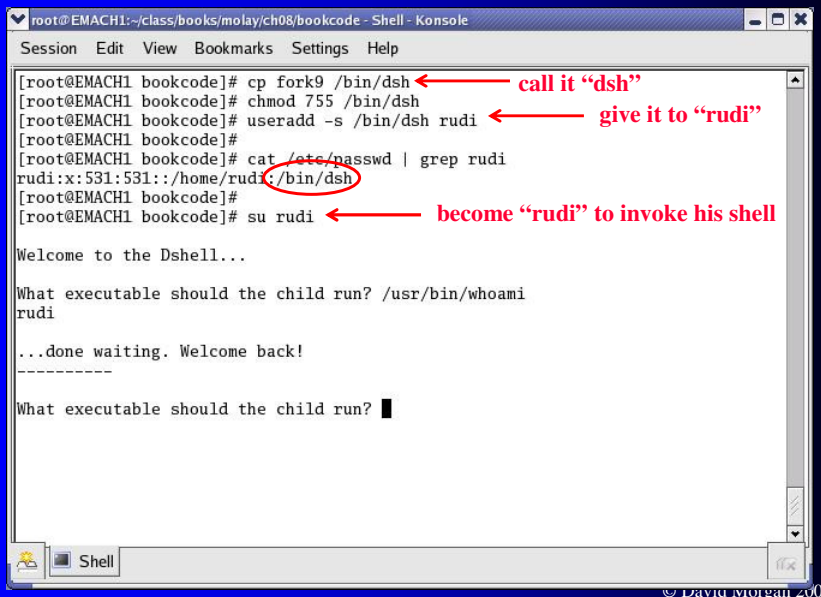
2005-18

Make it repetitive

stick it in a while ( ) loop



Now install it as a default shell



Limitations of this 15-line shell

- full-path commands only
- single-token commands only (no args!)
- no frilly bash stuff
  - no filename globbing (wildcarding)
  - no history (commandline recall)
  - no redirection/pipes
  - no variables
  - no builtins (cd, exit, etc)

You gotta problem with it??

- terms are open source-- you can outcode me
- have it run commands that *do* give frilly stuff
  - /bin/ksh
  - /bin/tcsh
  - /bin/zsh
  - /bin/bash