Structure of a command shell
--writing our own, homemade but full-fledged

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

the single essential, for a "command" shell

Review: process spawns process
the fork/exec formula (fork5 spawns ls)

```
[root@REMACHL bookcode]# cat fork5.c
#include <unistd.h>
#include <stdio.h>
main() { int result;
    printf("aParent does stuff and then...\n\n"");
    result = fork();
    if (result == 0 ) {
        printf("Child could run some executable...\n\n");
        exec1="/bin/ls","-l","/etc/httpd/conf",NULL); }
    else
        printf("...parent do something completely different...\n\n"); }
[root@REMACHL bookcode]# ./fork5
Parent does stuff and then...
Child could run some executable...

ls -l /etc/httpd/conf
(the real thing)
```
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

© David Morgan 2005
Make it orderly

*insert wait (parent) & exit (child)*

```
main() { int result;
    printf( "Parent does stuff and then...
.tiles\n" );
    if ( result == 0 )
        { printf("Child could run some time-consuming executable...
.tiles\n");
         execl("timecstar", NULL); exit(0); }
    printf("While parent might plunge ahead in parallel...
.tiles\n...and/or wait for child termination to continue.
.tiles\n");
}
```

stall till child is done

prog that wastes 10 sec

Make it interactive

*insert scanf() for program name*

```
main() { int result;
    char prog[100];
    printf( "Parent does stuff and then...
.tiles\n" );
    result = fork();
    if ( result == 0 )
        { printf("Child could run a user-specified executable...
.tiles\n");
         printf("What executable should the child run? ");
         scanf( "%s", &prog);
         execl(prog, prog, NULL);
         exit(0); }
    else
        wait(NULL);
    printf("...parent continues now, only when child is done.
.tiles\n");
}
```

10 sec goes by here

© David Morgan 2005
Make it repetitive

```c
int result;
char prog[100];

printf("welcome to the Dobell...\n\n");
while (1) {
    print("What executable should the child run? ");
    scanf("%s", &prog);
    result = fork();
    if (result == 0) {
        execl(prog, prog, NULL);
        exit(0);
    } else {
        wait(NULL);
        printf("...done waiting. Welcome back!\n\n\n\n");
    }
}
```

Now install it as a default shell

```bash
$ cp fork9 /bin/dsh
$ chmod 755 /bin/dsh
$ useradd -s /bin/dsh rudi
$ su rudi

Welcome to the Dobell...

What executable should the child run? /usr/bin/whoami
rudi

...done waiting. Welcome back!

What executable should the child run? 
```
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