

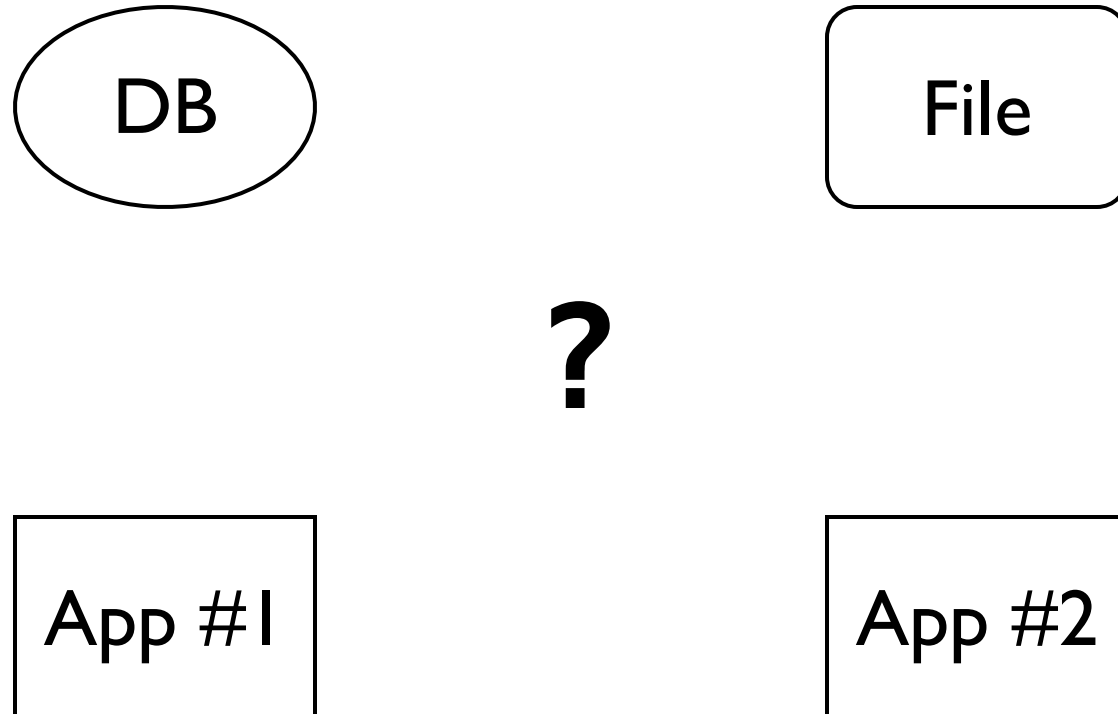
Day 9: System Interaction

Suggested reading: *Learning Perl* (4th Ed.)

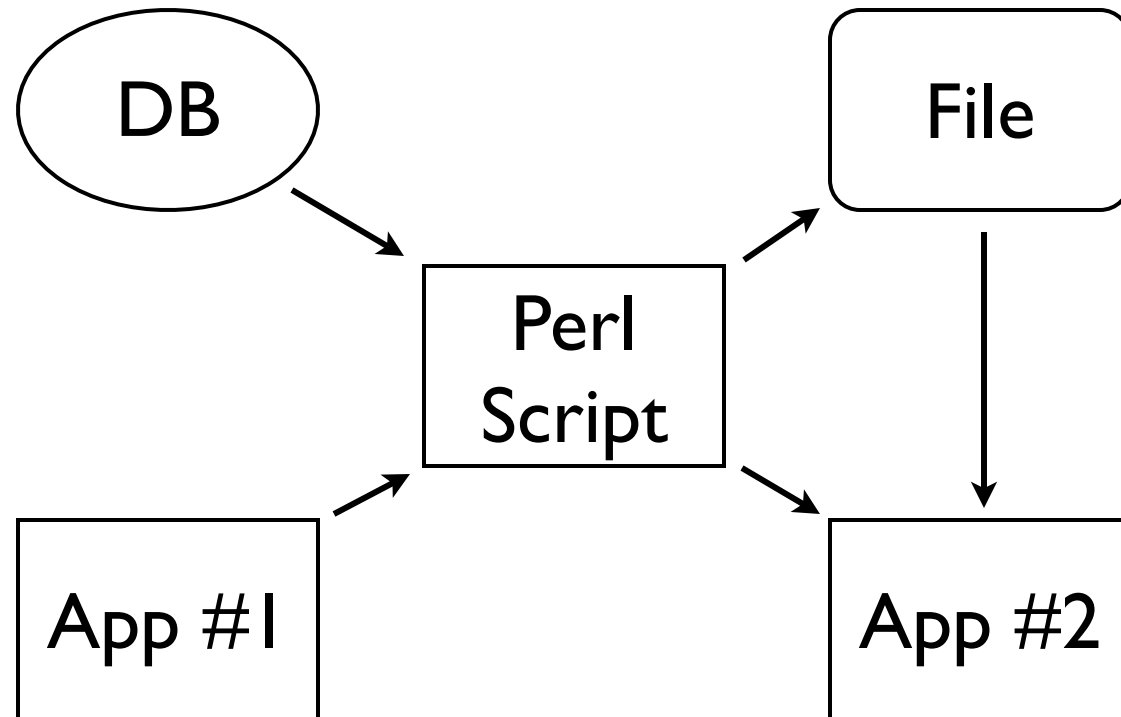
Chapter 14: Process Management

Homework Review

Problem



(One) Solution



Introducing ... The Shell

The Shell

- Is another scripting environment — e.g., `/bin/sh`
- Runs “commands” — most are separate programs
- Has variables, control structures, functions, ...
- Gives control over input and output
- Allows pipelines of commands (workflow)
- Interactive (command-line) or scripted

Common Shell Commands

- Built-in: **cd, echo**
- Manual pages: **man**
- List files: **ls, find**
- Change file privileges: **chown, chmod**
- File manipulation: **cp, mv, rm, ln, unlink**
- Directory manipulation: **mkdir, rmdir**
- View files: **cat, more, less, wc**
- Filter and/or manipulate files: **grep, sed, awk**
- Text editors: **pico, vi, emacs**
- Scripting: **perl, python, ruby**

Shells and Commands

command line (shell)

```
% foo -x  
%
```

```
PWD: ~/scripts  
PATH: /usr/bin:/usr/local/bin:...
```

foo (Perl)

```
chdir 'data';  
system('bar');
```

```
PWD: ~/scripts/data  
PATH: /usr/bin:/usr/local/bin:...
```

bar (C)

```
printf("hi\n");  
exit(2);
```

```
PWD: ~/scripts/data  
PATH: /usr/bin:/usr/local/bin:...
```


The System \Rightarrow Your Script

Command-Line Arguments

\$0 command (script) name
@ARGV command-line arguments

```
% my_script data-42.txt 1200
```

```
#!/usr/bin/perl
```

```
use strict; use warnings;
```

```
my ($filename, $threshold) = @ARGV;
```

```
foreach my $argument (@ARGV) { ... }
```

```
print "$0: arguments parsed ok!\n";
```

Arguably Better Arguments

```
% my_script --lo --with foo -aXb file1
```

```
use Getopt::Long;
```

```
my $lo = 0;
```

```
my $with = '';
```

```
GetOptions( 'lo'          => \$lo,  
            'with=s'     => \$with,  
            # etc.);
```

```
my $file = $ARGV[0];
```

Environment

- **%ENV** : environment variables
- Readable and writable

```
my @path = split(/:/, $ENV{'PATH'});
```

```
my @new = grep(! m{/sbin}, @path);  
$ENV{PATH} = join(':', @new);
```

```
delete $ENV{'BLAH'};
```

Your Script \Rightarrow The System

Running a Command

`system()`

- May create a shell to parse command
- Waits for command to finish
- Command inherits environment, etc.

```
system("gzip $output_file");
```

Sneaky system() Subtleties

Create a shell?

```
system('ls'); # no
system('ls >> my_file'); # yes
system('ls', '>>', 'my_file'); # no
```

Children do not affect parent

```
system('pwd'); # /home/cat/foo
system('cd bar');
system('pwd'); # ???
```

Sneaky system() Subtleties II

Watch out for quoting issues:

If a Perl script contains...

```
system("echo 'Don't say \"no\"!'");
```

... the actual string (command) is ...

```
echo 'Don't say "no"!'
```

... and the **echo** command prints ...

```
Don't say "no"!
```


Return Values

- `system()` returns exit code $\times 256$ ($\ll 8$)
- Exit code of `0` is good in shell,
- But `0` is *false* in Perl, so...

```
system(...) and die('fail');      # confusing  
!system(...) or die('fail');      # may miss !
```

- ... instead, strive for clarity:

```
system(...) == 0 or die('fail');  
if (system(...) != 0) { die('fail'); }
```

Errors

- `system()` return value \equiv `$?` \equiv exit code \ll 8
- `$? == -1` means it failed to execute
- `#!` is the system error message

```
if ($? == -1) {  
    print "failed to execute: $!\n";  
} elsif ($? & 127) {  
    print "died with signal";  
} else {  
    print "exited " . ($? >> 8) . "\n";  
}
```

Getting Output

```
my $sys_date = `date`;
```

- Like **system()** but returns standard output
- **\$?** and **\$!** are set in the same way
- Need standard error, too?

```
my $sys_date = `date 2>&1`;
```

- In list context, returns list of lines in output
- Backticks interpolate!

```
my @files = `find $directory`;
```

Miscellaneous

Quit script with given exit code

```
exit 1;
```

Print message to standard error and exit (non-zero)

```
die 'message';
```

Print message to standard error

```
print STDERR 'message';  
warn 'message';
```

When To Use

- Glue between existing commands
- More powerful shell
- Workflow management
- ***Not*** to replace Perl functions!
 - E.g., do not actually use ``date``
- Your ideas?

Wrap Up

Other Scripting Languages

- In command-line scripts, expect
 - Command-line arguments
 - Environment
 - Standard in, out, and error
 - System calls
 - Exit with status
 - Windows will be different...
- Embedded scripting (PHP, Lua, JavaScript, ...)
 - Expect more restrictions

Homework

- Find IP addresses for image downloads
- Use external commands for:
 - Fetching an HTTP document (web page)
 - Looking up IP addresses
- Middle section will involve regular expressions