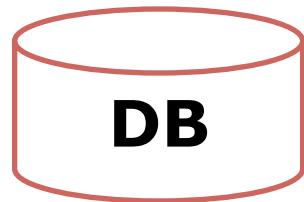


# **Day 11: System Interaction**

suggested reading:  
Learning Perl (4th Ed.),  
Chapter 14: Process Management

# **Homework Review**

# Problem

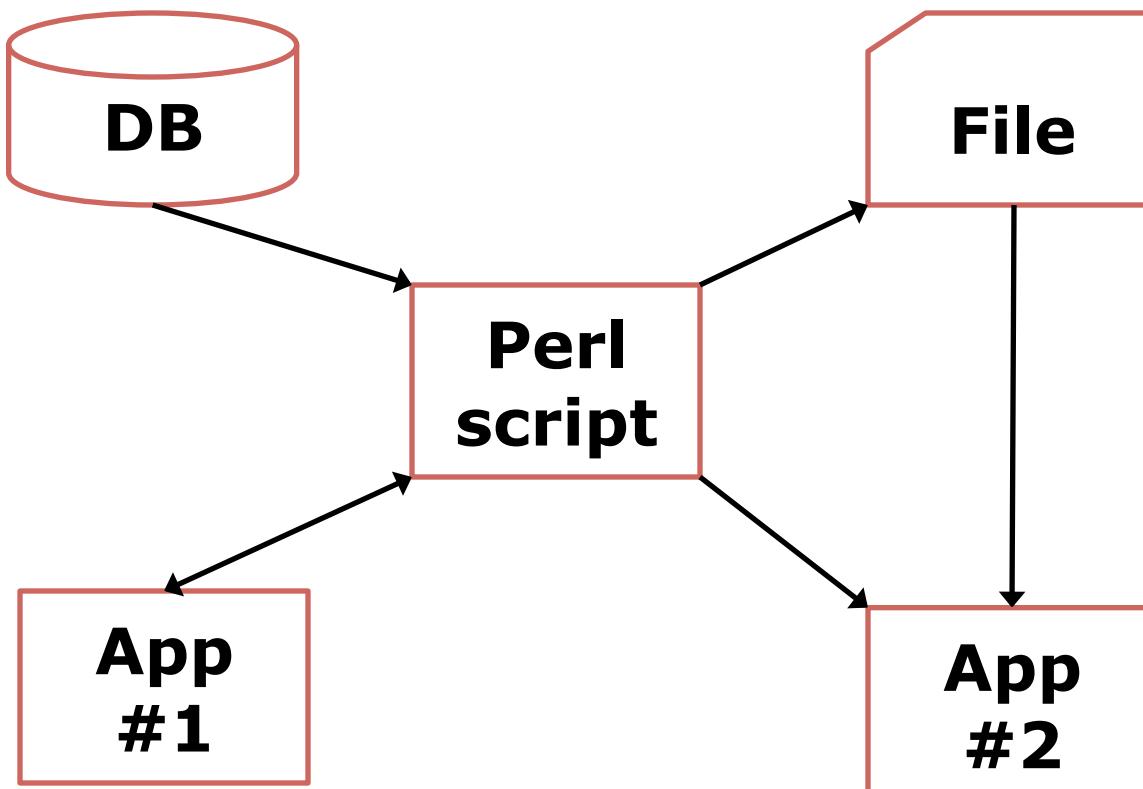


?

**App  
#1**

**App  
#2**

# Problem



## Arguments

- **\$0** : command name
- **@ARGV** : command-line arguments

```
#!/usr/bin/perl
use strict;
use warnings;

foreach my $arg (@ARGV) {
    # process argument
}
print "$0: arguments parsed\n";
```

# Arguably Better Arguments

```
% do-stuff -aXb --lo --with foo 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'};
```

# Running a Command

## **system()**

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

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

(does a fork & exec, FWIW)

# Sneaky system() Subtleties

- To shell or not to shell?

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

- Children do not affect parent

```
system('pwd');
system('cd subdirectory');
system('pwd');
```

## Sneaky system() Subtleties II

- Watch out for quoting

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

# Return Values

- `system()` returns exit code << 8
- Exit code of 0 is good in shell,
- But 0 is false in Perl...

```
system(...) && die('FAIL!'); # confusing  
or  
!system(...) || die('FAIL!'); # may miss !
```

- So, strive for clarity:

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

# Errors

- return value = \$? = exit code << 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 stdout
- **\$?** and **\$!** are set in the same way
- Need stderr, too?

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

- List context: one element per line
- Backticks interpolate!

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

# Miscellaneous

- Quit script with given exit code

```
exit 1;
```

- Exit nonzero with message

```
die('message');
```

- Print to standard error

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

## When To Use

- To glue existing systems together
- As a more powerful shell
- NOT to replace Perl functions!
  - I.e., `date` is probably a bad idea
- Examples from my work:
  - Automate a build
  - Run configuration scripts during install
  - Check command output for failure
- Your ideas?

# Security Considerations

- BE CAREFUL!
- Especially with non-literal commands
- **use taint**; may help (but is hard)
- what could go wrong with this?

```
system("mv $foo $bar");
```

# 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, JS, ...)
  - Expect more restrictions