

Unix for Perl Programmers

Pipes and Processes

Aaron Crane
London Perl Workshop 2009

Perl makes easy things easy

```
my $processes = `ps`;
```

```
open my $fh, 'ps |';
```

Perl makes hard things possible

fork
pipe
exec

Running a program

```
system 'update_web_server';
```

Program success and failure

```
system('update_web_server') == 0  
    or die "Failed to update web server\n";
```

```
system('update_web_server');  
die "Failed to update web server\n"  
    if $? != 0;
```

Program arguments

```
system('update_web_server --all') == 0  
    or die "Updating web servers failed\n";
```

```
# DANGER if $host is untrusted  
system("update_web_server --host=$host") == 0  
    or die "Updating $host failed\n";  
# what if $host eq "; rm -rf /" ?
```

Safe program arguments

```
my $host = $cgi->param('hostname');  
system 'update_web_server', "--host=$host";
```

Running a program in the background

```
system 'update_web_servers --all &;'
```

Forking a process

- Clone existing process
- Child is barely distinguishable
- Continue to run the same code

Fork in Perl

```
my $pid = fork;
die "Can't fork: $!\n" if !defined $pid;
if ($pid == 0) {
    # This is the child process
}
else {
    # This is the parent process
}
```

Fork in Perl

```
sub fork_child {  
    my ($child_process_code) = @_;  
    my $pid = fork;  
    die "Can't fork: $!\n" if !defined $pid;  
    return $pid if $pid != 0;  
  
    # Now in child process  
    $child_process_code->();  
    exit;  
}
```

Executing a process in the background

```
fork_child(sub {  
    exec 'update_web_server', "--host=$host"  
    or die "Can't run update_web_server: $!\n"  
});
```

Background process exit status

```
my $pid = fork_child(sub {  
    exec 'update_web_server', "--host=$host"  
    or die;  
});  
  
do_complex_calculations();  
  
waitpid $pid, 0;  
die "Failed to update web server $host\n"  
    if $? != 0;
```

What if a child process isn't reaped?



Fire-and-forget processes

```
my $child_pid = fork_child(sub {  
    # Child process  
    fork_child(sub {  
        # Grandchild process  
        exec $program, @arguments or die;  
    });  
});  
  
waitpid $child_pid, 0;  
die if $? != 0;
```

Running programs in parallel

```
my %children;
for my $h (hosts_to_update()) {
    my $pid = fork_child(sub {
        exec 'update_web_server', "--host=$h"
        or die;
    });
    $children{$pid} = 1;
}
```

```
delete $children{ waitpid -1, 0 }
while keys %children;
```

Run a program in a different directory

```
my $pid = fork_child(sub {
    chdir $dir
        or die "Can't chdir to $dir: $!\n";
    exec 'tar', '-cf', $tar_file, '.'
        or die "Can't execute tar: $!\n";
});

waitpid $pid, 0;
```

Capturing program output



Capturing program output

```
pipe my ($readable, $writable)
    or die "Can't create pipe: $!\n";

my $pid = fork_child(sub {
    dup2(fileno $writable, 1);
    exec 'ps' or die "Can't exec ps: $!\n";
});

close $writable;

print while <$readable>;
waitpid $pid, 0;
```

Sending input to a program

```
pipe my ($readable, $writable) or die;
```

```
my $pid = fork_child(sub {  
    dup2(fileno $readable, 0);  
    close $writable;  
    exec 'lpr' or die "Can't exec lpr: $!\n";  
});
```

```
print {$writable} $_  
    while defined($_ = get_printable_line());  
close $writable;  
waitpid $pid, 0;
```

Sending *and* capturing

```
pipe my ($send_readable, $send_writable);  
pipe my ($capt_readable, $capt_writable);  
  
my $pid = fork_child(\&bidi_pipe);  
  
close $send_readable;  
close $capt_writable;  
print {$send_writable} get_data();  
close $send_writable;  
print while <$capt_readable>;  
  
waitpid $pid, 0;
```

Sending *and* capturing

```
sub bidi_pipe {  
    dup2(fileno $send_readable, 0);  
    dup2(fileno $capt_writable, 1);  
    exec 'sort'  
        or die "Can't execute sort: $!\n";  
}
```

Sending *and* capturing: two processes

```
pipe my ($send_readable, $send_writable);
pipe my ($capt_readable, $capt_writable);

my $writer_pid = fork_child(sub {
    close $capt_readable;
    close $capt_writable;
    close $send_readable;

    print {$send_writable} get_data();
    close $send_writable;
});
```

Sending *and* capturing: two processes

```
my $filter_pid = fork_child(sub {  
    dup2(fileno $send_readable, 0);  
    dup2(fileno $capt_writable, 1);  
    exec 'tr', 'a-z', 'A-Z'  
        or die "Can't execute tr: $!\n";  
});
```

Sending *and* capturing: two processes

```
close $send_readable;  
close $send_writable;  
close $capt_writable;  
  
print while <$capt_readable>;  
  
waitpid -1, 0 for 1 .. 2;
```

Sending *and* capturing: temporary file

```
my $temp_fh = tempfile();
print {$temp_fh} get_data();
seek $temp_fh, 0, 0;
pipe my ($readable, $writable);
my $pid = fork_child(sub {
    dup2(fileno $temp_fh, 0);
    dup2(fileno $writable, 1);
    exec 'tr', 'a-z', 'A-Z';
});
close $writable;
print while <$readable>;
waitpid $pid;
```

<http://aaroncrane.co.uk/talks/>