IT 240
Shell Scripting for Administrators

Chapter 16
Process Management

Stan J. Senesy
IT Program/CCS
New Jersey Institute of Technology

Creating Child Processes

- Use the system function to launch a child process:

  system “date”;

- A child that is running the date command will be created
- Output will be directed to STDOUT
- If you need it to run as a background process, put the & after the command

Avoiding the Shell

- System may be invoked with more than a single argument:

```
my $tarfile = “something*wicked.tar”
my @dirs = qw(fred|flinstone <barney&rubble> betty);
system “tar”, “cvf”, $tarfile, @dirs;
```

- The net effect is to run the tar command, passing cvf, $tarfile and @dirs as arguments

More Process Creation

- With system, we relied on the OS to create the new process
- We can create the process directly in perl using the exec function:

```
exec “bedrock”, “-o”, “args1”, @ARGV;
```

Environmental Variables

- Any child process inherits information passed from the shell regarding settings such as the PATH, etc
- We can set these in perl with the ENV operator:

```
$ENV{‘PATH’} = “/home/senesy:
$ENV{‘PATH’};
```

Capturing Output

- We can grab the output of a command by using the back quotes ` ` and prevent it from going to STDOUT

```
my $now = `date`;
```

- This may also be performed when dealing with a list, producing one line of output per list element:

```
my @who_lines = `who`;
```

Processes as File Handles

- Our previous examples of creating processes have been blocking; execution of the parent halts until the child is complete

- We can also create a child as a parallel process that does not block the parent:

```
open date, “date|” or die “cannot pipe from date: $!”;
```

Signals

- Signals are messages that are sent to processes
- The kill command may be used to send a number of different signals to a process:

```
kill 0, $pid
```

- Signals may also be caught