Chapter 2

UNIX Processes

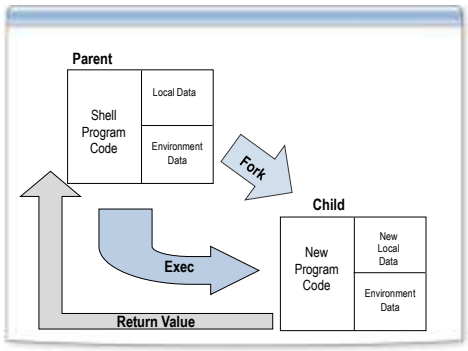
Overview

- What is a Process?
- The `ps` command
- Background processes
- Redirecting the Standard Error

Lesson: What is a Process

- A process is an instance of running a program
- Almost every time you issue a command, Unix starts a new process
  - Some commands are built into the shell (like `cd`). They are executed by the shell, without starting a new process.
- Every process has a Process Identification Number (pid)
- Every process (except init) has a parent process

The Subprocess Environment



Determine command type

- **Syntax:**

```
$ type [commands]
```

- **Example:**

```
$ type ls
ls is /bin/ls

$ type cd
cd is a shell built-in
```

Lesson: ps – report process status

- **Syntax:**

```
ps [options]
```

- **Example:**

```
$ ps
  PID TTY      TIME CMD
 1342 tty1     0:00 sh
 1396 tty1     0:00 ps
$ ps -ef
  UID   PID  PPID  C   STIME   TTY   TIME COMMAND
 root     0     0  0   Jan  1  ?     0:20 swapper
 root     1     0  0   Jun 23  ?     0:00 init
 root     2     0  0   Jun 23  ?     0:16 vhand
stu03  1342  1341  3   18:30   tty1  0:00 -sh
stu03  1396  1342 22   18:49   tty1  0:00 ps -ef
```

Lesson: Background processes

- Shell and background processes
- Running a job in background
- `wait` – await process completion
- `kill` – terminate or signal processes

Shell and background processes

- Syntax:

```
command line > cmd.out 2> err.out &
```

- Example:

```
$ grep user * > grep.out &
[1] 194

$ ps
  PID  TTY    TIME  COMMAND
  164  ttyp2  0:00  sh
  194  ttyp2  0:00  grep
  195  ttyp2  0:00  ps
```

`kill` – terminate or signal processes

- Syntax

```
kill [-s signal_name] PID ...
```

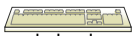
- Example

```
$ cat /usr/share/man/cat1/*> bigfile1 &
[1] 995
$ cat /usr/share/man/cat2/*> bigfile2 &
[2] 996
$ kill 995
[1] - Terminated cat /usr/share/man/cat1/*> bigfile1 &
$ kill -s INT %2
[2] + Interrupt  cat /usr/share/man/cat2/*> bigfile2 &
$ kill -s KILL 0
```

Lesson: Redirecting the Standard Error

- `stdin`, `stdout` and `stderr`
- Error Redirection

`stdin`, `stdout` and `stderr`

File	Device	File descriptor
`stdin`	 keyboard	0
`stdout`	 terminal	1
`stderr`		2

Error Redirection – 2> and 2>>

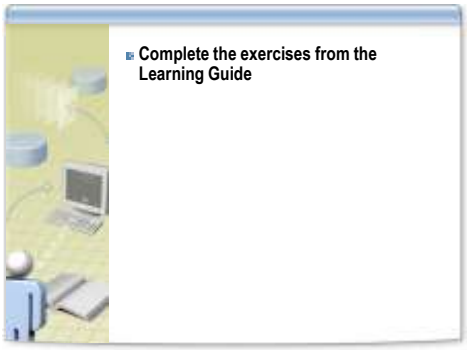
- Any program that produces error messages to `stderr` can have those messages redirected to another file.

Examples:

```
$ cp 2> cp.err
$ cp 2>> cp.err

$ cat cp.err
Usage:  cp [-f|-i] [-p] source target
        cp [-f|-i] [-p] source ... target
        cp [-f|-i] [-p] -R|-r source ... target
Usage:  cp [-f|-i] [-p] source target
        cp [-f|-i] [-p] source ... target
        cp [-f|-i] [-p] -R|-r source ... target
```

Review Exercises



Topics for Review