Chapter 13

Shell IPC

Overview

- Co-processes
- Signals
- Named Pipes
- The `wait` Command

Lesson: Co-processes

- What is Co-process
- Exchanging information

What is Co-process

- **Co-process is a way to run commands in parallel (asynchronously) and keep a way to exchange information with them.**
  - Available in Korn shell family

command `|&`	Execute command as co-process
n<&p	Redirect co-process input to file descriptor n (by default stdin)
n>&p	Redirect co-process output to file descriptor n (by default stdout)
`print -p`	Write to co-process through pipe
`read -p`	Read from co-process through pipe

Exchanging information

- **Korn shell family has alternate to `echo` command - `print`**
- **Both `print` and `read` commands can interact with co-process.**
- **Example:**

```
bc |&
print -p "3*4"
read -p answer
echo $answer
```

Lesson: Signals

- **Sending signals**
- **Common signals**
- **The `trap` Command**

`kill` – send signal to 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
```

Common Signals

```
$ kill -l
```

0	EXIT	(exit)
1	HUP	
2	INT	(ctrl-c)
3	QUIT	(ctrl-\)
6	ABRT	
9	KILL	
14	ALRM	
15	TERM	

The `trap` Command

- `trap` **catch a signal sent to script.**
- `trap` **can start actions on response to signal**
- **Syntax:**

```
trap 'action' signal ...
trap - signal ...
```

- **Example:**

```
trap 'echo interrupt received; exit' INT
trap 'echo "Error found" ; continue' 2
trap 'rm /tmp/myfile; clear; echo script ending' EXIT

$ trap 'echo "*** ERROR $ERRNO FOUND ***" ' ERR
$ grep NOVALUE /etc/passwd
*** ERROR 3 FOUND ***
$ echo $?
1
```

Lesson: Named Pipes

- Named pipe is special file that allows two processes communicate with each other.

```
# mkfifo named_pipe

[me@linux]~# mknod named_pipe p
```

- After creating, processes can read from and write to that file.

Lesson: `wait` – await process completion

- Syntax:

```
wait [job_id ...]
```

- Example:

```
$ sleep 1000 &
$ pid=$!

$ wait $pid ; echo Job exited with status $?
```

Review Exercises

- Complete the exercises from the Learning Guide

Topics for Review