Process and Job Control in Linux

Ben C. Roose

Electrical Engineering & Computer Science
Systems Administrator, Wichita State University

- Maintain Debian, Ubuntu, and Linux Terminal Server Project Labs

Kansas Linux Fest 2017

Sat & Sun, May 13 & 14, 2017

Wichita State University, Wichita, KS

kansaslinuxfest.org

What are we talking about?

We will look at:

- Processes (PID) and the ps command
- Parent and child processes (PPID)
- Terminals, pseudoterminals, and shells
- Process sessions (SID) and process groups (PGID)
- Orphan and zombie processes
- Signals, terminal keybindings, and the kill command
- Process job control

Linux Processes

What is a *process* in Linux?

- According to The Linux Information Project (www.linfo.org):
  - "A *process* is an executing (i.e. running) instance of a *program*."
  - "A *program* is a passive entity until it is launched, and a *process* can be thought of as a *program* in action."

- A process is identified in a system by a unique *PID* number (a non-negative integer).
- The system kernel maintains a list of active process PIDs

Parents and Children

- Process can be executed by *spawning/forking* from another process.
- *Spawning* creates *child* process(es) from a *parent*.
- Every process has a *parent* and hence has a *PPID* number.

- Processes can terminate by:
  - Exiting without error
  - Exiting upon error
  - System (kernel) forcing the process to terminate

Terminals and Pseudoterminals

- According to The Linux Information Project, the *terminal* (also known as a *virtual console)* is:
  - “a display mode that contains only text and no images and that occupies the entire screen of the display device (usually a CRT or LCD).”
- The *terminal* is an input/output character device often listed as: `/dev/tty#`
- A *Pseudoterminal* is a window within the GUI which “emulates” a terminal character device by using process file streams.
- The *pseudoterminal* is often listed as device: `/dev/pts` or `/dev/ptmx`

So how do we interact with programs using a terminal or pseudoterminal?

The Shell

- The *shell* is a program and runs as a *process*:
  - Connected to a controlling *terminal* (or *pseudoterminal*)
  - Acts as interface between the user and the operating system
  - Spawns (executes) other processes via commands
  - Contains additional functionality: file stream redirection, command history, keyboard shortcuts, and process control

What are some common shell programs?

- Borne Again Shell (`bash`) – default shell in Linux
- Borne Shell (`sh`)
- Korn Shell (`ksh`)
- C Shell (`csh`)
- TENEX C Shell (`tcsh`)

Example: The Shell as a Process

```
greyarea@MARTELL:~$ ps -o "pid ppid pgid sid tname command" $$
  PID  PPID  PGID   SID TTY      COMMAND
25376 25274 25376 25376 pts/2    /bin/bash
greyarea@MARTELL:~$
```

- A child process is given a unique process number (PID)
- A child process inherits from its parent:
  - Parent Process number (PPID)
- A child process inherits from its parent (or sets its own):
  - Process Group Identification number (PGID)
  - Session Identification number (SID)

Sessions and Groups

- **Process Session (SID)**
  - A process is a member of a session
  - Specific processes, such as the shell, can become *session leaders* (where SID = PID)
  - A session may have a maximum of one controlling terminal

- **Process Group (PGID)**
  - A group is a collection of one or more processes
  - Shell will execute a command with the first process in the command as *group leader* (where PGID = PID)
  - Within a session, only one process group can be in the *foreground* at any time and have access to the terminal
  - All other process groups in a session are in the *background*

Example: Process Relationships

Figure 34-1 shows the process group and session relationships between the various processes resulting from the execution of the following commands:

```
$ echo $$                          Display the PID of the shell
400
$ find / 2> /dev/null | wc -l &    Creates 2 processes in background group
[1] 659
$ sort < longlist | uniq -c        Creates 2 processes in foreground group
```

At this point, the shell (*bash*), *find*, *wc*, *sort*, and *uniq* are all running.

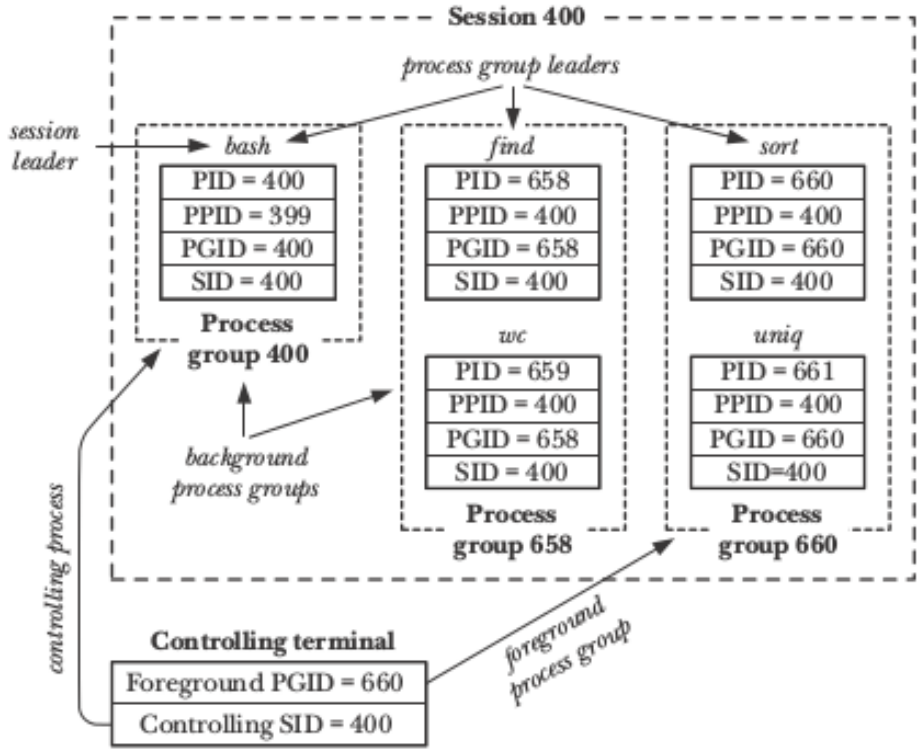


Figure 34-1: Relationships between process groups, sessions, and the controlling terminal

From: Kerrisk, M. (2010). The Linux Programming Interface (pp. 701)

Orphans and Zombies

What is an *orphaned* process?

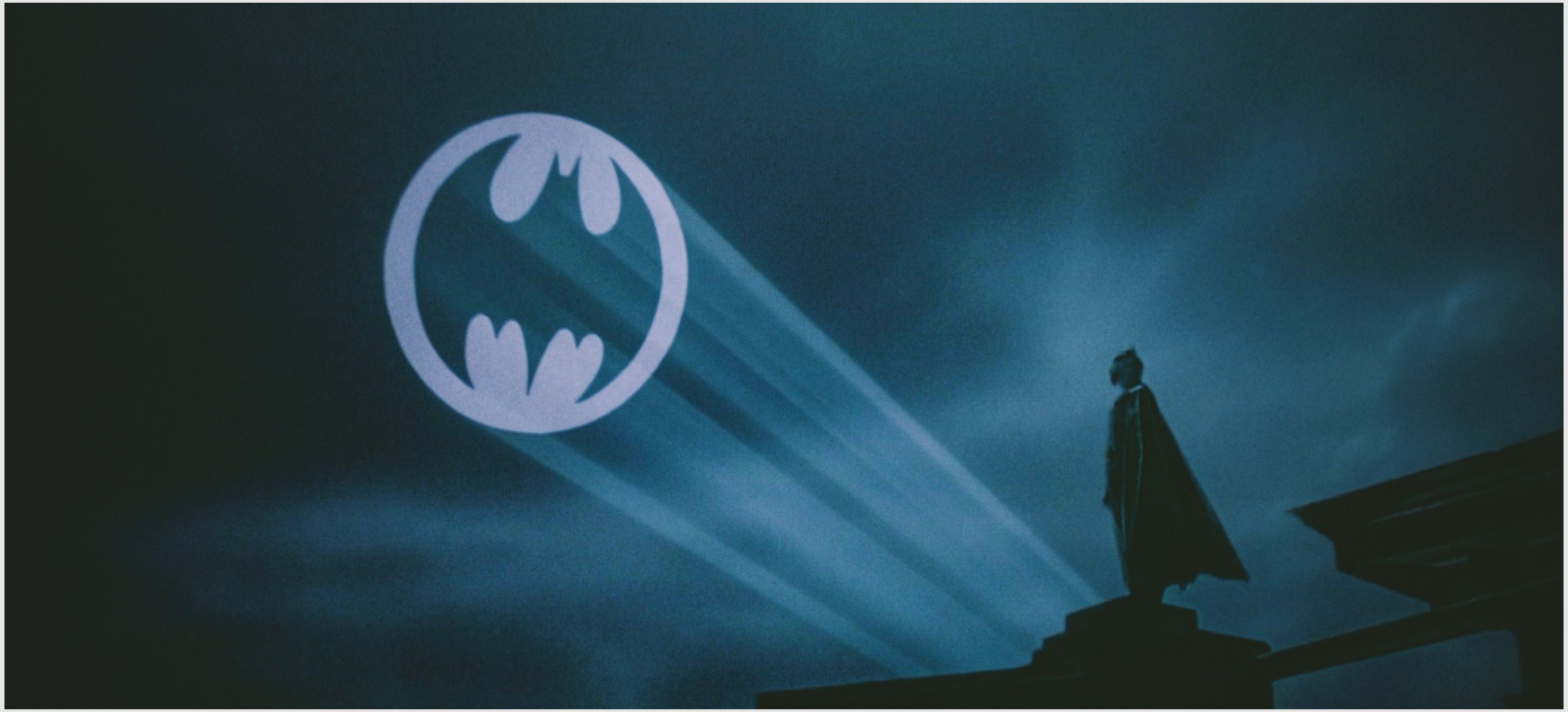
- First process run on a system is `/sbin/init` with PID = 1
- Any *orphaned* process in which its parent process has terminated will be “adopted” by the init process (and gain a PPID of 1).

Orphans and Zombies

What is a *zombie* process?

- Parent process performs a wait() system call to see if child terminated
- Until wait() is called, kernel holds PID of child in active process list
- System shows that PID is active, but cannot be killed by any signal!

Signals



Signals

- A software *signal* is a notification to process or process group
  - Similar to hardware interrupts
  - Denoted by a unique number and a symbolic name

- Software signals are sent by the system (kernel):
  - to a specific process
  - to the *foreground* process group within a session
  - to one or more *background* process groups within a session

Sending and Receiving Signals

- Signals are sent by the system:
  - using specific processes, i.e. `kill` command
  - using terminal keybindings, i.e. CTRL+C

- When a signal is received by a process, it can:
  - default to a standard behaviour: terminate, stop, or continue
  - ignore the signal (not all signals can be ignored)
  - run custom signal handler code, i.e. `ping` command

Common Signals

Sig Name	Sig #	Signal Description	Default Process Behaviour	Keybinding or command
SIGHUP	1	Terminal hangup	Process terminate	CTRL + D (EOL)
SIGINT	2	Terminal interrupt	Process terminate	CTRL + C
SIGQUIT	3	Terminal quit	Process core dump	CTRL + \
SIGTSTP	20	Terminal stop	Process stop	CTRL + Z
SIGTTIN	21	Term read from BG	Process stop	(sent by terminal)
SIGTTOU	22	Term write from BG	Process stop	(sent by terminal)
SIGTERM	15	Terminate process	Process terminate	kill <pid>
SIGKILL	9	Sure kill	Process terminate	kill -KILL <pid>
SIGSTOP	19	Sure stop	Process stop	kill -STOP -<pgid>
SIGCONT	18	Continue if stopped	Process continue	fg / bg / kill -CONT

Putting it together to control processes

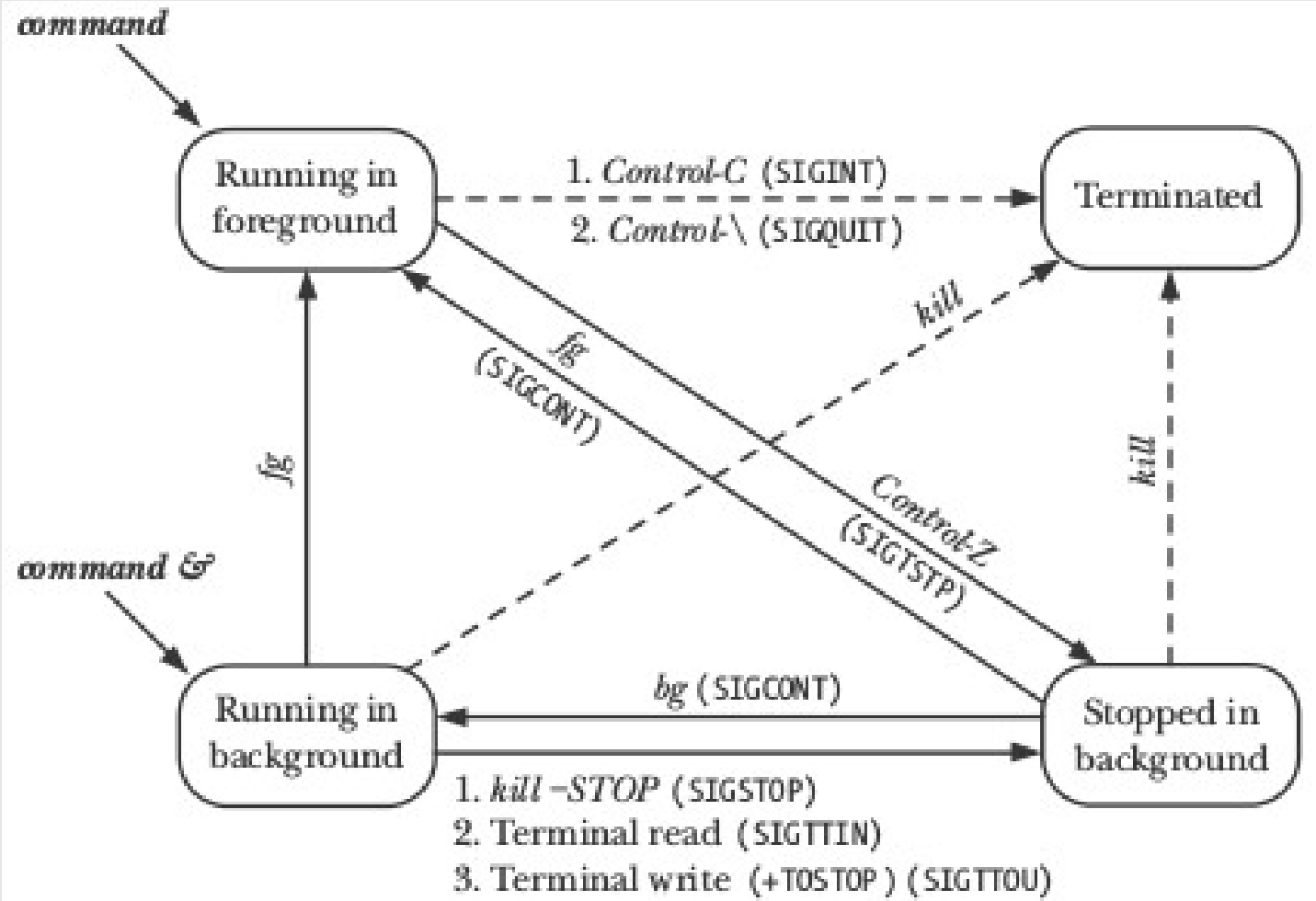


Figure 34-2: Job-control states

From: Kerrisk, M. (2010). The Linux Programming Interface (pp. 717)

Example on the Terminal: Job Control

Live Demonstration
by Ben if time allows!

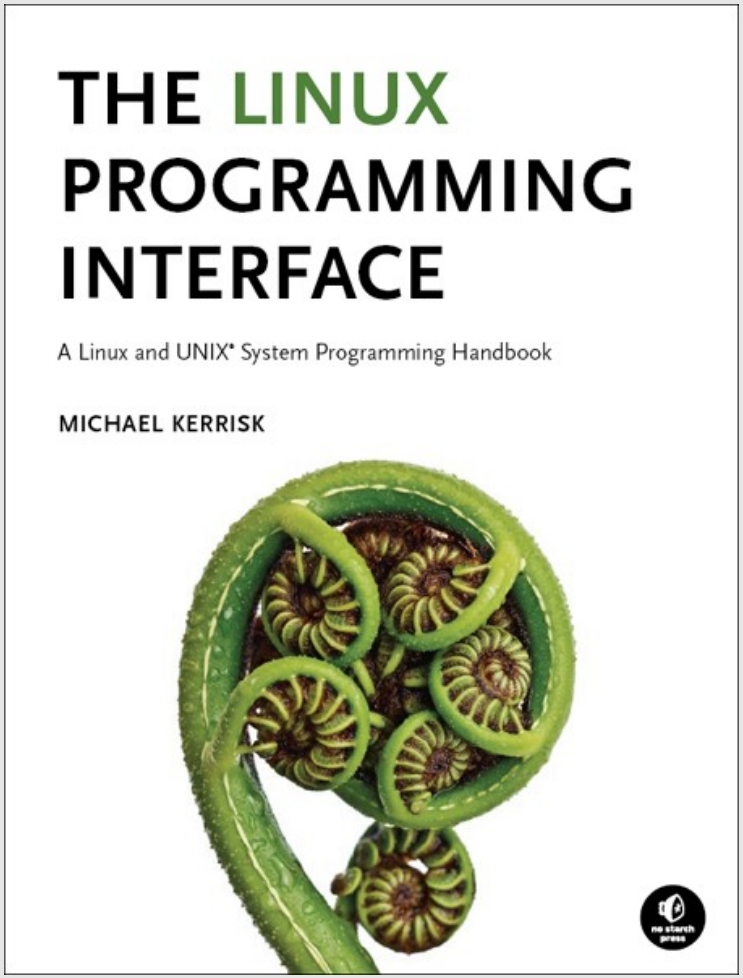
Kerrisk, Michael. (2010). The Linux Programming Interface. No Starch Press

Credit and plug...

A must read for Linux systems administrators, programmers, and enthusiasts looking for an in-depth discussion on the internal workings of UNIX.

For 30% sale discount and free ebook with the printed edition, go to:

http://man7.org/tlpi/



Thank You!

Ben Roose: ben.roose@wichita.edu

Any Questions?

Please attribute
Ben C. Roose, EECS, Wichita State University