System Programming

Topic 6: Process Relationships

Seongjin Lee
Updated by: Muhammad Faris Fathoni

Updated: 2018/11/08
06-process_rel

insight@gnu.ac.kr
http://open.gnu.ac.kr
Systems Research Lab.
Gyeongsang National University

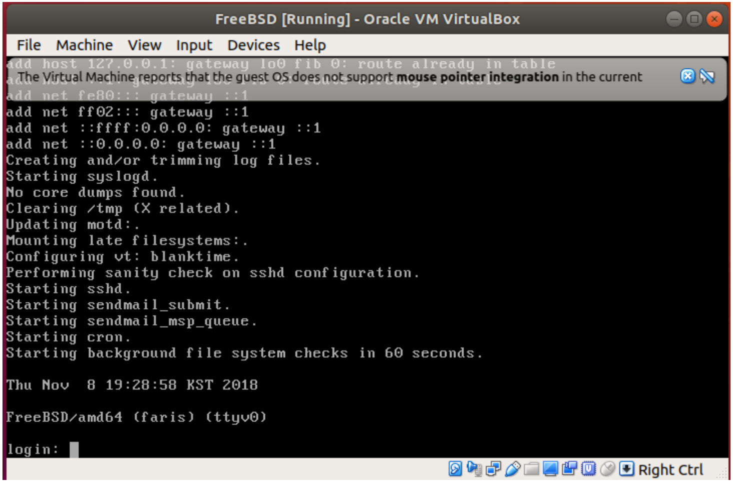
Table of contents

Overview

- Terminal Logins
- Sessions
- Controlling Terminal
- Job Control

Terminal Logins

BSD Terminal


FreeBSD [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

The Virtual Machine reports that the guest OS does not support **mouse pointer integration** in the current

```
add host 127.0.0.1: gateway lo0 fib 0: route already in table
add net fe80::: gateway ::1
add net ff02::: gateway ::1
add net ::ffff:0.0.0.0: gateway ::1
add net ::0.0.0.0: gateway ::1
Creating and/or trimming log files.
Starting syslogd.
No core dumps found.
Clearing /tmp (X related).
Updating motd:.
Mounting late filesystems:.
Configuring vt: blanktime.
Performing sanity check on sshd configuration.
Starting sshd.
Starting sendmail_submit.
Starting sendmail_msp_queue.
Starting cron.
Starting background file system checks in 60 seconds.

Thu Nov  8 19:28:58 KST 2018

FreeBSD/amd64 (faris) (ttyv0)

login:
```

Terminal Logins

In early UNIX systems

- Users logged in using dumb terminals that were connected to the host with hard-wired connections

Terminal Logins

In early UNIX systems

- Users logged in using dumb terminals that were connected to the host with hard-wired connections
- The terminals were either local or remote

Terminal Logins

In early UNIX systems

- Users logged in using dumb terminals that were connected to the host with hard-wired connections
- The terminals were either local or remote
- Users login through a terminal device driver in the kernel

Terminal Logins

In early UNIX systems

- Users logged in using dumb terminals that were connected to the host with hard-wired connections
- The terminals were either local or remote
- Users login through a terminal device driver in the kernel
- A host had a fixed number of terminal devices

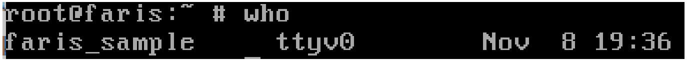
Terminal Logins

In early UNIX systems

- Users logged in using dumb terminals that were connected to the host with hard-wired connections
- The terminals were either local or remote
- Users login through a terminal device driver in the kernel
- A host had a fixed number of terminal devices

  type `who` in the shell

```
James    console  Oct 12 15:36
James    ttys000  Oct 13 22:02
James    ttys001  Oct 13 22:08
```



```
root@faris:~ # who
faris_sample     ttyv0        Nov  8 19:36
```

BSD Terminal Logins

Mac OS X and Linux login procedure follows essentially the same steps as the BSD

The system administrator creates `/etc/ttys`, `ttys(5)`, that has one line per terminal device

Each line specifies the name of the device and other parameters that are passed to the `getty(8)` program

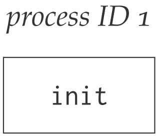
BSD Terminal Logins cont'd

process ID 1

1. the kernel creates process ID 1, the `init` process

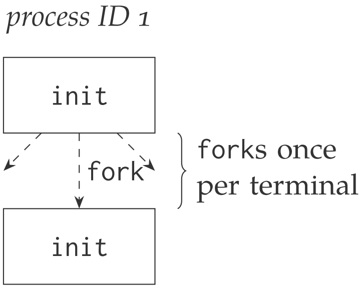
BSD Terminal Logins cont'd

1. the kernel creates process ID 1, the `init` process
2. the `init` process reads the file `/etc/ttys`
3. creates empty environment



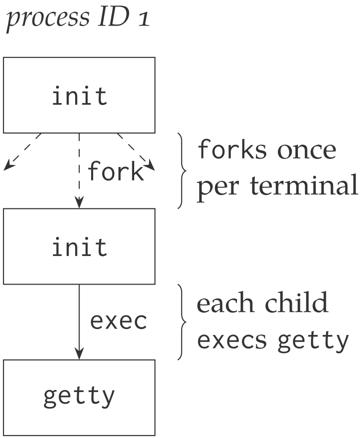
BSD Terminal Logins cont'd

1. the kernel creates process ID 1, the `init` process
2. the `init` process reads the file `/etc/ttys`
3. creates empty environment
4. `forks` for every terminal device



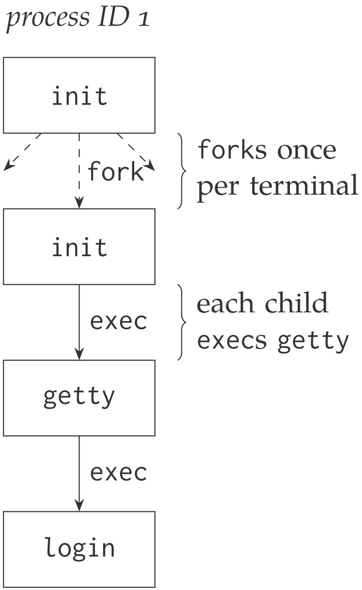
BSD Terminal Logins cont'd

1. the kernel creates process ID 1, the `init` process
2. the `init` process reads the file `/etc/ttys`
3. creates empty environment
4. `fork`s for every terminal device
5. followed by `exec` of the program `getty`
6. opens terminal device (fd 0, 1, 2)
7. reads user name
8. initial environment set



BSD Terminal Logins cont'd

1. the kernel creates process ID 1, the `init` process
2. the `init` process reads the file `/etc/ttys`
3. creates empty environment
4. `fork`s for every terminal device
5. followed by `exec` of the program `getty`
6. opens terminal device (fd 0, 1, 2)
7. reads user name
8. initial environment set
9. followed by `exec` of the program `login`



BSD Terminal Logins

`init(8)` PID 1 PPID 0 EUID 0

reads `/etc/ttys`

BSD Terminal Logins

`init(8)`	PID 1	PPID 0	EUID 0
reads `/etc/ttys`			
`getty(8)`	PID N	PPID 0	EUID 0

BSD Terminal Logins

`init(8)`	PID 1	PPID 0	EUID 0
reads `/etc/ttys`			
`getty(8)`	PID N	PPID 0	EUID 0
opens terminal			
prints “`login:`”			
read username			

BSD Terminal Logins

`init(8)`	PID 1	PPID 0	EUID 0

reads `/etc/ttys`

`getty(8)`	PID N	PPID 0	EUID 0

opens terminal
prints “`login:`”
read username

`login(1)`	PID N	PPID 0	EUID 0

`getpass(3)`, encrypt, compare with `getpwnam(3)`
register login in system database
read/display vaious files
`initgroups(3)/setgid(2)`, initialize environment
`chdir(2)` to home directory
`chown(2)` terminal device
`setuid(2)` to user’s uid, `exec(3)` `shell`

BSD Terminal Logins

`init(8)` PID 1 PPID 0 EUID 0

reads `/etc/ttys`

`getty(8)` PID N PPID 0 EUID 0

opens terminal

prints "`login:`"

read username

`login(1)` PID N PPID 0 EUID 0

`getpass(3)`, encrypt, compare with `getpwnam(3)`

register login in system database

read/display vaious files

`initgroups(3)/setgid(2)`, initialize environment

`chdir(2)` to home directory

`chown(2)` terminal device

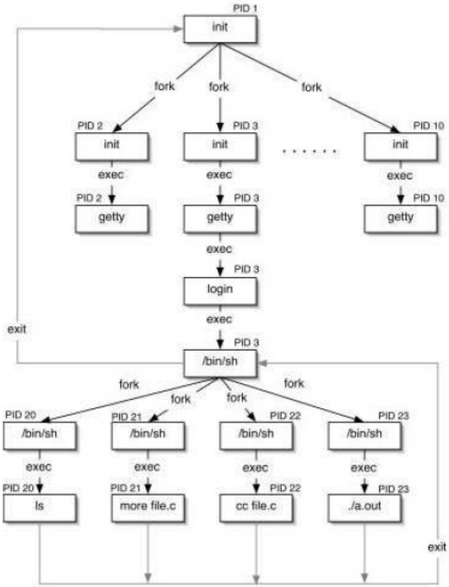
`setuid(2)` to user's uid, `exec(3) shell`

`$SHELL` PID N PPID 0 EUID U

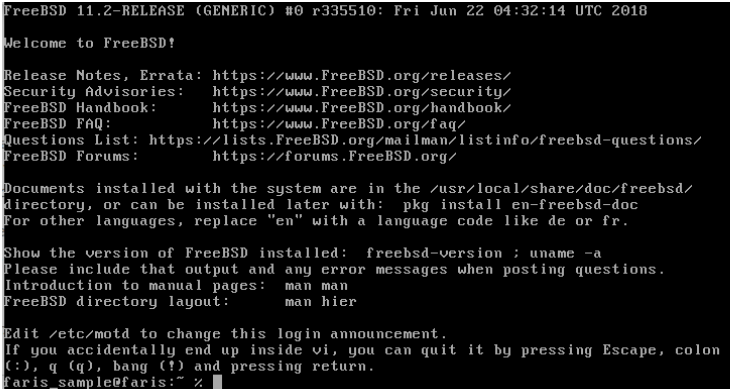
BSD Terminal Logins

`init(8)`	PID 1	PPID 0	EUID 0
reads `/etc/ttys`			
`getty(8)`	PID N	PPID 0	EUID 0
opens terminal			
prints "`login:`"			
read username			
`login(1)`	PID N	PPID 0	EUID 0
`getpass(3)`, encrypt, compare with `getpwnam(3)`			
register login in system database			
read/display vaious files			
`initgroups(3)/setgid(2)`, initialize environment			
`chdir(2)` to home directory			
`chown(2)` terminal device			
`setuid(2)` to user's uid, `exec(3)` `shell`			
`$SHELL`	PID N	PPID 0	EUID U
`ls(1)`	PID M	PPID N	EUID U

BSD Terminal Login Process



BSD Terminal



```
FreeBSD 11.2-RELEASE (GENERIC) #0 r335510: Fri Jun 22 04:32:14 UTC 2018

Welcome to FreeBSD!

Release Notes, Errata: https://www.FreeBSD.org/releases/
Security Advisories:   https://www.FreeBSD.org/security/
FreeBSD Handbook:      https://www.FreeBSD.org/handbook/
FreeBSD FAQ:           https://www.FreeBSD.org/faq/
Questions List: https://lists.FreeBSD.org/mailman/listinfo/freebsd-questions/
FreeBSD Forums:        https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/
directory, or can be installed later with:  pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed:  freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages:  man man
FreeBSD directory layout:      man hier

Edit /etc/motd to change this login announcement.
If you accidentally end up inside vi, you can quit it by pressing Escape, colon
(:), q (q), bang (!) and pressing return.
faris_sample@faris:~ %
```

Process Groups and Sessions

Process Groups

Each process belongs to a process group

- it is a collection of one or more processes
- usually associated with the same job
- the group has a unique process group ID
- the process group exist as long as one process is in the group

```
#include <unistd.h>
pid_t getpgrp(void);
pid_t getpgid(pid_t pid);
// Returns: process group ID of calling process
```

Process group cont'd

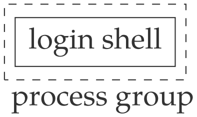
A process joins an existing process group or creates a new process group by calling `setpgid`

```
#include <unistd.h>
int setpgid(pid_t pid, pid_t pgid);
// Returns: 0 if OK, 1 on error
```

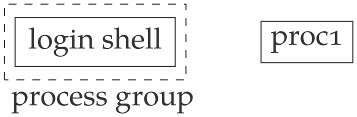
Each process group can have a process group leader

- Process group leader identified by its `pgid` (if `pgid` == `pid`)
- Leader can create a new process group, create processes in the group
- if `pid` == 0, caller process ID is used
- if `pgid` == 0, group ID == `pid`
- A porcess can set the process group ID of itself or its children

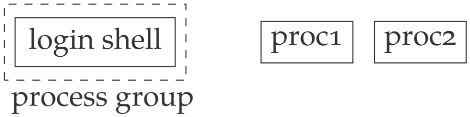
Sessions



Sessions



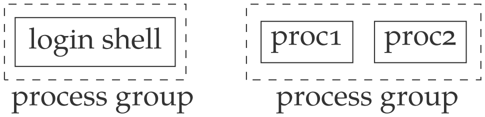
Sessions



Sessions

The processes in a process group are usually placed ther by a shell pipeline

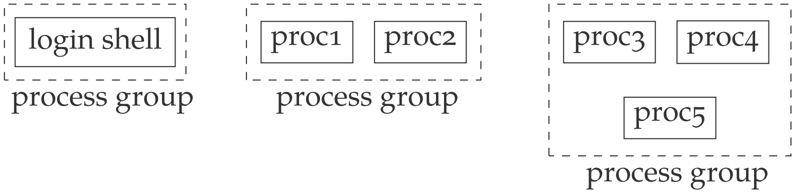
```
proc1 | proc2 &
```



Sessions

The processes in a process group are usually placed ther by a shell pipeline

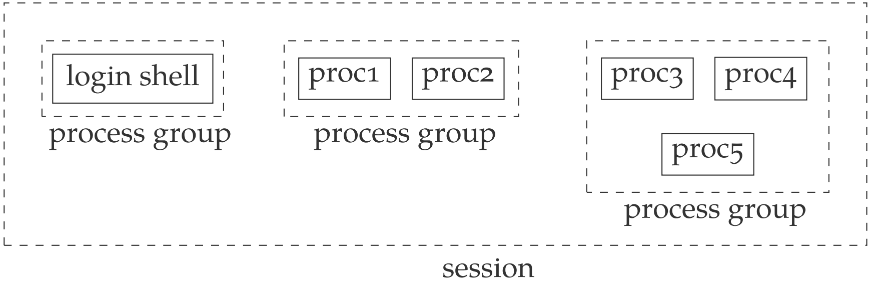
```
proc1 | proc2 &
proc3 | proc4 | proc5
```



Sessions

The processes in a process group are usually placed ther by a shell pipeline

```
proc1 | proc2 &
proc3 | proc4 | proc5
```



Session is a collection of one or more process groups

Sessions cont'd

A process establishes a new session by calling the `setsid` function

```
#include <unistd.h>
pid_t setsid(void);
// Returns: process group ID if OK, 1 on error
```

If the calling process is not a process group leader, this fuction creates a new session. Three things happen

Sessions cont'd

A process establishes a new session by calling the `setsid` function

```
#include <unistd.h>
pid_t setsid(void);
// Returns: process group ID if OK, 1 on error
```

If the calling process is not a process group leader, this fuction creates a new session. Three things happen

1. process becomes the *session leader*, and is only process in this new session

Sessions cont'd

A process establishes a new session by calling the `setsid` function

```
#include <unistd.h>
pid_t setsid(void);
// Returns: process group ID if OK, 1 on error
```

If the calling process is not a process group leader, this fuction creates a new session. Three things happen

1. process becomes the *session leader*, and is only process in this new session
2. the process becomes the process group leader (`pgid` ==`pid`)
   - if the caller is already a process group leader, then returns an error

Sessions cont'd

A process establishes a new session by calling the `setsid` function

```
#include <unistd.h>
pid_t setsid(void);
// Returns: process group ID if OK, 1 on error
```

If the calling process is not a process group leader, this fuction creates a new session. Three things happen

1. process becomes the *session leader*, and is only process in this new session
2. the process becomes the process group leader (`pgid` ==`pid`)
   - if the caller is already a process group leader, then returns an error
3. No contorlling terminal

Sessions cont'd

`getsid` function returns the process group ID of a process's session leader

```
#include <unistd.h>
pid_t getsid(pid_t pid);
// Returns: session leader's process group ID if OK, 1 on error
```

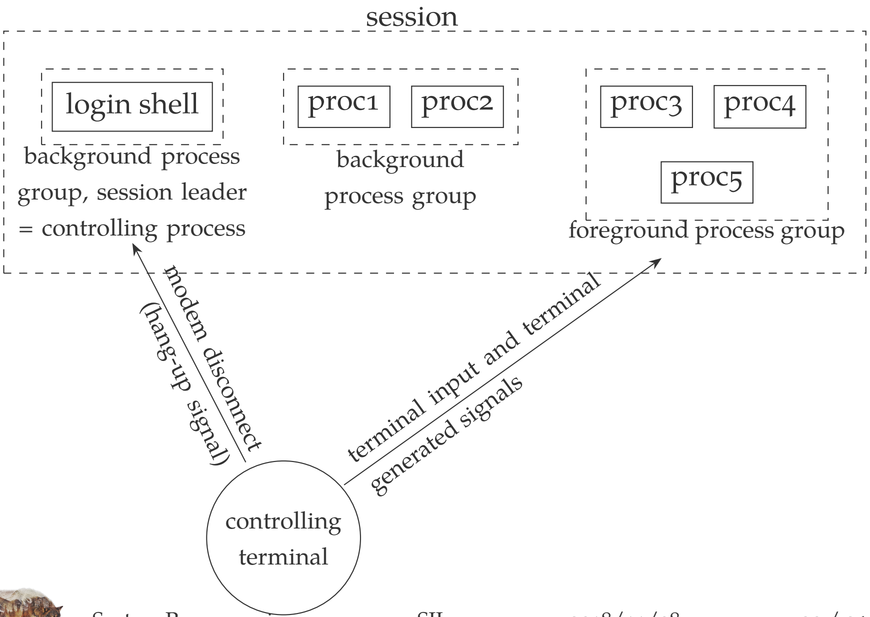
if `pid` == 0, it returns the `pgid` of calling process's session leader

Controlling Terminal and Job Controls

Controlling terminal

- Session can have a single controlling terminal
- session leader that connects to controlling terminal is *controlling process*
- process groups are divided into a single *forground process group* and one or more *background process groups*
- interrupt signals are sent to foreground process group

Controlling Terminal


session
login shell
background process group, session leader = controlling process
proc1
proc2
background process group
proc3
proc4
proc5
foreground process group
modem disconnect (hang-up signal)
terminal input and terminal generated signals
controlling terminal

Job Control

We can start a job in either the foreground or the background

Examples:

foreground `vi main.c` –> starts a job in the foreground

background `make all` –> start a job in the background

```
$ make all > Make.out &
[1] 1475
$ pr *.c | lpr &
[2] 1490
$              just press RETURN
[2] +  Done    pr *.c | lpr &
[1] +  Done    make all > Make.out &
```

Job Control cont'd

The foreground jobs are affected by some special characters, which generate signals

- Interrupt character (typically Ctrl + C) generates SIGINT
- Quit character (typically Ctrl + backslash) generates SIGQUIT
- Suspend character (typically Ctrl + Z) generates SIGTSTP
- Pause character (typically Ctrl + S) generates SIGSTOP
- Resume character (typically Ctrl + Q) generates SIGCONT

Job Control cont'd

```
1 $ cat > temp.foo &
   [1] 1681
2 $
   [1] + Stopped (SIGTTIN) cat > temp.foo &
3 $ fg %1
4 cat > temp.foo
5 hello, world
6 ^D
7 $ cat temp.foo
   hello, world
```

1. start in background, but it'll read from standard input
2. we press RETURN
3. bring job number 1 into the foreground
4. the shell tells us which job is now in the foreground
5. enter one line
6. type the end-of-file character
7. check that the one line was put into the file