

SYSTEM PROGRAMMING

WEEK 7: PROCESS RELATIONSHIPS

Seongjin Lee

Updated: 2016-10-19

06_process_rel

insight@hanyang.ac.kr

<http://esos.hanyang.ac.kr>

Esos Lab. Hanyang University



Table of contents

1. Last Words



introduction

This chapter covers following items

- Terminal Logins
- Sessions
- Controlling Terminal
- Job Control



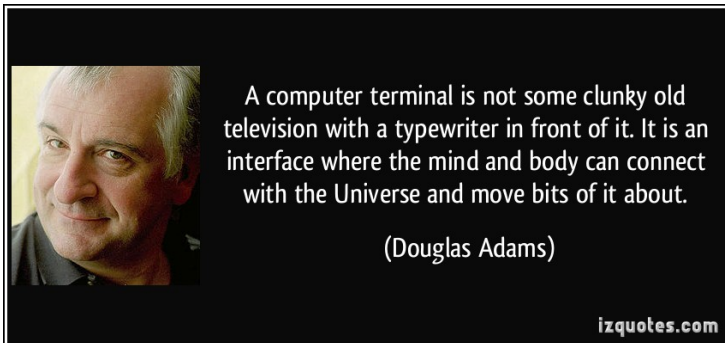


Figure: Douglas Noel Adams was an English author, scriptwriter, essayist, humorist, satirist and dramatist, best known as the author of *The Hitchhiker's Guide to the Galaxy* 1978



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



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

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

process ID 1



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

process ID 1

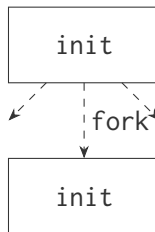
`init`



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

process ID 1



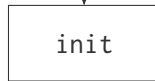
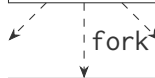
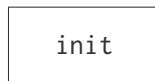
} forks once
per terminal



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
5. followed by exec of the program getty
6. opens terminal device (fd 0, 1, 2)
7. reads user name
8. initial environment set

process ID 1



} forks once per terminal

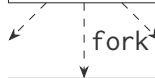
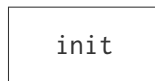
} each child execs getty



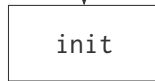
BSD Terminal Logins cont'd

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
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

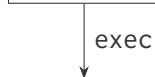
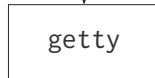
process ID 1



} forks once per terminal



} each child execs getty



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 various 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 various 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 various 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



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 exists as long as one process is in the group

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



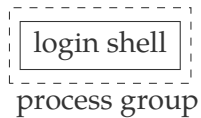
Process group cont'd

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

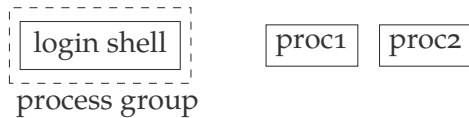
```
#include <unistd.h>
int setpgid(pid_t pid, pid_t pgid);
// Returns: 0 if OK, 1 on error
```

- sets the process group ID to *pgid* in the process whose process ID equals to *pid*
- if `pgid == pid`, then `pid ==` process group leader
- if `pid == 0`, caller process ID is used
- if `pgid == 0`, group ID == `pid`
- A process can set the process group ID of itself or its children





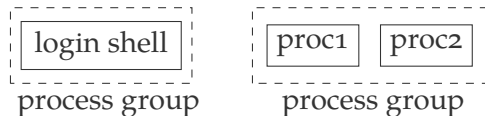




Sessions

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

```
proc1 | proc2 &
```

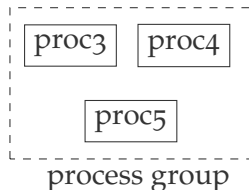
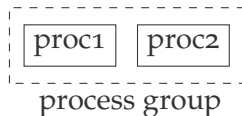
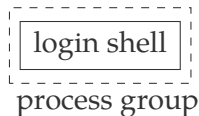


Sessions

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

```
proc1 | proc2 &
```

```
proc3 | proc4 | proc5
```

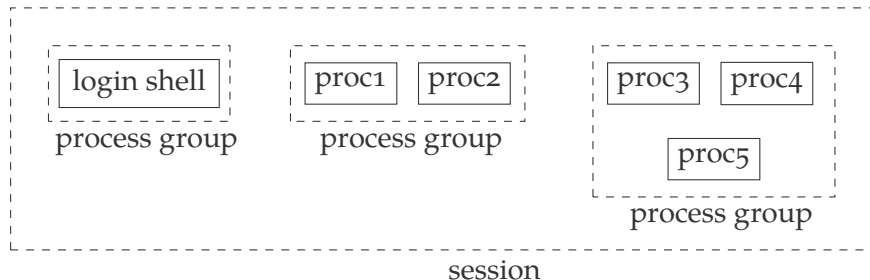


Sessions

The processes in a process group are usually placed there 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 function 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 function creates a new session. Three things happen

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



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 function 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



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 function 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 controlling 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

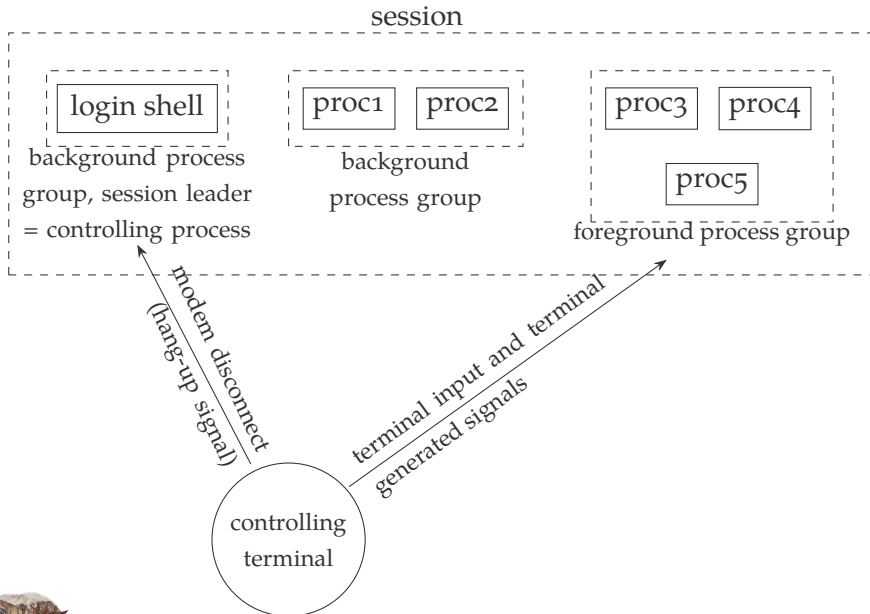


Controlling terminal

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



Controlling Terminal



Job Control

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

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 DELETE or Control-C) generates SIGINT
- Quit character (typically Control-bashslash) generates SIGQUIT
- Suspend character (typically Control-Z) generatea SIGTSTP



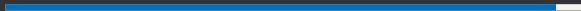
Job Control cont'd

```
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



LAST WORDS



Last Words

- Read chapter 10

