CMPS 105
Systems Programming

Prof. Darrell Long
E2.371
darrell@ucsc.edu
Chapter 9: Process Relationships
Chapter 9: Process Relationships

- Terminals
  - logins – local and remote

- Signals
  - process communication

- Details on process groups
  - group leaders
  - orphans

- Sessions
  - session leaders
Terminal Logins

- **Version 7**
  - local/remote terminals
  - Terminals had hard-wired connection to host
  - No windowing system until bitmapped graphical terminals

- **BSD**
  - Not changed much in past 35 years
  - `/etc/ttys` - one line per terminal
  - `init()` starts up `getty` for every line in `/etc/ttys` that allows login
    - `init` does `fork()` and `exec(getty())`
Figure 9.1 Processes invoked by init to allow terminal logins
login() does the following:

- getpwnam()
- getpass()
- crypt()
- exit() after too many fails
- or...
- chdir()
- chown()
- changes permissions
- setgid() and initgroups
- Initializes environment (HOME, SHELL, USER, PATH, ...)
- setuid() and invoke shell

Figure 9.2 State of processes after login has been invoked
Terminal logins cont...

- Mac OS X
  - Same as BSD except
    - `init()` work is done by `launchd()`
    - GUI login

- Linux
  - Similar to BSD
  - May have different terminal configuration

- Solaris
  - Either BSD style or ttymon login
  - `ttymon`: monitors all terminals listed in its config file and forks() when a login name is entered
Networking in BSD

- init() starts up shell script from /etc/rc
  - starts initd()

- initd() waits for TCP/IP connections
  - Forks when it gets a connection request
  -Execs appropriate program (telnet, ssh, …)

- Example: telnet connection
  - Telnet opens a pseudo terminal
  - Splits into two processes (using fork() )
    - Parent handles network communication
    - Child exec’s login
Remote logins

- Logins come through the kernel’s network interface drivers
- Don’t know ahead of time how many terminals
- Uses pseudo terminal to emulate serial terminal

Figure 9.3  Arrangement of processes after everything is set for a terminal login
Process groups: getpgrp(), getpgid(), setpgid()

- Each process group has a leader
  - Process group ID = leader’s process ID

- getpgrp() – returns the process group ID of the calling process
  - `pid_t getpgrp(void);`

- getpgid() – get process group for process specified by pid
  - `pid_t getpgid(pid_t pid);`

- setpgid() – allows a process to create a new process group or join existing group
  - `int setgid(pid_t pid, pgid);`
Sessions: setsid()

- New sessions
  - Process that calls setsid() is session leader
  - Process also becomes group leader
  - Process has no controlling terminal

- setsid() – establish a new session
  - pid_t setsid(void);

- getsid() – get process group ID
  - pid_t getsid(pid_t pid);
Controlling Terminal

- A session has a controlling terminal
- Session leader establishes connection to controlling terminal
  - Is called the controlling process
- Process groups within the session are divided into two groups
  - Foreground process group
  - Background process group
- Interrupt signals are sent to all foreground processes in the session
Controlling Terminal

Figure 9.7 Process groups and sessions showing controlling terminal
**tcgetpgrp(), tcsetpgrp, and tcgetsid()**

- **tcgetpgrp()** - returns the process group ID of foreground process group associated with terminal (fd)
  - `pid_t tcgetpgrp(int fd);`

- **tcsetpgrp()** - set the foreground process group
  - `int tcsetpgrp(int fd, pid_t pgrpId);`

- **tcgetsid()** - obtain the process group ID for the session leader
  - `pid_t tcgetsid(int fd);`
Job Control

- A job is a group of processes run from a single terminal

- Requirements for job control
  - A shell that supports job control
  - The terminal driver from the kernel support
  - Job related signal support from kernel

- Job control functionality is now commonly replaced by windowing systems
Figure 9.9  Summary of job control features with foreground and background jobs, and terminal driver
Orphaned Process Groups

- Orphaned children are inherited by init

- What happens when the child is stopped (job-control) when the parent terminates?

- Does the child know it’s orphaned?

Figure 9.11  Example of a process group about to be orphaned
Create an orphaned process group

```c
#include "apue.h"
#include <errno.h>

static void
sig_hup(int signo)
{
    printf("SIGHUP received, pid = %ld\n", (long)getpid());
}

static void
pr_ids(char *name)
{
    printf("%s: pid = %ld, ppid = %ld, pgrp = %ld, tpgrp = %ld\n", 
           name, (long)getpid(), (long)getppid(), (long)getpgrp(), 
           (long)tgetpgrp(STDIN_FILENO));
    fflush(stdout);
}

int main(void)
{
    char c;
    pid_t pid;
    pr_ids("parent");
    if ((pid = fork()) < 0) {
        err_sys("fork error");
    } else if (pid > 0) { /* parent */
        sleep(5); /* sleep to let child stop itself */
    } else { /* child */
        pr_ids("child");
        signal(SIGHUP, sig_hup); /* establish signal handler */
        kill(getpid(), SIGTSTP); /* stop ourself */
        pr_ids("child"); /* prints only if we're continued */
        if (read(STDIN_FILENO, &c, 1) != 1)
            printf("read error %d on controlling TTY\n", errno);
    
    exit(0);
}
```

Figure 9.12 Creating an orphaned process group
Figure 9.13  FreeBSD implementation of sessions and process groups