Process Relationships (Chapter 9)

Review:
☐ Every process has a parent
☐ Parent notified when child terminates
☐ Parent gets child’s exit status via wait*
☐ Parent dies before child, child inherited by process 1

History:
☐ 1970 ... central computers accessed by terminals
☐ Logins were via terminals .... real terminals
☐ Terminals not seen in quite a while (in most places)
How Logins were processed:

- init (using /etc/ttys) -> fork and exec getty
- getty gets user name -> execs login
- login verifies password -> execs login shell
- User uses login shell

Other Login Methods

Using an X display

- User logins in via getty/login, then runs startx
- xdm -- reads username & passwd, starts X as that user
  - Somewhat like a startx without the login shell
  - Can start a "terminal" (or shell) window
Network

- User connects to machine via telnetd, inetd or sshd
- telnetd/inetd/sshd -> fork/exec login -> exec shell

Shell

- In all cases "thinks" it is connected to a terminal
  - xterm window driver <-> shell, network <-> shell
  - fd’s 0, 1 and 2 set up for shell
- Real terminals
- Pseudo terminals
Process Group

Each process belongs to a process group.
Primary purpose ... signals
Secondary purpose ... terminal Read
  □ Every "terminal" has a "foreground" process group.
  □ That process group is the only process group allowed to read.

Most shells make each command line a different process group.

Example: foregd.c

System calls:
  □ pid_t getpgrp(void);
  □ pid_t getpgid(pid_t pid);
Setting up a process group

System calls:

- `int setpgid(pid_t pid, pid_t pgrp);`
- `int setpgrp(pid_t pid, pid_t pgrp); /* Old BSD */`

Typical use: Shell sets `pgrp` for a child before doing `exec`.

- `ls`
- `ls | sort`
- `grep xyz myfile | sort | uniq | cut -c3-25`
Sessions

Session is a collection of one or more process groups.
Session leader ... typically a shell

pid_t setsid(void);
- new session -- session leader
- new process group -- process group
- no controlling terminal
- error if calling process is a process group leader
Controlling Terminal (CT)

- Session has a single CT (real or pseudo)
- Session leader may establish a CT
- Session leader is controlling process
- Session may have many process groups
- If session has controlling terminal, then
  - single foreground process group
  - 0 or more background processes group
- "Keyboard" generated signals go to the foreground process group
- /dev/tty is CT
- pid_t tcgetpgrp(int fd); Get PGID of foreground process for fd
- int tcsetpgrp(int fd, pid_t pgrp_id); Set PGID for fd
- Signals
  - SIGTTIN - background read attempted on CT
  - SIGTTOU - background write attempted on CT
Job Control

BSD addition in 1980

- Job: cmd &
- Don’t wait!

- Originally:
  - Job
  - Interactive
    - Can’t "switch" between jobs
- Job Control -- "attach" different jobs to CT
- Work done by shell (session leader)
- ^Z -- SIGTSTP
- Built-in commands:
  - jobs
  - fg
  - bg
I/O from background

Controlling terminal as stdin (0)

Read:
- Need input from CT
- SIGTTIN -- stops jobs (if not caught)
- Shell restarts job by connecting CT
  - tcsetpgroup(), SIGCONT
- Example: `cat > file &`

Write:
- Depends on settings:
  - BSD -- normally lets output to CT
  - `stty tostop`
    - Send SIGTTO, stop process
  - `stty -tostop`
    - Allows bg writes to CT
Pipelines And Job Control!

- Want entire pipeline as a single process group.
  - `ps aux | grep dhcli | grep -v grep | cut -c5-10`
- Fork a process to do entire pipeline and be group leader
  - `sh -> fork -> sh1`
    - `sh1 -> fork -> exec ps`
      - `-> fork -> exec grep`
      - `-> fork -> exec grep`
      - `-> exec cut`
  - OR
    - `-> fork -> exec cut`
  - `-> wait for all children`