213 Recitation

Jiri Simsa

jsimsa@cs.cmu.edu
Outline

Tiny shell hints

- Executing programs
- Understanding process tree
- Reap all child processes
- Avoid race hazard
- I/O redirection
Execute program

```c
int execve(const char *fname,
            char *const argv[],
            char *const envp[]);
```

Examples:
```c
execve("/bin/ls", NULL, NULL);
execve("./mytest", argv, envp);
```
Each job has a unique process group id

```
int setpgid(pid_t pid, pid_t pgid);
setpgid(0, 0);
```
Process tree for a shell

Foreground job
receives SIGINT, SIGTSTP,
when you type ctrl-c, ctrl-z

```
int kill(pid_t pid, int sig)
pid > 0: send sig to specified process
pid = 0: send sig to all processes in same group
pid < -1: send sig to group -pid
```
Reaping child process

waitpid(pid_t pid, int *status, int options)

pid: wait for child process with pid (process id or group id)
-1: wait for any child process

status: tell why child terminated

options:
  WNOHANG: return immediately if no children zombied
  WUNTRACED: report status of terminated or stopped children

In Lab 4, use: waitpid(-1, &status, WNOHANG|WUNTRACED)

In sigchld_handler(): while ((c_pid = waitpid(...)) > 0)
Reaping child process

```c
int status;
waitpid(pid, &status, WNOHANG|WUNTRACED)
```

What to check in sigchld_handler:
- **WIFEXITED(status):**
  - child exited normally (the child finished, and quit normally on its own)
  - **WEXITSTATUS(status):** returns code when child exits
- **WIFSIGNALED(status):**
  - child exited because a signal is not caught (SIGINT, or typing CTRL-C)
  - **WTERMSIG(status):** gives the terminating signal number
- **WIFSTOPPED(status):**
  - child is stopped by the receipt of a signal (SIGSTOP, or typing CTRL-Z)
  - **WSTROPSIG(status):** gives the stop signal number
Reaping child process

- Where to put `waitpid(...)`?
- `eval()` vs. `sigchld_handler()`
- In `sigchld_handler()`: for both
- `eval()` should wait for a `fg` job
Busy wait for a fg job

In eval():

```c
if (fork() != 0) { /* parent */
    addjob(...);
    while (fg process still alive) {
        /* do nothing */
    }
}
```
Sleep

In eval():

```c
if(fork() != 0) { /* parent */
    addjob(...);
    while(fg process still alive){
        sleep(1);
    }
}
```
A data structure is shared by two pieces of code that can run concurrently.

Different behaviors of program depending upon how the schedule interleaves the execution of code.
An example of race hazard

```
sigchld_handler() {
    while ((pid = waitpid(...)) > 0){
        deletejob(pid);
    }
}
```

```
eval() {
    pid = fork();
    if(pid == 0)
    {
        /* child */
        execve(...);
    }
    /* parent */
    /* signal handler may run BEFORE addjob()*/
    addjob(...);
}
```
time

Shell

-fork()

addjob()

Signal Handler

execve()

exit()

deletejob()

Child

sigchld_handler()
Shell

fork()

Signal Handler

sigchld_handler()

deletejob()

addjob()

Child

execve()

exit()

Job added to job list *after* the signal handler tried to delete it!
Solution: blocking signals

```c
sigchild_handler() {
    pid = waitpid(...);
    deletejob(pid);
}
eval() {
    sigprocmask(SIG_BLOCK, ...)
    pid = fork();
    if (pid == 0)
    { /* child */
        sigprocmask(SIG_UNBLOCK, ...)
        execve(...);
    }
    /* parent */
    /* signal handler might run BEFORE addjob() */
    addjob(...);
    sigprocmask(SIG_UNBLOCK, ...)
}
```
I/O redirection

- Do it before call execve() in child process

eval() {
    pid = fork();
    if (pid == 0) {
        /* child */
        /* Redirect I/O */
        if (redirect_input)
            dup2(...); /* redirect STDIN */
        if (redirect_output)
            dup2(...); /* redirect STDOUT */

        execve(...);
    }
    addjob(...);
}
dup2(int oldfd, int newfd)

- Covered in Chapter 11
  oldfd: old file descriptor
  newfd: new file descriptor
- Dup2 makes newfd be a copy of the oldfd

Some examples:
Get input from in_fd instead of standard input
dup2(in_fd, STDIN_FILENO);

Print output to out_fd instead of standard output
dup2(out_fd, STDOUT_FILENO);
Reminders

❖ Some important system calls:
❖ fork, execve, waitpid, sigprocmask, setpgid, kill
❖ Check man pages for details about system calls
❖ man 2 kill
❖ Check return values of all system calls

❖ STEP by STEP
❖ Test your shell by typing commands first
❖ Each trace worth the same (work on easy ones first!)
❖ Start tomorrow!