

A Tiny Unix Shell (tsh) – Part III

Introduction

This is the third part of the project on process control and signaling. In this assignment you will fix the problem with the shell from the earlier assignment, and have the shell execute the `jobs` command properly. This assignment must be completed *individually*.

Background Children

The shell you have implemented in the second part of this project correctly waited for and reaped foreground jobs. But what about background jobs? Background jobs become zombies when they terminate, and they never get reaped because shell (typically) never terminates. This creates a memory leak that will eventually crash the kernel when it runs out of memory. Reaping background jobs requires a mechanism called a *signal*.

A *signal* is a small message that notifies a process that an event of some type has occurred in the system. When a child terminates, the kernel delivers a `SIGCHLD` signal to its parent (`tsh` in this case). Your shell should catch this signal, reap the zombie child and delete the child from the job list.

What to do

1. Add a new module called `siglib` (signal library) to your project. This module should implement the signal handler for `SIGCHLD`. The code for the signal handler is provided below.

```
/* File siglib.c */

/* Add appropriate headers */

/* sigchld_handler - The kernel sends SIGCHLD to the shell whenever
 * a child job terminates (becomes a zombie). The handler reaps
 * all available zombie children, but doesn't wait for any other
 * currently running children to terminate.
 */
void sigchld_handler(int sig)
{
    pid_t child_pid;

    printf("SIGCHLD processed");fflush(stdout);

    /* Detect any terminated jobs, but don't wait on the others. */
    while ((child_pid = waitpid(-1, 0, WNOHANG)) > 0)
    {
        deletejob(jobs, child_pid);
    }
    return;
}
```

2. Write an interface (header file) called `siglib.h` for this module and import it into any other module(s) that use signal handlers.
3. You will need to install the signal handler for `SIGCHLD` first thing in the main function of your shell (before entering the infinite loop). Add the following `Signal` function to your `siglib.c` module, and use it in `tsh.c` to install your handler:

```
typedef void handler_t(int); /* Move to siglib.h */

/*
 * Signal - wrapper for the sigaction function
 */
void Signal(int signum, handler_t *handler)
{
    struct sigaction action, old_action;

    action.sa_handler = handler;
    sigemptyset(&action.sa_mask); /* block sigs of type being handled */
    action.sa_flags = SA_RESTART; /* restart syscalls if possible */

    if (sigaction(signum, &action, &old_action) < 0)
        printf("Signal error");
    return;
}
```

4. Thoroughly test your code before moving on to step 5.
5. Once you have completed steps 1 through 4, you will notice that your shell sometimes fails to remove all terminated children from the job list (i.e., the built-in command `jobs` is inconsistent and sometimes it may not execute properly). This is due to a race condition where the child is reaped by `sigchld` handler (and thus removed from the job list) *before* the parent calls `addjob`.

To eliminate this problem, the parent must use `sigprocmask` to block `SIGCHLD` signals before it forks the child, and then unblock these signals, again using `sigprocmask`, after it adds the child to the job list by calling `addjob`:

```
/* In eval.c, define a signal mask: */

    ??? mask; /* Use the man pages to determine the datatype */

/* In eval.c, before forking a child: */
    if (sigemptyset(&mask) < 0)
        printf("sigemptyset error");
    if (sigaddset(&mask, SIGCHLD))
        printf("sigaddset error");
    if (sigprocmask(SIG_BLOCK, &mask, NULL) < 0)
        printf("sigprocmask error");

/* In eval.c, after parent calls addjob: */
    sigprocmask(SIG_UNBLOCK, &mask, NULL);
```

Also, since children inherit the blocked vectors of their parents, the child must be sure to unblock SIGCHLD signals before it `execs` the new program.

Evaluation

Your score will be computed out of a maximum of 50 points based on the following distribution:

40 Correctness.

10 Style points. We expect you to have good comments and to check for errors.

Hand In Instructions

Hand in a printed copy of the following:

1. Source code of `tsh.c`, `siglib.c` and `siglib.h`.
2. A sample output of your code.
3. A short README file explaining any bugs and deviations from the assignment specifications.

Make sure you have included your name and your Unix login name in the README file. Leave your source code in your Unix account. Good luck!