A Simple Linux Shell - ASH (Awesome SHell)

Andre Christensen (110033) og Jacob Pedersen (120374)

October 23, 2013

Contents

1 Introduction

This journal describes our implementation of a simple Linux shell, given as a mandatory lab exercise in the course 'Operating Systems and Embedded Linux - IOSLX4-E13'.

ASH contains the following features:

- Execute commands in the foreground.
- Execute commands (upto 20) in the background (by appending a '&' to the command).
- A built-in 'killbg' command which kills all processes running in the background.

2 Implementation details

ASH is capable of executing commands in both the foreground and in the background. Every command in read from the *stdin* and parsed into tokens seperated by spaces. A 'run-in-background' flag is set, if the character '&' is appended at the end of the command.

After the command has been read a new child is forked, and depending on the type of process (foreground or background) the shell waits for the PID to exit, or becomes ready to accept a new command from the user.

ASH keeps track of background processes, by keeping their PID's in an array. The status of all background processes is checked each time the SIGCHLD signal handler is fired. Checking the status of a process implies calling *waitpid* on the background process' PID with the WNOHAND flag, and checking if the system call returns the process' PID, which means the process and exited.

3 Testing

Figure 1 shows a screenshot of a testrun. The screenshots shows the following:

- 'ps' is executed in the foreground and lists running processes.
- The './test' program is started in the background (PID 7958), and the prompt is immediately ready for input again.
- 'ps' is executed again, and './test' is now listed as running.
- The built-in 'killbg' is now executed, and outputs shows that the './test' process (PID 7958) is killed.
- 'ps' is executed again, and shows the test program isn't running anymore.
- Finally the built-in 'exit' command is executed, and ASH exits.

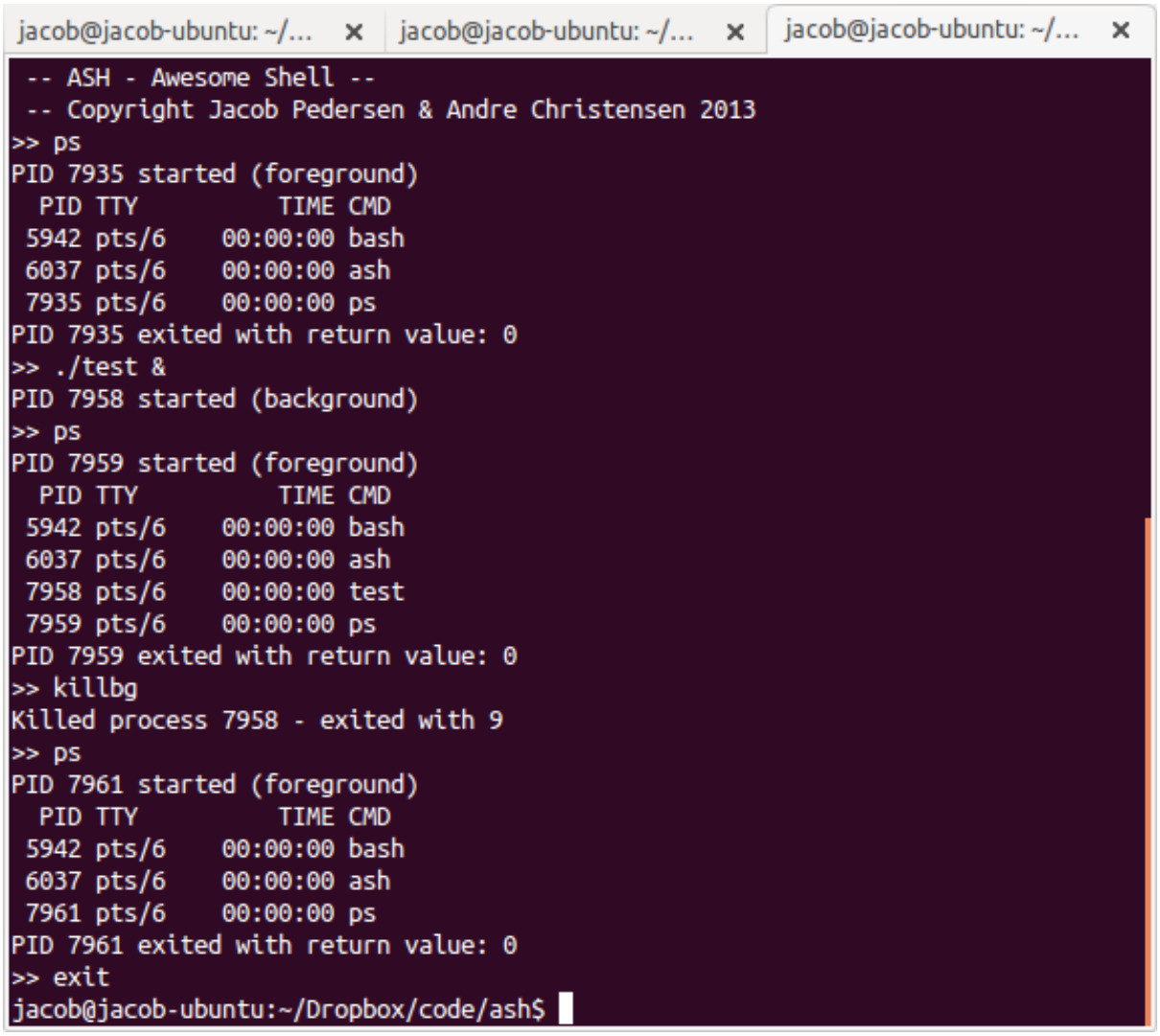

Figure 1: Execution of jobs in the foreground and background, and demonstration of the killbg command

3.1 Test system details

- OS: Ubuntu 13.10 (Linux 3.11.0)
- Compiler: gcc 4.8.1 (Ubuntu/Linaro)
- Makefile: See Listing 2

4 Source Code

Listing 1: ash.c

```
/**
 *
 * Awesome SHell - ASH
 * http://github.com/bomstrong/ash
 *
 * Copyright (C) Jacob Aslund Friis Pedersen & Andre Daniel Christensen
 * Technical University of Denmark - DTU
 *
 * ASH is a simple shell. It A process can be started in the background
 * by appending a & to the command. ASH has two built-in commands; killbg
 * which kills all processes running in the background, and exit which exits
 * ASH and returns to the 'real' shell.
 *
 */

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MAX_CHILDS                20
#define MAX_LEN                   255

static int background;
static pid_t fg_pid;
static pid_t pid_list[MAX_CHILDS] = {0};

static void check_bg_status();

/**
 * Signal handler for catching CTRL-C when a foreground process
 * is running. Notice that we ignore CTRL-C at any other time.
 *
 * |param sig The signal number
 */
static void sigint_handler(int sig)
{
        kill(fg_pid, SIGKILL);
}

/**
 * Signal handler for SIGCHLD. Checks the status of all background
 * processes when a SIGCHILD is received.
 *
 * |param sig The signal number
 */
static void sigchld_handler(int sig)
{
        check_bg_status();
}

/**
 * Kill all processes running in the backgound.
 */
static void kill_bg()
{
        int i, r, status;
        for (i = 0; i < MAX_CHILDS; i++)
        {
                if (pid_list[i] > 0)
                {
```

```
                        r = kill(pid_list[i], SIGKILL);
                        if (r < 0)
                        {
                                printf("Failed killing process %d\n", pid_list[i]);
                        }
                        else
                        {
                                // Wait for the process to exit and print info
                                waitpid(pid_list[i], &status, 0);
                                printf("Killed process %d - exited with %d\n", pid_list[
                                    i],
                                        status);
                        }
                        pid_list[i] = 0;
                }
        }
}

/**
 * Finds the index of the next available spot in the list of
 * processes running in the background.
 *
 * \return Index of the next available spot, or -1 if the list is full
 */
static int get_next_avail_index()
{
        int i;
        for (i = 0; i < MAX_CHILDS; i++)
        {
                if (pid_list[i] == 0)
                {
                        return i;
                }
        }
        return -1;
}

/**
 * Add `pid` to the list of PIDs for processes running
 * in the background.
 *
 * \param pid The PIP of the child
 * \return 0 on success, or -1 if if the PID could not be added to the list
 */
static int add_to_list(pid_t pid)
{
        int idx = get_next_avail_index();
        if (idx > -1)
        {
                pid_list[idx] = pid;
        }
        else
        {
                return -1;
        }
        return 0;
}

/**
 * Remove the `pid` from the list of processes running in the background.
 * The function doesn't return anything as we don't care about the result.
 *
 * \param pid PID to remove from the list
 */
static void remove_from_list(pid_t pid)
{
        int i;
        for (i = 0; i < MAX_CHILDS; i++)
        {
                if (pid_list[i] == pid)
```

```
                {
                        pid_list[i] = 0;
                }
        }
}

/**
 * Check status of all background processes and print
 * their return value if they have exited.
 */
static void check_bg_status()
{
        int i;
        int status;

        for (i = 0; i < MAX_CHILDS; i++)
        {
                if (pid_list[i] > 0)
                {
                        if (waitpid(pid_list[i], &status, WNOHANG) == pid_list[i])
                        {
                                // Child with PID `pid_list[i]` exited. Remove from list
                                printf("\nBackground process %d exited with status %d\n
                                    >> ",
                                        pid_list[i], status);
                                fflush(stdout);

                                remove_from_list(pid_list[i]);
                        }
                }
        }
}

/**
 * Main
 */
int main(int argc, char ** argv)
{
        char buffer[MAX_LEN];
        char * list[MAX_LEN];
        struct sigaction action;

        printf(" -- ASH - Awesome Shell -- \n");
        printf(" -- Copyright Jacob Pedersen & Andre Christensen 2013\n");

        // Handle child process termination
        signal(SIGCHLD, sigchld_handler);

        while(1)
        {
                int i, status;
                char * pch;

                background = 0;

                // Ignore Ctrl-C until a foreground process is started
                action.sa_handler = SIG_IGN;
                sigaction(SIGINT, &action, 0);

                //check_bg_status();

                printf(">> ");

                // Read input from the prom into a buffer.
                if (fgets(buffer, MAX_LEN, stdin) != NULL)
                {
                        // Skip empty string (check string termination and newline)
                        if (*buffer == '\0' || *buffer == '\n')
                        {
                                continue;
```

```
}

// Remove newline at the end of the command
for (i = 0; i < MAX_LEN; i++)
{
        if (buffer[i] == '\n')
        {
                buffer[i] = '\0';
        }
}


// Splits the string into tokens.
i = 0;
pch = strtok(buffer, " ");

// Loops until the last token
while (pch != NULL)
{
        list[i++] = pch;
        pch = strtok(NULL, " ");
}

// Check the last token for '&' which indicates the process
// should run in the background
if(strcmp(list[i-1], "&") == 0)
{
        background = 1;
        list[i-1] = 0;
}
else
{
        list[i] = (char *) 0;
}

//
// Check for built-in commands.
//
if (strcmp(list[0], "exit") == 0)
{
        exit(0);
}
else if (strcmp(list[0], "killbg") == 0)
{
        kill_bg();
}
//
// Or fork a new child and execute the requested command
//
else
{
        // Fork a new process and get the pid
        pid_t pid = fork();
        // The child
        if (pid == 0)
        {
                if (execvp(list[0], list) == -1)
                {
                        perror("The following error occurred");
                }
                exit(0);
        }
        // The parent
        else
        {
                printf("PID %d started (%s)\n", pid, (background
                     != 1)
                        ? "foreground" : "background");

                // If the process should run in the background
```

```
                                        if(background == 1)
                                        {
                                                // TODO Check if this succeds
                                                add_to_list(pid);

                                                // Quickly check if the background
                                                    process has already
                                                // exited. The WNOHANG tells the system
                                                    call to return
                                                // immediately, and is returning the pid
                                                    of the process
                                                // if it has exited.
                                                if (waitpid(pid, &status, WNOHANG) ==
                                                    pid)
                                                {
                                                        printf("Background process %d
                                                            exited\n", pid);
                                                        remove_from_list(pid);
                                                }
                                                else
                                                {
                                                        // Background process still
                                                            running
                                                }
                                        }
                                        // Or if the process should run in the
                                            foreground
                                        else
                                        {
                                                fg_pid = pid;

                                                // Catch CTRL-C signals
                                                action.sa_handler = sigint_handler;
                                                sigaction(SIGINT, &action, 0);

                                                // Wait till the process have changed
                                                    state
                                                waitpid(fg_pid, &status, 0);

                                                printf("PID %d exited with return value:
                                                    %d\n", pid,
                                                        status);
                                        }
                                }
                        }
                }
        }
        return 0;
}
```

Listing 2: Makefile

```
all:
        gcc -Wall -O0 -o ash ash.c

clean:
        rm ash
```