A Simple Linux Shell - ASH (Awesome SHell)

Andre Christensen (110033) and 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 is 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.

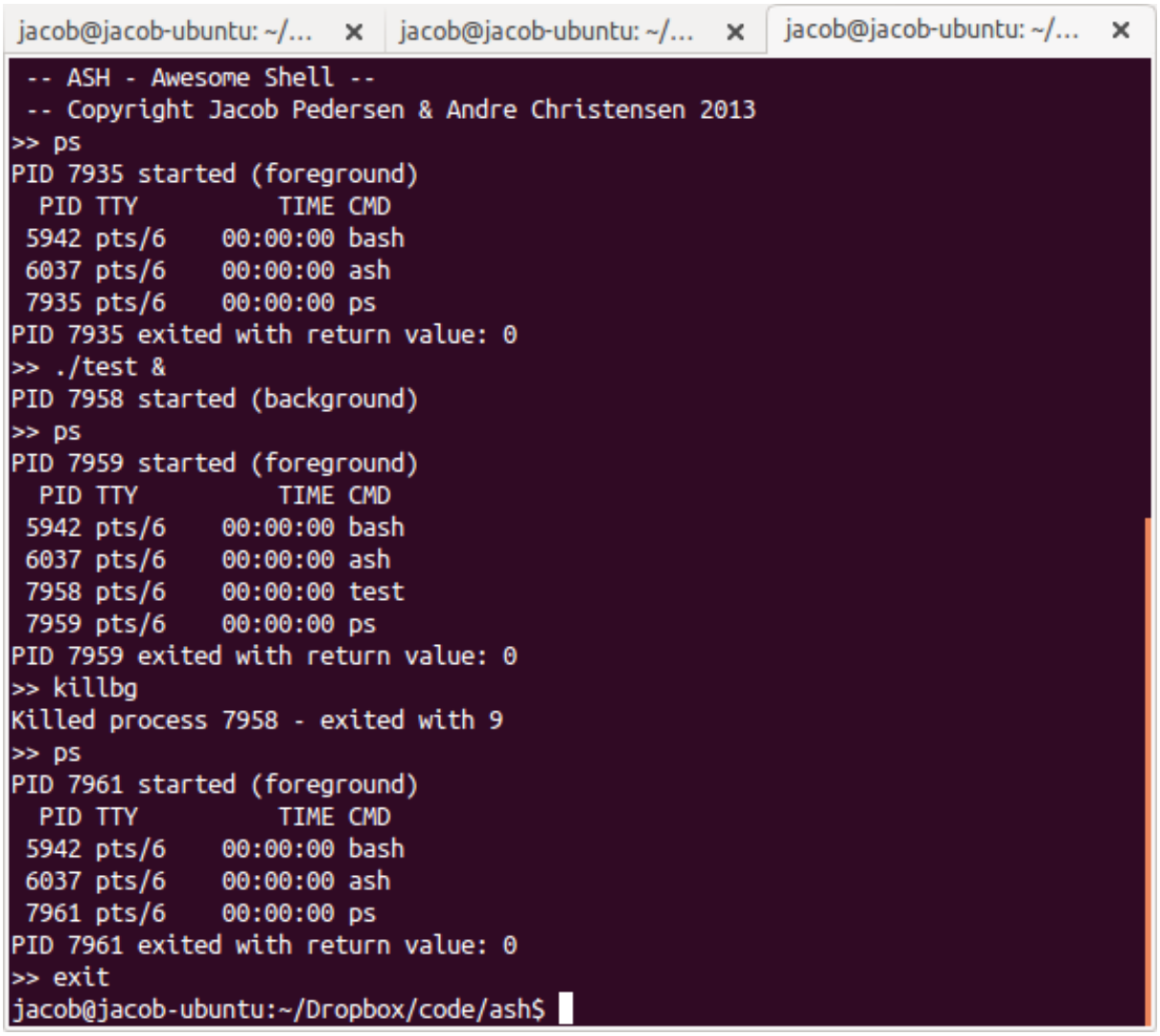

Figure 1: Execution of jobs in the foreground and background, and demonstration of the killbg command

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

```
1  /**
2   *
3   * Awesome SHell - ASH
4   * http://github.com/bomstrong/ash
5   *
6   * Copyright (C) Jacob Aslund Friis Pedersen & Andre Daniel Christensen
7   * Technical University of Denmark - DTU
8   *
9   * ASH is a simple shell. It A process can be started in the background
10  * by appending a & to the command. ASH has two built-in commands; killbg
11  * which kills all processes running in the background, and exit which exits
12  * ASH and returns to the 'real' shell.
13  *
14  */
15
16 #include <stdio.h>
17 #include <string.h>
18 #include <unistd.h>
19 #include <stdlib.h>
20 #include <unistd.h>
21 #include <signal.h>
22 #include <sys/types.h>
23 #include <sys/wait.h>
24
25 #define MAX_CHILDS      20
26 #define MAX_LEN      255
27
28 static int background;
29 static pid_t fg_pid;
30 static pid_t pid_list[MAX_CHILDS] = {0};
31
32 static void check_bg_status();
33
34 /**
35  * Signal handler for catching CTRL-C when a foreground process
36  * is running. Notice that we ignore CTRL-C at any other time.
37  *
38  * |param sig The signal number
39  */
40 static void sigint_handler(int sig)
41 {
42     kill(fg_pid, SIGKILL);
43 }
44
45 /**
46  * Signal handler for SIGCHLD. Checks the status of all background
47  * processes when a SIGCHILD is received.
48  *
49  * |param sig The signal number
50  */
51 static void sigchld_handler(int sig)
52 {
53     check_bg_status();
54 }
55
56 /**
57  * Kill all processes running in the backgound.
58  */
59 static void kill_bg()
60 {
61     int i, r, status;
62     for (i = 0; i < MAX_CHILDS; i++)
63     {
64         if (pid_list[i] > 0)
65         {
```

```
66            r = kill(pid_list[i], SIGKILL);
67            if (r < 0)
68            {
69               printf("Failed killing process %d\n", pid_list[i]);
70            }
71            else
72            {
73               // Wait for the process to exit and print info
74               waitpid(pid_list[i], &status, 0);
75               printf("Killed process %d - exited with %d\n", pid_list[i],
76                  status);
77            }
78            pid_list[i] = 0;
79         }
80      }
81   }
82
83   /**
84    * Finds the index of the next available spot in the list of
85    * processes running in the background.
86    *
87    * \return Index of the next available spot, or -1 if the list is full
88    */
89   static int get_next_avail_index()
90   {
91      int i;
92      for (i = 0; i < MAX_CHILDS; i++)
93      {
94         if (pid_list[i] == 0)
95         {
96            return i;
97         }
98      }
99      return -1;
100  }
101
102  /**
103   * Add `pid` to the list of PIDs for processes running
104   * in the background.
105   *
106   * \param pid The PIP of the child
107   * \return 0 on success, or -1 if if the PID could not be added to the list
108   */
109  static int add_to_list(pid_t pid)
110  {
111     int idx = get_next_avail_index();
112     if (idx > -1)
113     {
114        pid_list[idx] = pid;
115     }
116     else
117     {
118        return -1;
119     }
120     return 0;
121  }
122
123  /**
124   * Remove the `pid` from the list of processes running in the background.
125   * The function doesn't return anything as we don't care about the result.
126   *
127   * \param pid PID to remove from the list
128   */
129  static void remove_from_list(pid_t pid)
130  {
131     int i;
132     for (i = 0; i < MAX_CHILDS; i++)
133     {
134        if (pid_list[i] == pid)
135        {
```

```
136             pid_list[i] = 0;
137         }
138     }
139 }
140 
141 /**
142  * Check status of all background processes and print
143  * their return value if they have exited.
144  */
145 static void check_bg_status()
146 {
147     int i;
148     int status;
149 
150     for (i = 0; i < MAX_CHILDS; i++)
151     {
152         if (pid_list[i] > 0)
153         {
154             if (waitpid(pid_list[i], &status, WNOHANG) == pid_list[i])
155             {
156                 // Child with PID `pid_list[i]` exited. Remove from list
157                 printf("\nBackground process %d exited with status %d\n>> ",
158                     pid_list[i], status);
159                 fflush(stdout);
160 
161                 remove_from_list(pid_list[i]);
162             }
163         }
164     }
165 }
166 
167 /**
168  * Main
169  */
170 int main(int argc, char ** argv)
171 {
172     char buffer[MAX_LEN];
173     char * list[MAX_LEN];
174     struct sigaction action;
175 
176     printf(" -- ASH - Awesome Shell -- \n");
177     printf(" -- Copyright Jacob Pedersen & Andre Christensen 2013\n");
178 
179     // Handle child process termination
180     signal(SIGCHLD, sigchld_handler);
181 
182     while(1)
183     {
184         int i, status;
185         char * pch;
186 
187         background = 0;
188 
189         // Ignore Ctrl-C until a foreground process is started
190         action.sa_handler = SIG_IGN;
191         sigaction(SIGINT, &action, 0);
192 
193         //check_bg_status();
194 
195         printf(">> ");
196 
197         // Read input from the prom into a buffer.
198         if (fgets(buffer, MAX_LEN, stdin) != NULL)
199         {
200             // Skip empty string (check string termination and newline)
201             if (*buffer == '\0' || *buffer == '\n')
202             {
203                 continue;
204             }
205 
```

```
206             // Remove newline at the end of the command
207             for (i = 0; i < MAX_LEN; i++)
208             {
209                 if (buffer[i] == '\n')
210                 {
211                     buffer[i] = '\0';
212                 }
213             }
214
215
216             // Splits the string into tokens.
217             i = 0;
218             pch = strtok(buffer, " ");
219
220             // Loops until the last token
221             while (pch != NULL)
222             {
223                 list[i++] = pch;
224                 pch = strtok(NULL, " ");
225             }
226
227             // Check the last token for '&' which indicates the process
228             // should run in the background
229             if(strcmp(list[i-1], "&") == 0)
230             {
231                 background = 1;
232                 list[i-1] = 0;
233             }
234             else
235             {
236                 list[i] = (char *) 0;
237             }
238
239             //
240             // Check for built-in commands.
241             //
242             if (strcmp(list[0], "exit") == 0)
243             {
244                 exit(0);
245             }
246             else if (strcmp(list[0], "killbg") == 0)
247             {
248                 kill_bg();
249             }
250             //
251             // Or fork a new child and execute the requested command
252             //
253             else
254             {
255                 // Fork a new process and get the pid
256                 pid_t pid = fork();
257                 // The child
258                 if (pid == 0)
259                 {
260                     if (execvp(list[0], list) == -1)
261                     {
262                         perror("The following error occurred");
263                     }
264                     exit(0);
265                 }
266                 // The parent
267                 else
268                 {
269                     printf("PID %d started (%s)\n", pid, (background != 1)
270                         ? "foreground" : "background");
271
272                     // If the process should run in the background
273                     if(background == 1)
274                     {
275                         // TODO Check if this succeds
```

```
276                     add_to_list(pid);
277
278                     // Quickly check if the background process has already
279                     // exited. The WNOHANG tells the system call to return
280                     // immediately, and is returning the pid of the process
281                     // if it has exited.
282                     if (waitpid(pid, &status, WNOHANG) == pid)
283                     {
284                         printf("Background process %d exited\n", pid);
285                         remove_from_list(pid);
286                     }
287                     else
288                     {
289                         // Background process still running
290                     }
291                 }
292                 // Or if the process should run in the foreground
293                 else
294                 {
295                     fg_pid = pid;
296
297                     // Catch CTRL-C signals
298                     action.sa_handler = sigint_handler;
299                     sigaction(SIGINT, &action, 0);
300
301                     // Wait till the process have changed state
302                     waitpid(fg_pid, &status, 0);
303
304                     printf("PID %d exited with return value: %d\n", pid,
305                         status);
306                 }
307             }
308         }
309     }
310 }
311 return 0;
312 }
```

Listing 2: Makefile

```
1
2
3  all:
4      gcc -Wall -O0 -o ash ash.c
5
6  clean:
7      rm ash
```