

```

1 Handout for CS 372H
2 Class 5
3 31 January 2011
4
5 The first four panels, taken together, are meant to:
6
7     --communicate the power of the fork()/exec() separation
8
9     --illustrate how the shell itself uses syscalls
10
11     --give an example of how small, modular pieces (file descriptors,
12 pipes, fork(), exec()) can be combined to achieve complex behavior
13 far beyond what any single application designer could or would have
14 specified at design time.
15
16 1. Pseudocode for a very simple shell (see minish.c at the back of the
17 handout for a non-pseudocode version of what's below).
18
19     while (1) {
20         write(1, "$ ", 2);
21         readcommand(command, args); // parse input
22         if ((pid = fork()) == 0) // child?
23             exec(command, args, 0);
24         else if (pid > 0) // parent?
25             wait(0); //wait for child
26         else
27             perror("failed to fork");
28     }
29
30 2. Now add two features to this simple shell: output redirection and
31 backgrounding (see redirsh.c at the back of the handout for a non-pseudocode
32 version of the redirection handling).
33
34     By output redirection, we mean, for example:
35     $ ls > list.txt
36     By backgrounding, we mean, for example:
37     $ myprog &
38     $
39
40     while (1) {
41         write(1, "$ ", 2);
42         readcommand(command, args); // parse input
43         if ((pid = fork()) == 0) { // child?
44             if (output_redirected) {
45                 close(1);
46                 open(redirect_file, O_CREAT | O_TRUNC | O_WRONLY, 0666);
47             }
48             // when command runs, fd 1 will refer to the redirected file
49             exec(command, args, 0);
50         } else if (pid > 0) { // parent?
51             if (foreground_process) {
52                 wait(0); //wait for child
53             }
54         } else {
55             perror("failed to fork");
56         }
57     }
58

```

```

59 3. Another syscall example: pipe()
60
61     The pipe() syscall is used by the shell to implement pipelines, such as
62     $ ls | sort | head -4
63     We will see this in a moment; for now, here is an example use of
64     pipes.
65
66         // C fragment with simple use of pipes
67
68         int fdarray[2];
69         char buf[512];
70         int n;
71
72         pipe(fdarray);
73         write(fdarray[1], "hello", 5);
74         n = read(fdarray[0], buf, sizeof(buf));
75         // buf[] now contains 'h', 'e', 'l', 'l', 'o'
76
77 4. File descriptors are inherited across fork
78
79         // C fragment showing how two processes can communicate over a pipe
80
81         int fdarray[2];
82         char buf[512];
83         int n, pid;
84
85         pipe(fdarray);
86         pid = fork();
87         if(pid > 0){
88             write(fdarray[1], "hello", 5);
89         } else {
90             n = read(fdarray[0], buf, sizeof(buf));
91         }
92

```

Feb 02, 12 20:38

shell.txt

Page 3/4

```

93 5. Putting it all together: implementing shell pipelines using
94 fork(), exec(), and pipe(). (See pipesh.c at the back of the
95 handout for a non-pseudocode version of the pipeline handling.)
96
97
98 // Pseudocode for a Unix shell that can run processes in the
99 // background, redirect the output of commands, and implement
100 // two element pipelines, such as "ls | sort"
101
102 void main_loop() {
103
104     while (1) {
105         write(1, "$ ", 2);
106         readcommand(command, args); // parse input
107         if ((pid = fork()) == 0) { // child?
108             if (pipeline_requested) {
109                 handle_pipeline(left_command, right_command)
110             } else {
111                 if (output_redirected) {
112                     close(1);
113                     open(redirect_file, O_CREAT | O_TRUNC | O_WRONLY, 0666);
114                 }
115                 exec(command, args, 0);
116             }
117         } else if (pid > 0) { // parent?
118             if (foreground_process) {
119                 wait(0); // wait for child
120             }
121         } else {
122             perror("failed to fork");
123         }
124     }
125 }
126
127 void handle_pipeline(left_command, right_command) {
128
129     int fdarray[2];
130
131     if (pipe(fdarray) < 0) panic ("error");
132     if ((pid = fork ()) == 0) { // child (left end of pipe)
133
134         close (1);
135         dup2 (fdarray[1], 1); // make fd 1 the same as fdarray[1],
136                             // which is the write end of the pipe
137
138         close (fdarray[0]);
139         close (fdarray[1]);
140         parse(command1, args1, left_command);
141         exec (command1, args1, 0);
142
143     } else if (pid > 0) { // parent (right end of pipe)
144
145         close (0);
146         dup2 (fdarray[0], 0); // make fd 0 the same as fdarray[0],
147                             // which is the read end of the pipe
148
149         close (fdarray[0]);
150         close (fdarray[1]);
151         parse(command2, args2, right_command);
152         exec (command2, args2, 0);
153
154     } else {
155         printf ("Unable to fork\n");
156     }
157 }

```

Feb 02, 12 20:38

shell.txt

Page 4/4

```

156
157 6. Commentary
158
159 Why is this interesting? Because pipelines and output redirection
160 are accomplished by manipulating the child's environment, not by
161 asking a program author to implement a complex set of behaviors.
162 That is, the *identical code* for "ls" can result in printing to the
163 screen ("ls -l"), writing to a file ("ls -l > output.txt"), or
164 getting ls's output formatted by a sorting program ("ls -l | sort").
165
166 This concept is powerful indeed. Consider what would be needed if it
167 weren't for redirection: the author of ls would have had to
168 anticipate every possible output mode and would have had to build in
169 an interface by which the user could specify exactly how the output
170 is treated.
171
172 What makes it work is that the author of ls expressed his or her
173 code in terms of a file descriptor:
174     write(1, "some output", byte_count);
175 This author does not, and cannot, know what the file descriptor will
176 represent at runtime. Meanwhile, the shell has the opportunity, *in
177 between fork() and exec()*, to arrange to have that file descriptor
178 represent a pipe, a file to write to, the console, etc.

```

Feb 02, 12 15:11

our_head.c

Page 1/1

```

1  /*
2  * our_head.c -- a C program that prints the first L lines of its input,
3  *   where L defaults to 10 but can be specified by the caller of the
4  *   program.
5  *
6  *   (This program is inefficient and does not check its error
7  *   conditions. It is meant to illustrate filters.)
8  */
9  #include <stdlib.h>
10 #include <unistd.h>
11 #include <stdio.h>
12
13 int main(int argc, char** argv)
14 {
15     int i = 0;
16     int nlines;
17     char ch;
18     int ret;
19
20     if (argc == 2) {
21         nlines = atoi(argv[1]);
22     } else if (argc == 1) {
23         nlines = 10;
24     } else {
25         fprintf(stderr, "usage: our_head [nlines]\n");
26         exit(1);
27     }
28
29     for (i = 0; i < nlines; i++) {
30
31         do {
32
33             /* read in the first character from fd 0 */
34             ret = read(0, &ch, 1);
35
36             /* if there are no more characters to read, then exit */
37             if (ret == 0) exit(0);
38
39             write(1, &ch, 1);
40
41         } while (ch != '\n');
42     }
43
44     exit(0);
45 }
46

```

Feb 02, 12 15:11

our_yes.c

Page 1/1

```

1  /*
2  * our_yes.c -- a C program that prints its argument to the screen on a
3  *   new line every second.
4  *
5  */
6  #include <stdlib.h>
7  #include <string.h>
8  #include <unistd.h>
9  #include <stdio.h>
10
11 int main(int argc, char** argv)
12 {
13     char* repeated;
14     int len;
15
16     /* check to make sure the user gave us one argument */
17     if (argc != 2) {
18         fprintf(stderr, "usage: our_yes string_to_repeat\n");
19         exit(1);
20     }
21
22     repeated = argv[1];
23
24     len = strlen(repeated);
25
26     /* loop forever */
27     while (1) {
28
29         write(1, repeated, len);
30
31         write(1, "\n", 1);
32
33         sleep(1);
34     }
35 }
36

```

Feb 02, 12 15:08

minish.c

Page 1/2

```

1  /* By David Mazieres */
2
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <fcntl.h>
8  #include <sys/types.h>
9  #include <sys/wait.h>
10
11 char **av;
12 int avsize;
13
14 void
15 avreserve (int n)
16 {
17     int oldavsize = avsize;
18
19     if (avsize > n + 1)
20         return;
21
22     avsize = 2 * (oldavsize + 1);
23     av = realloc (av, avsize * sizeof (*av));
24     while (oldavsize < avsize)
25         av[oldavsize++] = NULL;
26 }
27
28 void
29 parseline (char *line)
30 {
31     char *a;
32     int n;
33
34     for (n = 0; n < avsize; n++)
35         av[n] = NULL;
36
37     a = strtok (line, "\t\r\n");
38     for (n = 0; a; n++) {
39         avreserve (n);
40         av[n] = a;
41         a = strtok (NULL, "\t\r\n");
42     }
43 }
44
45 void
46 doexec (void)
47 {
48     execvp (av[0], av);
49     perror (av[0]);
50     exit (1);
51 }
52
53 int
54 main (void)
55 {
56     char buf[512];
57     char *line;
58     int pid;
59
60     avreserve (10);
61
62     for (;;) {
63         write (2, "$", 2);
64         if (!(line = fgets (buf, sizeof (buf), stdin))) {
65             write (2, "EOF\n", 4);
66             exit (0);
67         }
68         parseline (line);
69         if (!av[0])
70             continue;
71
72         switch (pid = fork ()) {
73             case -1:

```

Feb 02, 12 15:08

minish.c

Page 2/2

```

74         perror ("fork");
75         break;
76     case 0:
77         doexec ();
78         break;
79     default:
80         waitpid (pid, NULL, 0);
81         break;
82     }
83 }
84 }

```

Feb 02, 12 15:12

redirsh.c

Page 1/2

```

1  /* By David Mazieres */
2
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <fcntl.h>
8  #include <sys/types.h>
9  #include <sys/wait.h>
10
11 char **av;
12 char *infile;
13 char *outfile;
14 char *errfile;
15 int avsize;
16
17 void
18 avreserve (int n)
19 {
20     int oldavsize = avsize;
21
22     if (avsize > n + 1)
23         return;
24
25     avsize = 2 * (oldavsize + 1);
26     av = realloc (av, avsize * sizeof (*av));
27     while (oldavsize < avsize)
28         av[oldavsize++] = NULL;
29 }
30
31 void
32 parseline (char *line)
33 {
34     char *a;
35     int n;
36
37     infile = outfile = errfile = NULL;
38     for (n = 0; n < avsize; n++)
39         av[n] = NULL;
40
41     a = strtok (line, "\\t\\r\\n");
42     for (n = 0; a; n++) {
43         if (a[0] == '<')
44             infile = a[1] ? a + 1 : strtok (NULL, "\\t\\r\\n");
45         else if (a[0] == '>')
46             outfile = a[1] ? a + 1 : strtok (NULL, "\\t\\r\\n");
47         else if (a[0] == '2' && a[1] == '>')
48             errfile = a[2] ? a + 2 : strtok (NULL, "\\t\\r\\n");
49         else {
50             avreserve (n);
51             av[n] = a;
52         }
53         a = strtok (NULL, "\\t\\r\\n");
54     }
55 }
56
57 void
58 doexec (void)
59 {
60     int fd;
61
62     if (infile) {
63         if ((fd = open (infile, O_RDONLY)) < 0) {
64             perror (infile);
65             exit (1);
66         }
67         if (fd != 0) {
68             dup2 (fd, 0);
69             close (fd);
70         }
71     }
72
73     if (outfile) {

```

Feb 02, 12 15:12

redirsh.c

Page 2/2

```

74     if ((fd = open (outfile, O_WRONLY|O_CREAT|O_TRUNC, 0666)) < 0) {
75         perror (outfile);
76         exit (1);
77     }
78     if (fd != 1) {
79         dup2 (fd, 1);
80         close (fd);
81     }
82 }
83
84 if (errfile) {
85     if ((fd = open (errfile, O_WRONLY|O_CREAT|O_TRUNC, 0666)) < 0) {
86         perror (outfile);
87         exit (1);
88     }
89     if (fd != 2) {
90         dup2 (fd, 2);
91         close (fd);
92     }
93 }
94
95 execvp (av[0], av);
96 perror (av[0]);
97 exit (1);
98 }
99
100 int
101 main (void)
102 {
103     char buf[512];
104     char *line;
105     int pid;
106
107     avreserve (10);
108
109     for (;;) {
110         write (2, "$", 2);
111         if (!(line = fgets (buf, sizeof (buf), stdin))) {
112             write (2, "EOF\\n", 4);
113             exit (0);
114         }
115         parseline (line);
116         if (!av[0])
117             continue;
118
119         switch (pid = fork ()) {
120             case -1:
121                 perror ("fork");
122                 break;
123             case 0:
124                 doexec ();
125                 break;
126             default:
127                 waitpid (pid, NULL, 0);
128                 break;
129         }
130     }
131 }

```

Feb 02, 12 15:23

pipesh.c

Page 1/3

```

1  /* By David Mazieres */
2
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <fcntl.h>
8  #include <sys/types.h>
9  #include <sys/wait.h>
10
11 char **av;
12 char *infile;
13 char *outfile;
14 char *errfile;
15 char *outcmd;
16 int avsize;
17
18 void
19 avreserve (int n)
20 {
21     int oldavsize = avsize;
22
23     if (avsize > n + 1)
24         return;
25
26     avsize = 2 * (oldavsize + 1);
27     av = realloc (av, avsize * sizeof (*av));
28     while (oldavsize < avsize)
29         av[oldavsize++] = NULL;
30 }
31
32 void
33 parseline (char *line)
34 {
35     char *a;
36     int n;
37
38     outcmd = infile = outfile = errfile = NULL;
39     for (n = 0; n < avsize; n++)
40         av[n] = NULL;
41
42     a = strtok (line, "\\t\\n");
43     for (n = 0; a; n++) {
44         if (a[0] == '<')
45             infile = a[1] ? a + 1 : strtok (NULL, "\\t\\n");
46         else if (a[0] == '>')
47             outfile = a[1] ? a + 1 : strtok (NULL, "\\t\\n");
48         else if (a[0] == '|') {
49             if (!a[1])
50                 outcmd = strtok (NULL, "");
51             else {
52                 outcmd = a + 1;
53                 a = strtok (NULL, "");
54                 while (a > outcmd && !a[-1])
55                     *--a = ' ';
56             }
57         }
58         else if (a[0] == '2' && a[1] == '>')
59             errfile = a[2] ? a + 2 : strtok (NULL, "\\t\\n");
60         else {
61             avreserve (n);
62             av[n] = a;
63         }
64         a = strtok (NULL, "\\t\\n");
65     }
66 }
67
68 void
69 doexec (void)
70 {
71     int fd;
72     int pipefds[2];
73

```

Feb 02, 12 15:23

pipesh.c

Page 2/3

```

74     while (outcmd) {
75         if (outfile) {
76             fprintf (stderr, "syntax error: > in pipe writer\\n");
77             exit (1);
78         }
79
80         if (pipe (pipefds) < 0) {
81             perror ("pipe");
82             exit (0);
83         }
84
85         switch (fork ()) {
86             case -1:
87                 perror ("fork");
88                 exit (1);
89             case 0:
90                 if (pipefds[1] != 1) {
91                     dup2 (pipefds[1], 1);
92                     close (pipefds[1]);
93                 }
94                 close (pipefds[0]);
95                 outcmd = NULL;
96                 break;
97             default:
98                 if (pipefds[0] != 0) {
99                     dup2 (pipefds[0], 0);
100                    close (pipefds[0]);
101                }
102                close (pipefds[1]);
103                parseline (outcmd);
104                if (infile) {
105                    fprintf (stderr, "syntax error: < in pipe reader\\n");
106                    exit (1);
107                }
108                break;
109            }
110
111         }
112
113         if (infile) {
114             if ((fd = open (infile, O_RDONLY)) < 0) {
115                 perror (infile);
116                 exit (1);
117             }
118             if (fd != 0) {
119                 dup2 (fd, 0);
120                 close (fd);
121             }
122
123         }
124
125         if (outfile) {
126             if ((fd = open (outfile, O_WRONLY|O_CREAT|O_TRUNC, 0666)) < 0) {
127                 perror (outfile);
128                 exit (1);
129             }
130             if (fd != 1) {
131                 dup2 (fd, 1);
132                 close (fd);
133             }
134
135         }
136
137         if (errfile) {
138             if ((fd = open (errfile, O_WRONLY|O_CREAT|O_TRUNC, 0666)) < 0) {
139                 perror (errfile);
140                 exit (1);
141             }
142             if (fd != 2) {
143                 dup2 (fd, 2);
144                 close (fd);
145             }
146
147         }
148
149         execvp (av[0], av);
150         perror (av[0]);
151     }

```

Feb 02, 12 15:23

pipesh.c

Page 3/3

```
147     exit (1);
148 }
149
150 int
151 main (void)
152 {
153     char buf[512];
154     char *line;
155     int pid;
156
157     avreserve (10);
158
159     for (;;) {
160         write (2, "$", 2);
161         if (!(line = fgets (buf, sizeof (buf), stdin))) {
162             write (2, "EOF\n", 4);
163             exit (0);
164         }
165         parseline (line);
166         if (!av[0])
167             continue;
168
169         switch (pid = fork ()) {
170             case -1:
171                 perror ("fork");
172                 break;
173             case 0:
174                 doexec ();
175                 break;
176             default:
177                 waitpid (pid, NULL, 0);
178                 break;
179         }
180     }
181 }
```