PSET 3

Jonathan Lam
Prof. Hakner
ECE 357
Operating Systems
10/23/19

Problem 1: Shell Script Invocation

1. The child of the interactive shell execs /bin/sh (from the shebang).

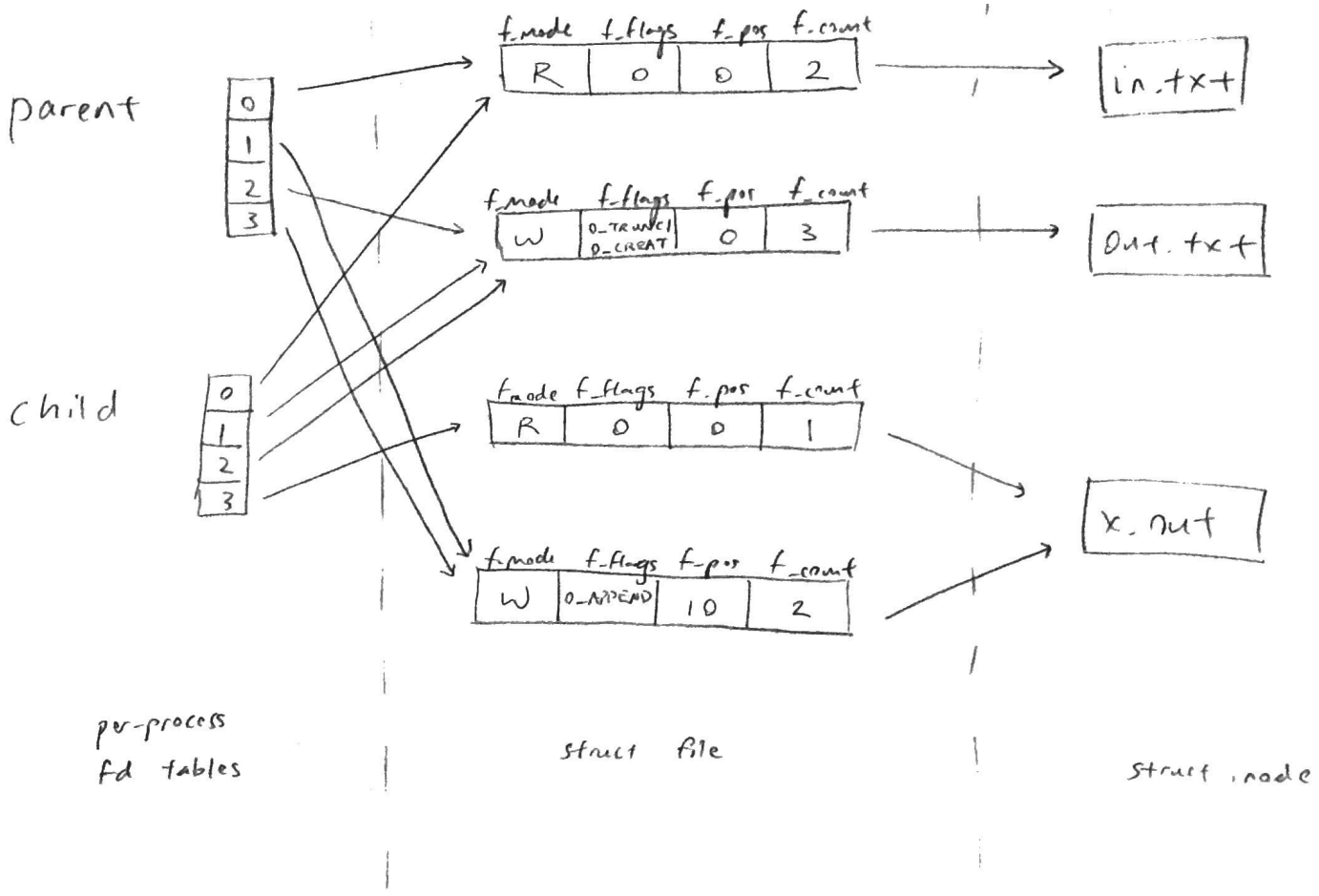
2. argc: 5
   argv: {“sh”, “./script.sh”, “f2.c”, “f3.c”, “f4.c”, NULL}

The sh shell expands “f[2-4].c” into separate arguments if the files exist, which they do. The argv is always NULL-terminated. It also seems that argv[1] is not the fully qualified path (as the lecture notes indicate), but the path of the script passed to the command.

3. Calls one of the wait syscalls to wait for the child to finish executing, and to get its return code.

4. Looking at man 1 ls, ls returns 1 for minor errors (such as not being able to access subdirs) and 2 for major errors (such as not being able to access the command-line argument). Since foobar is unreachable, this falls into the second category, so 2 is printed to the terminal.

Problem 2: File Descriptor Tables



Problem 3: Simple Shell Program

```
#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/wait.h>
#include <unistd.h>

#define SHELL "jsh"
#define PROMPT SHELL "$ "

#define ERR_FAT(prog, op, ctx, msg) {\
    dprintf(2, "%s: ERROR: %s \"%s\": %s\n", prog, op, ctx, msg);\
    exit(EXIT_FAILURE);\
  }
#define WARN(prog, op, ctx, msg)\
  dprintf(2, "%s: %s \"%s\": %s\n", prog, op, ctx, msg)

// structure parsed line
struct rd_out {
  // flag: 1 for append, 0 for trunc
  int append;
  char *file;
};
struct cmd_parse {
  char *cmd, **argv, *rd_in;
  struct rd_out rd_out, rd_err;
  int argc;
};

// global last status
int laststatus = 0;

// manage i/o redirection in file
void io_rd(char *file, int flags, int fd, char *rd_stream) {
  int rd_fd;
  char warn[40];
  if((rd_fd = open(file, flags, 0666)) < 0)
    sprintf(warn, "i/o redirection of (open) to %s", rd_stream);
  else if(dup2(rd_fd, fd) < 0)
    sprintf(warn, "i/o redirection of (dup2) to %s", rd_stream);
  else if(close(rd_fd) < 0)
    sprintf(warn, "unclean fd environment from i/o redirection (close) to %s",
            rd_stream);
  else
    return;
  WARN(SHELL, warn, file, strerror(errno));
  exit(1);
}

// handling line parsing; cmd_src is the source of cmds (stdin or interpreter)
void parse_line(char *line, FILE *cmd_src) {
```

```
char *token, path_buf[4097];
int argc_cap = 4, rd_fd;
struct cmd_parse cmd_parse = {
  .cmd = NULL,
  .argv = (char **) malloc(argc_cap * sizeof(char *)),
  .rd_in = NULL,
  .rd_out = { 0, NULL },
  .rd_err = { 0, NULL },
  .argc = 1
};
struct rusage rusage;
pid_t cpid;
struct timeval cp_start, cp_end;
int wstatus;

// very basic comments: ignore lines starting with "#"
if(*line == '#')
  return;

// very basic tokenizing by whitespace
token = strtok(line, " \t\n");
if(!token)
  return;

cmd_parse.cmd = token;
cmd_parse.argv[0] = token;
while(token = strtok(NULL, " \t\n")) {
  if(*token == '>') {
    cmd_parse.rd_out.append = *(token+1) == '>';
    cmd_parse.rd_out.file = token+1+cmd_parse.rd_out.append;
  } else if(*token == '2' && *(token+1) == '>') {
    cmd_parse.rd_err.append = *(token+2) == '>';
    cmd_parse.rd_err.file = token+2+cmd_parse.rd_err.append;
  } else if(*token == '<') {
    cmd_parse.rd_in = token+1;
  } else {
    if(cmd_parse.argc == argc_cap)
      if(!(cmd_parse.argv = (char **)
           realloc(cmd_parse.argv, (argc_cap*=2) * sizeof(char *))))
        WARN(SHELL, "allocating memory for argument parsing (realloc)",
             token, strerror(errno));
    cmd_parse.argv[cmd_parse.argc++] = token;
  }
}
// terminate argv with np
if(cmd_parse.argc == argc_cap)
  if(!(cmd_parse.argv = (char **)
       realloc(cmd_parse.argv, (argc_cap+1) * sizeof(char *))))
    WARN(SHELL, "allocating memory for argument parsing (realloc)",
         "end token (NULL)", strerror(errno));
cmd_parse.argv[cmd_parse.argc] = NULL;

// shell built-ins
if(!strcmp(cmd_parse.cmd, "pwd")) {
  if(!getcwd(path_buf, 4097)) {
    WARN("pwd", "getcwd", "", strerror(errno));
    laststatus = errno;
```

```
  } else
    dprintf(1, "%s\n", path_buf);
} else if(!strcmp(cmd_parse.cmd, "cd")) {
  if(chdir(cmd_parse.argc == 1 ? getenv("HOME") : cmd_parse.argv[1]) < 0) {
    WARN("cd", "chdir", cmd_parse.argc == 1 ? "" : cmd_parse.argv[1],
         strerror(errno));
    laststatus = errno;
  }
} else if(!strcmp(cmd_parse.cmd, "exit")) {
  // if invalid error code, return 2
  // bash does this; see https://askubuntu.com/a/892605/433872
  if(cmd_parse.argc > 1)
    for(char *c = cmd_parse.argv[1]; *c; c++)
      if(!(isdigit(*c) || (*c == '-' && c == cmd_parse.argv[1]))) {
        WARN(SHELL, "exit", cmd_parse.argv[1], "Numeric argument required");
        exit(2);
      }
  exit(cmd_parse.argc > 1 ? atoi(cmd_parse.argv[1]) : laststatus);
}

// fork, exec other programs
else {
  gettimeofday(&cp_start, NULL);
  switch(cpid = fork()) {
    case -1:
      WARN(SHELL, cmd_parse.cmd, "fork", strerror(errno));
      break;

    // child
    case 0:
      // if not interactive mode, close interpreted script fd
      if(cmd_src != stdin) {
        if(fclose(cmd_int)) {
          WARN(SHELL, "closing script fd to initiate clean child fd env",
               "close", strerror(errno));
        }
      }

      // i/o redirection
      if(cmd_parse.rd_in)
        io_rd(cmd_parse.rd_in, O_RDONLY, 0, "standard input");
      if(cmd_parse.rd_out.file)
        io_rd(cmd_parse.rd_out.file,
              O_WRONLY|O_CREAT|(cmd_parse.rd_out.append?O_APPEND:O_TRUNC),
              1, "standard output");
      if(cmd_parse.rd_err.file)
        io_rd(cmd_parse.rd_err.file,
              O_WRONLY|O_CREAT|(cmd_parse.rd_err.append?O_APPEND:O_TRUNC),
              2, "standard error");

      // exec; if unsuccessful, following lines to report error
      execvp(cmd_parse.cmd, cmd_parse.argv);
      WARN(SHELL, "exec", cmd_parse.cmd, strerror(errno));
      exit(127);

    // parent
    default:
```

```
        wait4(cpid, &wstatus, 0, &rusage);
        // return status from exit (laststatus) is the return value if normally
        // exited, and the whole status value if terminated with signal
        // (same behavior as bash)
        laststatus = WIFSIGNALED(wstatus) ? wstatus : WEXITSTATUS(wstatus);
        gettimeofday(&cp_end, NULL);
        dprintf(2, "%s: Child process %d exited ", SHELL, cpid);
        if(wstatus && !WIFSIGNALED(wstatus))
          dprintf(2, "with return value %d\n", WEXITSTATUS(wstatus));
        else if(wstatus)
          dprintf(2, "with signal %d (%s)\n",
                  WTERMSIG(wstatus), strsignal(WTERMSIG(wstatus)));
        else
          dprintf(2, "normally\n");
        dprintf(2, "%s: Real: %fs User: %fs Sys: %fs\n",
                SHELL,
                cp_end.tv_sec-cp_start.tv_sec+(cp_end.tv_usec-cp_start.tv_usec)
                  /1e6,
                rusage.ru_utime.tv_sec+rusage.ru_utime.tv_usec/1e6,
                rusage.ru_stime.tv_sec+rusage.ru_stime.tv_usec/1e6);
        free(cmd_parse.argv);
    }
  }
}

// driver function: handle interpreted scripts and start parse loop
int main(int argc, char **argv) {
  char *line_buf = NULL;
  size_t line_len = 0;
  FILE *cmd_in = stdin;

  // open command inputs as fd 3; will be closed to children after forking
  // expects first argument to be a fname since no other args are defined;
  // this is the same behavior as bash for non-option arguments
  if(argc > 1) {
    if(!(cmd_in = fopen(argv[1], "r")))
      ERR_FAT(SHELL, "Opening interpreter file", argv[1], strerror(errno));
  }

  // read, parse, execute command
  // prints out prompt if not reading from script file
  errno = 0;
  while(cmd_in == stdin && dprintf(1, PROMPT),
        getline(&line_buf, &line_len, cmd_in) != -1)
    parse_line(line_buf, cmd_in);
  if(errno)
    ERR_FAT(SHELL, "Reading line (getline)", line_buf, strerror(errno));

  // cleanup and exit
  // this will only be called if EOF from interpreter
  free(line_buf);
  dprintf(1, "\nEOF read, exiting shell with exit code %d\n", laststatus);
  exit(EXIT_SUCCESS);
}
```

Test cases

```
(base) [jon@archijon prog3]$ ./jsh
jsh$ # testing builtins
jsh$ cd
jsh$ pwd
/home/jon
jsh$ cd Documents/coursework/ece^C
(base) [jon@archijon prog3]$ ./jsh
jsh$ # testing builtins
jsh$ pwd
/home/jon/Documents/coursework/ece357/hw/programs/prog3
jsh$ cd
jsh$ pwd
/home/jon
jsh$ cd Documents/coursework/ece357/hw/programs/prog3/././/////.
jsh$ pwd
/home/jon/Documents/coursework/ece357/hw/programs/prog3
jsh$ exit 52
(base) [jon@archijon prog3]$ echo $?
52
(base) [jon@archijon prog3]$ ./jsh
jsh$ # test redirection
jsh$ ls -l >ls.out
jsh: Child process 8009 exited normally
jsh: Real: 0.014720s User: 0.013427s Sys: 0.000000s
jsh$ cat ls.out
total 68
-rw-r--r-- 1 jon jon    36 Oct 23 13:50 input.txt
-rwxr-xr-x 1 jon jon 17992 Oct 23 13:41 jsh
-rw-r--r-- 1 jon jon  7326 Oct 23 13:41 jsh.c
-rw-r--r-- 1 jon jon     0 Oct 23 14:30 ls.out
-rwxr-xr-x 1 jon jon 16536 Oct 12 02:24 sigsegv
-rw-r--r-- 1 jon jon    54 Oct 12 02:20 sigsegv.c
-rwxr-xr-x 1 jon jon    27 Oct 23 13:49 test2.jsh
-rw-r--r-- 1 jon jon   920 Oct 23 14:30 testcommands
-rwxr-xr-x 1 jon jon    42 Oct 23 13:48 testme.jsh
jsh: Child process 8010 exited normally
jsh: Real: 0.003175s User: 0.002788s Sys: 0.000000s
jsh$ ../prog2/rls / -v 2>rls.err >rls.out
jsh: Child process 8013 exited normally
jsh: Real: 33.907177s User: 8.273273s Sys: 12.156793s
jsh$ tail rls.err
rls: opening directory "/opt/anaconda/.cph_tmpxk03b1af": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmpwh81ttt3": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmpsdh_1e38": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmp5p78hp0x": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmpwvqtbno8": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmpmnzy858x": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmpxpla5b29": Permission denied
rls: opening directory "/opt/anaconda/.cph_tmpeb0kxuah": Permission denied
rls: mount point "/dev": Not crossing mount point
rls: opening directory
"/home/jon/Documents/coursework/ece357/hw/playground/testdir": Permission denied
jsh: Child process 8032 exited normally
jsh: Real: 0.002997s User: 0.000000s Sys: 0.002584s
jsh$ tail rls.out
```

```
 13406311    293 -rw-r--r--   1 jon      jon        296213 Oct 13 18:25
/home/jon/Downloads/diff guide.pdf
 13413185    309 -rw-r--r--   1 jon      jon        314946 Oct  5 13:05
/home/jon/Downloads/Studio Ghibli Medley (Animenz)_v2 (1).pdf
 13404302    133 -rw-r--r--   1 jon      jon        134675 Sep 16 10:11
/home/jon/Downloads/lr_template (1).pdf
 13418131    613 -rw-r--r--   1 jon      jon        624375 Sep 25 12:45
/home/jon/Downloads/richard-bib-out.txt
  6034852      5 drwxr-xr-x   2 root     root         4096 Oct  2 14:21 /home/alice
  2621441      5 drwxr-xr-x   4 root     root         4096 Jul 30 08:50 /srv
  2621443      5 drwxr-xr-x   2 root     root         4096 Oct  6 13:06 /srv/http
  2621442      5 dr-xr-xr-x   2 root     ftp          4096 May 23 10:18 /srv/ftp
       14      0 lrwxrwxrwx   1 root     root            7 Oct  6 11:44 /lib64 ->
usr/lib
       15      0 lrwxrwxrwx   1 root     root            7 Oct  6 11:44 /sbin ->
usr/bin
jsh: Child process 8035 exited normally
jsh: Real: 0.002904s User: 0.002589s Sys: 0.000000s
jsh$ echo helloworld >>rls.out
jsh: Child process 8038 exited normally
jsh: Real: 0.002749s User: 0.002341s Sys: 0.000000s
jsh$ tail rls.out
 13413185    309 -rw-r--r--   1 jon      jon        314946 Oct  5 13:05
/home/jon/Downloads/Studio Ghibli Medley (Animenz)_v2 (1).pdf
 13404302    133 -rw-r--r--   1 jon      jon        134675 Sep 16 10:11
/home/jon/Downloads/lr_template (1).pdf
 13418131    613 -rw-r--r--   1 jon      jon        624375 Sep 25 12:45
/home/jon/Downloads/richard-bib-out.txt
  6034852      5 drwxr-xr-x   2 root     root         4096 Oct  2 14:21 /home/alice
  2621441      5 drwxr-xr-x   4 root     root         4096 Jul 30 08:50 /srv
  2621443      5 drwxr-xr-x   2 root     root         4096 Oct  6 13:06 /srv/http
  2621442      5 dr-xr-xr-x   2 root     ftp          4096 May 23 10:18 /srv/ftp
       14      0 lrwxrwxrwx   1 root     root            7 Oct  6 11:44 /lib64 ->
usr/lib
       15      0 lrwxrwxrwx   1 root     root            7 Oct  6 11:44 /sbin ->
usr/bin
helloworld
jsh: Child process 8041 exited normally
jsh: Real: 0.002795s User: 0.002467s Sys: 0.000000s
jsh$
jsh$ # test error checking
jsh$ ls amskdlmaskdm
ls: cannot access 'amskdlmaskdm': No such file or directory
jsh: Child process 8043 exited with return value 2
jsh: Real: 0.003643s User: 0.000000s Sys: 0.003322s
jsh$ cd askdlmasldm
cd: chdir "askdlmasldm": No such file or directory
jsh$ ./sigsegv
jsh: Child process 8050 exited with signal 11 (Segmentation fault)
jsh: Real: 0.220205s User: 0.000000s Sys: 0.001683s
jsh$
EOF read, exiting shell with exit code 139
(base) [jon@archijon prog3]$ echo $?
139
(base) [jon@archijon prog3]$ cat testme.jsh
#!./jsh
cat >cat.out
```

```
cat cat.out
exit 123
(base) [jon@archijon prog3]$ ./testme.jsh
hello, world!
here is some text
1
1 1
1 1 1
1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
jsh: Child process 8288 exited normally
jsh: Real: 23.773219s User: 0.000000s Sys: 0.003095s
hello, world!
here is some text
1
1 1
1 1 1
1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
jsh: Child process 8298 exited normally
jsh: Real: 0.002143s User: 0.001872s Sys: 0.000000s
(base) [jon@archijon prog3]$ echo $?
123
(base) [jon@archijon prog3]$ cat test2.jsh
#!./jsh
cat >cat2.out
exit
(base) [jon@archijon prog3]$ cat input.txt
Hello, world!
This is in input.txt!
(base) [jon@archijon prog3]$ ./test2.jsh <input.txt
jsh: Child process 8316 exited normally
jsh: Real: 0.004048s User: 0.002154s Sys: 0.000000s
(base) [jon@archijon prog3]$ echo $?
0
(base) [jon@archijon prog3]$ cat cat2.out
Hello, world!
This is in input.txt!
```