CSCI 3150 Tutorial

Assignment One - Part II

HUANG Qun

SHB 120

qhuang@cse.cuhk.edu.hk

24/9/2012 – 28/9/2012

Outline

Phase 1

- Continuing on the Parser
- Build-in Commands: cd, exit

Phase 2

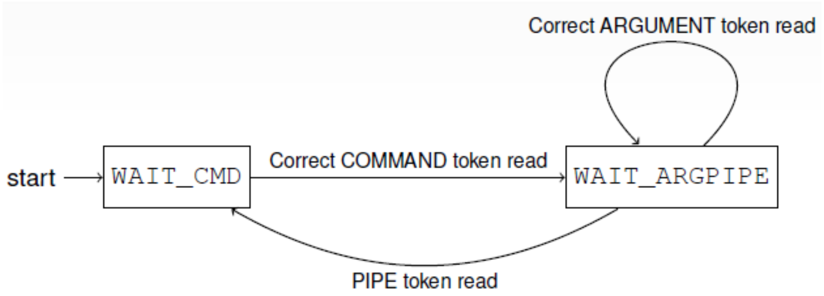
- Simple Shell: execute programs with arguments
- Error Handling

Another Example of FSM

- Suppose valid input is of the form:
  Command [Arguments] | Command [Arguments] . . .
- There is a set of illegal characters
- We can define 2 states
  - Waiting for command: next token read should be a command (WAIT_CMD)
  - Waiting for argument/pipe (WAIT_ARGPIPE)
- We start with waiting for command (initial state)
- State changes invoked by reading and processing a token

Writing A Parser

Another Example of FSM

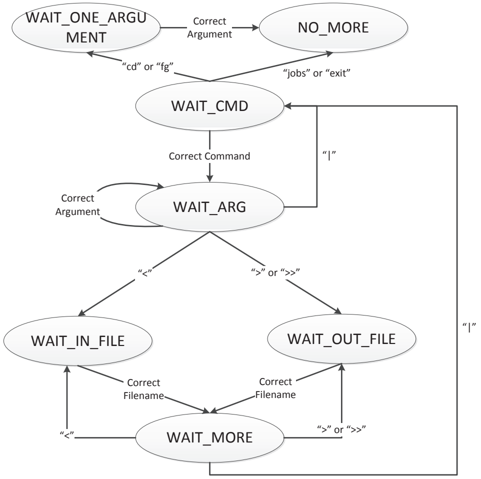


Writing a Parser

Sample Code

```
enum STATES {WAIT_CMD, WAIT_ARGPIPE};
enum STATES state = WAIT_CMD;
while (/* there is a next token to read */) {
    switch ((int)state) {
        case (WAIT_CMD):
        /* Check for illegal character (return 0 on
            error?)
        * Assign type COMMAND to token
        * ...
        * state = WAIT_ARGPIPE; */
        break;
        case (WAIT_ARGPIPE):
        // ...
        break;
        default:
        // ...
    }
}
```

A More Detailed FSM



Writing a Parser

Some Tips with This FSM

Single input source, single output channel

- invalid input: cat | cat < input.txt
- solution: use variables to record the number of input/ouput redirection seen

The FSM should not end up with some states

- invalid input: cat >
- solution: when the FSM stops, check the end state

Of course, you are encouraged to design your own FSM.

Remaining of Phase 1

Shell Prompt, cd, exit

- Simply getcwd(), chdir() and exit()
- Remember that *man page* is useful
- You can handle it :)

```
// getcwd - get current working directory
char *getcwd(char *buf, size_t size);

// chdir - change working directory
int chdir(const char *path);

// exit - process termination
void exit(int status);
```

Remaining of Phase 1

Marking Procedures

For Phase One

- Generate some (random) input, and
- Write a script to feed the input to your parser and compare
- So please follow the EXACT output format (All separated by single spaces)

```
Token %d: "%s" (%s)\n
```

For Phase Two

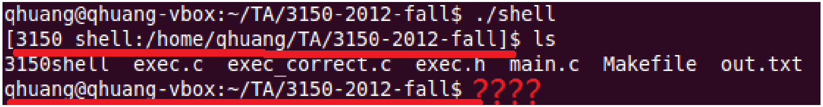
- Manual checking, (you) type in the test case commands letter by letter during the demonstration

Execute Programs with Arguments

Move on to Phase 2

- Start with a simple shell first
- Just creating a child process and executing one program with arguments

Why *fork* is Important???



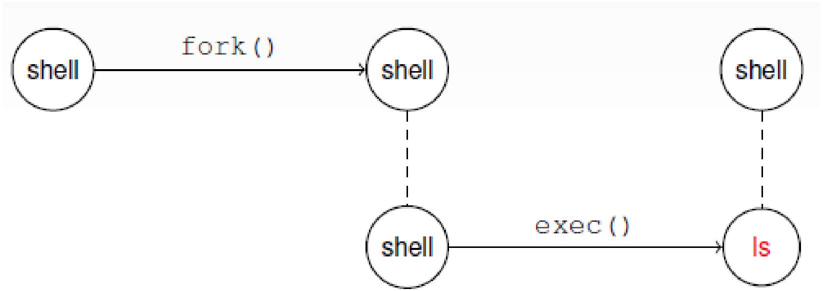
```
qhuang@qhuang-vbox:~/TA/3150-2012-fall$ ./shell
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls
3150shell  exec.c  exec_correct.c  exec.h  main.c  Makefile  out.txt
qhuang@qhuang-vbox:~/TA/3150-2012-fall$
```

Where's our shell???

Execute Programs with Arguments

Execution Flow

But fork is not enough, because we want something new, so ...



Execute Programs with Arguments

The *exec*()* Family

exec() family

- execl(const char *path, const char *arg, ...);
- execlp(const char *file, const char *arg, ...);
- execle(const char *path, const char *arg, ..., char *const envp[]);
- execv(const char *path, char *const argv[]);
- execvp(const char *file, char *const argv[]);
- execve(const char *file, char *const argv[], ..., char *const envp[]);

Differences

l	takes in argument list (arg1, arg2, . . . , NULL)
v	takes in pointer to array of strings (last in array must be NULL)
p	file - command supplied need not be absolute path
e	pass in environment variables as well

Execute Programs with Arguments

Which One to Choose?

- Arguments may be supplied as a list or an array of strings
- The programs may be absolute path or relative path
- So both *execvp* or *execlp* may be OK.

Important!

- The first argument must be the name of the program executed
- The strings must be either literals or with memory allocated
- Read the example in Lecture for details

```
args = (char **)malloc(3 * sizeof(char *));
args[0] = "ls"; args[1] = "-al"; args[2] = NULL;
execvp("ls", args);
// error handling because execvp should not return
```

Execute Programs with Arguments

Search Programs

- Environment variable: PATH
  PATH=/bin:/bin/usr:.
- How to put our PATH into the environment variables

```
// setenv - change or add an environment variable
int setenv(const char *name,
    const char *value, int overwrite);
```

Execute Programs with Arguments

Wait for Child Process

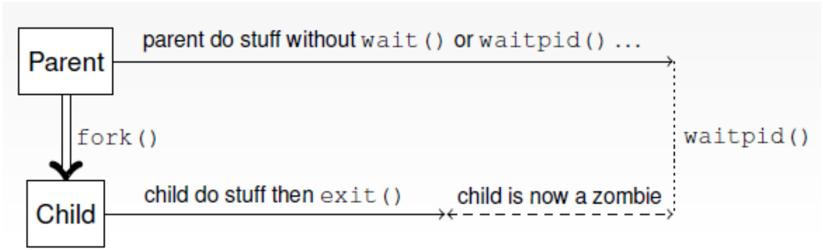
Parent should wait for child to suspend/terminate

- Lookup waitpid()

```
// waitpid - wait for process to change state
pid_t waitpid(pid_t pid, int *status, int options);
```

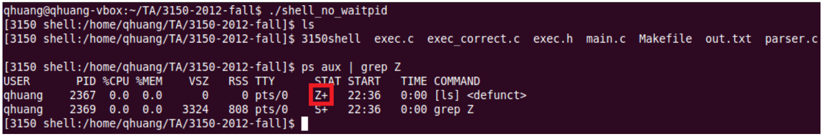
Otherwise...

- when a child process terminates, it becomes a zombie.


Parent
parent do stuff without wait() or waitpid() ...
fork()
waitpid()
Child
child do stuff then exit()
child is now a zombie

Execute Programs with Arguments

Detect and Killing Zombies



```
qhuang@qhuang-vbox:~/TA/3150-2012-fall$ ./shell_no_waitpid
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ 3150shell  exec.c  exec_correct.c  exec.h  main.c  Makefile  out.txt  parser.c

[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ps aux | grep Z
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
qhuang    2367  0.0  0.0      0     0 pts/0    Z+   22:36   0:00 [ls] <defunct>
qhuang    2369  0.0  0.0   3324   808 pts/0    S+   22:36   0:00 grep Z
[3150 shell:/home/qhuang/TA/3150-2012-fall]$
```

use **ps aux** | **grep Z** to detect,
the processes with state **Z+** are zombies

- No explicit handling (if you have use *waitpid()* correctly)
- waitpid() will take care of them
- Please read the man page of waitpid()!!
- Cost 1% course mark for roaming zombies (that is, IF your process execution order is right . . .)

Execute Programs with Arguments

Error Handling

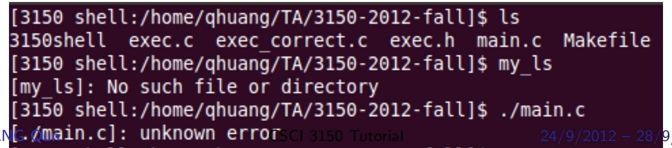
- fork(), exec*(), waitpid() ... all have return values
- Be sure to CHECK them, dont assume everything is OK
- When error occurs in a library/system call, variable errno[1] is set accordingly
- You can read from errno to determine what the error was, and
- Print the required error messages (according to the errno) to stderr
- The meaning of errno values for each function are described in the *man page*, take a look at them!
- Read the Specification carefully so that you will not miss any requirements on the error messages.

[1] you should #include<errno.h> to use the variable

Execute Programs with Arguments

An Example of *errno* variable for *execvp*

```
// set up char* command, char** arguments
if (execvp(command, arguments)) {
    if (errno == ENOENT) {
        fprintf(stderr,
            "[%s]: No such file or directory\n",
                arguments[0]);
        // others
    }
    else {
        fprintf(stderr,
            "[%s]: unknown error\n", arguments[0]);
        // others
    }
}
```



```
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls
3150shell  exec.c  exec_correct.c  exec.h  main.c  Makefile
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ my_ls
[my_ls]: No such file or directory
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ./main.c
[./main.c]: unknown error
```