CSCI 3150 Tutorial

Assignment One - Part I

HUANG Qun

SHB 120

qhuang@cse.cuhk.edu.hk

17/9/2012 – 21/9/2012

Outline

- Compiling a Program
- Introduction to Assignment One
- Writing a Parser

Compiling a Program

Preparing Your Source Code

- Our assignments must be completed on Linux (Ubuntu 12.04)
- On Linux we have editors like vim, emacs, nano . . .
- An easy (but troublesome) way would be to write your codes in Windows (notepad, Visual Studio whatever) and upload it to the Linux virtual machine

Compiling a Program

Using GCC

We have the source code ready.
We have to compile it into a program now.
For compilation on Linux, we use *gcc*.

Example

gcc -Wall -o 3150shell shell.c parser.c main.c

- gcc: the compiler
- -Wall: shows all warnings
- -o 3150shell: the complied executable file will be named as *3150shell*
- *.c: the source code files

Why is -**Wall** Important?

C Hazard #1: using return value of = Exercise

- Our hazard is **NOT** about **rand()**, but is about using functions which return a number in general.

rand_3.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(void) {
6    int result;
7    srand(time(NULL));
8    if( result = rand() % 2 == 0 )
9      printf("%d is even.\n", result);
10   else
11     printf("%d is odd.\n", result);
12
13   return 0;
14 }
```

This program is to tell whether a random number is odd or even.

Ready?

```
$ gcc rand_3.c -o rand_3
$ ./rand_3
0 is odd.
$ ./rand_3
1 is even.
$
```

OMG!!

78

The -**Wall** would help you detect such bugs.

Page 78, "Summer Preparatory Course on C - Day 1" by Dr. Wong

Introduction to Assignment One

What to Do?

A simple but functional shell, with command-line interface (CLI)

Example

```
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls -l
total 96
-rwxr-xr-x 1 qhuang qhuang 18117 2012-09-11 19:31 3150shell
-rw-r--r-- 1 qhuang qhuang  9106 2012-09-11 19:31 exec.c
-rw-r--r-- 1 qhuang qhuang    82 2012-09-07 16:17 exec.h
-rw-r--r-- 1 qhuang qhuang  1213 2012-09-11 19:31 main.c
-rw-r--r-- 1 qhuang qhuang   195 2012-09-07 19:10 Makefile
-rwx------ 1 qhuang qhuang    43 2012-09-07 20:12 out.txt
-rw-r--r-- 1 qhuang qhuang  5419 2012-09-07 19:10 parser.c
-rw-r--r-- 1 qhuang qhuang   398 2012-09-07 19:09 parser.h
-rwxr-xr-x 1 qhuang qhuang 18121 2012-09-07 19:58 shell
-rw-r--r-- 1 qhuang qhuang  9547 2012-09-07 11:25 tags
drwxr-xr-x 2 qhuang qhuang  4096 2012-09-07 11:30 tests
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ cat main.c | grep main
int main(int argc, char** argv) {
[3150 shell:/home/qhuang/TA/3150-2012-fall]$
```

Introduction to Assignment One

Overview

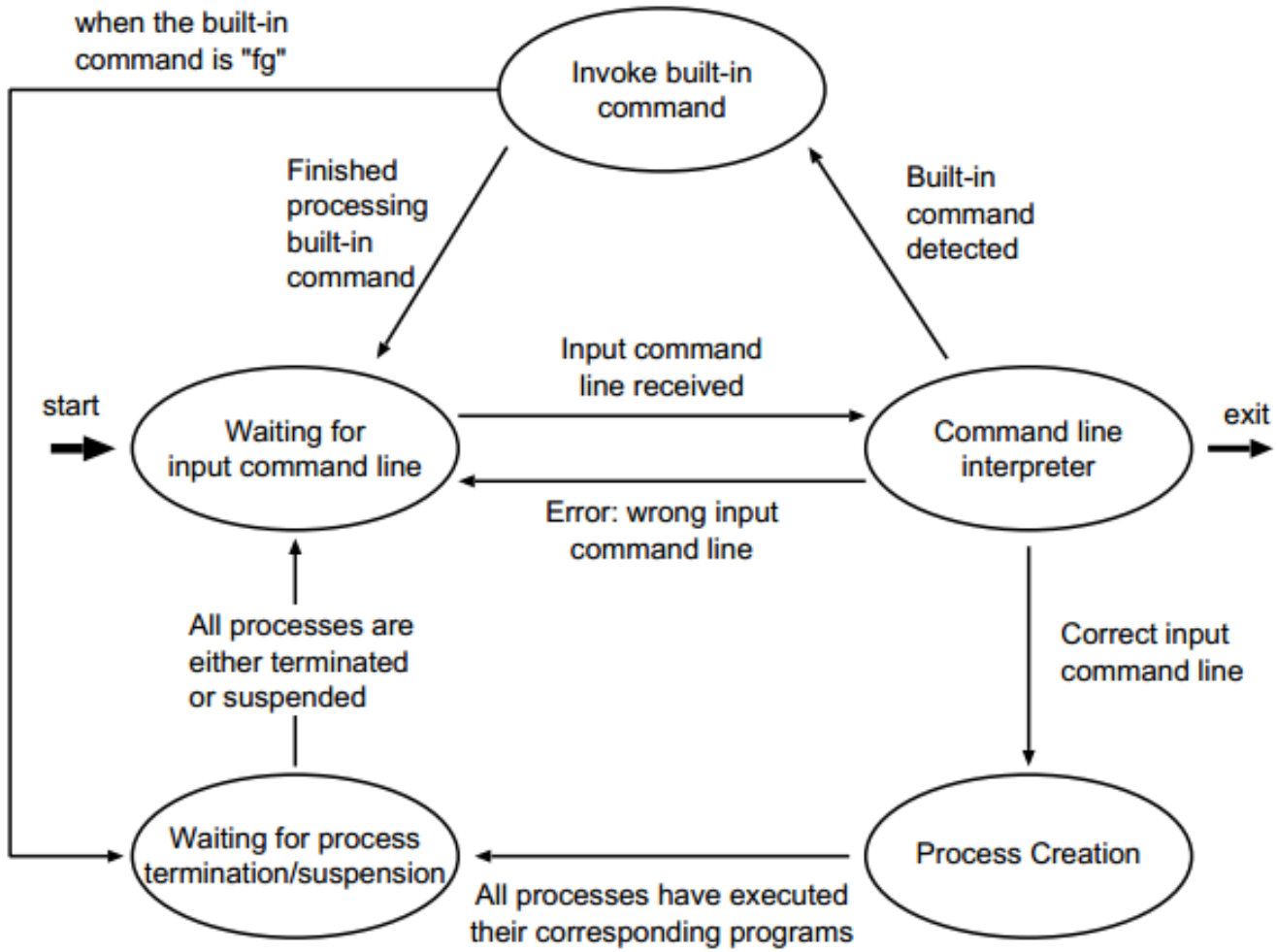


Figure 1: The shell's execution flow.

Introduction to Assignment One

Your Shell Should Be Able to

- A Parser to Accept and interpret user input
- Provide build-in commands. $\leftarrow$ Phase 1 [1]
- Execute programs with arguments
- More... $\leftarrow$ Phase 2

[1] But in Phase 1, only *cd*, *exit* are required.

Writing a Parser

Objectives

- Accept and interpret user input
  - Check against some predefined grammar
  - Each word is assigned a type
  - Output all the words and their types
- Phase 2 builds on this
- Dont worry about how to interact with the OS . . . yet, let's complete step by step :)

Writing a Parser

Types of Words

Built-in	Command Built-in command to execute
Command	Name Program to execute
Argument	Arguments supplied to command execution
Pipe	|
Redirect Input	<
Redirect Output	>, >>
Input Filename	Filename for input redirection
Output Filename	Filename for output redirection

Writing a Parser

Grammars

Read the specification for details.
Some tips on the grammar:

- Command [Arguments] [Redirections]
- Command [Arguments] [Redirection] | Command [Arguments] | Command [Arguments] [Redirection]
- Arguments (if any) follow a command immediately
- Input Filename (or Output Filename) follow $<$ (or $>$, $>>$) immediately
- A command must either at the begin or after |
- Single source of input, and single output channel
- Commands, Arguments and Filenames have strings but cannot contain spaces $> < | * !$ "

User Input Examples

Valid input

- ls -al
- cat | cat | cat
- cat < inFile > outFile
- cat > outFile < inFile

Invalid input

- cat > outFile | cat < inFile
- ls < inFile -al
- cat |
- ! < * > "|

Writing a Parser

Implementation Issues

To check the input grammar

- Read the input line
- Break the line into words (tokenize)
- Check the words sequentially
  - check illegal characters
  - check grammar with a finite state machine (FSM)

Writing a Parser

Tokenize

Any easy way to break the line up into words (tokens)?
Useful function:

```
// strtok() - extract tokens from strings
char *strtok(char *str, const char *delim);
```

Two things to keep in mind

- str will be modified by strtok(), pass in a copy of the original string if you want to keep it
- On subsequent calls to strtok(), supply NULL as str instead

Read the man page (e.g. man strtok)

Writing a Parser

Example

```
gets(cmdLine);
strcpy(tmp, cmdLine);
if (retval = strtok(tmp, " ")) {
    strcpy(tokens[i++], retval);
    while (retval = strtok(NULL, " ")) {
        strcpy(tokens[i++], retval);
    }
}
for (j=0; j<i; printf("%s\n", tokens[j++]));
```

```
$ ./tokenizer
 Hi   Woooorld, Bye !!
Hi
Woooorld,
Bye
!!
$ _
```

Detect Illegal Characters

Useful function:

```
// strcspn() - search a string for a set of characters
size_t strcspn(const char* str1, const char* str2);
```

- Scans str1 for the first occurrence of any of the characters that are part of str2
- returning the number of characters of str1 read before this first occurrence.

Detecting by comparing the return value and the length of the string (use strlen() function)

Example

```
char str[] = "fcba73";
char keys[] = "1234567890";
int i;
i = strcspn (str,keys);
printf ("The first number in str is at position %d.\n"
    ,i);

int length;
length = strlen(str);
if (i < length)
        printf("str contains characters in keys.\n");
```

```
The first number in str is at position 4.
str contains characters in keys.
```

Definition of FSM

Something that will read your input token by token

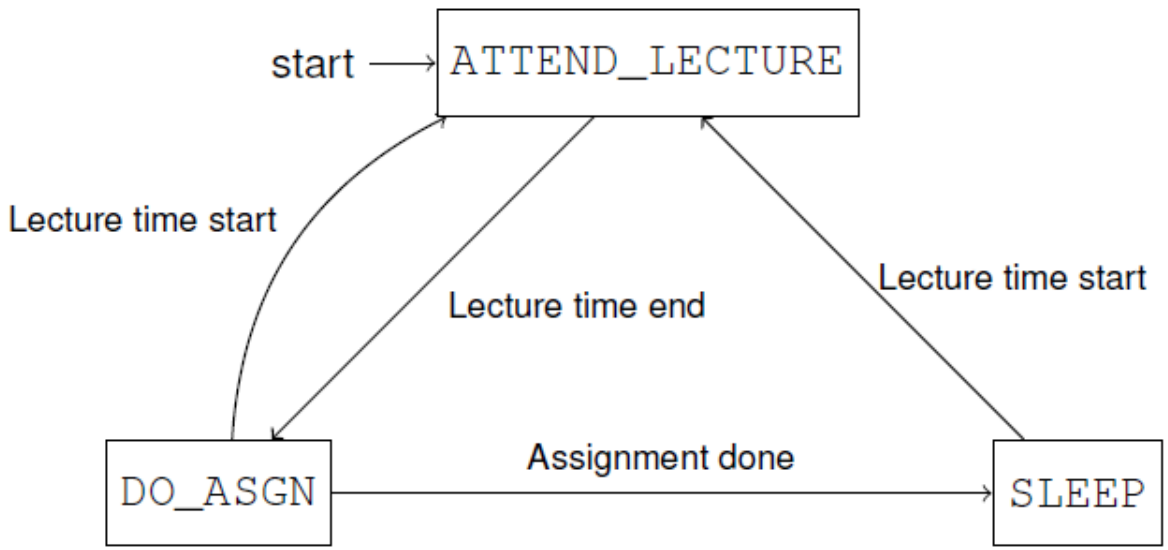
- consisting of some predefined states
- each time, the FSM is residing in exact one of the states
- on reading each token, carry out some actions depending on its current state
- Some actions could mean updating variables, deciding which state to jump to next

An Example of FSM

Example

- Suppose your life (a FSM) has only 3 states
  - Doing 3150 assignments (*DO_ASGN*)
  - Attending Moles lectures (*ATTEND_LECTURE*)
  - Sleeping (*SLEEP*)
- You start with attending lectures (initial state)
- State changes invoked by completion of assignments and time

An Example of FSM



Example

```
enum STATES {DO_ASGN, ATTEND_LECTURE, SLEEP};
enum STATES state = ATTEND_LECTURE;
while (1) {
    switch ((int)state) {
        case (DO_ASGN):
        if (lecture time starts)
            state = ATTEND_LECTURE;
        else if (assignment completes)
            state = SLEEP;
        break;
        case (ATTEND_LECTURE):
        if (lecture time ends)
            state = DO_ASGN;
        break;
        case (SLEEP):
        if (lecture time starts)
            state = ATTEND_LECTURE;
        break;
        default:
        // ...
    }
}
```

Design Your Own FSM

For your parser, think about

- What are the states?
- What are the actions in each state?
  - Which token type(s) do you expect, is the incoming token right?
  - What variables to keep and how will they be updated?
  - How/When to decide which state to jump to next?

Summary

If You Want to Learn More

- Wikipedia - Editor War
  http://en.wikipedia.org/wiki/Editor_war
- Wikipedia - Shell (Computing)
  http://en.wikipedia.org/wiki/Shell_(computing)
- A site that I visit for man page
  http://linux.die.net/