2013.2 LINUX System Programming Assignment

Project 3 : MiniShell 2 Implementation (Due : Dec. 9th, 11:59pm)

- **System** : LINUX or CYGWIN
- **Language** : C or C++
- **Submission** : File Upload (source codes including Makefile and README.txt) through eClass
- **What to submit ?**
  - Source code
    - Create at least three .c or .cpp source files, and add "`Makefile`"
    - README.txt : describe compilation environment (cygwin? or Ubuntu?) and how to execute
    - Make a directory whose name should be exactly your student_id# and put all source files and Makefile into the directory.
    - use `tar` and `gzip` command to compress the above directory and generate .tar.gz file
      (example) tar cvf 20113051.tar dir_name
      gzip 20113051.tar (.gz file은 gzip -d 로 압축을 풀 수 있음)
    - Submit the `.tar.gz` file to eClass page
- **Execution**
  - After compilation with make, MiniShell.exe should be created.
  - Execute MiniShell in a LINUX(ubuntu) or CYGWIN shell prompt.
  - MiniShell should print a shell prompt line and wait for a user input.
  - When user input is received, MiniShell interpret and process the input.
  - While MiniShell is running, it should not be killed abnormally.

- **기능 : Project 1에서 구현한 shell에서 아래의 내용을 추가하여 제출.**

(1) background process : &

- you may use the function `waitpid(pid, status, options)`

(example) `/home/bongbong/os> sleep 5 &`

- built-in command "`job`" should also be implemented

(example)
```
/home/bongbong/os> sleep 5 &
/home/bongbong/os> sleep 3 &
/home/bongbong/os> sleep 7 &
/home/bongbong/os> sleep 9
[Ctrl-Z pressed]
/home/bongbong/os> jobs
[1]   Done                 sleep 5
[2]   Done                 sleep 3
[3]   Running              sleep 7
[4]   Stopped              sleep 9
```

- the built-in command "`jobs`" should be executed just like UNIX "`jobs`".
  If a job is finished, "`Done`" should be displayed once.
  If a job is running, "`Running`" should be displayed.
  If a job is stopped using Ctrl-Z, "`Stopped`" should be displayed.

- build-in command "`kill`" should be implemented too.
  use a function "`kill(pid, SIGKILL)`".

(example)
```
/home/bongbong/os> jobs
[1]   Done                 sleep 5
[2]   Running              sleep 3
[3]   Running              sleep 7
/home/bongbong/os> kill %3
[3]   Terminated           sleep 7
```

(2) Signals

- MiniShell should not be killed unless a user commands to quit the MiniShell.
- Ctrl-C should kill currently running command. However, Ctrl-C should not kill MiniShell.
- ~~Ctrl-Z should stop currently running command.~~
- When a background job is done, MiniShell should display that.

(example)
```
/home/bongbong/os> sleep 2 &
/home/bongbong/os>
[1]+  Done              sleep 2
/home/bongbong/os> _
```

(example)
```
~~/home/bongbong/os> sleep 100~~
~~[Ctrl-Z pressed]~~
/home/bongbong/os> sleep 200 &
/home/bongbong/os> jobs
~~[1]  Stopped              sleep 100~~
[2]  Running              sleep 200
/home/bongbong/os> _
```

(3) pipe , redirection

- you may use the function dup2

(example1) `/home/bongbong/os> cat code.c | wc`
after execution, the standard output of "`cat code.c`" is put into the standard input of "`wc`"

(example2) `/home/bongbong/os> cat code.c > output.txt`
after execution, the standard output of "`cat code.c`" is written into output.txt file

bonus points => (example3) `/home/bongbong/os> cat code.c | grep printf | wc > output.txt`
multiple pipes and redirection

==========< Project 1 사항 >==========

(1) Basic execution of commands.

- MiniShell should receive a user input in a command line and execute the command. After the execution, MiniShell should wait for another input.
- To implement this, you should use `fork & exec` system call.
  You must not use the function **"system"**
- support multiple commands using ";"
- For wrong input command, your shell should display appropriate error messages.

(example) `/home/bongbong/os> ls -l ; whoami ; cat a.txt`

(2) Prompt : MiniShell should print current directory path in a prompt

- you may use the function `getcwd`

(example) `/home/bongbong/os> _`

(3) Implementation of other built-in commands

- cd : change current directory , ~ : home directory processing

(example)
```
/home/bongbong> cd os
/home/bongbong/os> _
/home/bongbong/os> cd ~bongbong
/home/bongbong> _
```

- exit : quit MiniShell
- alias : (example) /home/bongbong> alias dir ls -alF
- echo : print

(4) Wildcard processing (*,?)

- * : zero or more arbitrary characters

- ? : one arbitrary character

(example) /home/bongbong> ls *.c

a.c b.c code.c

(5) Shell variables and Environment Variables

- handling environment variables

(example) /home/bongbong> echo $USER

bongbong

/home/bongbong> setenv USER bssohn

/home/bongbong> echo $USER

bssohn

- handling shell variables

(example) /home/bongbong> set x = hello

(example) /home/bongbong> echo $x

hello

(6) Double Quotes, Single Quotes, and Backslash

- " " : double quotes remove the special interpretation of most metacharacters (e.g. <,>,|,*,?,...)

The exceptions are the dollar sign ($) in front of a variable name.

(example) /home/bongbong> find . -name "*.c" -print

- ' ' : single quotes operate like double quotes, but their effect is stronger.

Any enclosed metacharaters are treated as iteral characters.

(example) /home/bongbong> set x = hello

/home/bongbong> echo "< > $x *.c &"

< > hello *.c &

/home/bongbong> echo '< > $x "y" ? &'

< > $x "y" ? &

- ₩ (backslash) : converts special character into literal character

(example) /home/bongbong> find . -name ₩*.c -print

※meta-character : UNIX 쉘에서 특수한 의미를 갖는 문자 (예: $, *,?,[],~,!,;,&,|,<,>,&,% 등)

* you may assume that each component of a command is separated by one or more spaces.

(example) ls -l;cat a.txt (X) ls -l ; cat a.txt