COMP1521 18s2

Assignment 2
My Very Own Shell

Computer System Fundamentals

Objectives

- to give you experience writing C code to manipulate processes
- to give you experience with interprocess comminication (pipes)
- to give you further experience with data structures in C

Admin

Marks	11 (towards total course mark)
Group?	This assignment is completed **individually**
Due	by 11:59:59pm on Sunday 7th October
Submit	`give cs1521 ass2 mymysh.c history.h history.c` or via Webcms
Late Penalty	0.09 marks per hour late (approx 2.2 marks per day) off the ceiling (e.g. if you are 36 hours late, your maximum possible mark is 7.8/11)
Assessment	For a guide to style, use the code in the lectures and tute solutions, and the supplied code. 9 marks for correct performance, measured by auto-testing on a range of test cases, some of which will be provided for you to test your every line, 1 mar but roughly one comment on each block of C statements that does a meaningful task 1 mark for readable code, sensib use of indentation to highlight control structures

If your submitted code won't compile, your maximum possible "performance" mark is 3/9. The solution: make sure your code compiles and runs under `dcc` before submitting. If your submitted code fails all the performance tests, your maximum possible performance mark is 5/9. In both cases, the actual mark will be determined by a tutor's assessment on how close your code is to working.

Background

A shell is a program that executes other programs. Command-line based shells (e.g. `bash`) read lines of text, break each line into tokens, and execute the command indicated by the first token. For example:

```
ls -al ~cs1521/bin
```

is a command with three tokens ("`ls`", "`-al`" and "~cs1521/bin"). The shell works out where the executable for the `ls` command is located, and executes it, passing the second two tokens as command-line arguments.

However, shells do a lot more than just reading command lines and executing them. For example, they keep a history of previous commands to make it easy to re-execute them. They also allow users to capture the output of a command, by redirecting its output into a file. And, importantly on Unix/Linux, they allow users to build a pipeline of commands to achieve powerful effects without having to write a

program. For example, the following pipeline produces a list of the top ten most frequently used words in a text file:

```
cat blah.txt | tr -cs '[a-z]' '\n' | sort | uniq -c | sort -nr | head -10
```

For more details on Unix/Linux commands and how they can be combined in this way, see the COMP2041 web site.

Unix/Linux shells also typically provide a full programming capability, if pipelines aren't quite enough.

You will not be required to implement a shell as powerful as `bash` for this assignment, but you will build some of the core features.

Setting Up

Create a private directory for doing the assignment, and put the assignment files in it by running the following command:

```
$ unzip /home/cs1521/web/18s2/assignments/assign2/assign2.zip
```

If you're working on this at home, download the ZIP file and create the files on your home machine. It's fine to work on your own machine but remember to *always* test your code on the CSE machines before submitting.

The above command will create the following files:

`Makefile`

A file to control co

`mymysh.c`

A skeleton for the shell program that you are requi

`history.h`

A complete interface for the functions on history list data.

`history.c`

A skeleton for the history list functions.

Exercise

The aim of this exercise is to complete the supplied program skeleton in `mymysh.c`, giving an executable called `./mymysh`. This requires you to implement the command history list data structures and functions, and then use these in implementing the main program. You can add as many functions as you like to `mymysh.c` and `history.c` files.

The shell needs to be able to do the following:

Read and execute commands (1 mark)

The shell should print a prompt using the supplied `prompt()` function. It then reads a single line of text and interprets it as a command. A command is a sequence of space-separated tokens on a single line. The first token is treated as the name of a command, where the command exists as an executable file somewhere in the user's `PATH`. If no such executable is found, the shell should print a "`Command not found`" message.

The command is invoked via the `execve()` library function, with the full pathname of the command as the first parameter, the sequence of tokens as the second parameter, and the user's

environment (from the third argument of the the main program) as the third parameter. This is similar as the Week 08 Lab (except that the lab didn't use the available environment).

One difference to the Lab is that the shell should print some additional information before and after the output fom the command. Before the command it should show the full pathname of the command executable. After the command, it should print the command's return status. The output from the command should be delineated by twenty hyphens, as shown below:

```
mymysh$ ls -l
Running /bin/ls ...
--------------------
... output from the ls -l command ...
--------------------
Returns 0
mymysh$
```

Note that "`mymysh$`" is the shell's prompt.

There are a number of built-in "commands" that are not executed as described above, but are handled directly by the shell. The built-ins are described below.

If the command line has no tokens, it is simply ignored and a new shell prompt is printed.

Maintain a history of the previous 20 valid commands (2 marks)

The shell should maintain a persistent list of the most recent 20 valid commands that the shell has executed. Each command is associated with a sequence number; sequence numbers increase constantly over time, and persist between sessions with the shell (see the examples below).

While shell is exec -size data structure defined in `history.c`. A n and should be used in the `main()` function. If tures and functions, feel free to define your e precise implementation of the command history is not critical.

The command history has to persist between exec ogram. To achieve this, it saves the history in a file `$HOME/.mymysh_histo` ates and restores it from this file when it next starts. The `.mymysh_history` is simply a text file, containing the most recent 20 commands and their sequence numbers.

Commands from the history can be re-executed by using the special notation `!`*SeqNo* and giving the sequence number for one of the commands in the history. The command from the history should become the current command and then be treated as if it had been typed by the user. The special notation `!!` re-executes the previous command.

Note that, unlike most Unix/Linux shells, `mymysh` does not place invalid commands in the history, so commands should be checked for the following before being executed:

- an executable for the command (first token) actually exists
- stdin is redirected, but without giveng a filename to read from
- stdin is redirected, but with a filename that is nonexistent or not readable
- stdin is redirected, but without giving a filename to write to
- stdin is redirected, but with a filename that is not writeable
- using `!`*SeqNo* but with an invalid sequence number

If the command line fails any of the above, it should not be placed in the history. The built-in commands `h` (or `history`)

Implement shell built-in commands (1 mark)

The following commands are handled by the shell, and do not need to be searched for in the command path.

exit

terminate the shell (after saving the command history)

h or **history**

display the last 20 commands, with their sequence numbers

pwd

print the shell's current working directory
(hint: `getcwd()`)

cd *Directory*

change the shell's working directory (after filename expansion)
after changing, show the new working directory
(hint: `chdir()` and `getcwd()`)

The `exit` built-in is not placed in the command history.

Expand filename wildcards (2 marks)

If any of the following characters (' * ', ' ? ', ' [', ' ~ ') appears in one of the tokens, that token should be replaced by all of the tokens matching that token using the `glob()` library function. This may result in the tokens list becoming longer than initially. If there are no matches, use the token unchanged. This should be done before any of the actions described below.
(hint: use `GLOB_NOCHECK|GLOB_TILDE` as the second parameter of the `glob()` function)

Redirect command input (2 marks)

If the command line contains the tokens < and a filename as the last two tokens, the command should be executed with its standard input connected to the named file. If the file does not exist, or is not readable, tha re in the command-line is also an error.
(Hint: `pipe()` and `du`

Redirect command output (1 mark)

If the command line contains the tokens > and a fil okens, the command should be executed with its standard output connected to the named file. If the file does not already exist or exists and is writeable, then it is truncated to zero length and it current contents are overwritten. If the file exists and is not writeable, that is an error. Having > as the last token, or elsewhere in the command-line is also an error.
(Hint: `pipe()` and `dup2()`)

Hints

The main program of `mymysh` should be structured roughly as follows:

```
main() {
   restore the command history
   print prompt
   while (more commands) {
      if empty command, ignore
      handle ! history substitution
      tokenise
      handle *?[~ filename expansion
      handle shell built-ins
      check for input/output redirections
      find executable using first token
      if none, then Command not found
      sort out any redirections
      run the command
      print prompt
```

```
    }
    save command history
}
```

Trying to implement the whole of the above at once is difficult. I'd suggest implementing it in stages:

- get "normal" commands running properly (Week 08 Lab)
- then add shell built-ins
- then add command history
- then add history substitution
- then add filename expansion
- then add input/output redirection

Marks are available for each of the components. You can get full marks for a working component, even if other components don't work or aren't implemented.

The following example shows how the shell should work in practice:

```
$ ./mymysh
mymysh$ ls
Running /bin/ls ...
--------------------
Makefile   history.h  mymysh    mymysh.o   xxx
history.c  history.o  mymysh.c  mymysh1.c
--------------------
Returns 0
mymysh$ wc -l *.c
Running /usr/bin/wc -l *.c ...
--------------------
  129 history.c
  423 mymysh.c
  470 mymysh1.c
 1022 total
--------------------
Returns 0
mymysh$ cat xyz
Running /bin/cat ...
--------------------
cat: xyz: No such file or directory
--------------------
Returns 256
mymysh$ cat < xyz
... note: not recorded in history ...
Input redirection: No such file or directory
mymysh$ ls -l > xyz
Running /bin/ls ...
--------------------
--------------------
Returns 0
mymysh$ cat xyz
Running /bin/cat ...
--------------------
total 76
-rw-r--r-- 1 jas jas   303 Sep 16 21:59 Makefile
-rw-r--r-- 1 jas jas  3310 Sep 16 21:01 history.c
-rw-r--r-- 1 jas jas   377 Sep 16 21:02 history.h
-rw-r--r-- 1 jas jas  2640 Sep 16 22:41 history.o
-rwxr-xr-x 1 jas jas 17216 Sep 16 22:41 mymysh
-rw-r--r-- 1 jas jas 11242 Sep 16 22:36 mymysh.c
-rw-r--r-- 1 jas jas  7792 Sep 16 22:41 mymysh.o
-rw-r--r-- 1 jas jas 12734 Sep 16 20:54 mymysh1.c
-rw-r--r-- 1 jas jas   301 Sep 16 22:40 xxx
```

```
-rw-r--r-- 1 jas jas     0 Sep 16 22:48 xyz
--------------------
Returns 0
mymysh$ h
  11  make
  12  ls
  13  make clean
  14  cat xxx
  15  ls -l > xxx
  16  cat xxx
  17  echo Ooops ... mymysh executable is gone
  18  make
  19  h
  20  cd ~cs1521
  21  ls
  22  cd web/18s2
  23  ls
  24  h
  25  h
  26  ls
  27  wc -l *.c
  28  cat xyz
  29  ls -l > xyz
  30  cat xyz
mymysh$ exit
... stopped shell and then restart ...
$ ./mymysh
mymysh$ !28
cat xyz
Running /bin/cat .
--------------------
total 76
-rw-r--r-- 1 jas jas   303 Sep 16 21:59 Makefi
-rw-r--r-- 1 jas jas  3310 Sep 16 21:01 histor
-rw-r--r-- 1 jas jas   877 Sep 16 21:02 histor
-rw-r--r-- 1 jas jas  2640 Sep 16 22:41 histor
-rwxr-xr-x 1 jas jas 17216 Sep 16 22:41 mymysh
-rw-r--r-- 1 jas jas 11242 Sep 16 22:36 mymysh.c
-rw-r--r-- 1 jas jas  7792 Sep 16 22:41 mymysh.o
-rw-r--r-- 1 jas jas 12734 Sep 16 20:54 mymysh1.c
-rw-r--r-- 1 jas jas   301 Sep 16 22:40 xxx
-rw-r--r-- 1 jas jas     0 Sep 16 22:48 xyz
--------------------
Returns 0
mymysh$ pwd
/some/path/or/other/cs1521/ass/ass2
mymysh$ cd ..
/some/path/or/other/cs1521/ass
mymysh$ pwd
/some/path/or/other/cs1521/ass
mymysh$ !!
pwd
/some/path/or/other/cs1521/ass
mymysh$ control-D
$
```


To resolve any ambiguities in the above, and to give you a basis for testing, an executable for the shell is available as

```
$ ~cs1521/bin/mymysh
```

Please let me know asap if you think that there are bugs in the sample executable. Disagreeing with one of the above design choices does not constitute a "bug". However, if the sample executable behaves differently to what is stated above, then that is definitely a bug.

Challenge

(Worth kudos, but no marks)

Implement command pipelines (i.e. Cmd_1 **|** Cmd_2 **|** ... **|** Cmd_n, where $n \geq 2$).

Have fun, *jas*