The University of Queensland
School of Information Technology and Electrical Engineering

Semester One, 2010

COMP2303 / COMP7306 – Assignment 3.1

Due: 11:59pm Monday 17th May, 2010
Marks: 50
Weighting: 25% of your overall assignment mark (COMP2303)

Introduction

Your task is to write a program (called `suspect`) which checks the behaviour of other programs. For example it will check that output produced in response to given inputs. Your assignment must comply with the subject coding style.

This assignment will should use the C99 version of the language.

This is an **individual assignment**. You should feel free to discuss aspects of C programming and the assignment specification with your fellow students, but you shouldn't actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff – don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school website:
http://www.itee.uq.edu.au/about_ITEE/policies/student-misconduct.html

Specification

Your program (`suspect`) which will take a single *optional* filename as a parameter.
Any extra parameters should be ignored.
If the filename is not given input should be read from standard in. In either case the behaviour should be the same (aside from where the input is coming from).
Then `suspect` will read a series of multi-line *blocks* separated by blank lines. Each block represents an execution of a program. The first line of a block is the program to be executed. Successive lines represent input/output with the executing program or a test to be performed.
You will communicate with the program using pipes.
When a block ends, the command executed by that block must be terrminated (if it hasn't exited already). When `suspect` itself closes is should not leave any child processes or zombies.

For example – to check the first two files in the current directory are `suspect` and data:

```
./suspect           # has on the end of a line is a comment.
ls                  #execute the ls command
want suspect        #ls should output the text “suspect”
want data           #ls should output the text “data”
exit 0       #when ls exits, it should be with status 0
                    #... any other output from ls will be ignored (block ends)
```

The # comments are not part of the input and are just to provide explanation in this specification.

Once a block has begun the following commands can be sent. In this text, "program" refers to the program being run by `suspect` not `suspect` itself. Each command can pass or fail. When a command fails, the block should be finished (any additional commands should be ignored), and the program running should be killed. Once this is done, `suspect` should exit. Extra params to commands should be silently ignored. **Any command will fail if it is given incorrect parameters.**

Command	**Parameter**	**Explanation**
exit	Number *n*	Wait for the program to exit. Pass if exit status is n. Fail if program exits abnormally or exit status does not equal n. Fail if n is not a positive integer.
want	Rest of line of text (S)	Read a line of text from the program. Pass if it matches S. Fail if not.
send	Rest of line of text (S)	Send S to the input of the program. Pass provided there is no IO error and endinput has not been executed.
exists	Rest of line of text (S)	Use S as the name of a file. Pass if a file of that name exists. Fail otherwise.
size>	Number_n Rest_of_line (S)	Use S as the name of a file. Pass if the file exists and has size bigger than n. Fail otherwise.
echo	on/off	When output is read from the program it should be copied to stdout. The parameter determines whether echo is off or on. By default, your program should not echo. Passes if given correct parameter.
endinput	None	No more input will be sent to the program close the input pipe. Always passes.
interactive	Word W (sequence of chars with no whitespace in it).	Read lines of input and send them to the program. Stop doing this when W appears on its own. Passes provided the send operations succeed. If W contains spaces, then only chars before the space will be used. *Even if a filename is given, interactive input should be read from standard in.*
limit	Number of seconds n	Starts a timer. If ~~an exit command~~ the block has not completed before the timer runs out, then finish the block, kill the program and print an error message. Passes if given correct parameter.

For example:

```
./suspect
mylogin.sh                #Execute mylogin.sh
```

```
want username:                #Next output should be username
send fred                     #Send fred as input to mylogin.sh
want password:                #Next output should be password
interactive END           #Start input from keyboard
notmypassword                 # This is entered on the keyboard
END                           #End input from keyboard
exists loginok                #Check if file called loginok exists
send ./dostuff.sh
endinput                      #No more input to go to mylogin.sh
exit 0                    #Check that we exited ok

./taskcheck.sh                #Start a new block and execute taskcheck.sh
exit 1                    #Make sure that taskcheck.sh exits with status 1
```

Errors:

The table below describes how your program should respond to error conditions.
For this assignment, all output by `suspect` should be to standard out.
Where there is no error, `suspect` should exit with status 0.
If the file given on the command line cannot be read then exit with status 4.
If a block fails then `suspect` should exit with status 1 and print the line of input which failed.
This count should start from 1 but lines entered as part of the interactive command do not count.
For example, the following would fail (for this example, assume program.sh exists):

```
./program.sh                      # line 1
interactive   STOP                # line 2
text
text
STOP                              # still line 2
exists filethatisntthere      # line 3
```

`suspect` would exit with status 1 and output:

```
Test failed on line 3.            #Note the dot
```

If the end of a block is reached without seeing an exit statement then `suspect` should exit with status 2 and print:

```
Block X ended without an exit.
```

Where X is the number of the block (starting from 1).

If a timer expires (a block hasn't ended within the timelimit) then `suspect` should exit with status 3 and print:

```
Block X timed out.
```

Where X is the number of the block

If a line inside a block is not recognised as a command it should be treated as an error on that line.

In summary:

Exit status	**Condition**	**Output (with '\n')**

0	No error.	
1	Command in block failed	Test failed on line %d.
2	Block ended without an exit	Block %d ended without an exit.
3	Time limit expired	Block %d timed out.
4	Could not open file given on command line	Failed to open %s.

Notes/Suggestions:

- To begin with, you can take the whole first line of a block as the command to be executed. To get more marks however you should separate the line into space separated elements for argv.
- Some commands can be worked on independantly of each other. For example, exists and size> don't need to be able to talk to the other program while .exit does not require pipes.
- Read the documentation for the following functions carefully:
  - exec and friends
  - pipe, dup2, fdopen
- Executing another program can be a little tricky until you get used to it. Practice by writing a small program which runs: ls -l -a

Style:

You must follow **version 1.4** of the COMP2303/COMP7306 C programming style guide found at http://www.itee.uq.edu.au/~comp2303/resources/c_resources.html.
Note however that this assignment will be using the c99 form of the langauge (specifically -std=gnu99)

Submission:

Submission is via subversion. You should check out and commit to the **ass3/trunk** directory in your repository.

Your submission must include all source and any other required files (in particular you must submit a Makefile). Do not submit compiled files (eg .o or compiled programs).

Your Makefile must produce a program called `suspect` with the command:
make

Your program must be compiled under gcc with at least the following switches:
-Wall -std=gnu99 -pedantic

You are not permitted to disable warnings or use pragmas to hide them.
You are not permitted to import non-standard functions from standard libraries.
Your program must not use non-standard headers/libraries (except where they are written by you and submitted with your assignment).

The due date for this assignment is 11:59pm Monday 17th May, 2010. The policy on “grace days” and exceptional circumstances (e.g. illness) is as outlined in the course profile. Submissions completed before 11:59pm Sunday 16nd May, 2010 will earn additional grace days. Note that no submissions can be made more than 120 hours past the deadline under any circumstances.

Marks:

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, as outlined below. Partial marks will be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. **If your program does not allow a feature to be tested then you will receive 0 marks for that feature**, even if you claim to have implemented it. The markers will make no alterations to your code (other than to remove code without academic merit).

- Reading input from correct source and reporting errors if required (2 marks)
- Correctly executing the program for a block (2 marks)
- End of block handled correctly (6 marks)
- Correctly handling each of the commands (9 commands at 2 marks each)
- Correctly processing scripts with multiple blocks containing multiple commands (14 marks)

Style (8 marks)

Your style mark will be the minimum of:

$8 * 0.9^{(\text{Number of style guide violations + number of compilation warnings})}$
and
your functionality mark.

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of version 1.4 of the COMP2303/COMP7306 C Programming Style Guide. A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final – it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` tool. Your style mark can never be more than your functionality mark – this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

Late Penalties

Late penalties will apply as outlined in the course profile if no grace days remain at the time of submission.

Specification Updates:

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Changes in Version 3.1

1. Changed text for the size> command to better indicate that its two parameters should be space separated.
2. Fixed inconsistent description of limit command. Limit is timed to the end of the block not until the exit command.
3. Blocks should be separated by exactly one blank line.
4. Commands:
    1. Any line in a block (apart from the first line) which is not listed in the command table should be treated as a failed test.
    2. Commands should not be preceded by spaces.
    3. There can only be one limit command and one exit command per block. If more than one appears the second one should fail.
5. A line containing only whitespace is not treated as a blank line.
6. When reading lines of input (from anywhere), a line is ended either with '\n' or when the end of file is reached.