



MODULAR PROGRAMME ASSESSED COURSE-WORK SPECIFICATION

Module Details:

Module Code: uqc109s2	Module Title: Computer Networks & Operating Systems	
Module Leader: Nigel Gunton		
Module Tutors: John Counsell Graham Darke Nigel Gunton Ian Johnson		
Assignment CW2	Element Number: Weighting 25%	Total Assignment Time: 12 hrs + lab time

Dates:

Date assignment issued to students: Feb 1st	Date for return of marked work: May 23rd
Submission Place: post-box in N foyer, below the North stairs	Date of Submission: Thurs April 1st
	Time of Submission: 10.00am

Deliverables:

As listed on the Assignment spec sheet

Overview:

Even in these days of Graphical User Interfaces (GUIs) most modern operating systems still offer a command line interpreter, or shell. Many systems administrators are frequent users of command line interfaces, even on the Microsoft family of operating systems! From a learning point of view, command-line interfaces provide an opportunity to study the underlying structure of an operating system. To this end you will be undertaking the development of several utilities that will provide for interaction between a user and an operating system. You will also investigate the design and development of a simple command line interpreter.

- This assignment will be developed in stages and **signed off by your lab tutor as you proceed**.
- You will be expected to work on this during your lab sessions **AND** in your own time.

The criteria for awarding marks will be as follows :

Comments

should be clear and meaningful. They must explain **WHY** the code is doing something not **WHAT** it's doing. eg. which comment tells you more about the programmers intent

...

```
strcpy(a,b); /* copy string b*/
```

or

```
strcpy(a,b); /* copy user input into string 'a' for tokenizing */
```

Structure and design

Your code must be well structured with appropriate use of functions and variables, as well as meaningful variable names. Where you have provided a design then you should be able to justify your choice of design method. Your design should be clear and unambiguous and reflected in the structure of your code. *(Note that you can gain marks here even if your program does not execute correctly.)*

Correct functionality

How well does your code work? Can it handle input garbage from a user without crashing? Does it provide a default action or usage message? How well does it mimic the system equivalent? Can it handle the difficult cases eg `ls -l \dev`? Meaningful error messages?

Requirements:

You are expected to attempt part '1' and then to attempt **either** part '2a' or part '2b'. It will pay to think ahead and to consider functions that will be common to all or many of your programs. You will benefit from developing a design for the `ls` and `kill` programs rather than just forging ahead and attempting to code them directly.

For part 2 you are expected to use the code examples provided on http://www.hawk-lord.uklinux.net/**** as a starting point.

- 1) To develop the following elements, initially as stand-alone programs. They are listed in order of increasing difficulty.

`pwd` This should print, on stdout, the path to the current directory. Attempt this utility first as it is the most straightforward.

(5 marks)

`cd` This should take an optional path as an argument. If no argument is provided then the default behaviour is to change directory to the users home directory. Use the code from the previous command to print the path to the new current directory. Why does this program, as a 'stand-alone' behave in the way it does?

(5 marks)

`kill`

This command should respond as follows :

```
kill pid
send SIGTERM to process pid
```

`kill -l`

list the signals sent by this command. Your version of kill should recognise SIGTERM, SIGKILL and SIGHUP. It should provide a list of both the names of the signals and their numbers. (RTM Vol.7 signal[†])

`kill signal pid`

Send the specified signal to *pid*. It should recognise both the numeric value and the name of the signals.

(20 marks)

- 15 The 'list directory contents' command. It should accept the flags `-a` and `-l`. Read the manual page for details on the meaning of these flags. You should endeavor to emulate the behaviour of the system command in response to these two flags as closely as possible. This includes the ability to handle multiple flags in any order, ie `-al` or `-la` or even `-alllaaaaal`, as well as multiple paths and a default behaviour in the absence of any command line arguments.

(30 marks)

These should be demonstrated and explained to your lab tutor who will sign and date the relevant section of the sign-off sheet when s/he is satisfied with your code. This will accrue a maximum of 60%.

- 2) The design and structure of a simple command-line interpreter, or shell, based on the provided code examples. You should attempt **ONE** of the following:

- a) An analysis of the provided code culminating in a detailed design model. You should choose an appropriate design method from those that you have been introduced to in other modules and provide a clear justification for your chosen design method. Your design should also show how you would integrate your utilities from part 1 into the interpreter. One suitable design method would be the use of a Finite State Machine (FSM). This is similar to the use of Statechart Diagrams in UML.

OR

- b) The correction and integration of the provided code examples, along with your utilities into a functioning shell. The comments in the provided code must be removed and replaced with your own comments. The resulting shell should report all errors correctly.

These should be demonstrated and explained to your lab tutor who will sign and date the relevant section of the sign-off sheet when s/he is satisfied with your code. This will accrue a maximum of 40%. The criteria for marking are as described earlier.

Constraints :

- 1) All code and designs **MUST** be demonstrated and explained to your lab tutor before it will be signed off.
- 2) Remember, this is an individual assignment and that assessment offences are taken seriously. This does not prevent you from discussing problems and ideas with your peers and you are encouraged to do so as long as the final result is your own work. If you use sections of code from other sources then they must be clearly identified and you will be expected to demonstrate your understanding of the code to your lab tutor. This includes the use of the provided code.

Support :

C programming

The syntax of C and the syntax of Java are very similar. In addition there is an excellent C tutorial on Ian Johnson's home-page and many other web based tutorials. In general, picking up another programming language is something you should be confident about as students on a Computing degree. Most languages share the same concepts and control structures so you should only need to focus on the syntax.

[†] man -s 3HEAD signal on Solaris

system calls

These will be covered in the lectures. There is also a very good web-site that covers much of the assignment material.(see link from my home page). For your convenience, this site is also available on kenny. You will also find the manual pages useful. `man syscalls` will provide a list of all system calls for which manual pages exist. There is a separate manual page for each of these calls. `man undocumented` will provide a list of calls for which there is not yet any documentation. A search engine may be useful for information on these.

your lab tutor

Will provide support with syntax problems, design queries, the unix environment etc.

Deliverables :

- a) Your sign-off sheet, signed and dated for all completed work.
- b) Copies of all code and designs that have been demonstrated/explained to your tutor. These **MUST** be signed by your tutor.

UQC109S2 Computer Networks & O/S Coursework 2 2003 - 2004

Coursework Checklist

Student Number

	Total
pwd Comments : Structure : Function :	<input type="text"/>
cd Comments : Structure : Function :	<input type="text"/>
kill Comments : Structure : Function :	<input type="text"/>
ls Comments : Structure : Function :	<input type="text"/>
Demonstration of shell & integration of above :	<input type="text"/>
OR	
Walkthrough of extended design :	<input type="text"/>

Lab Tutor Signature.....