

**The University of Queensland  
School of Information Technology and Electrical Engineering  
Semester 2, 2009**

COMP3301 COMP7308

**ASSIGNMENT 2 – Filter Driver**

Due: **5pm Friday 25th September**  
Weighting: **25% (25 marks)**

## Objective

As part of the assessment for this course, students are required to complete three assignments which will assess your understanding of the Minix operating system and ability to program at a low level in C. This particular assignment will assess your ability to write simple device driver and kernel code.

## The Assignment

In many institutions it is necessary to restrict and remove certain phrases/words from being exchanged, whether from security concerns (e.g. ASIO interdepartmental email) or to ensure adherence to a strict language policy (e.g. local high-school IM client). Your task is to implement a simple character device driver in Minix, that will allow a user to write a sequence of characters to a device (`/dev/filter`), and then read-back the filtered sequence.

## Opening

When the device is opened (from a closed state), the device should be in a clean state (i.e. if a `read()` was performed, no characters would be returned, and the `mfilter` parameter has not been set). If the device is already open an appropriate error should be returned.

*For this assignment we will only be dealing with the case where a single instance of the device is opened at any one time in the entire OS.*

## Specifying Filter Parameters

Users will be able to specify what words will be filtered and what character to replace the characters of the filtered word with, by using an `ioctl` call. The request code will be `MIOCFILTER` and will take a pointer to an `mfilter` struct (defined below)

```
struct mfilter {
    char* words;           /*string of words to filter*/
    char replacement;     /*filtered word character replacement*/
};
```

The `words` field will be a string of words to filter; where words are separated by a single space (words may only consist of ASCII characters A-Z, a-z, 0-9). The `replacement` field specifies the character with which to replace each character of the filtered word read from `/dev/filter`. A new `mfilter` can be specified at any time during an open session, where the most recently specified `mfilter` is always the one applied. All replacements are case-sensitive.

## COMP7308 Only

*In addition to the above, you will implement a simple wildcard scheme. `*` will substitute for zero or more characters and `?` will substitute for a single character.*

## Write

Once the file has been opened, a sequence of ASCII characters may be written to the device. If more than 1024 bytes are written (before being read), the oldest bytes are discarded to make room for the newest bytes, such that reading the file will always return the most recent 1024 bytes (or less) that were written.

## Read

Once a file has been opened, a sequence of ASCII characters may be read from the device (up to 1024). Reading a byte from the device will consume it, such that it can never be read again.

When a complete word is read from the device (using a single `read()`) it will be filtered against the most recent `mfilter` specified using `ioctl`, and each character of an offending word will be replaced with the replacement character defined by the most recent `mfilter`. (words are defined as non-whitespace ASCII letter groupings separated by any number of whitespace characters).

Filtering will only apply to complete words (as previously defined) and not to substrings within words.

## Close

Upon closing the device any data that was written to the device should be discarded, including the most current filtering parameters, such that if the device was re-opened a `read()` would return nothing.

*Semantics for the values returned by `open()`, `write()`, `read()`, and `close()` should always follow standard Unix convention.*

You are required to alter the operating system source appropriately and generate a patch for a default kernel image. Your code should be appropriately commented to describe your code. In addition to this, you will be required to submit a user application(s) which will demonstrate the correct functionality of your code. This code should include a README file, describing your solution and how the user application demonstrates the scheduling algorithm.

### Example 1.

```
Filter words:      a happy
Replacement char: *
Plain-text:       I am a happy string
Filtered Text:    I am * ***** string
```

### Example 2.

```
Filter words:      a Man
Replacement char: $
Plain-text:       A man walked into a bar. OUTCH!
Filtered Text:    A man walked into $ bar. OUTCH!
```

### Example 3. (COMP7308 Only)

```
Filter words:      t* l?f*
Replacement char: #
Plain-text:       The Answer to the ultimate question of life, the
                  universe, and everything is ...
Filtered Text:    The Answer ## ### ultimate question of ####, ###
                  universe, and everything is ...
```

## Marking

Your submission will include the following four files:

- **s123456\_ass1\_final.patch** (where 123456 is replaced with your student number)
- **build.sh** (a shell script which will rebuild the kernel appropriately)
- **demo\_program.tar** containing,
  - application code/miscellaneous files which demonstrates all the functionality of your device driver
  - A Makefile to build the demo program(s)
- a **README** file containing two sections (plain text or PDF), 1-3 pages in 12 point font.
  - The first section will describe your design and implementation of your device driver
  - The second section will describe the use cases of your demo program(s), how they demonstrate all the required functionality, and how it is to be tested.

In order to mark your assignment, we will use a default Minix image (which we will provide for your own testing purposes). The patch file will be applied in the `/usr/src` directory with the following command:

```
patch -p0 < s123456_ass1_final.patch
```

The kernel will then be built using your supplied shell script and rebooted. Your programs will be tested by extracting them to `/root` and following the instructions in your README. Ensure that your submission will work correctly according to this procedure – particularly ensure that the patch file applies cleanly to the provided image.

## Hints

- Revisit Practical 4 for a guide to the memory driver
- Consider using the Standard C library to help you implement some of the string processing
- There are many ways of implementing this assignment – some of which will be more difficult than others. Examine the existing code *carefully* and do plenty of thinking *before* you start coding!

## Help sources

Help for this assignment is available from a number of sources. Help in understanding the Minix code is best obtained either from the textbook, or from the course newsgroup, `uq.itee.comp3301`. Help in understanding the algorithm is best obtained either from the internet or other textbooks. The tutor is available to provide assistance during the scheduled practical sessions in 78-110. If none of these methods provide the answers you need, the lecturers will be available for consultation - please email for an appointment.

## Submission Details

The due date for the assignment is 5pm Friday 25th September. The assignment must be submitted via the ITEE online submission system. Assignments may be resubmitted numerous times, but only the last submission will be marked.

## Late Submission

Late submissions will **not be accepted except with a doctor's note and even then by permission of the course coordinator.**

## Notification of Results

Students will be notified of their results via the course homepage. Students will be able to view their individual results only.

## Allocation of Marks

Students will be marked on their ability to implement changes in the kernel code and the demonstration of how those changes can be seen in the operation of the kernel.

	Marks available
<b>Coding and Testing (13 marks)</b>	
<code>write()</code> , <code>open()</code> , <code>close()</code> , operate in described manner	0-5
<code>ioctl()</code> is used to change filtering arguments	0-2
<code>read()</code> correctly filters output character sequence	0-4
Demonstration Program(s) adequately illustrates all use cases	0-2
Implementation of wildcard scheme ( <b>COMP7308 ONLY</b> )	0-5
<b>Documentation and Style (12 marks)</b>	0-5
Driver Design/Implementation Documentation	0-5
Demo Program Design/Implementation documentation	0-2
Coding style and commenting	
<b>Caps</b>	
System fails to compile	Capped at 10
System fails to boot / is unusable	Capped at 13

## Academic Merit, Plagiarism, Collusion and Other Misconduct

You should read and understand the statement on academic merit, plagiarism, collusion and other misconduct contained within the course profile and the document referenced in that course profile. You should note that this is an **individual assignment**.

## Reference

Tanenbaum and Woodhull, Operating Systems: Design and Implementation (3<sup>rd</sup> Edition), Prentice Hall, 2006.