University of Derby
Department of Electronics, Computing and Mathematics

Assessment Specification

Module Code and Title: Systems Programming (6CC514)	
Assignment No. and Title: Assessed Component 2: VGA Driver	
Assessment Tutor: Wayne Rippin	**Weighting Towards Module Grade:** 65%
Date Set: 16th November 2017	**Hand-In Date:** 5pm on Thursday 21st December 2017

Penalty for Late Submission

Any work submitted late will not be marked and will be treated as a non-submission.

The only exception to this rule is if you have been given permission for the work to be handed in later due to an appropriate support plan.

If you have been ill or there have been other exceptional extenuating circumstances, you may request a later submission. In this case, you should follow the EEC procedures. Please note that you must still submit something for this assignment by the deadline date.

Level of Collaboration:
This assessment must be an individual piece of work. No collaboration with other students is allowed. In addition, you should not ask for help on any Internet site such as Stack Overflow, etc

Learning Outcomes covered in this Assignment:

On successful completion of the module, students will be able to:

1. *Demonstrate critical awareness of the concepts and issues related to systems programming.*
2. *Design, implement, and evaluate programs employing aspects of systems programming.*

Criteria for Assessment:

See below.

Background

In other modules on your course, you are used to using graphics routines provided by the operating system that perform such functions as setting a pixel on the screen, drawing a line, drawing a rectangle, etc. For this assignment, you will be providing an implementation of such functions that operate on a VGA display.

Task and Assessment Criteria

You are to implement a library of low-level VGA drawing primitive functions to incorporate into the operating system you are developing in this module. You will also provide a demonstration of the primitives you have implemented. These will run under the Bochs emulator.

Details of the how to program to the VGA interface is given in the appendix to this specification. You should use the file Step9.zip as the starting point for this assessment. This includes the version of the Operating System we have seen so far in this module, as well as the files referenced in this specification.

In order to obtain a pass grade (i.e. 40% or higher), you must provide functions that perform the following in ring 0. You have only been told what the functions need to do. It is up to you to define suitable definitions for these functions. For this part of the assessment, you only need to provide implementations that work in resolution 320x200 at 256 colours.

Function	**Description**
Set Pixel	Sets the pixel at the specified x, y co-ordinate to the specified colour
Draw line	Draws a line between two co-ordinates. You may decide to implement this as one function or you may implement it in a similar manner to the Windows GDI library by providing a MoveTo function and then a LineTo function.
Draw rectangle	Draws a rectangle of a specified width and height with its top left-hand corner at co-ordinate x, y using the specified colour.
Clear screen	Clear screen to black.
Fill rectangle	Draws a rectangle of a specified width and height with its top left-hand corner at co-ordinate x, y using the specified colour.

You will need to research suitable algorithms for drawing lines between any two points on the screen (for example, the Bresenham line drawing algorithm).

In order to achieve a grade of 50% or greater, you must provide versions of your functions that can be called from code running in ring 3. Since the core routines will be directly accessing video memory, they will not be directly callable from ring 3 code. Therefore, you need to provide a layer that maps ring 3 versions of the functions through to ring 0 functions via a software interrupt (in a similar way to how you have seen other system calls being handled). To avoid interfering

with the current system call mechanism, you must use interrupt 0x81 to handle your calls for this assessment.

To achieve a higher grade than 60% in this assignment, you are expected to implement additional functions. Examples include, but are not limited to:

Drawing a circle
Filling a circle
Filling a polygon
Support for different line styles (dotted lines, dashed lines, etc)

You are free to define the parameters to be whatever you feel is appropriate for that function. To implement additional functions, you will need to research suitable algorithms.

In order to achieve a grade of 70% or higher, your code should provide support for additional functionality such as support the changing of VGA palette colours or support for different resolutions.

In order to demonstrate your functions, it is only required that you add code to kernel_main.c. However, if we get to the point where we can load user applications from disk, you may decide to write an application that is loaded by your operating system that demonstrates your VGA functions.

In addition to your code, you must also produce a Technical Design Document describing the functions you have implemented and the major technical features of your solution to this assignment. This will include the algorithms used for any additional functions that you may implement. It is not expected that this document will be very large since you are expected to provide extensive comments in your source code. The Technical Design Document should describe the testing you have carried out on your code.

The grading of your assessment will include looking at the efficiency and readability of the code, as well as the level of functionality implemented.

Submission

You need to submit a zip file containing your code to the submission point on Course Resources by the due date and time. The name of the zip file must be Assessment2_xxxxxxxxx.zip where xxxxxxxxx is your student number. For example, if your student number is 051234567, yout submission should be called Assessment2_051234567.zip.

Please do NOT use any other name for your submission or use any compression other than .zip. The tools used to download submissions from Course Resources for marking rely on a consistent naming scheme and compression mechanism.

The zip file must contain the complete folder containing your code and the Technical Design Document. Your code should include the code given to you and the associated makefiles so that the marking tutor can rebuild your code by running make. You must run ‘make clean’ before creating your zip file to ensure that only the source files are submitted and not any binary files created by the make process.

Failure to follow these submission requirements will result in a reduction of the grade awarded to your assignment by 20%.

Appendix

Programming the VGA Hardware

To handle all aspects of the VGA hardware and all of the different graphics modes is quite complex and you are not expected to handle all of it in this assessment. In particular, switching graphics modes requires writing to quite a few I/O ports. Code to do this has been provided to you. Details of the code provided to you and other information required for this assignment is provided below, so please read it carefully.

The Video Graphics Array (VGA) is the design of display hardware first introduced in 1987 for the IBM PS/2 computers, but was widely adopted by other companies as a display standard. The highest video mode resolution supported is 640x480x16 colour. Due to the widespread adoption by PC manufacturers, VGA has become one of the oldest standards still supported by modern PC's.

Changing the VGA Mode

The standard modes supported by VGA hardware are as follows:

Mode	Resolution	Colour depth	Mode	Resolution	Colour depth
0h	40x25 Text	16 Colour	Dh	320x200	16 Colour
1h	40x25 Text	16 Colour	Eh	640x200	16 Colour
2h	80x25 Text	16 Colour	Fh	640x350	2 Colour
3h	80x25 Text	16 Colour	10h	640x350	16 Colour
4h	320x200	4 Colour	11h	640x480	2 Colour
5h	320x200	4 Gray	12h	640x480	16 Colour
7h	80x25 Text	2 Colour	13h	320x200	256 Colour

The ‘mode’ number comes from the number passed to the VGA BIOS from code running in real mode to set the different resolutions. Because we are running in protected mode, we cannot use the BIOS and the different modes are set by sending values directly to VGA registers using I/O ports.

Two files have been provided for you that define routines that you can use to set the VGA mode. These files are vgamodes.c and vgamodes.h. These define a function called VGA_SetGraphicsMode which you should use to switch from the text mode that the operating systems starts up using into an appropriate graphics mode.

The signature of VGA_SetGraphicsMode is:

```
int VGA_SetGraphicsMode(uint16_t width, uint16_t height, uint8_t chain4Mode)
```

The parameters are:

width	The width of the mode required (in pixels).
height	The height of the mode required (in pixels)
chain4mode	1 if Chain4 mode is to be used, 0 otherwise (see below for more details)

Until you have done most of this assessment, you should call VGA_SetGraphicsMode as follows:

```
VGA_SetGraphicsMode(320, 200, 1);
```

This will select mode 13h. Do NOT attempt to try other modes until you have implemented everything using this mode.

Note that, at the time of writing, this function does not support the modes that support widths of 640 pixels and it does not support switching back to text mode. This may be added later, but for this assessment, it is sufficient for you to only use the modes supported by this function.

Writing Pixels to the Screen

In graphics mode, VGA uses memory mapped into the region 0x000A0000 - 0x000BFFFF. VGA memory is referenced as 4 planes each of 64k of memory. You can think of them as different memory banks that are connected to each other. In 16 colour modes, there are 4 bits per colour. The 4 bits are stored at the same location on each plane. Unfortunately writing a pixel requires also writing to a hardware register to select the plane to write to.

Fortunately, some VGA modes support a linear frame buffer. This is also referred to as Chain4 mode. Linear memory is an array of consecutive bytes; like an array in C. Mode 13h is an example mode that has a linear memory model. This means that we can set a pixel at a particular location on the screen just by writing a byte to a memory location in the 64KB starting at 0x0A0000 (remember that mode 13h supports 256 colours). This is why this mode has been chosen for this assessment – we do not need to worry about selecting different planes.

So to set a pixel at a particular x, y position, we need to write a byte containing the colour to the following location in memory:

0xA0000 + screenwidth * y + x

The Colour Palette

The values of the byte used for each pixel can be thought of as an index into a colour table. The first 16 values (0 through 15) are initially defined as the same colours as used for text colour attributes as follows:

Index	Colour name	Index	Colour name
0	Black	8	Dark Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	Light Gray	15	White

For most of this assessment, you can just use these colours. However, you may want to experiment with other colours.

The remaining values (16 through 255) are not initially defined, but act as indices into a colour palette. The colour palette is a set of RGB values where the red, green and blue components are represented by 6 bits (so you can define a value between 0 and 63 for each colour).

To setup the colour palette for one of the 256 colours requires writing bytes to two ports as follows:

Output the colour number (0 to 255) as a byte value to port 0x3C8
Output the red component (0 to 63) as a byte to port 0x3C9
Output the green component (0 to 63) as a byte to port 0x3C9
Output the blue component (0 to 63) as a byte to port 0x3C9

Note that once you change the colour palette values for a particular colour value on the screen, all pixels that are set to that value will immediately change colour. This can enable you to produce some interesting effects.

References

If you want to learn more about programming for VGA, the following are excellent sources of information.

http://wiki.osdev.org/VGA_Hardware
http://www.osdever.net/FreeVGA/vga/vga.htm