**Text based console:**

For this stuff below to work you need to be in the protected mode. The code below runs without using the BIOS.

As we are not using the BIOS, we need to write to the Text Mode Video Memory. Text Mode Video Memory is a buffer located at 0xB8000 for Colour Monitors and 0xB0000 for monochrome, hoping your kernel has the ability to detect the type of monitor and adjust the location of the video memory. Monochrome monitors are not used in the current standard, so you might directly write to B8000 Buffer. For a simple console we need to have a basic Scrolling function, Cursor and of course printing functions. The text mode takes 16bits to display each character. The 16bit is divided into 2 major parts of 8bits each. The lower 8bit tells the display controller what character to print on the screen and is the ASCII Code Byte. And the upper 8bits are for colour. Again, the upper 8bits are split into two minor parts, the lower 4 bits of the colour byte carry foreground colour and other upper 3 bits of the colour byte carry background colour. The upper 8bits are called attribute byte and the lower 8bits are called character byte.

Example: 0xb8002: ‘H’, Backgroundcolor, Foregroundcolor

The example above shows an element of the video memory.

The screen is divided into rows and columns which form a grid. As the screen is divided into separate cells, to access a particular cell in the memory we use a formula which is given below. Y axis is the height and X axis is the length of the screen.

A123456789 (Imagine, this as a Window opened in your Operating System) B123456789 C123456789

Here, The Letters A to A9 is the X axis and A1 to C1 is the Y axis. That is, X is the horizontal axis and Y is the vertical axis.

The Formula:

element = (y \* width\_of\_screen) + x

To have a basic text mode interface in a kernel we need functions to handle Scrolling of Window, Text Mode Cursor, and a function to setup the buffer and a function to print characters on the screen.

--------------------------------------Text Mode Cursor----------------------------------------

Cursor is a thick underscore which will be pointing our next column in the screen. We can see this cursor while the BIOS is booting. Now you must have realised what exactly the cursor is. We need to write to the VGA Hardware Controller to set the cursor. We send the command to the controller’s command port 0x3D4 and send a byte through 0x3D5.

void cursormove(int row, int col)

{

unsigned short location=(row\*80) + col;/\* Short is a 16bit type \*/

*//Cursor Low port*

outportb(0x3D4, 0x0F);//Sending: cursor low byte to the VGA Controller

outportb(0x3D5, (unsigned char)(location&0xFF));

*//Cursor High port*

outportb(0x3D4, 0x0E);//Sending: cursor high byte to the VGA Controller

outportb(0x3D5, (unsigned char )((location>>8)&0xFF));

//Char is a 8bit type

}

This function can be used to set the cursor location.

-------------------------------Scrolling Screen-----------------------------------------------

At some point we might almost fill up the entire screen, so we need to scroll the window by one line up, note that the top line or the first line will be plucked out of the memory when we move up by one line. You can use this bit of code to check and scroll up.

if(cury >= 25)

{

unsigned short space = 0x20 | (attributebyte << 8);

//attributebyte must be replaced your attributebyte

int tmp;

for (tmp = 0\*80; tmp < 24\*80; tmp++)

{

videomemory[tmp] = videomemory[tmp+80];//B8000 is the location of

// the buffer

}

// Setting the last line

for (tmp = 24\*80; tmp < 25\*80; tmp++)

{

videomemory[tmp] = space;//B8000 is the location of the buffer

}

// Move the cursor to last line

cury = 24;

}

This simple snippet of code will first move all the text up by one line, then set the last line and finally move the cursor to the last line.

---------------------------------------Clearing the Screen-------------------------------------

We sometimes may require clearing the whole screen into a new blank screen. That’s too simple; we will just have to replace all the elements in the video memory to blank spaces.

void clearscr(unsigned char \*attributebyte)

{

unsigned short space = 0x20 | (\*attributebyte << 8);

int i;

for (i = 0; i < 80\*25; i++)

{

videomemory[i] = space;

}

cursormove(0,0);

}

The above function will clear the whole screen for us with a fresh new blank screen with a cursor at the top.

---------------------------------------Printing a Character-------------------------------------

Here the code gets little more complex. It would be easier if we could understand it. Once again add some data to the text memory.

void print(char chr)

{

unsigned short attribute = attributebyte << 8;//attributebyte ?? U know

unsigned short \*location;

int numtab = 8;

// Handle a backspace

if (chr == 0x08 && cur\_x)

{

cur\_x--;//Moves one step back

}

// Handle a tab by

else if (chr == 0x09)

{

cur\_x = (cur\_x+ numtab) & ~( numtab -1);//Pushes the cursor by

//number in the numtab

}

// Handle carriage return

else if (chr == '\r')

{

cur\_x = 0;//You know it

}

// Handle newline

else if (chr == '\n')

{

cur\_x = 0;

cur\_y++;//You know

}

// Handle any other printable character.

else if(chr >= ' ')

{

location = video\_memory + (cur\_y\*80 + cur\_x);//Get the

//location

\*location = chr | attribute;//Set a element

cur\_x++;//Advance the cursor by 1

}

if (cur\_x >= 80)//Handles if you are at the end of the line

{

cur\_x = 0;

cur\_y ++;

}

// Scroll the screen if needed.

scroll();

cursormove(cur\_y,cur\_x);

}

This function takes the input and checks for tab, newline, etc. and runs the command for each of those. If there is a character, the function will copy it to the video memory with its attributes.

-----------------------------------------Printing Strings----------------------------------------

We can print a character but we also need to print multiple characters at once, this simple code will do it.

void printstring(unsigned char \*text)

{

int i;

for (i = 0; i < strlen(text); i++)//You know strlen(String length)

{

print(text[i]);

}

}

We can print a string now,

Example: printstring(“Hello, World”);

------------------------------------------Setting Up--------------------------------------------

We have to set the text memory to 0xB8000. Also set the screen for output. The code below is simple and easy to understand.

unsigned char myattriburebyte = (0 <<4) |(15 & 0x0F);

//An attribute byte which sets black as backg.. colour and white as foreg….

unsigned short \*videomemory;

void init\_console()

{

videomemory = (unsigned short \*) 0xB8000;//B8000 for Colour Moniters

clearscr(myattributebyte);

printstring(“Text based console loaded. \n”);

}

**Troubleshooting:**

**Missing Strings:**

*Sometimes printing individual characters works, but printing string fails. This usually is due to the .rodata section missing in you linker script. The GNU C Compiler option –fwritable-strings is a solution, but the real solution is to add .rodata to the linker script.*

**“vediomemory” undefined:**

*This is due to missing definition of videomemory. Define the videomemory as unsigned short (16bit) and as a pointer where all functions can access it.*

*Written by Yashas Samaga*