Printing To Screen

From OSDev Wiki

Contents

Basics

Assuming that you are in protected mode and not using the BIOS to write text to screen, you will have write directly to "video" memory.

This is quite easy. The text screen video memory for colour monitors resides at `0xB8000`, and for monochrome monitors it is at address `0xB0000` (see Detecting Colour and Monochrome Monitors for more information).

Text mode memory takes two bytes for every "character" on screen. One is the *ASCII code* byte, the other the *attribute* byte. so the text "HeLlo" would be stored as:

```
0x000b8000: 'H', colour_for_H
0x000b8002: 'e', colour_for_e
0x000b8004: 'L', colour_for_L
0x000b8006: 'l', colour_for_l
0x000b8008: 'o', colour_for_o
```

The *attribute* byte carries the *foreground colour* in its lowest 4 bits and the *background color* in its highest 3 bits. The interpretation of bit #7 depends on how you (or the BIOS) configured the hardware (see VGA Resources for additional info).

For instance, using `0x00` as attribute byte means black-on-black (you'll see nothing). `0x07` is lightgrey-on-black (DOS default), `0x1F` is white-on-blue (Win9x's blue-screen-of-death), `0x2a` is for green-monochrome nostalgics.

For colour video cards, you have 16kb of text video memory to use. Since 80x25 mode does not use all 16kb (80 x 25 x 2, 4000 bytes per screen), you have 8 display pages to use.

When you print to any other page than 0, it will *not* appear on screen until that page is *enabled* or *copied* into the page 0 memory space.

Printing Strings

If you have a pointer to video memory and want to write a string, here is how you might do it;

```
// note this example will always write to the top
// line of the screen
void write_string( int colour, const char *string )
{
    volatile char *video = (volatile char*)0xB8000;
    while( *string != 0 )
    {
        *video++ = *string++;
        *video++ = colour;
    }
}
```

This simply cycles through each character in the string, and copies it to the appropriate place in video memory.

For a more advanced print function, you need to store variables for x and y, as the display controller will not print a newline. This involves a switch statement or similar construct. You also have to test for x>80 or y>25 and in the case of x>80 setting x to 0 and incrementing y, or in the case of y>25 scrolling.

Printing Integers

Just like in any environment, you repeatedly divide the value by the base, the remainder of the division giving you the least significant digit of the value.

For example, since 1234 = 4 + 3* 10 + 2 * 100 + 1* 1000, if you repeatedly divide "1234" by ten and use the result of the division, you get the digits:

```
1234 = 123*10 + 4
123 = 12*10 + 3
12 = 1*10 + 2
1 = 1
```

As this algorithm retrieves the digits in the "wrong" order (last-to-first), you have to either work recursively, or invert the sequence of digits afterwards. If you know the numerical value of `number % 10`, you simply have to add this to the character '0' to have the correct character (e.g. '0'+4 == '4')

Here is an example implementation of the itoa() function (which is not standard, but provided by many libraries):

```
char * itoa( int value, char * str, int base )
{
    char * rc;
    char * ptr;
    char * low;
    // Check for supported base.
    if ( base < 2 || base > 36 )
    {
        *str = '\0';
        return str;
    }
    rc = ptr = str;
    // Set '-' for negative decimals.
    if ( value < 0 && base == 10 )
    {
        *ptr++ = '-';
    }
    // Remember where the numbers start.
    low = ptr;
    // The actual conversion.
    do
    {
        // Modulo is negative for negative value. This trick makes abs()
        *ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghi
        value /= base;
    } while ( value );
    // Terminating the string.
    *ptr-- = '\0';
    // Invert the numbers.
    while ( low < ptr )
    {
        char tmp = *low;
        *low++ = *ptr;
        *ptr-- = tmp;
    }
    return rc;
}
```

And here is a shorter one (http://www.strudel.org.uk/itoa/)

Troubleshooting

Nothing is Displayed

Keep in mind that this way of writing to video memory will _only_ work if the screen has been correctly set up for 80x25 video mode (which is mode 03). You can do this either by initializing every VGA register manually, or by calling the *Set Video Mode* service of the BIOS Int10h while you're still in real mode (in your bootsector, for instance). Most BIOS's do that initialization for you, but some other (mainly on laptops) do not. Check out Ralf Brown's Interrupt List for details. Note also that some modes that are reported as "both text & graphic" by mode lists are actually graphic modes with BIOS functions that plot fonts when you call char/message output through Int10h (which means you'll end up with plain graphic mode once in Protected Mode).

(GRUB does this setup for you.)

Another common mistake, e.g. in numerous tutorials spread across the net, is to link the .text section of your kernel/OS to the wrong memory address. If you don't have memory management in place yet, make sure you're using physical memory locations in the linker script.

Printing a Character

While in Protected Mode, try a simple command like:

```
// C
*((int*)0xb8000)=0x07690748;

// NASM
mov [0xb8000], 0x07690748

// GAS
movl $0x07690748,0xb8000
```

which should display 'Hi' in grey-on-black on top of your screen. If this does not work, check your paging / segmentation setup correctly maps your assumed video memory address to 0xB8000 (or 0xB0000).

Missing Strings

Sometimes printing individual characters works, but printing strings fails. This is usually due to the `.rodata` section missing in the linker script. The GCC option `-fwritable-strings` is a substitute workaround, but the real solution is to add `.rodata` to the script.

See Also

Text Mode

- Text UI

GUI

- GUI

Retrieved from "http://wiki.osdev.org/index.php?title=Printing_To_Screen&oldid=17401"
Category: Video

- This page was last modified on 1 January 2015, at 08:18.
- This page has been accessed 62,678 times.