Text UI

From OSDev Wiki

A text user interface is a user interface where by all output is presented in the form of text, in contrast with a Graphical UI that uses graphics along with text to display output.

Contents

Popularity and Use

With the rise of graphical UI's, text based user interfaces still remain practical in hobbyist operating system projects, as they are the most easy type of interface to implement, and are still in use in larger and commercial operating systems where a graphical user interface is unnecessary, for example, in large servers. A text user interface is also easy for quickly outputting results or debug data, and inputing commands to test new features, rather than writing complicated graphical applications.

Input

Input in a text user interface primarily involves the use of a command line shell, usually through entering commands via a keyboard. Other methods do exist, such as text-based menus that can scroll up and down, and prompts asking a user to press a specific key for a specific event to occur.

Video Mode

The most used VGA video mode for a text UI is "text mode", or "mode 0". This is the most commonly used, as it allows direct memory access to a linear address containing each character and it's associated attributes. Text mode 0 provides a text interface 80 characters wide and 25 characters lines per screen.

Video Memory

In text mode 0, the linear text buffer is located in physical at 0xB8000. Reading and writing to and from this address will provide direct manipulation of on screen text. To access a particular character on the screen from X and Y coordinates is simple using the following formula:

position = (y_position * characters_per_line) + x_position;

Each character takes up two bytes of space in memory. The first byte is split into two segments, the forecolour, and the backcolour. The second byte is a n 8-bit ASCII value of the character to print.

Colours

Each character has a colour byte. This colour byte is split up in forecolour and backcolour.

The layout of the byte, using the standard colour palette:

```
Bit 76543210
    ||||||||
    |||||^^^-fore colour
    ||||^----fore colour bright bit
    |^^^-----back colour
    ^--------back colour bright bit OR enables blinking Text
```

Its easy to write to BL, the Colour Nibbles(4Bit), in a Hex Value.
For Example:

```
0x01 sets the background to black and the fore colour to blue

0x10 sets the background to blue and the fore colour to black

0x11 sets both to blue.
```

The default display colours set by the BIOS upon booting are 0x0F: 0 (black) for the background and 7 (White) + 8 (Bright) for the foreground.

In text mode 0, the following standard colour palette is available for use. You can change this palette with VGA commands.

Number	Colour	Name	Number + bright bit	bright Colour	Name
0		Black	0+8=8		Dark Gray
1		Blue	1+8=9		Light Blue
2		Green	2+8=A		Light Green
3		Cyan	3+8=B		Light Cyan
4		Red	4+8=C		Light Red
5		Magenta	5+8=D		Light Magenta
6		Brown	6+8=E		Yellow
7		Light Gray	7+8=F		White

Example Console Class

A heavily documented and easy to understand example of a text console class is taken from Brandon's Kernel Tutorial, available at http://osdever.net/bkerndev/Docs/printing.htm

C Code for Print a Character

The following C code will print a character at X and Y coordinates:

```
void WriteCharacter(unsigned char c, unsigned char forecolour, unsigned c
{
     uint16_t attrib = (backcolour << 4) | (forecolour & 0x0F);
     volatile uint16_t * where;
     where = (volatile uint16_t *)0xB8000 + (y * 80 + x) ;
     *where = c | (attrib << 8);
}
```

Scrolling

Scrolling is achieved through copying the second line of characters onto the first, the third onto the second, etc. Then clearing the last line of text with null characters (usually a blank space with the fore/back colours of 7 and 0, respectively). In text mode 0, this is accomplished by copying characters 80-159 in to memory position 0-79, 160-239 into 80-159, etc.

Virtual Terminals

Virtual terminals are much the same as physical screens, but instead of writing to the linear text buffer referring to the screen, text is written to a linear text buffer stored in memory. Each virtual terminal has it's own text buffer, and the terminal currently being displayed on screen has it's buffer copied into the video buffer to display on screen. Having more than one virtual text buffer has other advantages, such as only having parts of multiple buffers being output onto the screen at the same time.

Buffering Data

When working with a text-based interface, writing text directly into the video memory may eventually begin to create issues, especially when handling keyboard input, processing input from menus or dealing with multiple text mode consoles. This problem can usually be overcome by writing characters and colour data into a separate buffer (unparsed), and periodically flushing the buffer to the screen. There are multiple ways to implement a text buffer; a simple solution might feature a flat block of memory in which text and colour bytes are stored before being copied to the screen, though more a more advanced approach may include a per-line buffer, planar-buffer or a system of tracking updated lines or portions of the screen.

Categories: Text UI | Video
