Nano-Kernel : A Bare Metal OS

## Part 3 - The Console Driver

Modify the console code that you started in the previous section. Your new console code should be a basic console driver. It should:

* Keep track of the cursor’s position (row and column)
* Change the cursor’s position
* Keep track of the foreground and background color
* Change the foreground and background color
* Clear the screen (fill with 0’s for character and foreground / background color)
* Allow you to print a character using the current foreground and background color
* Allow you to print a string (by calling the print character over and over)
* **Implement a version of “printf” using** [***variadic* *functions***](http://bfy.tw/GPUV) **(they are built-in to GCC)**
* Emulate the basics of an [ASCII terminal](http://bfy.tw/GPUX), so that the following characters get special treatment:
  + Newline - move cursor down one line OR scroll the display up one row
  + Carriage return - move the cursor to the beginning of a line
  + Tab - move the cursor 8 spaces forward
  + Backspace - move the cursor back one space, zapping the current character (except at beginning of line)
  + Form Feed - clear the screen
  + Vertical tab - move the cursor to the last row
  + Line wrap - after printing the last character on a line, move cursor to the next line, and possibly scroll up

### Color Graphics Adapter (CGA)

Since we lost the system’s BIOS in protected mode, things that were built-in aren’t. Printing characters directly to video RAM doesn’t move the cursor on the screen! We can do that by accessing the CGA’s control registers[[1]](#footnote-1),[[2]](#footnote-2):

|  |
| --- |
| #define CGA\_PORT (0x3d4)  static void cga\_output(uint8\_t value, uint16\_t port)  {  \_\_asm \_\_volatile ("outb %0, %w1" : : "a" (value), "d" (port) );  }  static void console\_cga\_set\_cursor(uint8\_t r, uint8\_t c)  {  cga\_output(14, CGA\_PORT);  cga\_output(r, CGA\_PORT+1);  cga\_output(15, CGA\_PORT);  cga\_output(c, CGA\_PORT+1);  } |

Since we are learning how to write device drivers, we should do this the right way! The console code should be in its own .C file, publicly callable functions should in an accompanying .H file, but non-publicly callable functions should be marked “static” and not in the “.H” file.

Furthermore, to minimize the number of global variables, you should have a single “typedef struct” that is contained in .H file, your kernel (not the console.c) should create one of these, initialize it, and then pass it into your console functions. For an extra challenge, we should be able to instantiate 8 of these, one for each page of video memory, and thus have 8 different console pages. Later, we could even switch between them with a keyboard shortcut, such as “Alt-F1” for the first, and “Alt-F8” for the last… you know, like Linux!

# Deliverables and Demos

Arrange a time for us to meet, and show be prepared to show me the following:

1. Show me your console driver in action
2. Give me a tour of your code, show me what you created, and why you created it.
3. I want to see good use of data structures, and use of the static and inline keywords on functions where appropriate.
4. As an overall perspective, I want to see a piece of software that represents the types of things you would expect of a text window – things that are “just handled” by your driver to make it easier for other programmers to use your driver.

Your grade for this part will be determined by the overall quality of your code… having things in a few huge functions will result in

Points: \_\_\_\_\_\_\_\_\_ / 50

1. https://en.wikipedia.org/wiki/Color\_Graphics\_Adapter [↑](#footnote-ref-1)
2. http://www.oldskool.org/guides/oldonnew/resources/cgatech.txt [↑](#footnote-ref-2)