UF UNIVERSITY of FLORIDA
Herbert Wertheim
College of Engineering

Computer Engineering
CEN3907C

Drivers for Operating Systems

Overview

When dealing with low-level programming and/or embedded devices, engineers often must directly interact with devices to send them information and/or poll them for their state. For example, The DOS platform of the 16-bit and 32-bit eras of computing is still in use today, particularly in spacecraft, as the x86 line of processors has been thoroughly tested for radiation-hardening, making CPUs usable in the sometimes hostile environments of space travel. In this activity, students will work in an emulated x86 real mode environment (DOS) to build a set of drivers for mouse input and video output.

Structure

This project involves three major components:

1) Writing driver to program the mouse and receive input
2) Writing driver to program the Video Graphics Adapter (VGA) and display output
3) Testing the drivers via a provided test script

Students are required to provide a Makefile to build their sources using the Open Watcom C/C++ toolchain suite (as provided) and header files, as described, with the appropriate declarations.

Environment

As part of this assignment, students are provided with the Open Watcom C/C++ cross-compiler toolchain for modern Windows platforms (NT-64) and the DOSBox emulator platform. The compiler targets DOS and real mode. As the DOSBox root ("C:\") is an ordinary directory, students can easily move source and binary files into and out of the DOSBox environment. The archive includes the following:

File / Folder	**Description**
`dosbox-x`	DOSBox executable build and 486 configuration
`root`	Root directory (C:\) for FreeDOS-DOSBox
`watcom`	Windows (NT-64) Open Watcom suite for DOS target
`“FreeDOS Environment.bat”`	Batch file to launch FreeDOS environment in DOSBox
`“Windows Watcom Dev Prompt.bat”`	Development prompt for Open Watcom NT-64 suite

Note that while development is in the Windows environment, testing code can only be done within the emulator itself. The DOSBox root is mounted at "V:\" when the development prompt is launched.

Though Open Watcom supports many features of modern C++, some C++11 features have not been implemented. For example, the **`cout`** stream cannot print directly **`std::string`** objects, so the **`std::string::c_str()`** method is required, and **`0`** must be used instead of **`nullptr`**. Additionally, because you have no debugger, we recommend using file logging for debugging.

Specification

Students will complete the mouse driver, video driver, and test program. Drivers will be built as a static library. Prototypes and type declarations should be included in header files.

Mouse Driver

The driver includes functions to directly poll hardware and functions that use BIOS state.

Data Structures

```
struct mouse_state            // Structure to hold device and cursor
data
   bool left, right, middle  // Button state; true is button is
pressed int16_t x, y            // Current cursor position
```

Hardware Polling

```
bool mouse_reset()
```

Initializes the mouse. If successful (mouse is present), returns **`true`**; otherwise, returns **`false`**.

```
int mouse_get_button_presses(int16_t button)
```

Get the presses of the specified **`button`** (left: 0; right: 1; middle: 2) since the last call.

```
int mouse_get_button_releases(int16_t button)
```

Get the releases of the specified **`button`** (left: 0; right: 1; middle: 2) since the last call.

```
void mouse_get_mickeys(int16_t *mickeys_x, int16_t *mickeys_y)
```

Retrieve the mickeys accumulated since the last call and add to values at the pointers provided.

```
void mouse_calculate_pos(int16_t* value, int16_t* mickeys, int16_t
rate)
```

Calculate mouse position value (x or y) based on current **`value`**, **`mickeys`**, and **`rate`**. The **`value`** should be updated for any whole pixel changes, and with the remainder re-stored in **`mickeys`**.

BIOS State

```
void mouse_show_cursor()
void mouse_hide_cursor()
```

Shows or hides the mouse cursor on/from the video display.

```
void mouse_set_limits_x(int16_t minimum, int16_t maximum)
void mouse_set_limits_y(int16_t minimum, int16_t maximum)
```

Set the maximum and minimum position values for the cursor on the horizontal/vertical axis.

```
void mouse_set_position(int16_t x, int16_t xy)
```

Sets the cursor position for use with BIOS-assisted cursor routines.

```
void mouse_get_state(struct mouse_state* state)
```

Obtains the mouse state from the BIOS and stores it in the **`state`** structure.

```
void mouse_set_movement_rate(int16_t rate_x, int16_t rate_y)
```

Set the mickeys-per-coordinate rates for the BIOS-assisted cursor routines.

```
void mouse_set_sensitivity(int16_t level_x, int16_t level_y)
```

Set the mouse sensitivity level for the BIOS-assisted cursor routines (1-100).

`void` **`mouse_get_sensitivity`**`(int16_t *level_x, int16_t *level_y)`
Retrieve the mouse sensitivity level for the BIOS-assisted cursor routines (1-100).

VGA Driver

The driver includes routines to synchronize hardware, set palette colors, and access video memory.

Video Configuration

`void` **`vga_set_mode`**`(uint8_t mode)`
Sets the video mode number of the video graphics adapter.

`void` **`vga_vertical_sync`**`()`
Blocks until the vertical refresh cycle has been completed (to prevent "tearing" on CRTs).

`void` **`vga_set_pallete_mask`**`(uint8_t mask)`
Sets the palette mask for the video graphics adapter (which prevents modification of colors).

`void` **`vga_set_color`**`(uint8_t index, uint8_t r, uint8_t g, uint8_t b)`
Sets palette entry at the specified **`index`** to the red (**`r`**), green (**`g`**), and blue (**`b`**) values. Color band values must fall in the range of `[0:63]`.

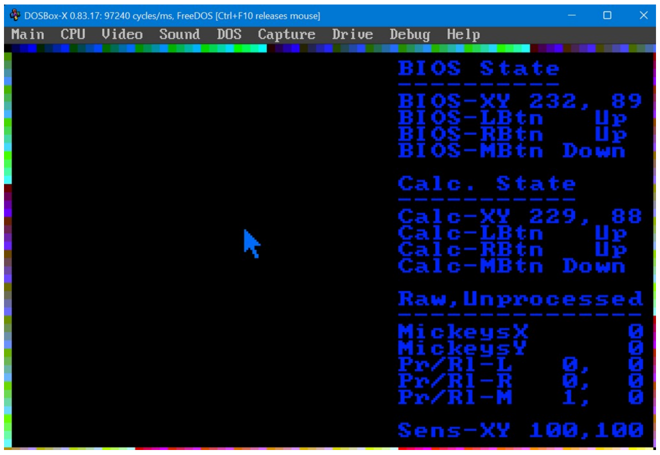
Adapter Query

`void` **`vga_get_color`**`(uint8_t index, uint8_t *r, uint8_t *g, uint8_t *b)`
Retrieves palette entry at the specified **`index`** and stores in the red (**`r`**), green (**`g`**), and blue (**`b`**) addresses. Color band values will fall in the range of `[0:63]`.

`uint8_t *`**`vga_get_address`**`()`
Gets address of video memory for the adapter.

Test Program

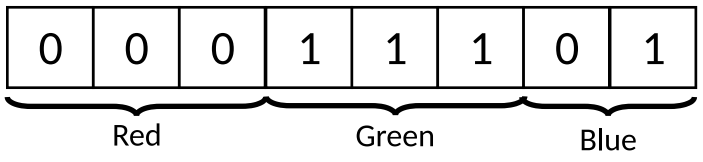
The test program – ***which is provided to students*** – utilizes most of the driver features to test the functionality of your code. It generates a standardized 3:3:2 8-bit RGB palette, displays the colors around the edge of the screen, shows the mouse, and updates the current mouse data on the screen continuously. The hope is that it will help students catch most issues. However, it is critical for students to write and run their own tests; the test program should not be considered exhaustive or sufficient.


DOSBox-X 0.83.17: 97240 cycles/ms, FreeDOS [Ctrl+F10 releases mouse]
Main CPU Video Sound DOS Capture Drive Debug Help

```
BIOS State
----------
BIOS-XY 232, 89
BIOS-LBtn    Up
BIOS-RBtn    Up
BIOS-MBtn  Down

Calc. State
-----------
Calc-XY 229, 88
Calc-LBtn    Up
Calc-RBtn    Up
Calc-MBtn  Down

Raw,Unprocessed
---------------
MickeysX      0
MickeysY      0
Pr/Rl-L   0,  0
Pr/Rl-R   0,  0
Pr/Rl-M   1,  0

Sens-XY 100,100
```

Color Palette

The color palette is taken from the bits of the index in a 3:3:2 RGB format:



Example - Color #29 (0x1D)

To scale colors to a range of **0:63**, red and green values are multiplied by **9** and blue by **21**.

Information Display

Information about the BIOS-assisted cursor / button state, unprocessed raw device data, and calculated cursor / button state is displayed in the right-hand pane of the program, starting at text column 25 in MCGA mode. The “BIOS State” section displays the cursor position and button states derived from the BIOS-assisted cursor routines. When the left button is pressed and released, the “Calc. State” section is updated to show the cursor position and button states based on the raw, but unprocessed, device data (button presses / releases and mickey counts). Mouse movement and/or button presses and releases is cumulatively tracked and displayed in the “Raw,Unprocessed” section until the left button is clicked. At the bottom, the mouse sensitivity rate is displayed; this is taken from the BIOS call to get the sensitivity (i.e., not merely populated directly). The first and last lines are left blank to prevent text from overwriting the palette block displays that are used to verify palette information.

```
01
02 BIOS State
03 ----------
04 BIOS-XY   123,
05 45
06 BIOS-LBtn   Up
07 BIOS-RBtn Down
08 BIOS-MBtn   Up
09
10 Calc. State
11 -----------
12 Calc-XY
13 54,123
14 Calc-LBtn   Up
15 Calc-RBtn   Up
16 Calc-MBtn Down
17
18 Raw,Unprocesse
19 d
20 --------------
21 -
22 MickeysX     -
23 12345
24 MickeysY
25 32767
```

To position text without overwriting graphical elements, the **`<graph.h>`** header is used:

```
_settextposition(2,   25);   //
Row,Col
printf("BIOS State");
```

The output must be flushed before moving to another location due to **`printf()`**’s functionality.

Program Execution

The program is initialized as follows:

1) Initialize mouse. If not found, print **`"Error: mouse not available. exiting.\n"`**
2) Switch to MCGA mode and set palette. Bits for RGB should be taken from index (see below).
3) Display the border colors as 4x4 pixel blocks:
    a. Colors 0:79 should be displayed along the top, from left to right.
    b. Colors 80:127 should be displayed along the left, top to bottom.
    c. Colors 128:175 should be displayed along the right, top to bottom.
    d. Colors 176:255 should be displayed along the bottom, left to right.
4) Set up the mouse / cursor variables as follows:
    a. Position: (50, 25)
    b. X Range: [8, 383]

c. Y Range: [4, 195]
d. Movement Rate (Mickeys / Coord): [x = 2, y = 4]
e. Mouse Sensitivity: [x = 100, y = 100]
f. Show the Cursor

While the program is running, it should perform the following steps in a loop:

1) Fetch mouse BIOS cursor data (position and buttons)
2) Fetch latest mickey counts and button presses and releases for left, right, and middle buttons
3) If right and left buttons have been pressed together, then both released, exit program.
4) If left button was pressed & released, calculate position & buttons from unprocessed data.
5) Display the latest data in the right-hand pane.

Analysis

Use the test program to answer the following questions within the report submitted.

1) If the mouse moves quickly and/or out of bounds, the BIOS coordinates and those calculated from device hardware packets (mickeys) will differ. How and why do you think this happens?
2) If the BIOS already has functions for tracking coordinates, button states, and for setting the cursor, why would a developer find it useful to instead use device hardware packets instead?

Submission

Your submission will be composed of the following elements:

- Report (**DriversReport.pdf**) on Canvas
- Source and build files as a zip archive file (**drivers.zip**) on Canvas

Report

The report explains the process used to develop the program and tools used. It describes how testing was performed and any known bugs. The report should be 500 words or fewer, cover all relevant project aspects, and be organized and formatted professionally – ***this is not a memo!***

Compressed Archive

Your archive should this structure:

```
drivers.zip
 └→ tests.c
 └→ drivers (directory)
      ├→ Makefile
      ├→ mouse.h
      ├→ vga.h
      └→ (Other sources/folders)
```

We will use these commands to test from the Dev Prompt:

```
V:\project$ unzip drivers.zip
V:\project$ cd drivers
V:\project\drivers$ wmake
V:\project\drivers$ cd ..
V:\project$ wcl tests.c drivers/drivers.lib
```

This should generate the “**`tests.exe`**” executable.

We should be able to use your library in our own programs (for further testing) as follows:

```
V:\project$ wcl our_proggy.c drivers/drivers.lib -ml
```