Cps 230: Computer Systems

Team Project: Multitasking Kernel

Team Members: Stephen Sidwell and Nathan Collins

12/7/2016

**Overview:** The goal of this team project was to write a multitasking kernel from scratch for an IBM 5150 PC. The IBM 5150 was an early PC that was based on Intel’s 8086 architecture, so the kernel had to be written in 16-bit x86 assembly and fulfill two key requirements. The first key requirement of the kernel is that it is to run independently of any operating system that is, the kernel must be boostrapped on startup to run independently. The second key requirement of the kernel is that it must demonstrate multi-tasking, either cooperative or preemptive. Essentially, the kernel must run several tasks in parallel. Overall, the kernel is to be demonstration of everything the team members have learned in the class and should showcase both their knowledge and creativity in creating x86 programs.

**Results:**

**Require Features Implemented:**

* Multi-tasking (4+ task possible)
* Bootstapped (Bootloader and Bootstrapper implemented)
* Each task has 256 bytes of stack space

**Elective Features Implemented:**

* Timer Preemption
* Graphics Demo (Mandlebrot Set, Moving blocks)
* Music Demo
* Real-Mode C (Open Watcom compiled code)

**Details:**

**Multi-tasking:** Our kernel uses timer preempted multitasking. The basics of our multi-tasking involve using our own interrupt handler for the IVT 8 timer interrupt which triggers about 18 times a second. Our interrupt handler uses an array of stack pointers to keep track of each task’s execution stack, switches the stack pointer to the next stack pointer in the array on every interrupt call, and wraps the index of the stack array as necessary.

**Bootloader:** Our bootloader functions fairly simply by using BIOS interrupts to read our kernel code from the disk into a particular memory location and then simply jumps to that location in memory to begin execution of the kernel code.

**Mandlebrot Demo:** This first graphics demo draws a graphical representation of the Mandlebrot set on the left half of the screen. This representation is generation by a simple iterative formula to determine the color of each point on the screen.

**Moving Boxes 1 and 2:** The second and third demos are a pair of moving white boxes on the right half of the screen. The boxes simply move from left to right and wrap around the screen when they reach the edge.

**Palette Shift Demo:** The fourth demo is a simple counter of sort to shift the palette color of the Mandlebrot set after each complete draw. Essentially, it changes the colors of the Mandlebrot set.

**Music Demo:** The fifth demo is music which plays in the background of the kernel. The demo plays the University hymn and a classical carnival theme.

**Contributions:**

**Stephen Sidwell**

* Bootloader
* Mandlebrot Set Demo
* Timer Preemption
* Real-Mode C debugging (Mainly data segment stuff)
* Build automation

**Nathan Collins**

* Music Demos
* Moving boxes demo
* Real-Mode C main implementation
* Build automation