Introduction

Kernel development is not an easy task. This is a testament to your programming expertise: To develop a kernel is to say that you understand how to create software that interfaces with and manages the hardware. A kernel is designed to be a central core to the operating system - the logic that manages the resources that the hardware has to offer.

One of the most important system resources that you need to manage is the processor or CPU - this is in the form of allotting time for specific operations, and possibly interrupting a task or process when it is time for another scheduled event to happen. This implies multitasking. There is cooperative multitasking, in which the program itself calls a 'yield' function when it wants to give up processing time to the next runnable process or task. There is preemptive multitasking, where the system timer is used to interrupt the current process to switch to a new process: a form of forcive switch, this more guarantees that a process can be given a chunk of time to run. There are several scheduling algorithms used in order to find out what process will be run next. The simplest of which is called 'Round Robin'. This is where you simply get the next process in the list, and choose that to be runnable. A more complicated scheduler involves 'priorities', where certain higher-priority tasks are allowed more time to run than a lower-priority task. Even more complicated still is a Real-time scheduler. This is designed to guarantee that a certain process will be allowed at least a set number of timer ticks to run. Ultimately, this number one resource calculates to time.

The next most important resource in the system is fairly obvious: Memory. There are some times where memory can be more precious than CPU time, as memory is limited, however CPU time is not. You can either code your kernel to be memory- efficient, yet require alot of CPU, or CPU efficient by using memory to store caches and buffers to 'remember' commonly used items instead of looking them up. The best approach would be a combination of the two: Strive for the best memory usage, while preserving CPU time.

The last resource that your kernel needs to manage are hardware resources. This includes Interrupt Requests (IRQs), which are special signals that hardware devices like the keyboard and hard disk can use to tell the CPU to execute a certain routine to handle the data that these devices have ready. Another hardware resource is Direct Memory Access (DMA) channels. A DMA channel allows a device to lock the memory bus and transfer it's data directly into system memory whenever it needs to, without halting the processor's execution. This is a good way to improve performance on a system: a DMA-enabled device can transfer data without bothering the CPU, and then can interrupt the CPU with an IRQ, telling it that the data transfer is complete: Soundcards and ethernet cards are known for using both IRQs and DMA channels. The third hardware resource is in the form of an address, like memory, but it's an address on the I/O bus in the form of a port. A device can be configured, read, or given data using it's I/O port(s). A Device can use many I/O ports, typically in the form of ranges like ports 8 through 16, for example.

Overview

This tutorial was created in an attempt to show you, the reader, how to set up the basics for a kernel. This involves:
1) Setting up your development environment