CS411 Operating Systems II Fall 2008

Project #2

Due 6:00 am, Saturday, October 18, 2008

Objectives:

1. understand the basics of the Linux Scheduler
2. understand how a shortest remaining time first scheduler operates
3. be able to use the Linux kernel's Linked List API
4. practice kernel coding skills

Introduction:

The CPU scheduler controls when processes are allowed to run and is the primary means by which the illusion of multitasking is maintained. In Linux, the CPU scheduler provides a means for the kernel to organize processes and control how and when they are allowed to execute within the CPU. This job is important to maintain fair usage of system resources amongst all running process, maintain interactivity, and ensure that critical processes are allowed access to the CPU in a timely fashion.

The scheduler you write will not be run within the Linux kernel. Instead, a virtual machine has been provided for you to use in writing the scheduler. The virtual machine provides all the same API calls used by the kernel for scheduling. Unlike programming within the kernel, you can use standard C functions, as well as debug the assignment in an environment of your choice. All relevant kernel structures and macros have been provided for you, and you will not need any additional source code outside of the package we provide.

This project utilizes a virtual machine because the current Linux scheduler is extremely complex. A realistic scheduler for the Linux kernel must be able to handle numerous types of process behaviors, multiprocessor support, and memory management. In order to make this project possible for the class, a virtual machine was written which provides a single-processor environment that does not require extensive memory management or interrupt handling. This will allow you to focus on learning how a scheduler works without having to also understand the additional complexities of a full operating system. This being the case, we feel that the virtual machine provides an experience nearly identical to programming within the Linux kernel. Every function and structure has been duplicated from the original kernel source code, and the virtual machine executes calls in the same fashion as the Linux kernel.

Materials:

1. Download the source package from http://classes.engr.orst.edu/eecs/fall2008/cs411/proj02. This contains
   a. the framework for the project
   b. a default *Makefile* provided for your convenience
   c. a guide to using the virtual machine

Tasks:

1. Write a scheduler implementing the **shortest remaining time first** (SRTF)algorithm.
   a. You should begin by familiarizing yourself with the `schedule.c` and `schedule.h` files. Each file is documented extensively to help you understand the roles of each function used within the scheduler. The `schedule.h` file lists which functions are **required** to be implemented by your scheduler. The virtual machine relies on these functions to perform its tasks. All other functions are optional, but have been included because they coincide with methods used in the Linux scheduler. (*It is highly recommended that you examine the* `sched.c` *file in the Linux kernel. At several thousand lines of code, this file can be daunting, but the source code will give you an excellent idea of what each function provided is supposed to accomplish.*)
   b. Make sure you understand what the emulated system calls provided by the virtual machine do. These calls are necessary for the proper functioning of your scheduler. These calls are listed within `schedule.h`.
   c. Familiarize yourself with the data structures defined in `schedule.h`. These structures are simplified versions of those found within the Linux kernel, but operate in an identical fashion.
   d. You may implement functions in any order you wish, though some are more necessary than others at the beginning. *initschedule* and *killschedule* provide for the setup and tear-down of the scheduler by the virtual machine, and are the bare minimum functions necessary to get the virtual machine to run properly. Next of importance are *activate_task* and *deactivate_task*. The *schedule* function is also a good place to start, as it contains the core functionality of the scheduler.
   e. Once you have enough code in place, you can compile it and link it to the virtual machine. The resulting code can be run from the command line. The virtual machine runs off of a process profile which controls when and how to create processes, as well as what type of processes to create. We have provided a test profile for you to use along with the expected output from the virtual machine. You may use these to test your scheduler. You can also create your own profiles to test a variety of situations. For more information consult the virtual machine guide.

Notes and Caveats:

1. There will probably be a help session for this project. It is suggested that you attend this if you are in any way confused about how the scheduler works. ***Start your work early!!*** The scheduler is very large and complex, and even the simplified environment we have provided will take you some time to become familiar with.
2. Do not hesitate to use the class mailing list. Open discussion is encouraged. The TAs are monitoring the list and will respond to your questions. Remember that you can fulfill part of your open source requirement for the course by contributing to the mailing list and the student wiki.

Useful resources:

There is a link on the class wiki to a cross-referenced HTML version of the kernel sources. The following files will be particularly useful:

- `kernel/sched.c`
- `kernel/sched.h`

What to turn in:

1. Create a gzipped tar archive containing the items below and submit it at http://engr.oregonstate.edu/teach.
   a. `schedule.c`
   b. `schedule.h`
   c. Any additional files you created or edited
   d. Your Makefile
2. Each member of your team must individually write a separate "review" document. (See the course wiki for the requirements.) This document must be submitted by 11:59 pm on Monday, October 20, at http://engr.oregonstate.edu/teach.
3. Before 9:00 am on Monday, October 20, your team must submit a signed hardcopy of your completed Credit Distribution Agreement. (See the link on the course wiki)

Credits:

Ian Oberst:

- Project design and testing
- VM / library development
- Project Presentation (in class, October 8)
- Project Seminar (TBA)