Fair Share Scheduler
CS 416: Operating System Design
Due: 11:55 PM, 03/27/2013

1 Announcements

- Assignment open date: February 21, 2013
- Group size: 4 people per group
- Due date: March 27, 2013 (11:55 pm)
- Submission: Electronically, via sakai
- Logistics: The assignment should be carried out in iLab machines with provided virtual machine

2 Background

This assignment requires you to modify scheduler in the Linux kernel (version 2.4.24) to ensure fair share per user. This means that all of a user's processes will share that user's portion of the CPU time and not infringe on other user's time. As an example, if user A has four times as many processes running as user B, each of user A's process would receive only one fourth of the allocation per second, say, given to processes for user B, so that users A and B split the CPU resource equally. However, a user may want to increase the resource share of an individual process. A system call is a convenient method to facilitate this. You need to implement a system call that will make double the share of the process that invokes the system. For example, under a completely even sharing scheduler, there are two users logged into a system, each running one process. User A's process will get scheduled 50% of the time, and User 2's process will get scheduled the other 50% of the time. If User 2 starts 2 other processes, User 1's process will continue to get scheduled 50% of the time, but each of User 2's processes will run 16.33% of the time. However, if a process of User 2 invokes the system call, it will receive 25% of the time and other two processes with receive 12.5% each.

3 Assignment

Modify the source code of Linux kernel 2.4.24 to implement the described fair scheduler and construct a set of example to demonstrate that the scheduling is indeed fair. In addition, add a new system call to the kernel that tells the scheduler that the process calling it is to receive twice as much time, within the user's allotted time slot.

4 Recommended Procedure

4.1 Kernel Modification

- Read over the kernel code that handles scheduling. You can find the entire kernel source online at: http://lxr.fsl.cs.sunysb.edu/linux/source/?v=2.4.24

At a minimum you should read kernel/sched.c, kernel/fork.c, kernel/user.c, and include/linux/sched.h. You should also look at include/asm-i386/unistd.h and arch/i386/kernel/entry.S, as you will need these files to add the system call.

- Develop your scheduling algorithm (the ready queue is a simple linked list, so you just need to figure out how to schedule the list of processes).
- Figure out what code needs to be changed/added/replaced to implement your algorithm.
- Implement your changes/additions/replacements.

4.2 System Call

- A system call creates a trap and the source is looked up at a vector table. So, you need to add an entry in the corresponding file (arch/i386/kernel/entry.S). A sample entry would look like this,

  .long SYMBOL_NAME(sys_myincreaseshare)
- You also need to add an entry in include/asm-i386/unistd.h. The entry should start with "__NR_" and end with a number. To avoid conflict with existing system call number, assume something higher than 252 (e.g. 259).
- Place your source code file in a meaningful directory. You would have to edit the corresponding Makefile as well. For example, if you add your source in the directory "kernel", you have to edit the file "kernel/Makefile"

  obj-y := sched.o ... myincreaseshare.o
- In order to be linked properly, your system call needs the word "asmlinkage" prepended to their function header and "sys_" prepended to the name. For example, you would have:

  asmlinkage int sys_myincreaseshare(paremeter list) {
  /* implementation of myincreaseshare, modification of your data structure */
  }
- Lastly, the user "stub" can be generated so that a user program can use your system call. There are macros defined in include/asm-i386/unistd.h. The format is "_syscallN(return type, function name, arg1 type, arg1 name ...)" where "N" is the number of parameters. You can add this declarations either in header file or separate source file.

5 Logistics

5.1 Virtual Machine

A Virtual Machine (VM) image is provided for you to be able to change the kernel and test. The VM image is stored at freespace of java.rutgers.edu. You will need a VMware player run it. All iLab machines have VMware player installed. To get the VM image and run on any iLab machine, follow these steps,

- scp <your-net-id>@java.rutgers.edu:/.freespace/CS416SP13/finaldebian3.tar.gz <your-current-directory>
- tar -zxvf finaldebian3.tar.gz
- vmplayer finaldebian3/finaldebian3.vmx

The root password of the VM is "admin". The VM also has a user account (name:"user", password:"user"). You can add more users using the command "useradd". There is a fresh kernel 2.4.24 source in /home/user. You can use 'apt-get' to install aditional software if you like. The source list will get the software that are compatible with Debian 3.1. You can also ssh/scp from the VM to the host machine/iLab machines.

5.2 Compiling and Booting Linux Kernel

To recompile the kernel after after a change, follow these steps when your current working directory is <your_working_directory>,

- Modify "<your_working_directory>/Makefile", set "EXTRAVERSION = test"
- make oldconfig (press enter to accept the default answer for all questions)
- make dep (You will typically have to do make dep once, but once in a while you may need to clear old old dependancies with a make clean and do make dep again)
- make bzImage
- make modules
- make modules_install
- Note that your VM uses GRUB for boot loader. Edit the "/boot/grub/menu.lst" file and add one boot entry for the new kernel

  ```
  title Kernel 2.4.24 for CS416 Project 2
  root (hd0,0)
  kernel /boot/vmlinuz-2.4.24test root=/dev/hda1 ro
  savedefault
  boot
  ```

  If you don't want to select kernels during booting time, set the default entry to the new entry number. Numbering starts from 0.

  default 2 (OR replace 2 by the correct number)
- reboot
- When the machine prompts you, make sure you are selecting the right kernel
- Type uname -a to verify that the correct kernel is running

5.3 Patch Generation

Generate a patch of your modification using the following command. Please try to create the patch when your modified directory does not have any object file. Generally this means, you take a fresh copy of kernel and copy only the modified source/header files to the copy. This is your modified directory.

```
diff -crB <fresh-kernel-directory> <your-modified-directory> > patch_name.patch
```

To test that you are submitting the right patch, apply the patch to a fresh kernel, build it and test it after booting. The command to apply a patch is,

```
patch -p1 -i patch_name.patch
```

6 Submission

You have to submit your report and patch in sakai. Please submit one tar file named "project2.tar". Upon extraction, it should should give a directory named "project2". The directory project2 should contain 2 files named "patch_name.patch" and "report.pdf" (All names are case-sensitive). You can put additional files in the directory but mention them in the report.

7 Grading

This assignment is worth 100 points that are distributed in the following way, 50 points for the scheduler, 20 points for the system call, 10 points for performance and 20 points for the report. The group(s) that will implement their scheduling algorithm with the lowest time complexity will get 5 points, then the second group and so on. Similarly, the group(s) that will use the lowest amount of dynamic memory will get 5 points and so on.