Print

lab 3 : scheduling algorithms in the MentorOS

Lab 3: Mon 22 Jan 2024

Scheduling Algorithms in Mentor OS

Today we are going to continue working with the Mentor operating system, MentOS. This is a cut-down version of Unix, ideal for educational usage.

We want to build this OS and investigate its process scheduling algorithms.

- Project website: https://mentos-team.github.io/
- Git repo: https://github.com/mentos-team/MentOS

Step 1a: Clone & Build for Ubuntu
This step is the same as last week - so skip it if you have already built MentOS for your Linux machine...

On your terminal, clone the MentOS repository from github into a working directory.

git clone https://github.com/mentos-team/MentOS

To build MentOS on Ubuntu, you can follow these instructions here: https://github.com/mentos-team/MentOS#3-prerequisites

Note the need to install a **cross-compiler** for x86 since we are running MentOS on qemu with 32-bit x86 emulation.

Use `cmake` to generate the makefiles in a build directory, then run `make` then run `make filesystem` (both in your build directory).

Step 1b: Clone & Build for macOS
(Skip this step for Ubuntu)
Last week we had a problem building for macOS. I have a partial workaround in a forked repo so please follow these instructions and see how you get on. You need homebrew installed, along with the brew formulas specified here:

- git
- gcc
- nasm
- make
- cmake
- e2fsprogs
- i386-elf-binutils
- i386-elf-gcc
- qemu-system-i386

Now clone my forked MentOS repo:
git clone https://github.com/jeremysinger/MentOS

cd MentOS
mkdir build && cd build
cmake -DCMAKE_TOOLCHAIN_FILE=../TC-x86.cmake ..

This generates the Makefiles - you need to add some exes to the PATH when you invoke make ...

make PATH=/opt/homebrew/Cellar/i686-elf-binutils/2.41_1/i686-elf/bin/:/opt/homebrew/Cellar/e2fsprogs/1.47.0/sbin/:$PATH

Then make the filesystem (same command and path, but with the `filesystem` target)

make PATH=/opt/homebrew/Cellar/i686-elf-binutils/2.41_1/i686-elf/bin/:/opt/homebrew/Cellar/e2fsprogs/1.47.0/sbin/:$PATH filesystem

Step 2: Test you can boot MentOS in qemu

Then you can run qemu with another make command:

make qemu. # <--- on Ubuntu

or
make PATH=/opt/homebrew/Cellar/i686-elf-binutils/2.41_1/i686-elf/bin/:/opt/homebrew/Cellar/e2fsprogs/1.47.0/sbin/:$PATH qemu # <--- on macOS

and it should bootup. Check you can login with root/root username/password combo.

Once you have logged in, run the `logo` command to pretty-print the MentOS ascii logo. You can also run `ls /` to list the root directory in the filesystem.

You can quit qemu to terminate the OS run. (On macOS, some programs may still cause MentOS to die unexpectedly ... sorry.)

Step 3: Create a CPU hogging process

You can write a nice C program to model a CPU-intensive process. How about something like this pseudo-random number generator? Add this file to the MentOS/programs directory and add it to the CMakelists.txt file in the programs directory, as the last entry in the PROGRAM_LIST. (Full details of how to do this here.) You now need to delete your build directory and re-run cmake / make / make filesystem / make qemu to build & boot a new OS image which includes your busy program in /bin

Step 4: Run the CPU hog

Login to MentOS on qemu and run
/bin/busy
maybe giving it an integer parameter to make it more random. How long does it take to complete?

You can run several concurrent busy processes with:
/bin/busy &

and they should all run together as background tasks.

Step 5: Explore the CPU Scheduling Policy

Check out the source code for the process scheduling in MentOS/mentos/src/process/scheduling*.c … inspect how this works. Which scheduling algorithm does MentOS use by default? Which other algorithms are available? Perhaps add some debugging printf (pr_notice) statements in the scheduling code and recompile and look at the output. What do you see?

Step 6: Change the CPU Scheduling Policy

To modify the CPU scheduling policy, you need to delete your build and re-run cmake with a scheduling policy specified as a compiler flag. See instructions here. How do the different scheduling policies affect the behaviour of concurrent busy processes?

Step 7: [Advanced / Optional / probably Linux only] Compute the Load Average

The *load average* for a POSIX operating system is the average number of processes in the ready/runnable queue at each scheduling decision point. You can normally see the load average with the `uptime` command on a Linux machine. What is your load average? Could you instrument / adapt the MentOS process scheduling code to compute the load average for MentOS on qemu? See wikipedia for more details.