Jeshua Kracht and Jacob Petrovic

CS 4500/5500 Operating Systems

Dr. Jia Rao

Project 4 – WRR Scheduler

For Project 4, we were tasked with created a new Weighted Round Robin CPU scheduler for our Linux Kernel. We first changed what scheduler the new kernel would need to use, as outlined in part 1. In the **sched\_rt.c** file, we modified the file to use the wrr\_sched\_class for the type of scheduler. The **sched\_class rt\_sched\_class** also had the enqueue and dequeue we need later on to schedule our processes. Since we were working on just getting our **wrr\_class** to be called and work at all, we didn’t worry about multi-core and load balancing issues as of yet.

To implement the WRR scheduling, we needed to have some type of back end scheduling system to update, so we had to work with both part 1 and part 2 together to get any reasonable data out or our kernel. We used the following basic WRR information as our source to mimc in the Linux kernel:

Supposing that there is a server set *S* = {S0, S1, …, Sn-1};

W(Si) indicates the weight of Si;

*i* indicates the server selected last time, and *i* is initialized with -1;

*cw* is the current weight in scheduling, and cw is initialized with zero;

max(S) is the maximum weight of all the servers in S;

gcd(S) is the greatest common divisor of all server weights in S;

while (true) {

i = (i + 1) mod n;

if (i == 0) {

cw = cw - gcd(S);

if (cw <= 0) {

cw = max(S);

if (cw == 0)

return NULL;

}

}

if (W(Si) >= cw)

return Si;

}

<http://kb.linuxvirtualserver.org/wiki/Weighted_Round-Robin_Scheduling>

It became pretty clear though that the algorithm was going to be much more complicated in the linux kernel though. We had to modularize and have the majority of our scheduling logic take place in the **sched\_wrr.c** class we made. In here we are essentially adjusting the timing, inserting (enqueue), removal (dequeue), and load balancing logic. In the **sched.c,** we essentially initiate a system call that sends tasks by their PIDs and weights to be scheduled via the WRR. We have struct in **sched.c**  that holds the data of the overall queue, giving a central place to reference and hold data about the run queue using the WRR scheduling.

In **sched.c** we also have some logic to add to the total weight of the tasks to complete, but the actual priority and rescheduling will occur within the **sched\_wrr.c.** We simply look at how much time it will take each process to finish, and with WRR we can decide that the process with the least amount of time left needs to be scheduled, thus the priority will changed based on the changes and newly scheduled process. In **sched.c** we also set the affinity of each process to run on the same processor. It’s achieved by fetching the same **cpumask** for each of the processors, and sending that mask and the pid of the process to the **sched\_setaffinity** function. The difficulty of this part was more in that the function is outdated and has changed massively over previous version, and thus it took time to research and find where else in the kernel we could get the **cpumask** of a specific core.

Our test program titled **set\_wrr\_scheduler.c** simply creates new processes through forking, creating a total of 7 processes to schedule. We simply send the PID’s and weights of the processes to the **sched.c** class to be scheduled, and of course then the **sched\_wrr.c** class actually schedules and balances the load. We are pretty confident that the scheduler is working for the most part. It’s really difficult to debug because tracing data without a debugger and having to recompile the linux kernel every time can waste a lot of time and break kernels. For the actual load balancing of the weights, we are unsure how close we really are. We attempted to balance to load according to time slice and the weight we sent through, but we imagined that the weight was more of a test for how if we could retrieve the weight again from our structs and the scheduler. Our test program will use the “top” command to print out 7 times all the processes, which allows the user to see how the process CPU load changes over time. In our example, the CPU load doesn’t really change as we believe it should, but the answer to our problem may be that the way we are balancing weights needs to be fleshed out more in the **sched.c** according to the algorithm posted above.

Overall this was a pretty painful experience, but mostly just for the debugging. Implementing a weighted round robin algorithm would not be incredibly difficult in simulation, but the time and effort put into debugging such a massive build like the linux kernel can be frustrating, especially when the kernel is older and documentation is limited.

We learned a lot from this project, even if we didn’t get it 100% working. It’s clear to see that the template for implementing a different scheduler is there for linux, it’s just weather users are willing to debug through many hours to get it working, and that’s just for a simple scheduler like WRR.