Name: Calvin Deutschbein

User Weighted Round Robin

Methodology:

User weighted round robin was implemented as a kernel module in uwrr.c and compiled and inserted into the kernel with the reset.sh script and the makefile. Uwrr.c was based off of pre-existing kernel modules and used the rebound write function to simulate system call. In addition to the "uwrr {n}" call specified in the assignment, an "enable" command to mark a task as eligible for scheduling and a "ping" command to test that the module was live were written. These were used to validate the module for testing. Not that the scheduler also printed timestamps to the /var/log/kern.log with printk commands on every calling

Testing was performed by the test.sh file that used 8test.c (based on the provided spinner.c) to create multiple processes with different weights to call the module. Synchronization was achieved by spinning in the module until all 8 tasks had been scheduled in the module. This light weight solution seemed highly effective compared to including additional weight queue code appears to have had no negative impact on the results.

The testing was performed on the provided virtual machine without any other user processes running. Unfortunately the recommended command:

```
% sudo echo -1 >/proc/sys/kernel/sched_rt_runtime_us
```

was unsuccessful, as were attempts to debug. (Author's note: I regret to say I forgot about this command I was too close to due time to get adequate help on this issue.)

For a note on implementation, weighting was determined by modifying the time_slice attribute of the struct sched_rt_entity element rt_se of the struct task_struct associated with the current process. This was performed in a wrapper around the pre-existing task_tick function. Each time a process was first called after another process had been executing, it's time_slice was scaled to be its weight times the minor frame size of ten milliseconds required by the specification.

For the experiment, weights were selected as the first 8 even numbers for no particular reason at all.

Data:

I refitted the provided spinner.c code in 8test.c to run for only 10 seconds given the amount of data that was being generated. When considered iteration bound, rather than time bound, loops, each process was permitted 500000 iterations.

Weight	Time-bound Iterations	Iteration-bound Times (s)
2	77846	9
4	272405	10
6	358514	9
8	311065	10
10	387214	7
12	445743	5
14	544486	6
16	620008	7

Results:

While not exactly displaying a ratio in perfect proportion with the weights, higher weight does, in general, results in more work being done in less time, or in the same amount of work being done more quickly. Note that in the iteration bound case, as processes exited as soon as completing, there was a bit of clumping in completion time. I suspect processes exiting with potentially large amounts of time remaining allocated to them could also be responsible for some irregularities.

Perhaps a more interesting metric can be done by dividing by weight (iterations rounded to integer, seconds to hundreths):

2	38923	4.50
4	68101	2.50
6	59752	1.50
8	38883	1.30
10	38721	0.70
12	37145	0.42
14	38892	0.43
16	38751	0.44

This shows weights do tend to highly accurately predict performance in general though smaller weights may perhaps have too small of a granularity.

Conclusions:

Given the degree of amateurism in implementation, the degree of consistency especially in the four highest weights is truly remarkable. There may be a variety of systems things going on at lower levels than have been adequately controlled and the erratic results at lower weights likely results from complications of this nature and should not necessarily reflect poorly on the operating system. Overall, the kernel allocates time slices highly accurately.

Time Slice Analysis (Provisional Extra Credit Attempt?):

Overall, there were only 37 context switches in the 10 seconds with the order also proceeding as follows: 10 - 2 - 16 - 8 - 12 - 14 - 4 – 6. This repeated 4 times and then 14 ran one additional time and 4 and 6 each ran two additional times. This goes a long way to explaining why 4 and 6 had such unexpectedly high iterations per weight but should only explain an upward skew of up to 50% especially in the phase of 14 remained on trend. The raw data extracted from kern.log is included in slices.csv but not graphed (Author's note: I couldn't get a program working that made it look nice at all.)

Time slice over weight in milliseconds was always within 5% of 40 except the final 14 weighted slice which was 2.9 instead. This further explains the deviations seen by weights 4 and 6 and gives the desired ~50% error expected for 6 though 4 still seems unusually high.