Assignment – Homework 3
Class – Computer Science 446: Section 1001
Author - Samuel Mouradian
Professor – Christos Papachristos
Assignment Due Date – 10/16/2024

Homework 3 Questions

1. Run the program with an amount of Threads as many as the number of CPU's of your system (if you don't know you can use the **nproc --all** command in your terminal, or the **sysconf(_SC_NPROCESSORS_ONLN);** syscall in C. Write your observations.

    Each thread is executing on a different processor, and that is, therefore, displaying consistent progress bars.

2. Run the program again with the same amount of Threads. Open a different terminal and issue the command:
    watch -n.5 grep ctxt /proc/&lt;pid&gt;/status
    This allows you to see the number of Context Switches (voluntary and involuntary) that a Task with PID &lt;pid&gt; has undergone so far and get an update of it every 0.5 seconds. You can find the PID of a specific task through pstree -p, or more easily by the running sched.c which should now be printing out the PID of each thread alongside each progress bar.
    Now (as you are observing the Context Switches of a specific Thread), switch its Scheduling Policy to a Real-Time one. Try both Policies, and 1-2 different Priority Levels. Write the sequence of commands you used for this question, and your observations.

    Code used:
    ```
    pstree -p
    watch -n.5 grep ctxt /proc/32304/status
    chrt -f 1 32304
    chrt -r 1 32304
    ```

    Switching to a real-time policy has created an increase in context switches for the thread. There also seemed to be a decrease in latency with the real-time policy.

3. Run the program again with the same amount of Threads. Create a system cpuset with all CPUs except 1 (as described in Method b) in the section about Linux Multi-Processor

Control) and move all Tasks (all User-Level and Kernel-Level ones that are possible to move) into that set. Create a dedicated cpuset with only the CPU that is excluded from the system one. Move one of the Threads of sched.c to the dedicated cpuset. Write the sequence of commands you used for this question, and your observations.

Code used:

```
sudo mkdir /sys/fs/cgroup/cpuset/my_cpuset
echo 0-<num_cpus_excluding_1> | sudo tee /sys/fs/cgroup/cpuset/my_cpuset/cpuset.cpus
echo 1 | sudo tee /sys/fs/cgroup/cpuset/my_cpuset/cpuset.mems
sudo mkdir /sys/fs/cgroup/cpuset/my_dedicated_cpuset/cpuset.cpus
echo 1 | sudo tee /sys/fs/cgroup/cpuset/my_dedicated_cpuset/cpuset.cpus
echo 1 | sudo tee /sys/fs/cgroup/cpuset/my_dedicated_cpuset/cpueset.mems
echo 32304 | sudo tee /sys/fs/cgroup/cpuset/my_dedicated_cpuset/cgroup.procs
```

The thread that moved to the dedicated CPU had great performance, but the remaining thread that was grouped with all the other tasks had poor performance.

4. While 3) is still executing with one Thread on the dedicated cpuset, observe the Context Switches of that Thread with:

   watch -n.5 grep ctxt /proc/&lt;pid&gt;/status

   and then change its Scheduling Policy to a Real-Time one at 1-2 Priority Levels. Write the sequence of commands you used for this question, and your observations.

Code used:

```
watch -n.5 grep ctxt /proc/32304/status
chrt -f 1 32304
chrt -r 1 32304
```

The context switch count remained lower while one thread was exclusively using CPU 1. Once it went to a real-time scheduling policy, the context switch count rose, because it was competing with the other thread.