KERNEL OPTIMIZATION

OPERATING SYSTEMS
Prof. Víctor Manuel Rodríguez Bahena

Enrique Anaya Bovio - A01630317

Abstract

The kernel is a computer program that has complete control over the operating system. It handles the input/output requests and translates them into data-processing instructions for the CPU. It handles not only the memory, but also peripherals.

The kernel controls the tasks that are managed in the running system, some of these are running processes, hardware management and handling interrupts, in the kernel space. On the other hand, the user performs in the user space. This separation helps to prevent data interfering that can cause instability and slowness, or worse, malfunctioning application programs that can crash the entire operating system.

The process scheduler decides which is the next task to run. In the following project we analyzed the behavior of the scheduler changing the default value of the runtime scheduling, this value is 950000μs for the scheduler real time running variable. According to this, 5% of the CPU time is reserved for processes that are not running under a realtime or deadline scheduling policy. On this project, this value specifies how much of the period time could be used by real-time and deadline scheduled processes on the system.

Theorical Framework

The Kernel

The kernel is the central module of an operating system. It is the part of the operating system that loads first, and it remains in main memory. The kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system.

The kernel is responsible for memory management, process and task management, and disk management. The kernel connects the system hardware to the application software.

The Scheduler

The scheduler is in charge of keeping busy the CPUs in the system. The Linux scheduler implements a number of scheduling policies, which determines how long a thread runs on a particular CPU core and when it runs.

The scheduling policies are divided into two major categories:

Realtime policies:
- SCHED_FIFO
- SCHED_RR

Normal policies:
- SCHED_OTHER
- SCHED_BATCH
- SCHED_IDLE

SCHED_FIFO policy

In the Linux kernel, the SCHED_FIFO policy includes a bandwidth cap mechanism. This protects realtime application programmers from realtime tasks that might monopolize the CPU. This mechanism can be adjusted through the following /proc file system parameters:

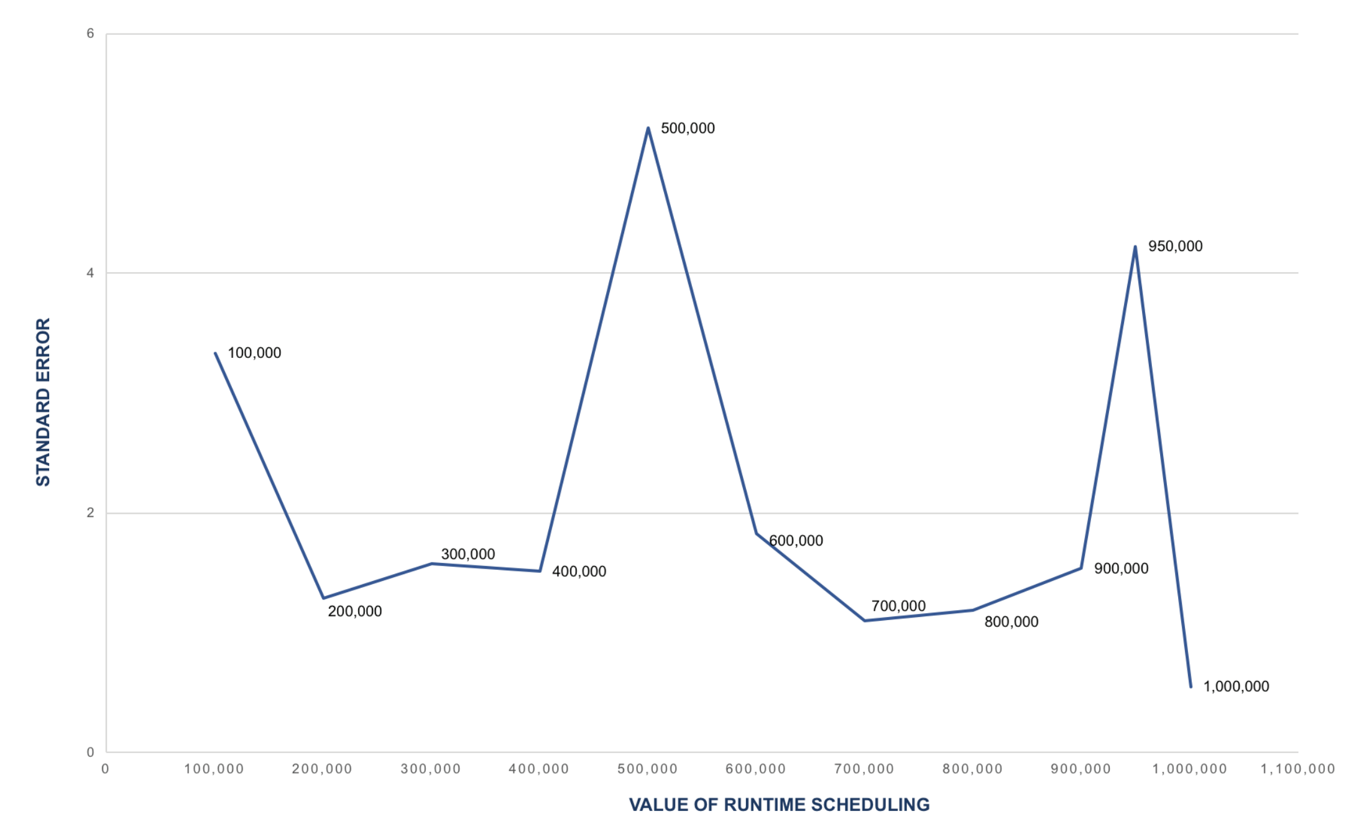
/proc/sys/kernel/sched_rt_period_us
Defines the time period to be considered one hundred percent of CPU bandwidth, in microseconds ('us' being the closest equivalent to 'μs' in plain text). The default value is 1000000μs, or 1 second.

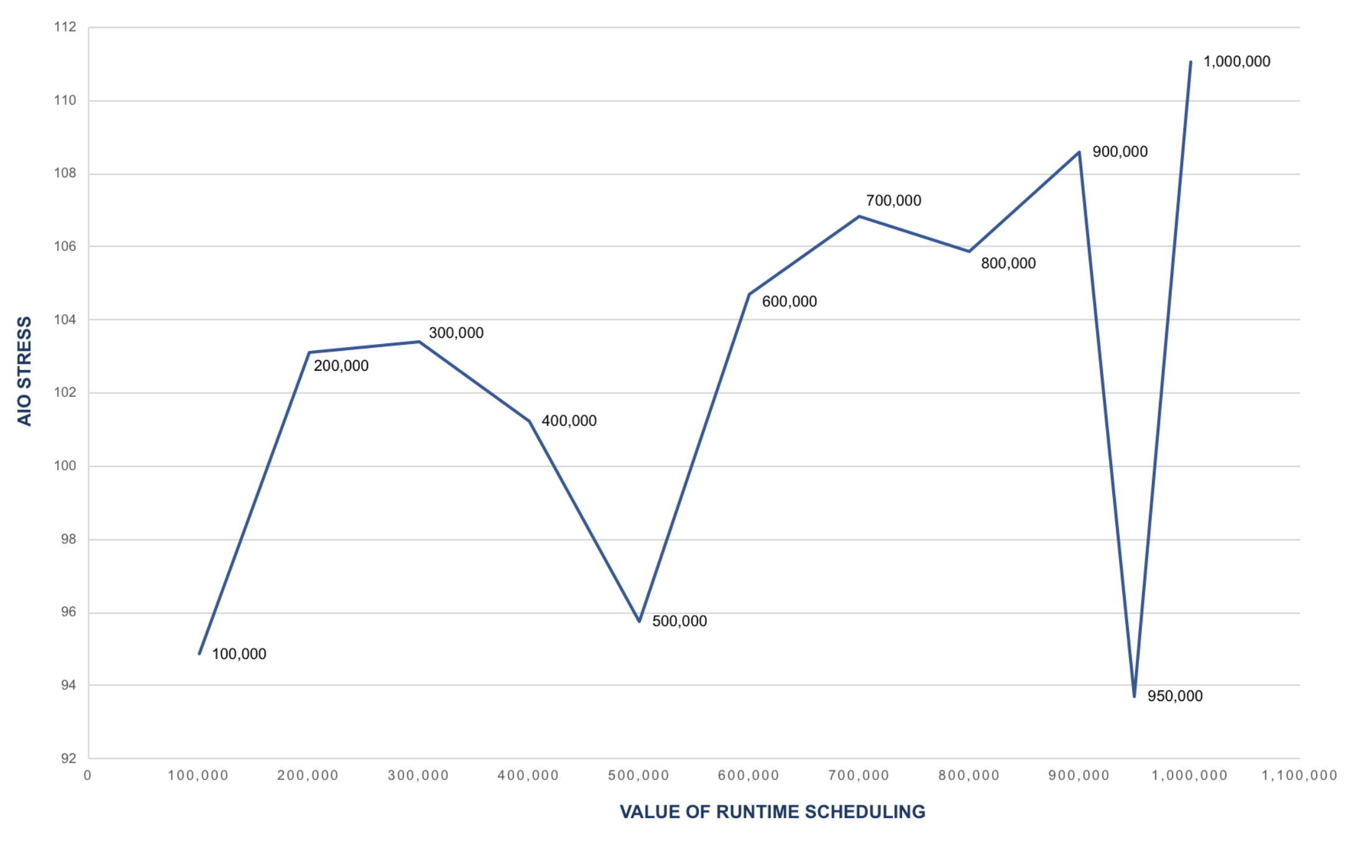
/proc/sys/kernel/sched_rt_runtime_us
Defines the time period to be devoted to running realtime threads, in microseconds ('us' being the closest equivalent to 'μs' in plain text). The default value is 950000μs, or 0.95 seconds.

Development and Results

The default value of sched_rt_runtime_us is 950000μs (0.95 seconds). According to this, 5% of the CPU time is reserved for processes that are not running under a realtime or deadline scheduling policy; this value specifies how much of the period time could be used by real-time and deadline scheduled processes on the system. The value can range from -1 (that makes the run-time the same as the period) to INT_MAX-1, making the run-time the same as the period there's no CPU time set aside for non-realtime processes.

Value of Runtime Scheduling	AIO Stress	Standard Error	Standard Deviation
1,000,000	111.06 mb/s	0.55	0.86%
950,000	**93.71 mb/s**	**4.23**	**11.08%**
900,000	108.6 mb/s	1.54	2.46%
800,000	105.89 mb/s	1.19	1.95%
700,000	106.84 mb/s	1.1	1.78%
600,000	104.71 mb/s	1.83	3.03%
500,000	95.76 mb/s	5.21	13.32%
400,000	101.25 mb/s	1.51	2.99%
300,000	103.39 mb/s	1.58	2.64%
200,000	103.13 mb/s	1.29	2.17%
100,000	94.87 mb/s	3.34	8.64%





System Specifications

- **Processor:** Intel Core i7-4500U @ 3.00GHz (4 cores)
- **Memory:** 2 x 4096 MB DDR3-1600MHz
- **Disk:** 1000GB Western Digital WD10JPVX-75J
- **Network:** Realtek RTL8101/2/6E + Qualcomm Atheros QCA9565 / AR9565
- **Motherboard:** Dell 03VVKX
- **Chipset:** Intel Haswell-UTL DRAM
- **OS:** Ubuntu 16.04
- **Kernel:** 4.4.0-43-generic (x86_64)
- **Compiler:** GCC 5.4.0 20160609Motherboard: Dell 03VVKX
- **Display Server:** X Server 1.18.4
- **Display Driver:** Intel 2.99.917
- **File-System:** ext4

Conclusion

With the realization of this project that was focused on research and analysis of results, it was obtained, among other things, the better understanding of the use and importance of a scheduler within the kernel of a computer. The role of the scheduler is also very important, it is responsible for keeping busy the CPU of the system, it is a small part of the kernel, but it is the one that coordinates the fluidity of the processes that occur in the system by the user requests through the Shell.

But with this project we can realize something else, the scheduler, with all the importance it has within the operating system, can be improved because it provides configuration variables that are available so that a user can modify them and achieve an improvement in their OS.

In the end, a lot was learned, but there is more: the developer of an operating system. External to the practice, unconsciously in the end, those people who are dedicated to carry this out are more appreciate. It takes a lot of research and some basic knowledge about the operating system to only modify a variable given by them.