**CSCI 447: Case Study**

**Riot OS**

**Caleb Ouellette**

# Introduction

The OS I chose to research is Riot OS. Riot OS is a tiny OS that is tailored for the challenges of Internet of Things development. Often in IoT resource such as RAM and power are limited, so Riot OS tries to be power efficient and have a low memory footprint. The minimum amount of RAM required to run Riot is just 1.5kB. Riot is an OS to use if Linux is too big. Riot OS is a micro kernel operating system. This helps aid the goal of lower memory overhead because developers can pull in only the features they need. Riot is written in C and has roughly 1 million lines of code. It started in 2008 and is still being worked on today.

Even though Riot is small it still tries to conform to well know standards of programing and implements some of the POSIX standards were possible. The goal is to make it easy to transition from Linux to Riot, which allows programmers to program many more devices. Riot also can run as a native process in Linux which means you don’t need IoT hardware to develop.

Another key focus of Riot OS is portability. Riot can run on 32-bit, 8-bit and 16-bit platforms. Riot is vendor independent and is easy to port to new hardware. This is especially important in the IoT space where new devices are showing up daily.

# CPU Scheduling

The key goals for Riot’s CPU Scheduling are to, keeping things fast in order to support real-time operations and to stay efficient on battery life. This means Riot keeps things pretty simply when it comes to CPU Scheduling and data structures. A core principle of Riot design is to keep all kernel tasks to an O(1).

Riot uses a tick-less, pre-emptive, priority-based scheduler. Let’s break down what all that means.

In operating systems such as Linux, the kernel will grant a process CPU time. However, when this happens the kernel sets a timer to interrupt that process and take back over after a set interval or tick. This allows the kernel to change what process is running or perform other tasks before giving back the CPU to a process. This is a great feature for conventional operating systems that don’t have to be real time and are often running many tasks at once. It also prevents any one bad process from taking all of the CPU. However, in the IoT world many of these things are not as big of a concern. Generally, IoT devices don’t have many processes running so the need to change the current process is minimal. Also, in IoT scenarios the processes running on the device are known, and probably won’t be hogging the CPU. Users generally won’t be downloading any programs unknown programs onto these machines. But most importantly Switching Context is expensive on power. By eliminating the ticker, the device can be much more power efficient. It will instead relay on interrupts or processes giving back the CPU to re-evaluate scheduling.

For the algorithm itself Riot implements a very simple priority-based scheduler to decide what process get to run. When a thread is created it is given a priority value. Higher Value means lower priority. There are is only one queue in Riot. The process with the lowest value gets to run first. The scheduler is also pre-emptive meaning that if a high priority thread (lower value) is created and the scheduler will switch to that thread before finishing its current thread has completed and will finish its current work after the higher priority process has completed.

# Memory Management

Riot OS does not support virtual memory and does not include any code for an MMU. This is due to the fact that there are often no MMUs on IOT devices. Riot OS simply assigns memory to the stack of each thread at initialization and then just leaves it. It offers no protection for that memory region and an overflow can crash everything. Riot does not even provide a separation of kernel space from user space, so all threads have the same privilege of execution. This loose approach to memory management means that Riot is very fast. There are no abstractions to take up extra time, so when everything is working correctly all is well. However, this approach has large security issues and offers little protection for processes.

# IPC/Messaging

A core pillar of a micro kernel is the Messaging system. All processes need this system in order to communicate with things the kernel would normally do. Riot Implements both a blocking and non-blocking message protocols. Blocking blocks, the sending thread until it receives a response.

# File System and I/O

Riot OS is a micro kernel and actually does not have a file system included in it. You have to include the file system you want to use as a user space process. You can then access this by using Riot’s IPC. In most IoT there will not be a file system needed, so it makes sense that it’s not included in the kernel.

The most popular file system package for Riot seems to be FAT file system package. FAT uses a file allocation table which is allocated at the time of formatting. This table contains the data for where the next block of data is stored. This allows you traverse the table to get all the data for a file that is stored in a list of blocks. The table also contains the information for if a block is free or not. The data for where each file starts is stored in a directory file. And to get all this started there is a root directory which is always stored at a fixed location. Once you know where to start you used the directory files to see where files start, then the File allocation table to find where all the data is for a given file.

Another file system that is popular with Riot is Littlefs. Which is a robust file system that is designed to be power-loss resilient and minimize updates. The core idea around Littlefs is to store meta data in twice. Once is the last revision and the other is the current. On each update the old state is update, meaning there is always a copy of the data even if the current write fails due to power loss. Writing data works similarly where the new data is written then the meta data is updated to point at the new data.

Overall, it’s pretty handy to be able to switch file systems and use multiple at once. The micro kernel allows this to happen. On the negative side it does introduce more work if you want to develop for these machines.

# Conclusions

Summarize and critique this OS. Do you agree with all decisions made? What would you do differently and why?

~0.5 page

Overall, I Riot is a really neat idea for an OS and fills its niche role very well. It solves the microkernel overhead problem by simply not implanting a user/kernel space mode. It gives great flexibility by allowing developers to only load in what they need. It keeps things simple to keep its resource needs low. Next time I do any IoT work I will defiantly consider this OS. It also seems to have large adoption and an active community. I’d say I was impressed by the decisions made around this OS and hope to see it gain more adoption.

To critique the OS, I would say it looks difficult to setup. Having to load in the packages you need is nice, but also more work. I would hope they have a prepackaged section to get you started. Another short coming on this OS is security. From what I can tell there is none. If a malicious software got onto this hardware it would be able to read all of main memory and send messages as it pleased.

One item you read a lot about RIOT is that it is only designed for machines smaller then a raspberry pi. It does not have 64 bit support. I might change that. I love how simple the OS is and might interested in running it on a more powerful machine or perhaps in a docker container. Having a super simple OS is something I find really interesting and want to work more with.

# References

<https://www.riot-os.org/docs/riot-infocom2013-abstract.pdf> \*I recommend reading this one.

<http://riot-os.org/#nutshell>

<https://www.youtube.com/watch?v=Cwf3y2Xj0og>

<https://github.com/RIOT-OS/RIOT/wiki>

<https://github.com/RIOT-OS>

<https://riot-os.org/api>

<https://hal.inria.fr/hal-00768685v3/document>

<https://github.com/geky/littlefs/blob/master/DESIGN.md>

<https://named-data.net/wp-content/uploads/2016/07/ndn-0043-1-ndn-riot.pdf>