Crossroads
The ACM Student Magazine

RTLinux: An Interview with Victor Yodaiken

By ***Kevin Fu***

RTLinux is an extension to the Linux kernel for real-time control. That is, RTLinux assists timing-critical tasks to finish within hard deadlines. General Editor Kevin Fu interviews the principal architect of RTLinux and extracts advice for budding computer scientists. Below is the interview conducted with Victor Yodaiken.

KF: In your own words, explain briefly what is a Realtime Operating System (RTOS).

VY: A realtime operating system gives application programmers a method of designating some tasks as "realtime." And "realtime" tasks should run with minimal event latency (the time between an event such as an interrupt that triggers a task and the start of the task) and minimal jitter (the variation in running times of a task that is supposed to run at a fixed period).

KF: What is your role in RTLinux?

VY: RTLinux was my solution to not having enough money for a realtime operating system when I wanted to do some research on realtime. About 90% of RTLinux has been written by my students and ex-students and myself -- although there are some important parts that have been contributed by others.

KF: RTLinux takes an interesting stance to give Linux realtime capabilities. How do you interface RTLinux with Linux? How does this affect development of realtime programs?

VY: The RTLinux philosophy is that the realtime subsystem needs to be as simple and predictable and close to the raw machine as possible. So we provide a couple of ways to connect realtime tasks to ordinary Linux tasks. We split realtime applications between the part that is really realtime and the part that just needs to run fast. For Linux tasks, the realtime system can look just like a device that can read and written. You can write a realtime application as a Perl program or even a shell script that gets data from the realtime "device" and sends commands to that "device."

KF: What are the factors to consider when selecting a RTOS? [versus non-RTOS and other RTOS]

VY: See above. I also think that realtime is one of those areas where mysterious "black box" software is less acceptable. If you depend on a component to always complete within a time bound, you better understand how it works.

KF: The description "realtime" has received much misuse recently. What does it mean to be realtime? Are online stock quotes realtime?

VY: I think a lot of confusion results from different, but reasonable, uses of "realtime". The simulation community uses "realtime" to mean that a simulation simulates in the time that the real process would take. They say a simulation is "hyper-realtime" if the simulation is fast enough to predict or even help control the process. There are other, not so reasonable, uses. Sometimes "realtime" is used just as an advertising word with no semantic content at all. The definition we use is that realtime tasks have predictable worst case timing. That's the key to using software to control machinery. If you look at the advertising for realtime operating systems, however, you will often see "typical" times used. That's not a very useful measure for many applications. If your robot arm gets a one-degree error for every microsecond delay and the average delay is less than one microsecond but the worst case is 20 microseconds, then you are in trouble.

KF: How did you become involved with the RTLinux project?

VY: I wanted a realtime operating system to experiment with so I could understand something about the field. The options seemed to be either not-too-robust academic systems or systems that are too expensive and have closed source. So I looked for an alternative and Linux was just starting to be widely available.

KF: How well has Linux allowed you to implement a RTOS?

VY: Linux has been wonderful. The strength of the Linux kernel development group is scary.

KF: What interesting concepts from computer science does RTLinux address? Threads? Scheduling? Atomicity?

VY: All those and more. Operating systems is, of course, the most interesting field in computer science.

KF: How do you obtain predictability of program completion?

VY: The keys for RTLinux are (A) Linux cannot delay the progress of the RT component, and (B) the RT component is simple.

KF: What was/is the hardest part in designing and implementing RTLinux?

VY: Trying to avoid overdesign. It is really tempting to put features into code either because they seem clever or because they are standard. I've thrown out of lot of my code and many nice ideas. I keep re-reading Butler Lampson's article on system design for inspiration[**2**].

KF: How many designs and implementations did you go through before settling on the current one?

VY: The current version is the result of three total re-writes and there is a new version in the works. The number of minor revisions is much larger than three.

KF: Which features of Linux have been helpful for the implementation and which have been detrimental?

VY: Loadable kernel modules have been critical.

KF: Which applications work well in RTLinux and which do not? [KURT claims that RTLinux cannot easily handle the ARTS system or multimedia viewers]

VY: KURT is the project of a colleague from graduate school at UMASS, so I won't say

anything bad about it! RTLinux does not fit the current realtime programming model in which realtime is added to a standard programming environment. I think that we get better performance and cleaner designs by forcing this decoupling. But some people really don't like the model. Since RTLinux is easily extensible, you can add features even if I disapprove.

KF: What further reading do you recommend to students interested in RTOS?

VY: Stankovic and Ramamrithram's tutorial is a good starting point [**4**]. It's really important to have a strong understanding of operating systems. I recommend Vahalia's book for the sections on synchronization and Henry Massalin's Ph.D. thesis for sheer inspiration [**5, 3**]. Then read some of the RT conference proceedings and also some of the industrial journals -- Embedded Programming is pretty good [**1**].

KF: Any other advice to students interested in RTOSs?

VY: Read Lampson's paper every night before bed.

KF: So you read his paper every night too? :-)

VY: Well, not every night. I've got it memorized and just need to compensate for the effects of age on memory.

References

1

Embedded Systems Programming. **http://www.embedded.com/**

2

Butler W. Lampson. Hints for computer system design. *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, Bretton Woods, New Hampshire (October 10-13, 1983), pages 33-48. Published as *Operating Systems Review 17*, 5 (1983).

3

H. Massalin. *Synthesis: An Efficient Implementation of Fundamental Operating System Services*. PhD thesis, Columbia University, 1992.

4

J. Stankovic and K. Ramamrithram. "Hard Real-Time Systems," *IEEE Tutorials*, vol. 819, 1988.

5

U. Vahalia. *UNIX Internals: The New Frontiers*. Upper Saddle River, NJ : Prentice Hall, 1996.

Biography

Victor Yodaiken (yodaiken@nmt.edu) is an Associate Professor of Computer Science at New Mexico Tech.

Kevin Fu (fubob@mit.edu)is a Crossroads General Editor and a graduate student at MIT.