

FlightLinux Project

POSIX Report

Aug 15, 2000

Updated

Sept, 2001

Patrick H. Stakem
QSS Group, Inc.

Revision History

August 15, 2000
August 15, 2001
Sept 2001

Initial Release
Revision; updated references; major technical edit
minor revisions

Introduction

This report is a deliverable of the task NAS5-99124-564. It is associated with the contract AIST-0083-0075, "FlightLinux Operating System for Use with Spacecraft Onboard Computers." This report 1) discusses the Portable Operating System Interface (POSIX) standard, 2) illustrates how the FlightLinux Operating System will be POSIX compliant, 3) discusses the POSIX-compliant flight software developed by GSFC, Code 582, and 4) documents POSIX references.

The FlightLinux Project is concerned with implementing the Linux Operating system in a spacecraft onboard computer environment. This report defines what Posix is, what the advantages are, and how Linux meets the Posix standards. It also defines the ongoing GSFC efforts to produce Posix-compliant applications code, and to develop coding guidelines for such code.

The following section explains the role of this report in the larger context of the FlightLinux Project.

Steps to FlightLinux Implementation

The purpose of this section is to define the reports which will be produced as milestones in the FlightLinux process, and their interrelationships. The goal of the FlightLinux Project is an on-orbit flight demonstration and validation of the operating system.

We have defined the steps to a space flight demonstration of the Linux operating system. Regardless of the implementation architecture, certain pivotal issues must be define. This will be done in a series of reports. These reference reports will collect together in one place information and ongoing research related to the topics. The key issues include the architecture of the target systems, the nature of application software, the architecture of an onboard LAN, and the requirements for support, the architecture of the onboard storage system, and the requirements for support, and the nature and design of the software development testbed.

In the Target Architecture Technical Report, we examine the current, near term, and projected computer architectures that will be used onboard spacecraft. From this list, we examine the feasibility and availability of Linux. The choice of the actual architecture for implementation will be determined more by opportunity of a flight than by choice of the easiest or most optimum architecture.

The POSIX Report will examine and document the POSIX-compliant aspects of Linux and other Flight Operating systems, as well as the POSIX-compliant nature of legacy flight application software. This is an ongoing effort by GSFC Code 582, the Flight Software Branch.

The Onboard LAN Architecture Report discusses the physical level interfaces on existing and emerging missions, and the device drivers required to support IP over these

interfaces. Ongoing work in this area is being done by the CCSDS committee, and the OMNI Project (GSFC, Code 588). The choice of a demonstration flight will define which interfaces will need to be implemented first. In addition, those interfaces with cots drivers, and those for which device drivers need to be defined will be delineated.

The Bulk Memory Device Driver Report will define the approach to be taken to implement the Linux file system in the bulk memory ("tape recorder") of the spacecraft onboard computer. It will define which elements are COTS, and which need to be developed.

The Embedded Testbed Report will define the requirements and architecture for the facility to develop and validate the operating system code for the flight experiment. Guidance will be drawn from similar past facilities.

These reports will be living documents, updated as required to document new developments. The reports will be stand-alone, but will reference the other reports as required. A major purpose of the reports will be to collect in one area the COTS aspects of the specific aspect of the FlightLinux implementation, so that attention may be focused on the remaining "missing pieces."

POSIX

POSIX is an IEEE standard for a Portable Operating System Interface [Ref. 3]. The use of POSIX-compliant operating system and applications has many benefits for flight software. Among these are software library reuse between missions and software commonality between ground and flight platforms. For compliant code, the function calls, arguments, and resultant functionality are the same from one operating system to another. Source code does not have to be rewritten to port to another environment. Most Linux variants are mostly but not completely POSIX compliant. The posix standards are now maintained by an arm of the IEEE called the Portable Applications Standards Committee (PASC) with the associated website <http://www.pasc.org/>.

For example, Linux can now support POSIX-like Access Control Lists (ACLs). Access Control Lists allow specifying fine-grained per-user or per-group permissions for files and directories. These implement POSIX 1003.1e Draft Standard 17, Access Control Lists. The Linux kernel has also been updated to support POSIX clock and timer functions. POSIX shared memory features are not yet fully support in Linux.

POSIX compliance is certified by running a Posix Test Suite, available from the National Institutes of Standards and Technology (NIST). At the moment, we have no plans for POSIX compliance testing of various flavors of Linux. We will, however, collect and document third party and manufacturer's information on Posix compliance.

FlightLinux

FlightLinux addresses the area of operating systems for onboard computers for traditional and constellation missions. A project website has been set up at the location: [http: FlightLinux.gsfc.nasa.gov](http://FlightLinux.gsfc.nasa.gov) .This report, among others, will be found on the site.

The advantages of Linux are numerous, but the requirements for spacecraft flight software are unique and non-forgiving. Traditional spacecraft onboard software has evolved from being monolithic (without a separable operating system), to using a custom operation system developed from scratch, to using a commercial embedded operating system such as VRTX or VxWorks. None of these approaches have proved ideal. In many cases, the problems involved in the spacecraft environment require access to the source code to debug. This becomes an issue with commercial vendors. Cost is also an issue.

As a variation of Linux and thus Unix, FlightLinux is open source. This means the source code is not only readily available, it is free. FlightLinux currently addresses soft real time requirements, and is being extended to address hard real time, for applications such as attitude control. There is a wide experience base in writing Linux code that is available to tap.

The use of FlightLinux will simplify several previously difficult areas in spacecraft onboard software. For example, FlightLinux imposes a file system on onboard data storage resources. In the best case, Earth-based support personnel and experimenters may network-mount onboard storage resources to their local file systems. FlightLinux provides a path to migrate applications onboard and it enforces a commonality between ground-based and space-based resources.

As the Operating Missions as Nodes on the Internet (OMNI) project at GSFC has demonstrated, the TCP/IP protocol can be successfully used with orbiting spacecraft. Support for TCP/IP is built into Linux as a standard feature. Functions such as PING, FTP, TELNET, and Web services are standard features. FlightLinux also enables the implementation of an onboard LAN architecture and provides the missing last link in the end-to-end satellite networking scheme.

We are pursuing the IEEE POSIX compliance issues of standard embedded Linux, in parallel with an effort in GSFC Code 582, which is collecting a library of POSIX-compliant flight applications software.

FlightLinux will also enable the implementation of the Java Virtual Machine, allowing for the uplink of Java applets to the spacecraft.

Real-Time Extensions to Linux

Linux is not by nature or design a real-time operating system. Spacecraft embedded flight software needs a real time environment in most cases. However there are shades of real-time, specified by upper limits on interrupt response time, and interrupt latency. We can generally collect these into hard real time and soft real time categories. Examples of hard real time requirements would be for attitude control, and spacecraft clock maintenance, and telemetry formatting. Examples of soft real time requirements would include thermal control, data logging, and bulk memory scrubbing.

The Unix operating system, on which Linux is based, is a multi-tasking system. Each process being run is assigned a priority that determines when the process gets resources and attention from the cpu. In general, a process is running, waiting to run, or asleep. The key to the operating system task switching, and its real-time response, is the mechanism by which processes that are not running get to run. Unix, and Linux, were not designed as real-time operating systems, but do support multitasking. Modifications or extensions to support and enforce process prioritization are necessary to apply Linux to the embedded real-time control world.

In one model, a process may yield up the cpu to another pending task. In a preemption scheme, a running process is stopped, and a pending process is started. In another scheme, time slicing, a round robin priority scheme allows equal access to all tasks, or a variation, with a high priority and a low priority queue. It is generally agreed that a preemptive scheduling scheme allows for greater concurrency in a real time system.

Beyond the process-switching scheme is the interrupt prioritization. Here, we mean asynchronous interrupts from external sources. Interrupt prioritization is determined and enforced by the hardware configuration. Also, interrupt servicing supersedes software process execution in general.

The Lynx Operating System (LynxOS) is produced by LynuxWorks, the company that also makes BlueCat Linux. LynxOS is not open source, but the LynxOS and BlueCat Linux products are moving closer together. The problems originate from the fact that the tradition Unix or Linux kernel is a monolithic entity that governs process prioritization. Interrupt drivers and the kernel itself do not participate in the prioritization scheme. The kernel typically has large stretches of non-preemptible code. This is necessarily in the design so that data structures can be modified in an atomic fashion. In a Linux kernel, all interrupt handlers run at a higher priority than the highest priority task. In the Unix view, the kernel is the top level and most important task. In the real-time control world, this is not necessarily true.

The Linux Scheduler

Before discussing how various RTOSs implement scheduling, taking a look at how Linux does its scheduling provides an interesting reference point. The scheduler in `/usr/src/linux/kernel/sched.c` of the Linux source tree works with three scheduling modes (which are part of the POSIX standard!): `SCHED_RR`, `SCHED_FIFO` and `SCHED_OTHER`. `SCHED_OTHER` is the default. The scheduling mode is set by the POSIX `sched_setscheduler` system call.

`SCHED_RR` is the round-robin time slicing algorithm. After a task finishes its time slice, it is moved to the tail of its priority queue, such that another task in the same priority level can start running. If there is no other task at this priority, the interrupted task can continue.

`SCHED_FIFO` is a First-In, First-Out scheduling algorithm: the tasks in one priority level are scheduled in the order they get ready to run; once a task is scheduled, it keeps the processor until pre-empted by a higher priority task, until it releases the processor voluntarily, or until it has to wait to get access to some resource. This scheduler mode is often called “POSIX softreal-time” because it corresponds to the most common real-time scheduling approach with static priorities, but without the other necessary real-time components.

`SCHED_OTHER` tries to combine two conflicting performance measures: maximum throughput and good response to interactive users. It calculates a “goodness” value for each candidate task, based on a number of heuristic rules.

Tasks with the `SCHED_OTHER` scheduling policy receive priority “0”, while the `SCHED_RR` and `SCHED_FIFO` policies can use priority levels from “1” to “99”. User space tasks have to use the `SCHED_OTHER` policy. The portable POSIX way to find out about the minimum and maximum scheduling priorities are the `sched_get_priority_min()` and `sched_get_priority_max()` system calls; they take one of the priority policies as their argument.

The scheduling for Symmetric Multi-Processor (SMP) systems is basically the same as for the uni-processor case. What does become more complicated in a system with multiple processors is the interprocess synchronisation." [Ref. 22]

One approach to correcting the deficiencies in the Linux kernel is to implement a threaded execution approach for the kernel and the interrupt handlers. The question arises as to how much the Linux Kernel can be modified, and still be referred to as a Linux kernel. Another approach is to treat the kernel itself as a scheduled task, under a Real Time Task manager that manages process prioritization and takes over control of interrupts. This has been referred to as kernel cohabitation.

At least two real time schedulers for Linux are available for download. These are a Rate Monotonic Scheduler, that treats tasks with a shorter period as tasks with a higher

priority, and an Earliest Deadline First (EDF) scheduler. Other approaches are also possible. It is not clear which approach will provide the best approach in the spacecraft operating environment. This area will continue to be tracked.

POSIX Flight Software

The POSIX Flight Software Project is an effort of GSFC, Code 582 (Flight Software). Their web site (see reference 1) describes their project plan, their activities, and results. At this time, a core subset of the MIDEEX flight software has already been ported to Posix compliance and tested. As this project progresses, it is expected that a library of POSIX compliant flight software routines will be collected and available.

"Linux today is a POSIX-compliant OS and its constituent subsystems support all relevant ANSI, ISO, IETF and W3C standards. However, certification is a different issue, and the Linux community is against having to pay standards bodies for something that doesn't really benefit them. Therefore, Linux is currently in the state of being compliant with some standards without actually being certified." [Ref. 21]

Future Directions

Linux is evolving in the direction of full POSIX compliance. The GSFC Flight Software Branch, Code 582, is building a collection of Posix compliant application software.

The question remains, as to how much POSIX-compliance is enough. Complete compliance with the standard for applications and the operating system is probably not required nor warranted.

The activities that will be pursued during the course of this project include:

1. Track POSIX compliance of the various flavors of Linux and its embedded and real-time variations.
2. Track the progress of the POSIX-compliant flight software project of GSFC Code 582.
3. Track the real time characteristics of Linux variations, in conjunction with Code 584.
4. Use this information in the implementation of FlightLinux for flight testing.

REFERENCES

1. <http://posixfsw.gsfc.nasa.gov> (temporarily, <http://mongoose5.gsfc.nasa.gov/posixfsw>).
2. <http://FlightLinux.gsfc.nasa.gov>.
3. <http://standards.ieee.org/catalog/posix.html#gen22>.
4. <http://hegel.ittc.ukans.edu/projects/posix/index.html>.
5. <http://www.ukuug.org/sigs/linux/newsletter/linux@uk21/posix.html>.
6. POSIX threads: <http://www.humanfactor.com/pthreads/pthreadlinks.html>.
7. <http://members.aanet/~mtp/PCthreads.html>.
8. Newmarch, Jan "Unix Systems Programming Using Java," Distributed Information Laboratory Information Sciences and Engineering, University of Canberra.
<http://pandonia.canberra.edu.au/java/posix/paper.html>.
9. <http://hegel.ittc.ukans.edu/projects/posix/>.
10. <http://www.pasc.org/>.
11. Butenhof, David R., *Programming with POSIX Threads*, May 1997, Addison-Wesley Publishing Co, ISBN: 0201633922.
12. Gallmeister, Bill O., *POSIX. 4: Programming for the Real World*, January 1995, O'Reilly & Associates, ISBN: 1565920740.
13. A library that provides a VxWorks style interface to Linux Posix Threads:
<http://www.gb.nrao.edu/GBT/MC/ygor/libraries/TaskLib/TaskLibdoc.html>.

Real Time Linux References

14. Yodaiken, Victor, *The RT-Linux Approach to Hard Real-Time*, Department of Computer Science, New Mexico Institute of Technology.
<http://www.rtlinux.org/rtlinux.new/documents/papers/whitepaper.html>.
15. Epplin, Jerry, "Linux as an Embedded Operating System," *Embedded Systems Programming*, Oct. 97. <http://www.embedded.com/97/fe39710.html>.
16. Rajkumar, Ragnathan, *Synchronization in Real-Time Systems: A Priority Inheritance Approach*, Kluwer Academic Publishers, ISBN 0-7923-9211-6, 1991, 208 pp.

17. Mercer, Clifford W., and Ragnathan Rajkumar, "An Interactive Interface and RT-Mach Support for Monitoring and Controlling Resource Management," *Proceedings of the Real-Time Technology and Applications Symposium*, May 1995.
18. Mercer, Clifford W., Jim Zelenka, and Ragnathan Rajkumar, *On Predictable Operating System Protocol Processing* Technical Report CMU-CS-94-165, School of Computer Science, Carnegie Mellon University, May 1994.
19. Mercer, Clifford W., Ragnathan Rajkumar, and Jim Zelenka, "Temporal Protection in Real-Time Operating Systems," *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, May 1994.
20. Tiemann, Michael, "POSIX Eases Route to Embedded Linux," *Electronic Engineering Times (EE Times)*, 04/09/2001, No. 1161, Pg. 102, Section: EMBEDDED SYSTEMS -- FOCUS EMBEDDED SYSTEMS CONFERENCE/RTOSes.
21. http://zandura.net/~dustind/propaganda/Guide_To_Linux/guide_to_linux.html.
22. Real Time and Embedded HOWTO,
<http://www.mech.kuleuven.ac.be/~bruyinc/rthrowto/sched-linux.html>.