DOI. No. 10.1109/MAES.2017.160182

Feature Article:

Current Use of Linux in Spacecraft Flight Software

***Hannu Leppinen*, Aalto University, Espoo, Finland**

INTRODUCTION

Spacecraft on-board computers are responsible for controlling the spacecraft platform, payloads, or other on-board devices. Their mission-specific software allows communication with ground or other on-board computers. Traditionally, on-board software has been written close to the hardware in assembly language, Ada, C, or C++, with or without a real-time operating system (RTOS) [1].

As the spacecraft computer hardware capabilities are increasing, spacecraft software is becoming larger and more complex, handling more tasks from payload data processing to landing a first stage of a launch vehicle on an ocean-going barge. Spacecraft will still continue to include very small embedded systems that can be developed without operating systems (OS), but some systems will also have large software bases, requiring efficient software development processes and reuse of existing software modules.

The last decade has seen increasing use of Linux in spacecraft on-board software. This article presents common features of spacecraft on-board computers and software and discusses potential benefits and drawbacks of on-board Linux use. The focus of this article is on spacecraft on-board avionics software, that is, spacecraft-controlling code that flies into orbit with the spacecraft. Other types of computers are not included in this analysis; for example, many laptops on the International Space Station run Linux [2].

BACKGROUND

SPACECRAFT ON-BOARD COMPUTERS

Spacecraft on-board computers are responsible for handling telecommands sent by ground, providing telemetry to the ground, processing on-board data, and controlling the spacecraft platform and payload devices. A spacecraft may have one or several on-board computers that handle different tasks. Similar to other embedded computers, a spacecraft on-board computer usually has at least a processor, random access memory (RAM), read-only memory for boot code, mass memory, data bus interfaces, and a power supply. Perhaps unlike many other embedded computers, some parts of the on-board computer may be made redundant to circumvent any possible hardware malfunctions.

According to Eickhoff [1], radiation-hardened SPARC, PowerPC, MIPS, and Intel x86 architectures have been popular in traditional industrial and governmental space applications. University small satellite missions have tended to use a larger variety of processors, especially from the ARM family. According to a 2010 analysis [3], low-power microcontrollers such as the Microchip PIC series and the Texas Instruments MSP series have been popular in CubeSats before 2010, while ARM microcontrollers were becoming more popular due to their greater processing power. Some inexpensive commercial off-the-shelf (COTS) microcontrollers, such as the Texas Instruments Hercules family, have safety-critical features previously usually seen in expensive high-reliability processors, making them attractive for space applications [4].

SPACECRAFT ON-BOARD SOFTWARE

The on-board software running on the on-board computer is responsible for utilizing the computer's hardware resources and interfaces to achieve its specified mission, such as controlling the spacecraft or some platform or payload instrument. On-board software must typically provide at least a telecommand and telemetry interface that is utilized by ground control or some other on-board software running on another on-board computer. On-board software can be roughly divided into three parts: hardware driver software for providing abstractions of the underlying hardware, OS for providing task and resource management, and application software for providing the mission-specific functionality. This division is illustrated in Figure 1. Some simple on-board software may not need an OS at all.

The hardware driver software is usually provided by the hardware manufacturer as a software abstraction of their hardware, and various users of the same hardware in different industrial domains may use the same driver software.

The OS is an optional component, and in many cases it has not been used at all. In some cases, the OS itself may provide the abstraction of hardware instead of separate hardware driver software. Historically, much of spacecraft flight software was written from ground up using assembly language. Ada programming language, which has long had advanced tasking features, has been used to build on-board software systems running several threads of execution. However, recently the use of Ada has been reduced in favor of using real-time OS along with C and C++ programming languages due to the larger availability of developers for these languages. When most of the software has been written from scratch, the end

Authors' address: Aalto University, Department of Radio Science and Engineering, Otakaari 5A, 02150 Espoo, Finland. E-mail: (hannu.leppinen@aalto.fi).
Manuscript received August 18, 2016, revised October 12, 2016, November 27, 2016, and ready for publication January 24, 2017.
Review handled by M. Jah.
0885/8985/17/$26.00 © 2017 IEEE

result has tended to be very tightly coupled, and not well suited for later use in other projects [5].

When OS have been used in spacecraft, common selections have included VxWorks and RTEMS. Both VxWorks and RTEMS are RTOSs designed for embedded systems and commonly used in safety-critical hard real-time applications [1]. University small satellite missions have also used other general purpose RTOSs, such as FreeRTOS [4].

EMBEDDED LINUX

The term Linux strictly refers to the OS kernel originally developed by Linus Torvalds since 1991. Linux is used in embedded systems because of quality and availability of code, wide hardware support, implementations of communication protocols and application programming interfaces, available development tools, favorable licensing conditions, vendor independence, and cost. The Linux kernel can be run in various Linux systems from very small embedded computers to supercomputer clusters. Each system has a different purpose and different software packages and applications.

Linux has been ported to many architectures: as of this writing, the Linux kernel readme file lists at least 22 supported architectures. According to Yaghmour [6], architectures mostly used in embedded Linux applications include ARM, AVR32, Intel x86, M32R, MIPS, Motorola 68000, PowerPC, and SuperH. Linux is also available for the LEON processors commonly used in the European space industry. Thus Linux is available for most of the architectures commonly used in space applications mentioned in the previous section.

A generic on-board software architecture using Linux is depicted in Figure 2. Linux provides OS facilities such as virtual memory, processes, communication sockets, and files. The kernel is responsible for driving the devices, managing input-output access, scheduling processes, enforcing memory sharing, and handling signals. The kernel is typically started by a bootloader, and is never stopped while the embedded device is running. The Linux kernel has support for many data buses and communication protocols useful for spacecraft developers, including at least CAN bus, Modbus, SPI, I²C, and serial port [6]. CAN is one of the on-board data buses already used in the space industry, and is directly supported by Linux [1].

Many commercial embedded Linux distributions exist for various platforms, but embedded developers may also build their custom distributions using the kernel source code. Buildroot is one commonly used tool for this purpose. uClibc, a lightweight C library, is often used to replace the standard GNU C library used

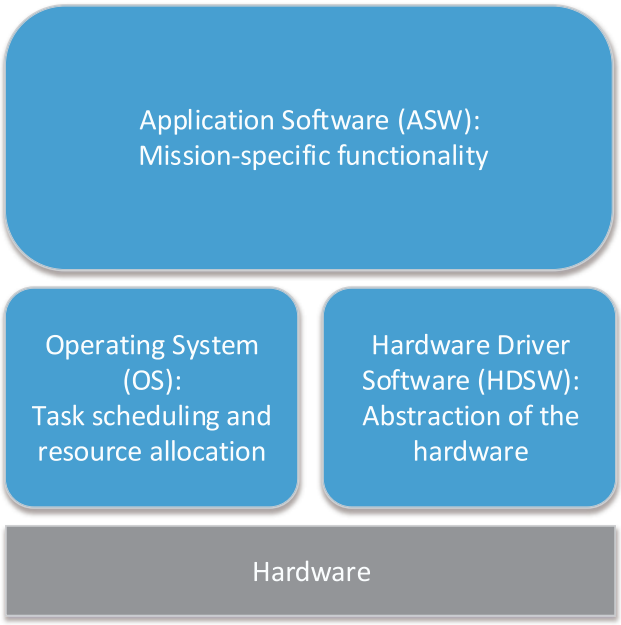

Figure 1.
Generic on-board software architecture.

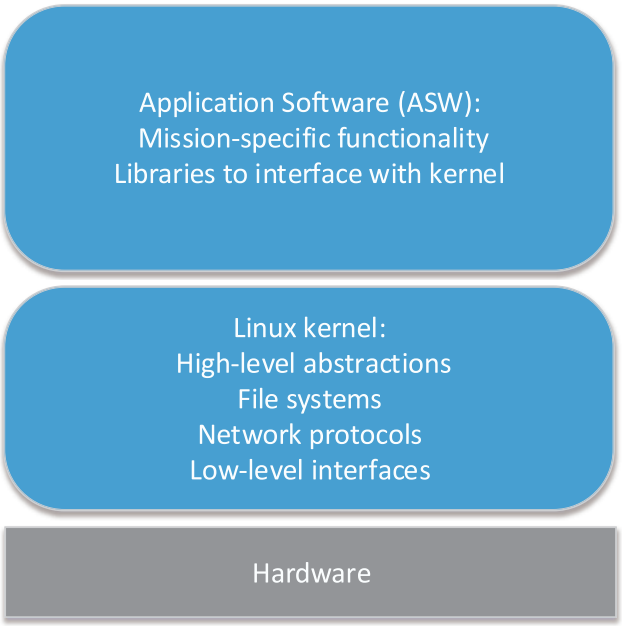

Figure 2.
Generic on-board software architecture with Linux kernel. Adapted from [6].

in desktop applications. BusyBox is often used to include many UNIX-like tools in embedded applications where only limited resources are available [6]–[8].

μCLinux is a version of Linux developed for processors that lack a memory management unit, but provides the same basic functionalities as Linux [6].

Linux has been designed as a general purpose OS and thus does not directly support hard real-time applications [6]. However, solutions have been designed to address this limitation and we discuss these in the next section about advantages and drawbacks.

LINUX IN ORBIT

This section reviews Linux use in various spacecraft. The analysis is limited to a selection of missions that have academically published their Linux use. The selection includes educational, governmental, and commercial missions. More than a hundred spacecraft, many by Planet and SpaceX, have flown Linux. The discussed missions are summarized in Table 1.

Possibly the first study of using Linux in spacecraft dates to the NASA FlightLinux project that ran from 1999 to 2002. The project aimed to provide an on-orbit demonstration of the Linux OS; the demonstration target was the UoSat-12 satellite operated by Surrey Satellite Technology Ltd. The rationale for this demonstration was to try extending the COTS philosophy used in hardware to the software domain by using an off-the-shelf OS with many available software packages, that is, Linux. The kernel and a decompression program were fitted within a space of approximately 400 kB, and the system only had 4 MB of RAM available. Linux provided networking and filesystem facilities and supported high-level programming languages, such as Java. The kernel setup routine was modified to support the UoSat-12 hardware. The planned UoSat-12 test was very primitive, such as printing "Hello World!" to a serial port [30].

Many CubeSat missions, including the Aalto-1 mission operated by Aalto University [10], have used Linux in their flight computers. Aalto-1 uses an AT91RM9200-based computer with a customized 3.4 kernel prepared using Buildroot. Aalto-1, shown in Figure 3, was launched on June 23, 2017. As the mission began recently and is still ongoing, it is not analysed in this article.

A microsatellite example is TacSat-1, which used Linux in its Copperfield-2 payload in PowerPC MPC823, PowerPC PowerQuicc II 8260, and StrongArm SA1110 computers. Linux was considered possible since there were no hard real-time requirements for the payload software, and thus no real-time adaptations of Linux were used. The payload computers, communicating via TCP/IP, processed sensor data and provided payload data storage and handling infrastructure. Linux also provided scripting engines as an additional benefit; Perl and Python were evaluated, but shell scripting was decided to be used. Many payload functionalities were implemented as scripts, and the data flow between software was handled via standard input and output. This allowed quick interfacing with existing, off-the-shelf Linux software. Most of the custom software developed for TacSat-1 handled conversion between TCP/IP and OX.25 protocols. Development was possible on x86 PCs, while the target itself had PowerPC architecture. However, TacSat-1 was never launched [9].

The following subsections discuss some of the missions in Table 1.

QUAKESAT

One of the earliest CubeSats to fly Linux was QuakeSat, which was a 3U CubeSat built at Stanford University for QuakeFinder LLC. It aimed to study extremely low-frequency magnetic signals in order to possibly predict earthquake activity. QuakeSat also aimed to demonstrate the usefulness of COTS electronics and the CubeSat standard for the development of low-cost space missions.

QuakeSat's command and data handling system operated with timed commands to perform payload data gathering when historically active earthquake areas were within the range of the instrument. QuakeSat had no digital on-board mission data processing. All electronics were embedded on a single circuit board. Diamond Systems Prometheus PC-104 processor board was used as the command and data handling system hardware, and a version of Linux provided by Diamond Systems was used as the OS. Linux was chosen since many of the required device drivers were already available. The processor was clocked down to 66 MHz, and the system had 32 MB of RAM and a 192 MB Flash disk for storage. In addition to Linux, the flight software consisted of some 10,000 lines of code, some of which included already existing AX.25 and modem drivers. Flight software features included communication with an on-board UHF modem, AX.25 libraries, other radio utilities, payload data collection and compression, data downlink, and command execution. Bzip2 compression was used for the payload data.

The satellite had only passive magnetic attitude control. The QuakeSat command and data handling system performed time-based operations, with operation times selected on the predicted position of the satellite. The command and data handling system was able to store commands and data for one day. Schedule files were uploaded to the satellite, and payload data files were compressed and downlinked from the satellite. The satellite operator was also able to query real-time telemetry from the satellite. The operator also needed to synchronize the satellite clock every three days. The satellite performed well in orbit, and collected useful scientific data [11], [12].

UWE-1 AND UWE-2

UWE-1 and UWE-2 were 1 kg CubeSats built by the University of Würzburg, and they were launched in 2005 and 2009, respectively. The UWE satellites aimed to test various small satellite technologies, including the use of internet protocols in space. Both used a similar on-board computer, which included a 16-bit Hitachi H8S 2764 microprocessor. The computer had 8 MB of RAM and 4 MB of nonvolatile flash memory. The 16-bit processor did not have a memory management unit, requiring the use of μClinux. The computer consumed only 300 mW during nominal operations; the low power consumption was one of the reasons to select the microcontroller, as it allowed more power for other experiments. The

Table 1.

List of Analysed Spacecraft				
Name	**Operator**	**Launch Date**	**Details**	**Sources**
TacSat-1	Naval Research Laboratory	Not launched	PowerPC MPC823 and PowerQuicc II 8260, StrongArm SA1110	[9]
Aalto-1	Aalto University	2017	AT91RM9200-based embedded computer, Linux 3.4 prepared with Buildroot	[10]
QuakeSat	QuakeFinder LLC	2003	Diamond Systems Prometheus PC/104 x86 CPU module with Red Hat Linux	[11] [12]
MidSTAR-1	United States Naval Academy	2007	ARM Linux on a payload controller	[13]
UWE-1[1], UWE-2[2]	University of Würzburg	[1]2005, [2]2009	Hitachi H8S 2674 16-bit microprocessor, µClinux	[14] [15]
X-Sat	Nanyang Technological University	2011	Payload controller: Linux on several StrongArm SA1110 processors	[16]
IPEX	Cal Poly San Luis Obispo, JPL	2013	400 MHz Atmel ARM9 CPU based computer, Linux 2.6.30	[17] [18]
STRaND-1	Surrey Space Centre	2013	Digi-Wi9C with µClinux and Google Nexus One with Android	[19] [20]
LightSail-1	The Planetary Society	2015	Tyvak Intrepid v6 single-board computer	[21] [22]
PhoneSat satellites	NASA	Several since 2013	Google Nexus One and Nexus S with Android 2.2	[23]
Dove satellites	Planet	Several since 2013	Low-power COTS x86 processors, Ubuntu server	[24] [25]
Falcon 9, Dragon	SpaceX	Several since 2010	Multiple COTS computers, custom Linux 3.2 with real-time patches	[26] [27] [28] [29]

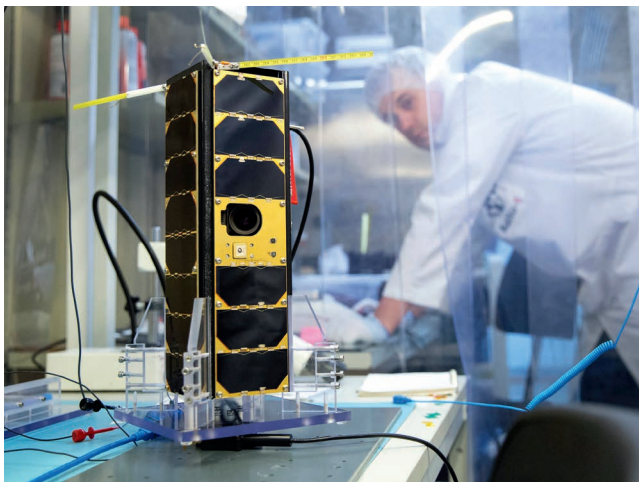

Figure 3.
Aalto-1 is one of the CubeSats using Linux. Photo: Aalto University.

processor also had the required interfaces, such as I^2C and serial port, to communicate with other hardware on the satellite. µClinux was selected because of the Linux portability and availability of software; only the application-specific software modules needed to be developed. Linux also had the internet protocol stack available for the technology demonstrations.

Although the UWE-1 on-board software performed well, it was rewritten for UWE-2 in order to be more extensible. Software was written in C due to its good integration with Linux, and due to the availability of C cross-compiler for the selected processor. The application software was realized as several Linux programs running in parallel, including programs for system control, radio control, battery control, housekeeping, logging, and sensors. The programs communicated through the Linux interprocess communication facilities.

UWE-1 completed its demonstration mission within a few weeks after launch, and UWE-2 was also operated successfully [14], [15].

IPEX

IPEX was a 1 kg CubeSat developed by Cal Poly San Luis Obispo and Jet Propulsion Laboratory (JPL) that aimed to validate technologies for on-board instrument processing and autonomous operations. The main computer had a 400 MHz Atmel ARM9 processor with 128 MB RAM, a few megabytes of radiation resistant PRAM, 512 MB NAND Flash, and a 16 GB microSD card. Linux 2.6.30 was used. The satellite additionally carried a Gumstix Earth Storm single-board computer with an 800 MHz ARM processor, 512 MB RAM, 512 MB NAND Flash, and also running Linux. The satellite had several cameras as payload, and had high processing power requirements. The two computers were linked with a serial link. An on-board software, CASPER, managed the spacecraft resources autonomously.

The flight software of IPEX was based on extending and adapting Linux facilities. The System V init process was used to start and restart main components of the flight software. Operations were based on uploading observation and ground contact schedules. Uplink and downlink bandwidth was limited, which made operations harder. The developers noted that more software could have been preloaded on the computer, since nonvolatile storage was abundant, while it was very hard to upload large pieces of software during the mission. IPEX computer used reboots to clear problems possibly caused by radiation, which grew more frequent as the mission progressed. The satellite stopped communicating in early 2015, but its mission of demonstrating on-board autonomy was successfully completed [17], [18].

STRAND-1 AND PHONESAT SATELLITES

The Android OS is based on Linux, and at least two projects have used Android smartphones to build satellite on-board computers. Smartphones contain many technologies useful for small satellites, such as low-power yet capable processors, gyroscopes, accelerometers, cameras, and communication interfaces.

STRaND-1 was a CubeSat built by Surrey Space Centre and Surrey Satellite Technology Ltd that aimed to explore new ways of including smartphone technologies to CubeSat platforms. While the satellite main computer ran FreeRTOS, STRaND-1 used in its payload μClinux in a Digi-Wi9C single board computer and Android on a Google Nexus One smartphone. The satellite main computer communicated with the Digi-Wi9C via I^2C, while Digi-Wi9C communicated with the smartphone via USB and Wi-Fi using for example the telnet protocol. One of the aims of the mission was to test transferring control of the satellite to the smartphone and its Android applications. The satellite was launched in early 2013, and experienced initial communication problems but was recovered and continued its mission in summer 2013 [19], [20].

The NASA PhoneSat technology demonstration project aimed to demonstrate building very low-cost nanosatellites using smartphones and other consumer technology. The project has launched several smartphone-based nanosatellites that used Google Nexus One and Nexus-S smartphones as on-board computers. Several PhoneSats have been launched since 2013, and the smartphone cameras have successfully taken pictures of the Earth [23].

Figure 4.
Two Dove satellites running Linux being deployed from the International Space Station. Photo: NASA.

LIGHTSAIL-1

LightSail-1 was a CubeSat developed by the Planetary Society and launched in 2015. Its purpose was to test the deployment of a solar sail in space. The LightSail-1 avionics consisted of two processor boards with different tasks. The main board handled the spacecraft command, control, data collection, and telemetry, and the payload interface board focused on attitude control and managing the solar sail payload deployment. The main board was a Tyvak Intrepid v6 single-board computer and used Linux, and the payload interface board used a 16-bit PIC33 processor. Flight software was written in C for both the Linux-running Intrepid and the PIC processor. While the Linux system could run several processes and support many libraries and interfaces, the PIC processor ran a simple 5 Hz control loop for attitude control and solar sail deployment.

During the mission, a problem in the Linux system was discovered that caused a temporary loss of control. A comma-separated values file grew out of bounds due to a bug, causing the system to halt; fortunately, a radiation-induced single-event effect caused the main computer to reboot, temporarily fixing the problem, and allowing a bug fix to be applied in orbit. After the bug fix, the solar sail deployment was successfully performed just in time before the low-flying satellite deorbited [21], [22].

DOVE SATELLITES

Planet, formerly Planet Labs, aims to provide high-resolution imagery of the whole Earth at high refresh rates by using constellations of tens to hundreds of CubeSats. The Dove satellite family used by Planet consists of three-unit CubeSats that are in effect telescopes with cameras, support electronics, and attitude control. Two Dove satellites are shown in Figure 4. Planet has extensively used components from the smartphone industry to compress the electronics to a very small size, thus making as much room as possible for the camera optics. Like smartphones and other consumer electronics, Planet does not use separately boxed electronics, but all the spacecraft electronics are integrated to a single package. Each satellite is very low cost, and expendable; there is thus a satellite-level redundancy built into the constellation.

Figure 5.
Falcon 9 uses Linux in its avionics. Photo: NASA.

The Dove satellites use COTS x86 processors in their main computers and run Ubuntu server. 0.5 terabytes of solid-state storage is available to support the imaging operations. Planet chose Linux to be able to rapidly reconfigure the OS properties for their mission-specific purposes. Some of the image processing is performed on-board, using open source imaging processing software. Part of the processing includes discarding images that are unusable due to cloud cover; avoiding downlinking such images can save downlink bandwidth. The constellations operated by Planet have successfully provided Earth imagery for several years [24], [25].

SPACEX

SpaceX, founded by Elon Musk in 2002, operates the Falcon family of launch vehicles and the unmanned Dragon spacecraft. Falcon 9, shown in Figure 5, is a commercially operated orbital launch vehicle. Dragon, shown in Figure 6, is currently used to supply the International Space Station and is planned to be man-rated in the near future. The goal of the company, as stated by Elon Musk, is to eventually establish human presence on Mars [27], [28].

SpaceX uses ordinary electronics parts in their flight electronics, as opposed to dedicated radiation-hardened parts, since systems can be made radiation tolerant with redundancy, and COTS components are easier to work with. The Dragon flight computer consists of three computer units, each of which has two indepen-

Figure 6.
Dragon, controlled by Linux-based avionics, arriving at the ISS. Photo: NASA.

dent processors. In total, the Dragon spacecraft has at least 54 different off-the-shelf processors, and Falcon 9 has at least 30. Radiation-hardened parts are not avoided because of cost, but rather because they often cannot match with commercial hardware in terms of size, power consumption, performance, tools, and support. SpaceX also selected Linux and C++ to be able to tap into the huge developer community for these environments—there are many more Linux and C++ developers than, for example, VxWorks and Ada developers. SpaceX also expects that greater availability of hardware leads to greater familiarity with the system, thus reducing bugs; flight software developers have several of the flight computers on their desks. They are also iterating and developing new versions of the computer to be used in future missions, and aiming to learn from experience. Developing new versions of the computer also allows using the latest parts available in the market. New industrial-grade parts become available much more frequently than radiation-hardened parts [27].

SpaceX began their flight software development with a combination of VxWorks for the primary computer and Linux for running communication gateways, but moved on to use a highly customized Linux everywhere after becoming comfortable with the Linux scheduler and kernel real-time patch progress. Reasons for selecting Linux included the availability of source code and thus programmability, its enterprise-level stability, availability of soft real-time patches, and its wide user community [26]. SpaceX uses Linux on their primary flight computers for Dragon spacecraft and Falcon 9 launch vehicle, and also for their test vehicles, such as Grasshopper. Their version of Linux is based on the 3.2 kernel with real-time patches. Only those functionalities needed for the SpaceX implementation have been carried from the original kernel—only around 10-15 percent of original code. SpaceX has also made their own mission-specific modifications to the kernel, and custom drivers have also been added. The kernel has been carefully evaluated, especially focusing on the scheduler performance [29].

SpaceX started spacecraft software development with the Falcon series of rockets, and transitioned the avionics code to the Dragon spacecraft. Parts of the transferred avionics code had thus already been proved in flight. Sharing the code bases also means bugs get fixed on all platforms. Reaction times are the main difference with a launch vehicle and an orbital spacecraft; a launch vehicle must typically react much faster [28].

SpaceX flight software developers use a lot of standard GNU tools such as gcc, gdb, ftrace, netfilter, and iptables. SpaceX includes extensive metrics gathering to their software, including but not limited to performance, network utilization, and CPU load. This information is collected and stored along with spacecraft telemetry and software versions in use; this allows reproduction of any encountered situation, especially useful when analyzing failures. Metrics data is automatically parsed to raise alarms if software behavior is out of ordinary. Software development processes, such as enforcing coding standards, are automated where possible [28].

POTENTIAL BENEFITS AND DRAWBACKS

As mentioned before, the idea of using Linux in spacecraft on-board computing dates back at least to 1999 to the FlightLinux project, and some of the issues identified then are still relevant. This section considers potential benefits and drawbacks of Linux use.

BENEFITS

The benefits of using Linux in spacecraft flight software are similar to those for embedded systems in general. Yaghmour [6] and Hallinan [8] have identified the following benefits for using Linux in embedded systems.

- Support for many hardware architectures–perhaps widest support by any OS–allows freedom in selection of the hardware platform
- The Linux code base can be relied upon due to its very wide adoption, and availability of source code allows verifying correctness and correcting found problems without delay
- Support for many communication protocols and standards make interfacing with external devices easy
- Availability of standardized development tools for example from the GNU project
- Large developer community provides support and developer recruitment pool
- Possibility of vendor independence, open source licensing, and low cost of adoption,

furthermore, Birrane, et al. [5] have found more specific benefits of using Linux in spacecraft flight software:

- Optional (soft) real-time capabilities
- Parts of software can be developed and debugged on desktop Linux computers and ported to flight environment with little effort
- Existence of huge catalogue of Linux software from various industries that can be taken into use in the flight computer with little effort, including data compression, file systems, operations scheduling, data processing, security-related software, algorithms, and scripting
- Standards such as POSIX allow flight software to be decoupled by modularizing it to sets of programs or processes that run on Linux, reducing development time and allowing per-program updates to the system.

Many of the projects analyzed in the previous section also noted their reasons for choosing Linux, which usually reflected the items mentioned above.

Open-source software may sometimes be perceived as error prone; however, since the software code is freely available, any possibly present bugs are detected and fixed more quickly, and the fixes can be provided back to the community. Developers may also add useful features to the software, and make them available to other users. Nearly all projects studied in this article considered the availability of Linux source code beneficial to them, as they could find problems, test modifications, and make customizations easily [30], [31].

Linux has many useful general-purpose facilities available, such as file systems, data bus interfaces, process scheduling, and interprocess communication. The POSIX abstractions have existed for decades, and most POSIX compatible software can be made to run on the embedded Linux system. Generic spacecraft software modules–handling, for example, attitude control–targeting Linux could be written and distributed commercially or noncommercially across projects. As mentioned, SpaceX has reused the same software in their launch vehicle and orbital spacecraft. Linux could thus make it easier to produce hardware-independent satellite software systems.

Shell scripting in Linux can substitute on-board control procedures that are traditionally used to automate spacecraft operations [1]. This approach was used for example in TacSat-1 [9].

DRAWBACKS AND MITIGATING THEM

Choosing Linux limits the choice of hardware; the processor must have a memory management unit, and 8-bit and 16-bit processors are mostly ruled out. The exception is µClinux, which can run on 16-bit processors without memory management units [6]. However, this is becoming a nonissue: for example, many mobile phones use 32-bit and 64-bit processors with very low power consumption. Low-power 32-bit processors can be used in place of 8-bit and 16-bit ones.

Birrane, et al. [5] noted problems in the package dependencies of many Linux programs they wanted to port to their systems. The programs were dependent on underlying packages, which contained much more features and code than what the actually needed program would use, thus leading to code bloat. When using nontailored Linux distributions, much of the available software might be unused in the actual application. If this extra software is flown, it either must be verified and validated along with the actually used software, or it must be verified that the extra software is never used during flight. While possibly time consuming, it is possible to remove the additional quality assurance burden by tailoring the custom distribution to completely remove any unused extra software [6]. As processing hardware increases in capability, the performance cost of using a general-purpose OS may become nearly negligible. The existence of "extra" software is not a performance problem if the system has enough spare memory and processing capability; it may be less expensive to pick more capable hardware than to carefully hand-tailor the distributions. On the other hand, it is also possible to customize Linux to run on very little resources [9].

A major drawback is that Linux has not been designed to be an RTOS. However, additions exist to make Linux more real-time friendly. In some cases, the number of hard real-time constraints can be reduced with design changes; and for those hard real-time constraints that remain, one option is to use a dedicated controller to handle them, or to use some real-time Linux variant. To handle the hard real-time problem, projects such as RTLinux, RTAI, xLuna, and µITRON have used a similar architectural solution of using a separate microkernel below the Linux partition, which runs Linux as the idle task of the microkernel, thus Linux is executing when no other higher priority tasks (presumably with hard real-time constraints) are blocking it [7].

Craveiro, et al. [7] and Rufino, et al. [32] describe the AIR project, which has studied ARINC 653 standard compatible methods of using Linux kernel in safety-critical hard real-time systems. Their method is based on time-space partitioning. A simplified version of the AIR architecture is shown in Figure 7. In this scenario, a microkernel running on the on-board computer executes partitions sequentially, and each partition contains its own data and executable code. One partition may then contain Linux kernel, while other partitions run ordinary RTOS kernels. The scheduling of partitions is strictly deterministic: each partition is cyclically allocated a fixed time slice. This determinism allows the development of hard real-time systems, while benefiting from the availability of Linux: Craveiro, et al. especially note that this allows using much of the software developed for Linux without tediously porting it to an RTOS, and additionally the scripting capability of Linux is made available, without specifically porting an interpreter. However, it must be made certain that the use of Linux does not affect the consistency and safety of the time-space partitioning. The authors also note that the AIR architecture ensures a fixed amount of processing time for the Linux partition and real-time partitions during each cycle, while earlier implementations that run Linux as the idle task may block Linux applications indefinitely if the higher priority tasks are under heavy load [7].

DISCUSSION

While early use of Linux in space was experimental, SpaceX and Planet have already successfully used it in commercial missions. They are possibly the most influential space industry entities to have declared using Linux in their avionics. Their success will likely influence the rate of Linux adoption elsewhere in the indus-

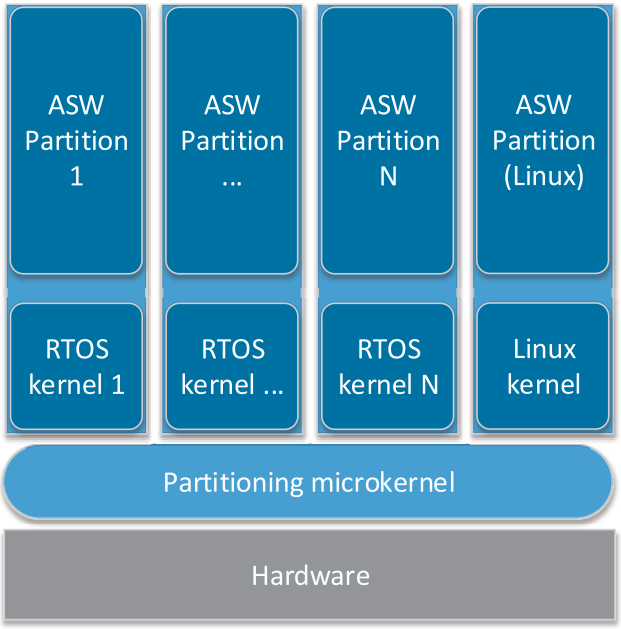

Figure 7.
Linux kernel and time-space partitioning. Adapted from [7].

try. It seems likely that use of Linux in spacecraft flight software will continue to grow, driven by the following factors:

1. Increasing complexity of demands placed on the flight software requires reuse of existing software modules, tools, and processes.
2. Computing power continues to become cheaper and is requiring less power, reducing the need for detailed optimization of software.
3. Skilled labor familiar with Linux-based mobile development is available more readily than RTOS developers

However, the hard real-time suitability of Linux still remains questionable; while we have described some possible solutions, there is no de facto standard way of making Linux hard real-time compatible. However, SpaceX has demonstrated with their Falcon 9 rocket that Linux can be used very well in hard real-time systems. Linux will not be the solution for all systems; some are still best written at low level without OS. Bringing Linux to the embedded platform allows utilizing the huge catalogue of Linux-compatible software developed during the past decades. This benefit must be weighed with the real-time considerations.

If Linux is to be used, hardware needs to be capable of running it. COTS hardware, due to its prevalent use across all industries, evolves much faster than the so-called space-grade or radiation-hardened hardware. Consumer electronics win in all performance, power consumption, and size categories; robustness, durability, and radiation tolerance is where "traditional" space hardware stands out. However, redundancy can be used to build robust systems with off-the-shelf electronics by duplicating the hardware components, or in case of Planet, by duplicating the satellites themselves.

Using Linux in space does not necessarily differ much from many terrestrial applications, and much "terrestrial" software and utilities can be reused for space applications. Many problems encountered during development may already have been solved, and the solution may be available as open source.

The ease of deploying Linux may be deceptive: it may be relatively easy to get the Linux kernel to boot on the target platform and some applications running, but deeply understanding the kernel well enough to use Linux in safety-critical software is a much bigger effort. Linux is complex, and systems built on Linux will be complex, and thus hard to test. Quality assurance remains important in preventing even simple problems such as the LightSail-1 disk overflow problem already discussed. Some systems may be quick to develop with Linux, but very hard to test; on the other hand, the same systems could be written from ground up with the same effort, and very easily tested. The difficulty of verifying and validating Linux-based systems is probably the reason for its current nonadoption in traditional space industry avionics.

CONCLUSIONS

This article discussed the use of Linux in spacecraft. Several cases found in literature were presented and studied.

It was found that Linux has several benefits for spacecraft software development, including availability of source code and development tools, large user community and pool of developers, deployment across several industries thus leading to off-the-shelf availability of many utilities and software, and support for many types of hardware. Platform-independent, Linux-targeted software could even be developed and used across various missions.

However, before Linux is to be used, questions that need to be considered include limiting hardware selection mostly to 32-bit and 64-bit architectures, uncertain and unstandardized hard real-time capabilities, and the possible presence of unused software modules that only consume resources without additional value.

It is likely that the use of Linux in all kinds of spacecraft will continue to grow. SpaceX and Planet have already commercially proven that Linux can be used equally well in all categories of spacecraft: launch vehicles, CubeSats, and, in the future, manned spacecraft. Thus Linux software could be traded between spacecraft, and everything "terrestrial" developed for Linux can also be taken into space with very little effort. The popularity of Linux is possibly one of the greatest reasons why it will continue to be taken to space. ◈

REFERENCES

[1] Eickhoff, J. *Onboard Computers, Onboard Software and Satellite Operations – An Introduction*. Heidelberg: Springer-Verlag, 2012.

[2] Anthony, S. International Space Station switches from Windows to Linux, for improved reliability. *ExtremeTech,* May 9, 2013. [Online]. Available: http://www.extremetech.com/extreme/155392-international-space-station-switches-from-windows-to-linux-for-improved-reliability

[3] Bouwmeester, J., and Guo, J. Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology. *Acta Astronautica,* Vol. 67, 78 (2010), 854.

[4] Leppinen, H., Kestilä, A., Pihajoki, P., Jokelainen, J., and Haunia, T. On-board data handling for ambitious nanosatellite missions using automotive-grade lockstep microcontrollers. In *Small Satellites Systems and Services - The 4S Symposium 2014,* May 2014.

[5] Birrane, E., Bechtold, K., Krupiarz, C., Harris, A., Mick, A., and Williams, S. Linux and the spacecraft flight software environment. In *Proceedings of the AIAA Small Satellite Conference, SSC07-XII-10,* 2007.

[6] Yaghmour, K., Masters, J., Ben-Yossef, G., and Gerum, P. *Building Embedded Linux Systems* (2nd ed). Sebastopol, CA: O'Reilly, 2008.

[7] Craveiro, J., Rufino, J., Almeida, C., Covelo, R., and Venda, P. Embedded Linux in a partitioned architecture for aerospace applications. In *2009 IEEE/ACS International Conference on Computer Systems and Applications,* May 2009, 132–138. [Online]. Available: http://dx.doi.org/10.1109/AICCSA.2009.5069315

[8] Hallinan, C. *Embedded Linux Primer* (2nd ed). Upper Saddle River, NJ: Prentice Hall, 2011.

[9] Huffine, C. Linux on a small satellite. *Linux Journal* (Mar. 1, 2005). [Online]. Available: http://www.linuxjournal.com/article/7767

[10] Kestilä, A., Tikka, T., Peitso, P., Rantanen, J., Näsilä, A., Nordling, K., et al. Aalto-1 nanosatellite - Technical description and mission objectives. *Geoscientific Instrumentation, Methods and Data Systems,* Vol. 2, 1 (2013), 121–130. [Online]. Available: http://www.geosci-instrum-method-data-syst.net/2/121/2013/

[11] Long, M., Lorenz, A., Rodgers, G., Tapio, E., Tran, G., Jackson, K. et al. A CubeSat derived design for a unique academic research mission in earthquake signature detection. In *Proceedings of the AIAA Small Satellite Conference, SSC02-IX-6,* 2002.

[12] Flagg, S., Bleier, T., Dunson, C., Doering, J., DeMartini, L., Clarke, P. et al. Using nanosats as a proof of concept for space science missions: QuakeSat as an operational example. In *Proceedings of the AIAA Small Satellite Conference, SSC04-IX-4,* 2004.

[13] Surratt, M., Loomis, H. H., Ross, A. A., and Duren, R. Challenges of remote FPGA configuration for space applications. In *2005 IEEE Aerospace Conference,* Mar. 2005.

[14] Schmidt, M., and Schilling, K. An extensible on-board data handling software platform for pico satellites. *Acta Astronautica,* Vol. 63 (2008), 1299–1304. [Online]. Available: http://dx.doi.org/10.1016/j.actaastro.2008.05.017

[15] Schilling, K. Networked distributed pico-satellite systems for Earth observation and telecommunication applications. In *IFAC Workshop on Aerospace Guidance, Navigation and Flight Control Systems,* 2011.

[16] Ramesh, B., Bretschneider, T., and McLoughlin, I. Embedded Linux platform for a fault tolerant space based parallel computer. In *Proceedings of the 2004 RealTime Linux Workshop,* 2004.

[17] Chien, S., Doubleday, J., Thompson, D., Wagstaff, K., Bellardo, J., Francis, C. et al. Onboard autonomy on the intelligent payload experiment (IPEX) Cubesat mission: A pathfinder for the proposed HyspIRI mission intelligent payload module. In *12th International Symposium in Artificial Intelligence, Robotics and Automation in Space,* 2014.

[18] Doubleday, J., Chien, S., Norton, C., Wagstaff, K., Thompson, D. R., Bellardo, J. et al. Autonomy for remote sensing – Experiences from the IPEX CubeSat. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS),* July 2015, 5308–5311. [Online]. Available: http://dx.doi.org/10.1109/IGARSS.2015.7327033

[19] Kenyon, S., Bridges, C., Liddle, D., Dyer, R., Parsons, J., Feltham, D. et al. STRaND-1: Use of a $500 smartphone as the central avionics of a nanosatellite. In *62nd International Astronautical Congress 2011 (IAC 2011),* Oct. 2011. [Online]. Available: http://epubs.surrey.ac.uk/26828/

[20] Bridges, C., Yeomans, B., Lacopino, C., Frame, T. E., Schofield, A., Kenyon, S. et al. Smartphone qualification and Linux-based tools for CubeSat computing payloads. In *Aerospace Conference, 2013,* Mar. 2013. [Online]. Available: http://dx.doi.org/10.1109/AERO.2013.6497349

[21] Manyak, G., and Bellardo, J. M. Polysat's next generation avionics design. In *2011 IEEE Fourth International Conference on Space Mission Challenges for Information Technology (SMC-IT),* Aug. 2011, 69–76. [Online]. Available: http://dx.doi.org/10.1109/SMC-IT.2011.13

[22] Ridenoure, R., Munakata, R., Diaz, A., Wong, S., Plante, B., Stetson, D. et al. LightSail program status: One down, one to go. In *Proceedings of the AIAA Small Satellite Conference, SSC15-V-3,* 2015.

[23] Salas, A., Attai, W., Oyadomari, K., Priscal, C., Schimmin, R., Gazulla, O. et al. PhoneSat in-flight experience results. In *Small Satellites Systems and Services - The 4S Symposium 2014,* May 2014.

[24] Boshuizen, C., Mason, J., Klupar, P., and Spanhake, S. Results from the Planet Labs Flock constellation. In *Proceedings of the AIAA/USU Conference on Small Satellites, SSC14-I-1,* 2014.

[25] Everard, B. Planet Labs: Putting Linux in space. *Linux Voice*, 8 (June 2015).

[26] Gruen, J. Linux in space. Presentation, 2012. [Online]. Available: https://events.linuxfoundation.org/images/stories/pdf/lcna_co2012_gruen.pdf

[27] Svitak, A. Dragon's "Radiation-Tolerant" design, Nov. 19, 2012. [Online]. Available: http://aviationweek.com/blog/dragons-radiation-tolerant-design

[28] Edge, J. ELC: SpaceX lessons learned, Mar. 6, 2013. [Online]. Available: https://lwn.net/Articles/540368/

[29] Keller, J. Introduction to SpaceX. Presentation, 2014. [Online]. Available: http://retis.sssup.it/rts-like/program.html#spacex-keller

[30] Stakem, P. Flight Linux: A viable option for spacecraft embedded computers. In *Earth Science Technology Conference,* Pasadena, CA, June 11–13.

[31] Wooster, P., Boswell, D., Stakem, P., and Cowan-Sharp, J. Open source software for small satellites. In *Proceedings of the AIAA Small Satellite Conference, SSC07- XII-3,* 2007.

[32] Rufino, J., Craveiro, J., and Verissimo, P. Building a time- and space-partitioned architecture for the next generation of space vehicle avionics. In *Software Technologies for Embedded and Ubiquitous Systems: 8th IFIP WG 10.2 International Workshop, SEUS 2010,* Waidhofen/Ybbs, Austria, October 13-15, 2010. Berlin: Springer Berlin Heidelberg, 2010, pp. 179-190. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16256-5_18