USENIX Association

Proceedings of the FREENIX Track: 2004 USENIX Annual Technical Conference

Boston, MA, USA
June 27–July 2, 2004

Towards Carrier Grade Linux Platforms

Ibrahim Haddad
Ericsson Research
8400 Decarie Blvd, Montreal
Quebec H4P 2N2, Canada
ibrahim.haddad@ericsson.com

Abstract

Traditionally, communications and data service networks were built on proprietary platforms that had to meet very specific availability, reliability, performance, and service response time requirements. Today, communications service providers are challenged to meet their needs cost-effectively for new architectures, new services, and increased bandwidth, with highly available, scalable, secure, and reliable systems that have predictable performance and that are easy to maintain and upgrade. This paper presents the technological trend of migrating from proprietary to open platforms based on software and hardware building blocks. The paper focuses on the ongoing work by the Carrier Grade Linux (CGL) working group at the Open Source Development Labs, examines the CGL architecture, the requirements from the latest specification release, and presents some of the needed kernel features that are not currently supported on Linux.

1. Open, standardized, and modular platforms

The demand for rich media and enhanced communications services is rapidly leading to significant changes in the communications industry such as the convergence of data and voice technologies. The transition to packet-based, converged, multi-service IP networks require a carrier grade infrastructure based on interoperable hardware and software building blocks, management middleware and applications, implemented with standard interfaces.

The communications industry is witnessing a technology trend moving away from proprietary systems toward open and standardized systems, built using modular and flexible hardware and software (operating system and middleware) common off the shelf components. The trend is to proceed forward delivering next generation and multimedia communication services, using open standard carrier grade platforms. This trend is motivated by the expectations that open platforms are going to reduce the cost and risks of developing and delivering rich communications services; they will enable faster time to market and ensure portability and interoperability between various components from different providers.

One frequently asked question is: How can we meet tomorrow's requirements using existing infrastructures and technologies? Proprietary platforms are closed systems, expensive to develop, and often lacking support of the current and upcoming standards. Using such *closed* platforms to meet tomorrow's requirements for new architectures and services is almost impossible. A uniform, open software environment with the characteristics demanded by telecom applications, combined with commercial off-the-shelf software and hardware components is a necessary part of these new architectures.

Three key industry consortia are defining hardware and software high availability specifications that are directly related to telecom platforms:

- The PCI Industrial Computer Manufacturers Group [1] (PICMG) defines standards for high availability (HA) hardware.

- The Open Source Development Labs [2] (OSDL) Carrier Grade Linux [3] (CGL) working group was established in January 2002 with the goal of enhancing the Linux operating system, to achieve an Open Source platform that is highly available, secure, scalable and easily maintained, suitable for carrier grade systems.

- The Service Availability Forum [4] (SA Forum) defines the interfaces of HA middleware and focusing on APIs for hardware platform management and for application failover in the application API. SA compliant middleware will provide services to an application that needs to be HA in a portable way.

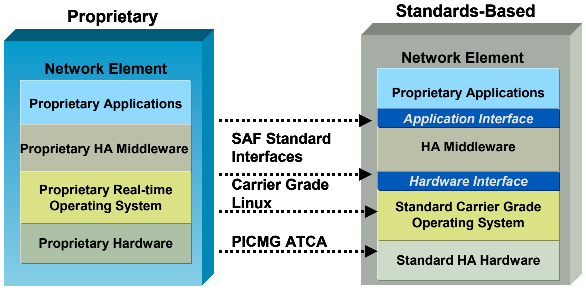

Figure 1: From Proprietary to Open Solutions

The operating system is a core component in such architectures. In the remaining of this paper, we will be focusing on CGL, its architecture and specifications.

2. The term Carrier Grade

In this paper, we refer to the term Carrier Grade on many occasions. Carrier grade is a term for public network telecommunications products that require a reliability percentage up to 5 or 6 nines of uptime.
5 nines of uptime refer to 99.999% of uptime (i.e. 5 minutes of downtime per year). This level of availability is usually associated with Carrier Grade servers.
6 nines of uptime refer to 99.9999% of uptime (i.e. 30 seconds of downtime per year). This level of availability is usually associated with Carrier Grade switches.

3. Linux versus proprietary operating systems

This section describes briefly the motivating reasons in favor of using Linux on Carrier Grade systems, versus continuing with proprietary operating systems. These motivations include:

- Cost: Linux is available free of charge in the form of a downloadable package from the Internet.
- Source code availability: With Linux, you gain full access to the source code allowing you to tailor the kernel to your needs.
- Open development process (Figure 2): The development process of the kernel is open to anyone to participate and contribute. The process is based on the concept of “release early, release often.”
- Peer review and testing resources: With access to the source code, people using a wide variety of platform, operating systems, and compiler combinations; can compile, link, and run the code on their systems to test for portability, compatibility and bugs.
- Vendor independent: With Linux, you no longer have to be locked into a specific vendor. Linux is supported on multiple platforms.
- High innovation rate: New features are usually implemented on Linux before they are available on commercial or proprietary systems.

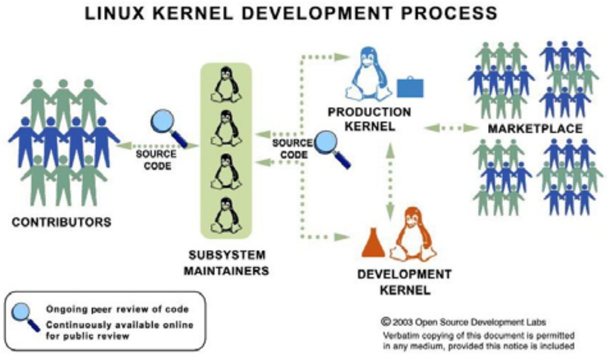
LINUX KERNEL DEVELOPMENT PROCESS
CONTRIBUTORS
SOURCE CODE
SUBSYSTEM MAINTAINERS
SOURCE CODE
PRODUCTION KERNEL
MARKETPLACE
DEVELOPMENT KERNEL
Ongoing peer review of code
Continuously available online for public review
© 2003 Open Source Development Labs
Verbatim copying of this document is permitted in any medium, provided this notice is included

Figure 2: Open development process of the Linux kernel

Other contributing factors include Linux’ support for a broad range of processors and peripherals, commercial support availability, high performance networking, and the proven record of being a stable, and reliable server platform.

4. Carrier Grade Linux

The Linux kernel is missing several features that are needed in a telecom environment, and it is not adapted to meet telecom requirements in various areas such as reliability, security, and scalability. To help the advancement of Linux in the telecom space, OSDL established the CGL working group. The group specifies and help implement an Open Source platform targeted for the communication industry that is highly available, secure, scalable and easily maintained, suitable for carrier grade systems.
The CGL working group is composed of several members from network equipment providers, system integrators, platform providers, and Linux distributors, all of them contributing to the requirement definition of Carrier Grade Linux, helping Open Source projects to meet these requirements, and in some cases starting new Open Source projects. Many of the CGL members companies have contributed pieces of technologies to Open Source in order to make the Linux Kernel a more viable option for telecom platforms. For instance, the Open Systems Lab [5] from Ericsson Research has contributed three key technologies: the Transparent IPC [6], the Asynchronous Event Mechanism [7], and the Distributed Security Infrastructure [8]. In a different

direction, there are already Linux distributions, MontaVista [10] for instance, that are providing CGL distribution based on the CGL requirement definitions. Many companies are also either deploying CGL, or at least evaluating and experimenting with it.

Consequently, CGL activities are giving much momentum for Linux in the telecom space allowing it to be a viable option to proprietary operating system. Member companies of CGL are releasing code to Open Source and making some of their proprietary technologies open, going forward from closed platforms to open platforms that use CGL.

5. Target CGL applications

The CGL Working Group has identified three main categories of application areas into which they expect the majority of applications implemented on CGL platforms to fall. These application areas are *gateways*, *signaling,* and *management servers*.

- *Gateways* are bridges between two different technologies or administration domains. For example, a media gateway performs the critical function of converting voice messages from a native telecommunications time-division-multiplexed network, to an Internet protocol packet-switched network. A gateway processes a large number of small messages received and transmitted over a large number of physical interfaces. Gateways perform in a timely manner very close to hard real-time. They are implemented on dedicated platforms with replicated (rather than clustered) systems used for redundancy.

- *Signaling servers* handle call control, session control, and radio recourse control. A signaling server handles the routing and maintains the status of calls over the network. It takes the request of user agents who want to connect to other user agents and routes it to the appropriate signaling. Signaling servers require soft real time response capabilities less than 80 milliseconds, and may manage tens of thousands of simultaneous connections. A signaling server application is context switch and memory intensive due to requirements for quick switching and a capacity to manage large numbers of connections.

- *Management servers* handle traditional network management operations, as well as service and customer management. These servers provide services such as: a Home Location Register and Visitor Location Register (for wireless networks) or customer information (such as personal preferences including features the customer is authorized to use). Typically, management applications are data and communication intensive. Their response time requirements are less stringent by several orders of magnitude, compared to those of signaling and gateway applications.

6. Overview of the CGL working group

The CGL working group has the vision that next-generation and multimedia communication services can be delivered using Linux based open standards platforms for carrier grade infrastructure equipment. To achieve this vision, the working group has setup a strategy to define the requirements and architecture for the Carrier Grade Linux platform and develop a roadmap for the platform and to promote development of a stable platform upon which commercial components and services can be deployed.

In the course of achieving this strategy, the OSDL CGL working group, is creating the requirement definitions, and identifying existing Open Source projects that support the roadmap to implement the required components and interfaces of the platform. When an Open Source project does not exist to support a certain requirement, OSDL CGL is launching (or support the launch of) new Open Source projects to implement missing components and interfaces of the platform.

The CGL working group consists of three distinct sub-groups that work together. These sub-groups are: specification, proof-of-concept, and validation. Explanations of the responsibilities of each sub-group are as follows:

Specifications: The specifications sub-group is responsible for defining a set of requirements that lead to enhancements in the Linux kernel, that are useful for carrier grade implementations and applications. The group collects, categorizes, and prioritizes the requirements from participants to allow reasonable work to proceed on implementations. The group also interacts with other standard defining bodies, open source communities, developers and distributions to ensure that the requirements identify useful enhancements in such a way, that they can be adopted into the base Linux kernel.

- Proof-of-Concept: This sub-group generates documents covering the design, features, and technology relevant to CGL. It drives the implementation and integration of core Carrier Grade enhancements to Linux as identified and prioritized by the requirement document. The group is also responsible for ensuring the integrated enhancements pass, the CGL validation test suite and for establishing and leading an open source umbrella project to coordinate implementation and integration activities for CGL enhancements.

- Validation: The sub-group defines standard test environments for developing validation suites. It is responsible for coordinating the development of validation suites, to ensure that all of the CGL requirements are covered. This group is also responsible for the development of an Open Source project CGL validation suite.

7. CGL architecture

Figure 3 presents the scope of the CGL Working Group, which covers two areas:

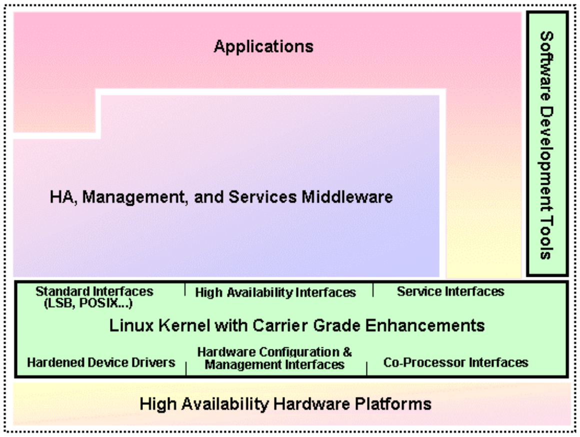
Applications
HA, Management, and Services Middleware
Software Development Tools
Standard Interfaces (LSB, POSIX...)
High Availability Interfaces
Service Interfaces
Linux Kernel with Carrier Grade Enhancements
Hardened Device Drivers
Hardware Configuration & Management Interfaces
Co-Processor Interfaces
High Availability Hardware Platforms

Figure 3: CGL architecture and scope

1. *Carrier Grade Linux*: Various requirements such as availability and scalability are related to the CGL enhancements to the operating system. Enhancements may also be made to hardware interfaces, interfaces to the user level or application code and interfaces to development and debugging tools. In some cases, to access the kernel services, user level library changes will be needed.

2. *Software Development Tools*: These tools will include debuggers and analyzers.

On October 9, 2003, OSDL announced the availability of the OSDL Carrier Grade Linux Requirements Definition, Version 2.0 (CGL 2.0). This latest requirement definition for next-generation carrier grade Linux offers major advances in security, high availability, and clustering.

8. CGL 2.0 requirements

The requirement definition document of CGL version 2.0 introduced new and enhanced features to support Linux as a carrier grade platform. The CGL requirement definition divides the requirements in main categories described briefly below:

8.1 Clustering

These requirements support the use of multiple carrier server systems to provide higher levels of service availability through redundant resources and recovery capabilities, and to provide a horizontally scaled environment supporting increased throughput.

8.2 Security

The security requirements are aimed at maintaining a certain level of security while not endangering the goals of high availability, performance, and scalability. The requirements support the use of additional security mechanisms to protect the systems against attacks from both the Internet and intranets, and provide special mechanisms at kernel level to be used by telecom applications.

8.3 Standards

CGL specifies standards that are required for compliance for carrier grade server systems.

Examples of these standards include:

- Linux Standard Base
- POSIX Timer Interface
- POSIX Signal Interface
- POSIX Message Queue Interface
- POSIX Semaphore Interface
- IPv6 RFCs compliance
- IPsecv6 RFCs compliance
- MIPv6 RFCs compliance
- SNMP support
- POSIX threads

8.4 Platform

OSDL CGL specifies requirements that support interactions with the hardware platforms making up carrier server systems. Platform capabilities are not tied to a particular vendor's implementation.

Examples of the platform requirements include:

- Hot insert: supports hot-swap insertion of hardware components.
- Hot remove: supports hot-swap removal of hardware components.
- Remote boot support: supports remote booting functionality.
- Boot cycle detection: supports detecting reboot cycles due to recurring failures.

- Diskless systems: support diskless systems which load and run applications via the network.

8.5 Availability

The availability requirements support heightened availability of carrier server systems, such as improving the robustness of software components or by supporting recovery from failure of hardware or software.
Examples of these requirements include:

- RAID 1: support for RAID 1 offers mirroring to provide duplicate sets of all data on separate hard disks.
- Watchdog timer interface: support for watchdog timers to perform certain specified operations when timeouts occur.
- Support for Disk and volume management: to allow grouping of disks into volumes.
- Ethernet link aggregation and link failover: support bonding of multiple NIC for bandwidth aggregation and provide automatic failover of IP addresses from one interface to another.
- Support for application heartbeat monitor: monitor applications availability and functionality.

8.6 Serviceability

The serviceability requirements support servicing and managing hardware and software on carrier server systems. These are wide-ranging set requirements that, put together, help support the availability of applications and the operating system.
Examples of these requirements include:

- Support for producing and storing kernel dumps.
- Support for dynamic debug of the kernel and running applications.
- Support for platform signal handler enabling infrastructures to allow interrupts generated by hardware errors to be logged using the event logging mechanism.
- Support for remote access to event log information.

8.7 Performance

OSDL CGL specifies the requirements that support performance levels necessary for the environments expected to be encountered by carrier server systems. Examples of these requirements include:

- Support for application (pre) loading.
- Support for soft real time performance through configuring the scheduler to provide soft real time support with latency of 10 ms.
- Support Kernel preemption.
- Provide Raid 0 support to enhance performance.

8.8 Scalability

These requirements support vertical and horizontal scaling of carrier server systems such as the addition of hardware resources to result in acceptable increases in capacity.

8.9 Tools

The tools requirements provide capabilities to facilitate diagnosis. Examples of these requirements include:

- Support the usage of a kernel debugger.
- Support for Kernel dump analysis.
- Support for debugging multi-threaded programs

9. CGL 3.0

The work on the next version of the OSDL CGL requirements, version 3.0, started in January 2004 with focus on advanced requirement areas such as manageability, serviceability, tools, security, standards, performance, hardware, clustering and availability. With the success of CGL's first two requirement documents, OSDL CGL working group anticipate that their third version will be quite beneficial to the Carrier Grade ecosystem. Official release of the CGL requirement document Version 3.0 is expected in October 2004.

10. CGL implementations

There are several enhancements to the Linux Kernel that are required by the communication industry, to help adopt Linux on their carrier grade platforms, and support telecom applications. These enhancements (Figure 4) fall into the following categories availability, security, serviceability, performance, scalability, reliability, standards, and clustering.

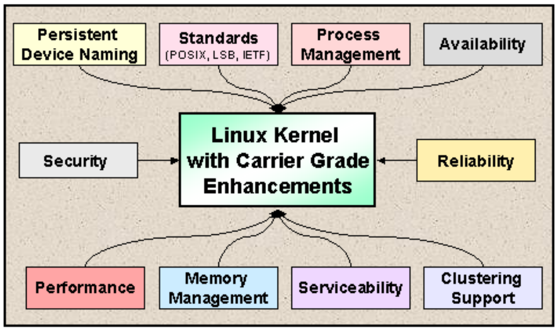

Figure 4: CGL enhancements areas

The implementations providing theses enhancements are Open Source projects and planned for integration with the Linux kernel when the implementations are mature, and ready for merging with the kernel code. In some cases, bringing some projects into maturity levels takes a considerable amount of time before being able to request its integration into the Linux kernel. Nevertheless, some of the enhancements are targeted for inclusion in kernel version 2.7. Other enhancement will follow in later kernel releases. Meanwhile, all enhancements, in the form of packages, kernel modules and patches, are available from their respective project web sites.

The CGL 2.0 requirements are in-line with the Linux development community. The purpose of this project is to form a catalyst to capture common requirements from end-users for a CGL distribution. With a common set of requirements from the major Network Equipment Providers, developers can be much more productive and efficient within development projects. Many individuals within the CGL initiative are also active participants and contributors in the Open Source development community

11. Examples of missing features from the Linux Kernel

In this section, we provide some examples of missing features and mechanisms from the Linux kernel that are necessary in a telecom environment.

11.1 IPv4, IPv6, MIPv6 forwarding tables fast access and compact memory with multiple FIB support

Linux should be able to run in a routing environment with fast recovery of routes when network failure is detected. This is achievable by having around 2000 routes/sec. Latency is not really an issue in a PC environment (a few ms doesn't make a big difference). What is important is to have a predictable performance from 10.000 to 500.000 routes. However, the faster is always better.

The actual implementation of the IP stack in Linux works fine for host or small router. However, with the high requirements in telecom, it becomes impossible to develop using Linux a high-end router for large network (core/border/access router) or a high-end server with routing capabilities.

The problem we are facing with Linux is the lack of support for multiple forwarding information bases (multi-FIB) with overlapping interface's IP address and appropriate interfaces for addressing FIB(VR). The route table is not scalable.

Another objective is to support multi-FIB with overlapping IP address. We can have on different VLAN or different physical interface, independent network in the same Linux box. For example, you can have 2 HTTP servers serving 2 different networks with potentially the same IP address. One HTTP serves the network/FIB 10, and the other serves the network/FIB 20. So the advantage you have is to have 1 Linux box serving 2 different customers with the own networks. (i.e. ISP with big companies using there services). So the only way to achieve that is to have an ID to completely separate the table in memory. (i.e. can be separate table or the ID is just append at the beginning of the key).

Another problem arise when we are not able to predict access time, with the chaining in the hash table of the routing cache (and FIB). This problem is of particular interest in environment that requires predictable performance.

Another aspect of the problem is that the route cache and the routing table are not kept synchronized most of the time (path MTU, just to name one). The route cache flush is executed regularly therefore any updates on the cache are lost. For example, if you have a routing cache flush, you have to rebuild every route that you are currently talking to. To achieve that, you need to go for every route in the hash/try table and rebuild the information. So you first have to lookup in the routing cache, if you have a miss, you need to go in the hash/try table. It's a very slow and not predictable because in the hash/try table with linked list with also a lot of potential collision when a large number of routes are present. This design is perfect for a home PC with a few routes, but it is not scalable for a large server.

To support the various routing requirements of telecom platforms, Linux should support:

- Implementation of multi-FIB using tree (radix, patricia, etc.). It is very important to have predictable performance in insert/delete/lookup from 10 to 500k routes. And, if possible, to have the same data structure for both IPv4 and IPv6.
- Socket and ioctl interfaces for addressing multi-FIB, and
- Multi-FIB support for neighbors (arp)

Providing these implantations in Linux will affect a large part of net/core, net/ipv4 and net/ipv6; these subsystems (mostly network layer) will need to be rewritten. Other areas will have minimal impact at the source code level, mostly at the transport layer (socket, TCP, UDP, RAW, NAT, IPIP, IGMP, etc).

There is no Open Source solutions or patches that are available.

11.2 Efficient low-level asynchronous event mechanism

Operating systems for telecom applications must ensure that they can deliver a high response rate with minimum downtime, less than five minutes per year of downtime, including hardware, operating system and software upgrade. In addition to this goal, a carrier-grade system also must take into account such characteristics as scalability, high availability and performance.

For such systems, thousands of requests must be handled concurrently without affecting the overall system's performance, even under extremely high loads. Subscribers can expect some latency time when issuing a request, but they are not willing to accept an unbounded response time. Such transactions are not handled instantaneously for many reasons, and it can take some milliseconds or seconds to reply. Waiting for an answer reduces applications' abilities to handle other transactions.

Many different solutions have been envisaged to improve Linux's capabilities using different types of software organization, such as multithreaded architectures, by implementing efficient POSIX interfaces or by improving the scalability of existing kernel routines. We think that none of these solutions are adequate for true Carrier Grade servers.

As a result, Ericsson has designed and developed the needed mechanism for telecom application and released it to Open Source under the GPL license. The solution is called Asynchronous Event Mechanism (AEM); it provides asynchronous execution of processes in the Linux kernel. AEM implements a native support for asynchronous events in the Linux kernel and aims to bring carrier-grade characteristics to Linux in areas of scalability and soft real-time responsiveness. In addition, AEM offers event-based development framework, scalability, flexibility and extensibility.

AEM has been announced on the Linux Kernel Mailing List (LKML) and received feedback that resulted in some changes to the design and implementation. AEM is not yet integrated with the Linux kernel. More information on AEM is available from [7].

11.3 Transparent inter-process and inter-processor communication protocol

Today's telecommunication environments are increasingly adopting clustered servers to gain benefits in performance, availability, and scalability. The resulting benefits of a cluster are greater or more cost-efficient than what a single server can provide. Furthermore, the telecommunication industry's interest in clustering originates from the fact that clusters ad-dress carrier-class characteristics such as guaranteed service availability, reliability and scaled performance, using cost-effective hardware and software. Without being absolute about these requirements, they can be divided in these three categories: short failure detection and failure recovery, guaranteed availability of service, and short response times.

The most widely adopted clustering technique is use of multiple interconnected loosely coupled nodes to a single highly available system.

One missing feature from Linux in this area is a reliable and efficient inter-process and inter-processor communication protocol. However, there exist an Open Source project, Trans-parent Inter Process Communication (TIPC) protocol, which is specially designed for efficient intra cluster communication, leveraging the particular conditions present within loosely coupled clusters. It runs on Linux and is provided as a portable source code package implementing a loadable kernel module.

TIPC is unique from the perspective that there seems to be no other protocol providing a comparable combination of versatility and performance. The functional addressing scheme is an original innovation, as is the topology subscription services and its "reactive connection" concept. TIPC is a useful toolbox for anyone wanting to develop or use Carrier Grade or Highly Available clusters on Linux. It provides the necessary infrastructure for cluster, network and software management functionality, as well as a good support for designing site-independent, scalable, distributed, high-availability and high-performance applications.

Some of the most important TIPC features include full location transparency, lightweight connections, reliable multicast, signaling link protocol, topology subscription services and more.

TIPC is a contribution from Ericsson to Open Source. It will be announced to LKML in mind-May 2004, two weeks after I submit the paper the USENIX. However, more recent news regarding TIPC will be included in the USENIX presentation. TIPC is licensed under a dual GPL and BSD license. More information on TIPC is available from [6][11].

11.4 Run-time authenticity verification for system binaries

Linux has generally been considered immune to the spread of viruses, backdoors and Trojan programs on the Internet. However, with the increasing popularity of Linux as a desktop platform, the risk of seeing viruses or Trojans developed for this platform are rapidly growing. To alleviate this problem, the system should prevent on run time the execution of untrusted software. One solution is to digitally sign the trusted binaries and have the system check the digital signature of binaries before running them. Therefore, untrusted (not signed) binaries are denied the execution. This can improve the security of the system by avoiding a wide range of malicious binaries like viruses, worms, Torjan programs and backdoors from running on the system.

DigSig Linux kernel module checks the signature of a binary before running it [9][12]. It inserts digital signatures inside the ELF binary and verifies this signature before loading the binary. It is based on the Linux Security Module hooks (main stream Linux kernel from 2.5.X and higher).

Typically, in this approach, vendors do not sign binaries; the control of the system remains with the local administrator. S/he is responsible to sign all binaries s/he trusts with his/her private key. Therefore, DigSig guarantees two things: (1) if you signed a binary, nobody else than you can modify that binary without being detected, and (2) nobody can run a binary which is not signed or badly signed.

There have already been several initiatives in this domain, such as Tripwire, BSign, Cryptomark [14][15][16]. We believe the DigSig project is the first to be both easily accessible to all (available on SourceForge, under the GPL license), and it operates at kernel level on run time. The run time is very important for CGL as this takes into account the high availability aspects of the system.

The DigSig approach has been to use the existing solutions like GPG [13] and BSign [15] (a Debian package) rather than reinventing the wheel. However, in order to reduce the overhead in the kernel, the DigSig project only took the minimum code necessary from GPG. This helped much to reduce the amount of code imported to the kernel in source code of the original (only 1/10 of the original GnuPG 1.2.2 source code has been imported to the kernel module).

12. Conclusion

There are many challenges accompanying the migration from proprietary to open platforms. The main challenge remains to be the availability of the various kernel features and mechanisms needed for telecom platforms and integrating these features in the Linux kernel.

Carrier Grade Linux is a cooperative initiative aiming to advance the Linux in the communications space and provide an alternative away from proprietary carrier grade operating systems. The participation in OSDL CGL is open to everyone. For more information, please visit the OSDL web site.

References

[1] PCI Industrial Computer Manufacturers Group,
http://www.picmg.org

[2] Open Source Development Labs,
http://www.osdl.org

[3] Carrier Grade Linux,
http://osdl.org/lab_activities/carrier_grade_linux

[4] Service Availability Forum,
http://www.saforum.org

[5] Open System Lab,
http://www.linux.ericsson.ca

[6] Transparent IPC,
http://tipc.sf.net

[7] Asynchronous Event Mechanism,
http://aem.sf.net

[8] An Event Mechanism for Linux, Linux Journal, July 2004,
http://www.linuxjournal.com/print.php?sid=6777

[9] Distributed Security Infrastructure,
http://disec.sf.net

[10] MontaVista Carrier Grade Edition
http://www.mvista.com/cge/index.html

[11] Make Clustering Easy with TIPC, LinuxWorld Magazine, April 2004,
http://www.linux.ericsson.ca/papers/tipc_lwm/

[12] Stop Malicious Code Execution at Kernel Level, LinuxWorld Magazine, January 2004,
http://www.linux.ericsson.ca/papers/digsig_lwm.pdf

[13] GnuPG,
http:// www.gnupg.org

[14] Tripwire,
http://www.tripwire.com

[15] Bsign,
http://packages.qa.debian.org/b/bsign.html

[16] Cryptomark,
http://www.immunix.org/cryptomark.html