Linux for Safety Critical Systems in IEC 61508 Context

Nicholas Mc Guire
Safety Coordinator OSADL
Distributed and Embedded Systems Lab
Lanzou University
safety@osadl.org

Abstract

"If computers systems technology is to be effectively and safely exploited, it is essential that those responsible for making decisions have sufficient guidance on the safety aspects on which to make these decisions" [IEC 61508-1 Introduction]

Is there enough guidance on COTS/OSS ?

The simple answer is no - but IEC 61508 is designed in a relatively open way - considering when it was written and that the authors were aware of a standard needing to be flexible enough to accommodate emerging technologies without breaking the fundamental concepts. *"...has been conceived with a rapidly developing technology in mind..."* [IEC 61508-1 Introduction]

So are the fundamental concepts of IEC 61508 applicable in COTS/OSS based systems ?

There is no simple answer to this one - but we believe it is yes.

In this article we will point out some main issues of using COTS/OSS software in the context of 61508 (and derivative) compliant safety-related systems. We will sketch what basic arguments are available, what the shortcomings of GNU/Linux and specifically of the Linux kernel are and what is available to address these shortcomings. Then we follow 61508s criteria and see what fits and what could be problematic followed by a brief outline of a general strategy in developing of COTS/OSS based safety cases based on the concept of cross-mapping application sector specific standards, concluded by a somewhat speculative view of in what direction we believe standards are developing and why this is good for COTS/OSS.

1 Introduction

The issue of using COTS software component is of quite general interest for the past 10 years at least. One can see not only in articles [7] [6], reports, i.e. by HSE [8] [9] as well as recent publications on use of COTS/Linux [**?**] [4], but also in standards that evolved around IEC 61508, that there is a continuous growing interest in finding safe strategies to integrate COTS software into safety critical systems. Open-Source is a special case of COTS software, in some respect better suited for validation and verification in some respect skeptically observed because of the lack of a formal vendor and the typically associated information available for products.

In this article we outline 61508 and its related standards in a very brief way, the relevance of this relation for efforts to integrate OSS components - specifically GNU/Linux - in 61508 context, and try to outline possible paths to guide such activities through 61508.

It should be noted that this paper is neither claiming that a particular OSS component can under all circumstances be included nor are we claiming that OSS is "the better solution" by any principal - it has its potentials and it has its shortcomings - in a safety context this means managing it just like any other component and adjusting strategies to select the right components for a give system.

As we are only interested in software in 61508 context we give a very rough overview of 61508 here to set the context for the discussion following:

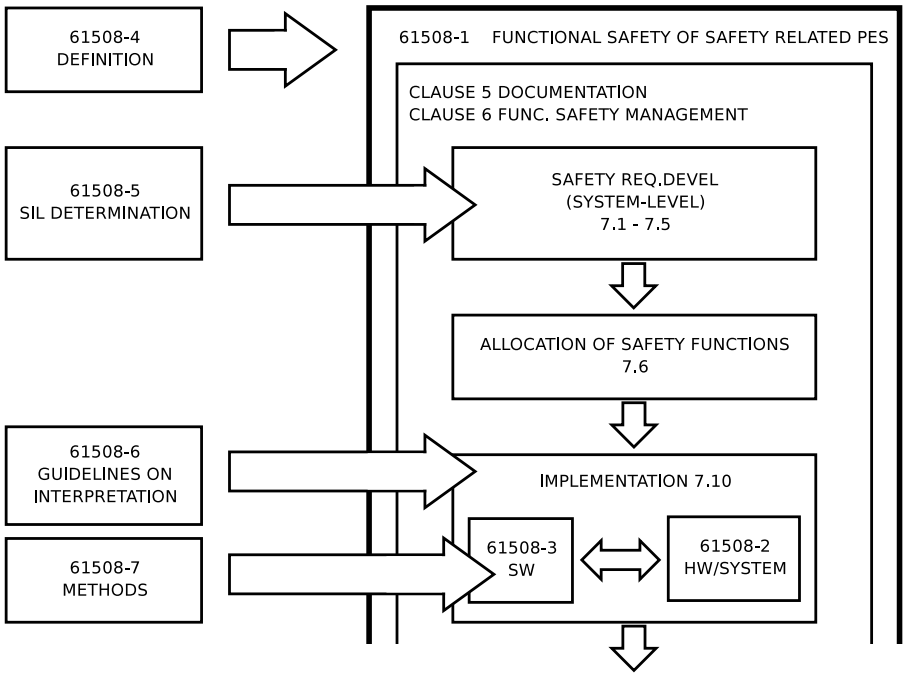

FIGURE 1: *61508 big-picture (SW only)*

2 Proven-in-use

While many people have a gut feeling that GNU/Linux is maturing and has a de-facto test-coverage that is superior to commercial counterparts - there obviously are problems quantifying this reliability. In this section we will first look at factors we consider supporting GNU/Linux in safety critical applications and then look into the problem areas that could be show-stoppers in some cases. Before we do that we would like to point out the safety related development in GNU/Linux that happened basically due to many of the needs of safety related systems matching with general needs of complex systems and high-availability in specific.

Although this should be clear, we will explicitly state here that proven-in-use for Linux, only based on "operational evidence" will not suffice to argue SIL1 or higher. The proven-in-use argument based only on operational hours and 10 use-cases (61508-2) are, in our opinion, quite clearly aimed at low-complexity hardware and not at a highly configurable and variable software as versatile as the Linux kernel.

2.1 GNU/Linux evolution

[**?**] published a study on Linux for safety critical systems identifying a number of problem areas and giving a carefully positive conclusion on using GNU/Linux in SIL1 and SIL2, with SIL3 being at least problematic. Many of the questions raised in the study, related to the 2.4.X series of kernels, have been addressed in 2.6.X - not due to the requirements of safety systems though. The main shortcomings identified by Piercen were:

- Tractability of the source code, documents and specifications
- There is no single specification
- Lack of hard real-time capability (temporal predictability)
- There is only limited overload tolerance

One of the main conclusion: Linux is not suitable for SIL4.

Aside from the problem that SIL3/SIL4 in this publication is not really clearly defined (multiple standards are listed - some of which don't specify SIL4 level (i.e. 61511/6261) - the criticism is not only well established but also quite clearly attributable to Linux 2.4.X "qualities" or rather the lack of these. The evolution of Linux in the 2.6.X kernel series, especially since the introduction of git [2] and a well defined kernel development life-cycle along with the technological advances in the area of real-time and improvement of robustness relativate much of the criticism expressed. The issue of the non-existing

single specification is of course still valid - but that is an inherent property of many aggregated safety critical systems that utilize COTS components (i.e. IEC 6261 Clause 6.5). It should be noted though that Linux is targeting POSIX:

```
What is Linux?
  Linux is a clone of the operating system
  Unix, written from scratch by Linus
  Torvalds  with assistance  from a
  loosely-knit team of hackers across
  the  Net.  It aims towards POSIX and
  Single UNIX Specification compliance.
```

This does not mean that Linux is well specified (at least not in 61508s sense of well specified) - but it is highly questionable if COTS OS, based on UNIX, used in safety-related systems had much more of a specification that Single Unix Specification - and with the availability of test-suits and regression tests focused on the single-unix specification [3] we believe this provides a acceptable mitigation to this problem - provided of course restricted use is applied in application context. Note especially that susV is an accepted IEC standard [ISO/IEC 9945-2003] that actually includes a rational....

2.2 61508 Criteria

61508 and its derived standards are a bit obsessed with:

- system size
- system complexity
- novelty of design
- novelty of technology

These for items are cited over and over though all of 61508 and derived standards as "depending on....". If we look at Linux with these criteria we would make the following claim:

- Linux as a system component is large , by 61508 standards huge (note annex E of 61508-6 refers to a COTS kernel with 30000 LoC...off by two orders of magnitude. Though the size of the Linux kernel core is actually not that large:

  Though still large - this conservative estimation indicates that Linux source is not any where near the 5million lines of code that can be found as claims on the Internet. Linux is currently 24 main CPU families with hundreds of CPUs supported - de facto even the above estimation of all LoC in `arch/i386` being included in the kernel binary is unrealistic even with an excessive configuration. The real size of Linux comes quite close to the COTS kernel mentioned in 61508-6 sample safety-case for SIL3 (appendix-E).

- Linux kernel is complex - I doubt anybody will dispute this even if we give no evidence here. But depending on the software architecture the internal complexity may not be that relevant in some cases. IEC 62061 allows subsystem-elements (software components) of high internal complexity to be treated as low-complexity systems provided it complies with IEC 61508-2/3 and "...its relevant failure modes, behavior on detection of a fault, its failure rate and other safety-related informations are known.." [IEC 62061 Clause 6.7.4.2.3].

- The design is conservative and has a long history - it hardly qualifies as revolutionary when it comes to design - monolithic kernel model is roughly 1970, primary design decisions follow the needs of the guiding standards like POSIX (which also does not qualify as revolutionary).

- Linux developers are conservative with respect to technologies used - much of the concepts moving into the kernel now have been published before Linux 0.1 was released in 1991 ! The actual implementations are of course not simply ports of old implementations - but the technologies that do go in are well tested and well understood (though there surely are exceptions).

So we believe that the focus of supportive evidence will need to be rested on justifying why the complexity and sheer size can be accepted - this is non-trivial and should not be underestimated. Mitigation of the first two clearly are:

- Level of documentation - there hardly is any kernel around that is documented as rigorously as the Linux kernel, both at the general level as well as the detailed implementation level.

- Tractability of the Linux kernel - especially in 2.6.X has been improved to a level that makes the size manageable - but companies must be well aware of the need to invest in there engineers.

2.3 Non technical issues

A issue, to our knowledge typically underestimated or plainly ignored is listed in Appendix B of 61508-1 in detail - the issue of "..the training, experience and qualification of all persons involved..." - OSS is a paradigm change mandating an appropriate response. I personally would claim that the probability of a OSS/COTS

based safety-related system failing is at least equally probable due to the lack of understanding of the nature and specifics of OSS as it is with respect to standards and regulatory issues.

The move towards OSS in safety-related systems must be managed just like the introduction of any fundamentally new technology - underestimating the specifics of OSS, or assuming that experience in overall safety-related systems is sufficient is one of the critical points in the process of introducing OSS.

Critical issues we will briefly note here are:

- adhering to the rules of the community to actually get access to the claimed benefit of OSS - i.e. peer review nature of the community.
- the issue of highly asynchronous development - OSS systems like GNU/Linux are built of a large number of packages developed independent, at different speeds and not "bundled" like classical proprietary vendor provided environments.
- fundamental change of tooling in OSS - there is no point in trying to run a OSS based development if the management refuses to accept current technologies like git. Such refusals can put a tremendous burden on the project management and reduce accessibility to safety related information in a critical way.
- OSS selection "..selection based on prior use.." [61511] or "Requirements for selection of existing (pre-designed) subsystems" [62061 6.7.3] must be investigated - typically these sections were not consulted in bespoke software life-cycles and there is insufficient established practice - these processes must thus be expected to be relatively slow during the first projects.
- In general one should expect that there is a certain shift in the SW life-cycle, if OSS is to be utilized, towards the investigation phase or teams will suffer the classical "reinvented wheel trauma" - typically resulting in triangular wheels.
- Community interaction - commercial entities need policies to interact with the community - it is absurd to utilize OSS and prohibit employees from joining the respective community mailing lists - this is though not that uncommon !
- The inherent danger of de-coupling from the community effort "because there work did not fit our needs" - well thats no how it works and trying to go that path will easily cause a fork, resulting in loss of arguments for proven-in-use

Without claiming completeness here - this is just a short rant to call to your attention that OSS is not simply a pool of freely available code but that it is much more - it is a fundamental decision that is needed early in the project life-cycle if a OSS based project should be successful.

2.4 Available evidence

The specifics of the development cycle of GNU/Linux mandate a certain set of tools so that development does not fall apart. By all standards of company practice I would claim that the development of Linux kernel by now has a level or rigor that is quite hard to find in industrial projects - this is not only due to the sheer size and complexity, but also to the very wide platform support and the large number of independently operating individuals and groups. We see some developments in Linux that facilitate high-quality evidence - some of these developments are:

- advances in the kernel software life-cycle:
  - introduction of subsystem maintainers
  - developer branches and arch branches for early testing of features (i.e. arm.linux.org.uk)
  - well defined experimental tree (-mm) and the introduction of the merge-window
  - early testing in the release candidates (rcX)
  - and long term road-maps for feature introduction (i.e. RT-preempt is being merged in steps since early 2.6.X)
- high-level management elements introduced in 2.6 - beyond LKML
  - Annual kernel summit for strategic decisions
  - domain specific groups (i.e. CELinuxForum Architecture Group for consumer electronics)
  - Auditing introduced for critical API (i.e. raw_spin_lock usage)
  - change-log management
  - improved maintenance of kernel specific information (i.e. lwn.net, kerneltrap.org changelogs)
- Testing and validation
  - critical resources include built-in-tests (i.e. RCU torture test, lock-dependency validator), especially in 2.6.X the development of built-in-tests have resulted in detection of a large number of bugs without that these ever struck in the field.

- Linux Test Project (LTP) providing a high-level test-coverage of the Linux kernel [1] providing roughly 3000 tests for the Linux OS (ltp-20070831)
- crackerjack - kernel code coverage test-suit
- `http://test.kernel.org` - Autotest is a framework for fully automated testing of the latest linux kernel releases - published online and available to the public.
- POSIX test-suit

Along with this we see the tendency to actually enforce long standing policies like kernel coding rules, in-source documentation (i.e. which commercial kernel can compete with "make psdocs" ?). These developments are not targeting safety critical systems, but are rather the consequence of the way the development is organized, the "loose gang" of developers only can succeed in a project of this complexity by adhering to a very strict set of rules regarding source management and software modification.

3 Strategy for Justification

61508 does not fit OSS/COTS that well - in part because it is a high-level approach towards functional safety that is not based on constraints like "fail-safe" or "low demand mode" only - thus many of the requirements will not be found in the application sector specific standards while the overall justification methodology does continue to adhere to 61508.

Application domain specific standards like 50128/62061/61511/etc. are based on consensus of industrial users of these standards and anticipate covering the "mainstream" of the respective domain. While this strategy is quite obviously sensible, to prevent special case overload in standards, it does raise the question how to handle cases that are not explicitly, or worse, not even implicitly, covered by the standard. In this case we propose the following strategy:

- select one of the other domain specific standards that better fits your application context (reactive/composite safety, level of complexity, mode of failure (i.e. fail-safe), etc.)
- derive the justification according to this domains standard model, of course adjusting it to the specifics of the domain under consideration where needed.
- argue ("justify") the non-standards compliance of the safety case based on the procedure being derived from the same top-level standard (61508) and the respective SIL claims of the standards (which basically are in sync with minor variations).

This proposal only makes sense if we also provide guidance of what could be the suitable "cross-selection". This is work-in-progress and definitely not completed, but we do believe that we can give some guidance that is of help.

It should also be noted that Linux has be certified in projects that conform to other standards (i.e. ATC systems guided by CAP 670), and this, though based on a completely different standard, does constitute a strong indication of the maturity and the available evidence base (CAP 670 is a evidence based safety case).

3.1 Relation of Standards

To show the relation of application sector standards we of course need to show the relation to the top level first - the top level is not 61508 (which is limited to functional safety), rather it is overall safety in the context of social, economic and regulatory constraints.

The three main influences on any overall safety concept will be social, economic and regulatory issues. I guess regulatory and economic issues are quite self explanatory - the issues of social influence is at the core of risk-assessment, fundamentally tolerable risk is the guiding term that is deeply routed in the societies acceptance of risk, thus changes - and we have seen fundamental changes in the past 10 years - in the acceptance of risk in society will influence the directions of safety standards and the interpretation.

61508 it self is covering one part of overall safety "functional safety". As a procedural safety standard it starts out with rules on documentation and management of functional safety which are the foundation of a procedural safety approach. The actual core of 61508 then starts in section 7 of 61508-1 that outlines the safety life-cycle. Following the basic pattern of the initial part of the safety life-cycle:

- Develop safety requirements (Clause 7.1-7.5)
- Allocate safety functions to systems (Clause 7.6)
- implement systems (Clause 7.10)

As 61508-1 is not concerned with the specifics of safe implementations but rather with the global strategy of safety life-cycle, clause 7.10 simple refers to 61508-2 (HW/System) and 61508-3 (Software). The other parts of 61508-1 provide guidance and definitions as well as a description of accepted methods. From this very generic development cycle (note that we only followed up to the actual development, the safety life-cycle of course continues on parallel to development with assessment activities and post-development with

commissioning maintenance and disposal. Those issues are de-facto unchanged for COTS/OSS in safety critical systems, thus we will not discuss them too much here. As 61508 is a generic approach it is relatively strict in its approach, and for different classes of systems further constraints can be added allowing to simplify methods and requirements - this simplified, or adjusted versions are the application sector standards for Machinery, Rail, Processing etc.

The relation ship between the application sector standards we are trying to establish here is not a hard-relation in the sense that standard X is "only" for a particular case, but what we are pointing out is the main focus of individual standards and there suitability for a more abstract property of the device to be certified. The intention of these tables is to point out the focus of specific application sector standards to aid in locating helpful concepts - it is not to claim that a specific standards is i.e. only concerned with reactive systems or composite safety - read it as "carrying a focus of".

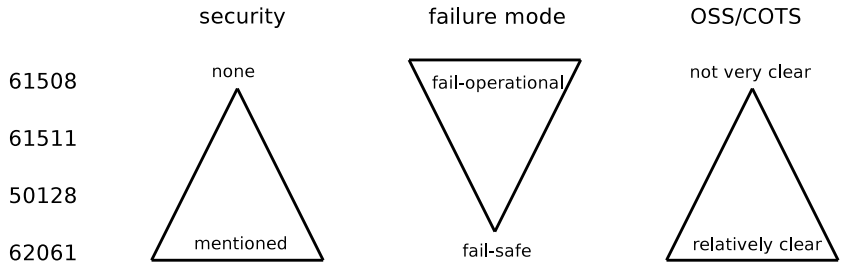

FIGURE 2: *relatsions of standards*

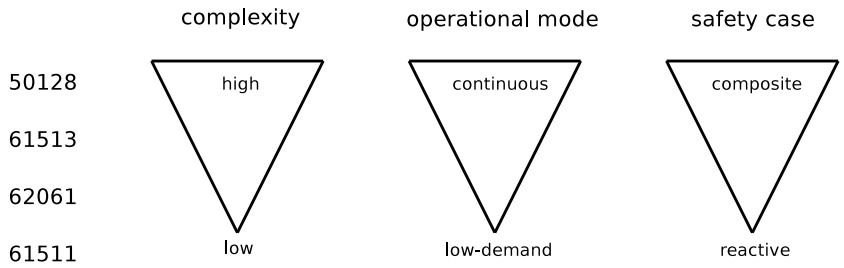

FIGURE 3: *relatsions of standards*

Not too surprising security is ignored in the older standards and addressed in the newer ones (62061 was published in 2005) likewise this coincides with the coverage (and acceptance) of COTS. It also should be noted that the definition of COTS (pre-existing software, prior use, etc.) has evolved over time. While the early definitions clearly are considering products, later definitions (i.e. "embedded software" [62061]) is not so much concerned with the origin of the components rather focuses on the qualities of components.

Note that though some application sector standards simply don't define low-demand mode - de facto almost no safety critical application of even only moderate complexity will be able to provide continuous mode only - thus a certain amount of cross-selection regarding justification strategies is more or less inadvertable any way.

"The life cycle model of EN 50126-1 does not take into account the iterative process necessary to make it applicable to reality [EN 50126-2 9.5]. We assume that this statement though pertaining to 50126-1 (50128) can be applied to all of the 61508 derived standards.

This comparison is incomplete - not too surprising - but what we hope to point out with this glimpse at specific aspects is that there are possibilities to get more-to-the-point information for a specific component if one considers related standards. Again this is not suggesting that a machine tool can simply use the standard from a nuclear power plant - but if the complexity of the problem fits a related standard well then the approach and especially the guidance offered for the "application sector constraint implementation of 61508" may provided vital help on how to approach the safety life-cycle details.

4 Arguing OSS GNU/Linux

If these elements are applied to GNU/Linux now then one can see that different standards show a quite different suitability in arguing OSS in there context. While 50128 explicitly addresses COTS, 62061 explicitly discusses "embedded software", 61511 referees to "prior use" though with a focus on hardware (references to 61508-1 and -2 NOT -3). None of the standards directly addresses OSS (obviously) but they do address categories of software that fit certain aspects of OSS and GNU/Linux in particular. in the following list we provide our view of this association:

```
61508 - preexisting software, standard
        software, proven-in-use
61511 - prior use,
        selection based on prior use
50128 - COTS, proven-in-use
62061 - embedded software , proven-in-use
```

The Linux kernel most obviously would be described as "preexisting software" and "selected based on prior use" - it should be noted though that these terms are most of the time not precisely defined in the standards (62061 does define embedded software though) - making the actual interpretation of statements non-trivial. Fundamentally this is a question of providing a convincing argument and concise justification - it will hardly be possible to prove strict adherence to an accepted procedure for GNU/Linux.

4.1 Possible approaches

Fundamentally we see two posible approaches to utilizing GNU/Linux and OSS capabilities.

- GNU/Linux mainstream:
  i.e. Unmodified GNU/Linux "as-is" justified by evidence and argued by advances of community monitoring and bug tracking - applications constraint to well established standard compliant subsets (i.e. POSIX threads)

- Linux virtualization technologies:
  i.e. Paravirtualized GNU/Linux on top of diverse RTOS core systems based heavaly on diversity of the RTOS/HW layer and supported by community peer review capabilities.

Of course there are variations of these two options - we will list some scenarios and detail only one here du to space constraints.

This section is rough and obviously incomplete - its intention is to give a rough idea of what directions are possible - and show that there are a number of possible ways to approach the problem.

One obviously available path is to follow 61508-6 Appendix E, and focus on diversity of the OS layer - thus in theory eliminating specification and design related common cause failures - this mapped to Linux could be based on a system outlined below:

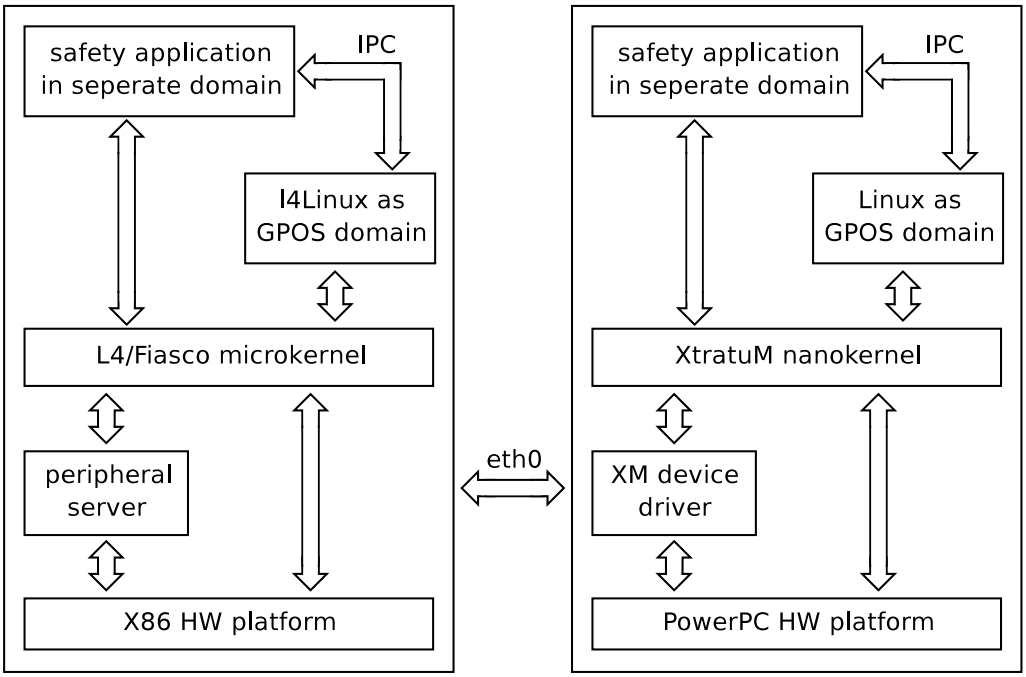

FIGURE 4: *OSS based maximum-diversity*

Resulting in "maximum-diversity":

```
X86 / PowerPC                    - hardware diversity
GRUB / GNU/Linux boot-system     - initialization diversity
L4 Fiasco / XtratuM nanokernel   - resource manager, runtime diversity
L4Linux / GNU/Linux intercepted - GPOS diversity with respect to IPC
L4-domain / RTLinux/GPL          - divers safety domains
```

While this is a fairly complex approach it is quite straight forward to map it into 61508s requirements and provides the full benefit of GNU/Linux as GPOS for maintenance and non-safety related tasks (monitoring, upload, etc.). This is a rough proposal for a very conservative view of Linux in 61508 context.

4.2 further options

Space does not allow to detail all others - but we would like to list some options that we see as possibilities:

- Build a 61508 compliant COTS argument, which would be heavily based on proven-in-use (refer to clause 3.4 - definition of COTS, which clearly points towards Proven-in-Use), and section 9.4.5 sub-clause ii and iii for requirements.
- Build a evidence based safety case which is clearly non-61508 compliant and argue the divergence from 61508 which is entirely procedure based. Fundamentally 61508 allows divergence at almost all places provided justification is given.
- Build a 61508 compliant procedural safety case for a nano-kernel that runs GNU/Linux as one of its (user-space) tasks running safety critical apps (or at least the safety responsible components) under direct control of the nano-kernel. GNU/Linux is then ideally only a SIL0 component in the overall system with no safety responsibility.
- put the safety responsibility completely into the application and argue the OS as a gray-channel based on diversity of the safety critical application (diverse OS usage, diverse languages, N-version programming). Within GNU/Linux it is also possible to select two functionally equivalent OSS components that are implemented independently (i.e. arguing diversity between apache2 and boa http servers should be doable).
- Document the Linux development life cycle (the kernel that is) in a suitable way and argue that it provides comparable if not superior quality even though it does not follow the procedural requirements. In fact the stability of Linux-2.6 can in our opinion be argued in this way - the main issue really is if this is acceptable to the safeties.

One essential point of all of these options though is that naive proven-in-use as "there is so much Linux in use" will not due, uptimes of even years of some Linux 2.2.X and 2.0.X systems are nice - but not a usable source of evidence for arguing a 2.6.22 kernel on Debian 4.0 ! We believe field experience will help, but basing a safety case on field data only - especially with none from safety-related systems - will be futile.

Detailing these options and investigating the limitations and certification strategies will be one of the goals of the Safety Critical Linux Working Group of OSADL.

4.3 Note on safety case

While this paper is not the place to detail a safety case for the Linux kernel one should consider a layered safety case:

- Generic Product Safety Case
- Generic Application Safety Case
- Specific Application Safety Case

safety case structures i.e. in 50126/50129

Building a monolithic safety case for a specific version of the Linux kernel with a specified configuration would be more or less unmaintainable and an effort that would be lost at the first upgrade. The specifics of OSS "release early - release often" mandate a somewhat different approach to the safety case than would be suitable with "bundled" commercial software.

Taking the safety case layering from above we would see this as

- provide a constraint OS definition - i.e. "pure-POSIX" and a set of kernel functions satisfying these based on a well defined standard (i.e. open-group specification).
- from this generic POSIX layer introduce further constraints (minimum POSIX real-time profile -

PSE 51) and map this to a particular implementation of the Linux kernel i.e. Linux with real-time preemption extension.

- finally justify a specific configuration selection (kernel config) in the context of the "generic application safety case Linux-RT PSE 51" as a basis for a well defined safety application running on top of this kernel.

Justification of these safety cases - as noted above - will hardly rest on field history only - not only does 61508 not clearly define what field history data would need to look like, the concept of field data, at least for the specific application safety case, but it simply will not be arguable due to lack of data fitting the specific configuration.

Rather justification will need to build on black-box testing and analytical methods outlined in 61508-7 and referenced in 61508-3 (i.e. table A3 software development and table A9 and A10) Note that especially here the application sector standards have a lot to offer on guidance of methods and in fact on additional methods considered appropriate.

5 Evolution of standards

A tendency towards evidence based approaches can be seen. Standards like MOD 00-55 (procedural) were replaced with evidence based counterparts MOD 00-56. Even within 61508s derived standards, evolution of domain specific standards can be seen. 62061 is almost modular [i.e. 62061 Clause 6.7.5] compared to 61508 - allowing subsystem-elements to be integrated as components (either developed or COTS). 61508 of course stays a system level safety case - but while 61508 is directly concerned with design and specification, 62061 (released in 2005) is more of a safety strategy. We believe this development will continue and of course it is up to industry to promote development in the direction best suited for its needs - if development continues in the direction visible now we expect OSS and COTS components to be much easier to integrate in the future than they are now.

Our expectation is that standards will evolve in the next decade in the direction of higher level of acceptance of evidence. This is not only due to the fact that increasing system complexity (both software and hardware) in safety-related systems impact the applicability of a strict procedural approach more and more, but also to the fact that there is growing evidence and theoretical works that indicate that COTS/OSS development may well be as good if not better than bespoke software development.

Of course it is up to industry to move the standards in a direction suitable for the use of COTS/OSS in safety critical systems. This is not suggesting that standards should be less rigorous in any way - quite the contrary - they need to be much more precise in defining COTS especially with respect to software, and provide better guidance on the use of COTS especially with respect to the types and quality of evidence as well as the use of risk assessment and validation methods (i.e. FMEA, HAZOP) in relation to COTS products.

Again this is one of the issues the Safety Critical Linux Working Group of OSADL will be focusing on. Never the less it is clearly up to industry to recognize the potential in COTS/OSS and especially in GNU/Linux and support efforts in standardization bodies to tweak standards to supply the means needed for its use.

6 Conclusion

Even though the material outlined here is far too general to make any claims that OSS and specifically Linux is usable in 61508 context, we do think that there is sufficient evidence that it is not excluded from 61508 compliant systems and that if the effort to achieve acceptance of Linux and other OSS components in safety-related systems is coordinated at a suitable level, that it well could constitute a sound basis for building safety-related systems in the future. There is plenty of work to be done and there are efforts under way to make it happen, both the advances in standardization and the formation of organizations like OSADL are encouraging indicators that there is not only a wide need for OSS in safety-related systems but that there is a certain acceptance in industry.

7 List of Acronyms

ATC - Air Trafic Control
CAP - CAA Publications
COTS - Comercial Off The Shelf
FDL - Free Documentation License
GDB - GNU DeBugger
GNU - GNU Not UNIX (recursive accronym)
GPL - General Public License
HSE - Health and Safety Executive
IPC - Inter Process Communication
KFI - Kernel Function Instrumentation
KGDB - Kernel GDB
LTT - Linux Trace Toolkit
OSADL - Open Source Automation Development Lab
POSIX - Portable Operating System Interface (for UNIX)
OSS - OpenSource Software
RT - Real Time
LTP - Linux Test Project
SIL - Safety Integrity Level

References

[1] Linux Test Project, Kenrel Code Coverage, *http://ltp.sourceforge.net/documentation/how-to/UsingCodeCoverage.pdf*

[2] git (variable - undeined acronym that sounds good) and cogito *http://www.kernel.org/pub/software/scm/git/*

[3] IEC 9945, *Single Unix Specification*,`http://www.unix.org/`, 2003

[4] DONALD WAYNE CARR, RUBA.N RUELAS, *COTS and Free Software Components for Safety Critical Systems in Developing Countries*, Universidad de Guadalajara, Guadalajara, Mexico RAA.L AQUINO SANTOS, APOLINAR GONZALEZ POTES Universidad de Colima, Colima, MACxico.

[5] Fan Ye,*Justifying the use of COTS Components within safety critical applications*, Thesis, University of York, 2005

[6] Weining Gu, Zbigniew Kalbarcyk, Ravishankar K. Iyer, Zhenyu Yang,*Characterization of Linux Kernel Behavior under Errors*, University of illinois at Urbana-champaign, 2002 (?)

[7] J-C Fabre, F. Salls, M. Rodriguez-Moreno, J. Arlat, *Assessment of COTS microkernels by Fault Injection*, LAAS-CNRS, Toulouse, 1998 (?)

[8] C. Jones, R.E. Bloomfield, P.K.D. Froome, P.G. Bishop, *Methods for assessing the safety integrity of safet-related software ofuncertain pedigree (SOUP*, HSE 337/2001

[9] R.E. Bloomfield, P.K.D. Froome, P.G. Bishop,*Justifying the use of software of uncertain pedigree (SOUP) in safety-related applications*, HSE 336/2001