

## Assessing Operating Systems for Use in Safety Related Systems

R. H. Pierce, MSc.; CSE International Ltd; Flixborough, UK  
M. Nicholson, PhD.; Department of Computer Science, University of York, UK  
A. G. Faulkner, MSc.; CSE International Ltd; Flixborough, UK

Keywords: Operating systems, integrity, reliability, safety

### Abstract

This paper presents a general model for the assessment of the suitability of COTS and other pre-existing operating systems for use in safety related software systems. A taxonomy of operating system features is presented which is used both for the description of an operating system's behavior and for the presentation of its safety integrity, so that the application designer may have objective information on which to base design decisions. The contents of an assessment report created in accordance with the model are also described.

### Introduction

Developers of software for safety related systems have considerable incentive to use pre-existing operating systems, either commercial off the shelf (COTS) products or freeware. There are inevitably concerns among users and regulators about the safety integrity of such pre-existing systems, few of which were designed with safety applications in mind. This concern is also felt about other COTS software components, and has been the topic of many conference papers and other publications, notably the research reports produced by Bloomfield (references 1, 2). Operating systems however pose unique problems. The reason for this is that, in general, it is not possible to write application software which can defend itself against certain kinds of operating system failures. This is fairly obvious, because the operating system forms a complete layer between the application software and the hardware and consequently failure of the operating system is equivalent in most cases to hardware failure from the perspective of the application.

An initiative has been launched by three UK authorities, the Health and Safety Executive, the Civil Aviation Authority (CAA) Safety Regulation Group and the Defence Procurement Agency to assess the safety integrity of pre-existing operating systems which might be used in safety related applications. Two preliminary studies have already been published as a result of this work, one on Linux (reference 3) and one on Microsoft Windows XP™ (reference 4).

CSE has recently developed, under CAA Contract 730, a common model for operating system assessments. This paper discusses the operating system assessment model and indicates what should appear in an assessment report. Eventually, it is hoped, one or more contracts will be awarded to carry out operating system assessments, with the results being published for the common good of the safety related systems community.

Operating system assessments conducted under this model are "context free". In other words, the assessment model makes no assumption about the application which will use the operating system, so the resulting reports will contain information of interest to any potential user, safety assessor or regulator. The model is also suitable for any kind of operating system ranging from a complex general purpose operating system to a simple run-time executive or tasking kernel.

### What needs to be assessed?

Obviously, the safety integrity of an operating system is an important matter, but it is not the only issue to be considered where the suitability of an operating system for a safety related application is being assessed. In the Linux study referred to above, three criteria were set out to decide on the suitability of an operating system (or indeed any pre-existing component). A somewhat simplified statement of these criteria is as follows:

- C1 The operating system must be sufficiently well defined.
- C2 The operating system must be suitable for the characteristics of the application.
- C3 The operating system must be of sufficient integrity to allow the system safety integrity requirements to be met.

Considering first C1, it is important for the designers of the safety related application to have a full knowledge of the intended behaviour of the operating system so that hazards do not arise because of mismatches between the belief of the application designers and the true of the operating system. Such mismatches are most likely to be experienced in unusual circumstances and testing alone cannot in general show their absence. The possibility also exists of unexpected behaviour from the operating system from either undeclared functionality authorized by the supplier, or inserted in an unauthorized manner by operating system development staff (“Easter Eggs”, time bombs or other malicious code).

It is also important to have sufficient information to allow an adequate hazard analysis of the interaction between the operating system, the application software, and the hardware (especially peripheral devices). It is therefore important for all aspects of operating system behavior to be thoroughly specified and understood. The topic of operating system failure behavior is discussed in more detail later.

It is also clear that C2 is a necessary criterion, since no matter how well specified an operating system or how reliable it is, if it does not provide the correct facilities to support the software design chosen for the safety related application, it will not be suitable for use. This can be most clearly seen in the timing domain: if the application has hard real time requirements and the operating system cannot support hard real time deadlines, it cannot be used with confidence.

Finally, criterion C3 is fairly self evident, but it must be noted that what is “sufficient” will depend on the complete system design, including any system level measures which can mitigate operating system failures and thus allow the operating system to have a lower safety integrity requirement than would be the case without system mitigation measures.

An operating system assessment, if it is to be useful, should provide sufficient information to allow a reasonable judgement to be made of whether the combination of the specific application software, the operating system and the system design features meets the three criteria for a given safety related application.

### Operating System Reliability and Robustness

The reliability of a component is defined as the probability that it will provide a defined service for a given period of time under a given set of operating conditions.

Operating systems typically provide many services to application programs (see the taxonomy of operating system services described below), and each service is potentially susceptible to a number of failure modes. In strict terms, therefore, the reliability of each service/failure mode combination should be considered.

Common experience of widely used operating systems indicates that the prevalent operating system failure mode is simultaneous complete loss of all services (colloquially known as a “crash” or “blue screen of death”), generally caused by a design error in the operating system leading to a unrecoverable exception or a corruption of vital operating system data structures. This is the failure mode which the average engineer will think about when discussing the reliability of an operating system, and may be the only significant failure mode where relatively low integrity systems are concerned.

It is not however impossible for more subtle failure modes to occur without a crash. For example, a specific device driver may lose outgoing or incoming messages without affecting the continued liveness of the operating system as a whole. Users should be aware of the need to analyse the effects of specific operating system failure modes as well as the “crash” failure mode (reference 6). An operating system assessment should therefore, where practicable, provide an assessment of the safety integrity of particular functions, services or groups of services, and not just of the operating system as a whole.

Furthermore, an assessment should (where feasible) consider the internal structure of the operating system. Information should be provided on the failure characteristics of the overall operating system in the event of application errors, internal errors and hardware errors. This kind of analysis is particularly important for systems of higher integrity.

The following aspects of operating system robustness (failure characteristics of the operating system) are relevant to an operating system assessment.

- Response of the operating system to internal application errors detected by hardware (for example, arithmetic overflow) in an application program, and in particular its effect on other executing programs.
- Response of operating system to application program memory addresses outside partition/process limits (where this is possible).
- Illegal invocation of the operating system itself (invalid system call, or incorrect parameters to a system call).
- Detection mechanisms for failures of connected items of hardware (other than processor or main memory), and actions taken on external device failure, including notification of applications or system reconfiguration.
- Response to internal operating system exceptions or inconsistent internal states (an inconsistent kernel state would generally result in an operating system crash, however specific operating systems may be resilient to failures in device drivers, communications protocol packages or file management systems and may have recovery mechanisms to allow at least some services to continue).

- Action (if any) on processor or main memory failure. Note that in a uni-processor system, this will normally mean complete loss of service from the operating system.

An important part of the assessment will be to provide this information so that the designer can take suitable design measures to mitigate operating system failures. In reference 5, Pygott describes some such design measures relevant to COTS operating systems.

### A Taxonomy of Operating System Facilities

Operating systems vary widely in the details of their functionality, but they are all basically concerned with the management of computer resources on behalf of application software. A taxonomy of operating system services has been developed to support the assessment process. There are a number of reasons for creating such a taxonomy.

Firstly, it allows the assessor (and the users of the resulting assessment reports) to distinguish the essential “operating system-like” features of a software product distribution from facilities which may be provided in connection with the operating system but are not logically part of it. Obvious example of such facilities are the text processing utilities provided with UNIX systems (such as *awk*, *grep*, *vi* and so on). Although many users will think of these as essential parts of UNIX, they are in fact merely utility programs which happen to be hosted on the Application Programming Interface (API) features of UNIX operating systems and Linux.

Secondly, the taxonomy provides a common structure for the presentation of the operating system assessment results, and thus facilitates the comparison of operating systems which may be candidates for use in a particular safety related application.

Finally, the taxonomy provides a systematic means of considering operating system failures and their effects on application programs. Thus it allows different safety integrity levels to be assigned to different operating system features (where it is valid to do this).

The taxonomy considers the facilities of any operating system at three levels: six very high level FUNCTIONS, a number of supporting SERVICES, and at the lowest level the individual application programming interface routines (which generally map onto system calls) or other detailed means of invoking the operating system. There is a many-to-many mapping between functions and services, as is shown in Figure 1. This model is based upon work on the analysis of operating system failures by Conmy and McDermid, described in reference 6, but has been extended to cater better for general purpose operating systems since the original work was specifically concerned with operating systems for Integrated Modular Avionics.

The six functions are as follows:

**Provision of secure and timely data flow.** This function is concerned with reliable and timely data flow to and from applications and input/output devices

**Controlled access to processing facilities.** The access of applications to the underlying hardware processing resources must be managed so that, for example, any timing deadlines can be met.

**Provision of secure data storage and memory management.** The aim of this function is to secure memory storage (dynamic or persistent) from corruption or interference by other applications or the actions the operating system takes on their behalf.

**Provision of consistent execution state.** This concerns the consistency of executable programs and data and is mostly concerned with the state of the system after initialisation or reconfiguration.

**Provision of health monitoring and failure management.** This function covers the detection and response to partial and controlled failures of the system (operating system, application, hardware).

**General provision of computing resources.** This function covers the provision of any of the services of the operating system. A failure of this function would imply an uncontrolled failure of the operating system (in other words a “crash”).

The services level categories are as follows.

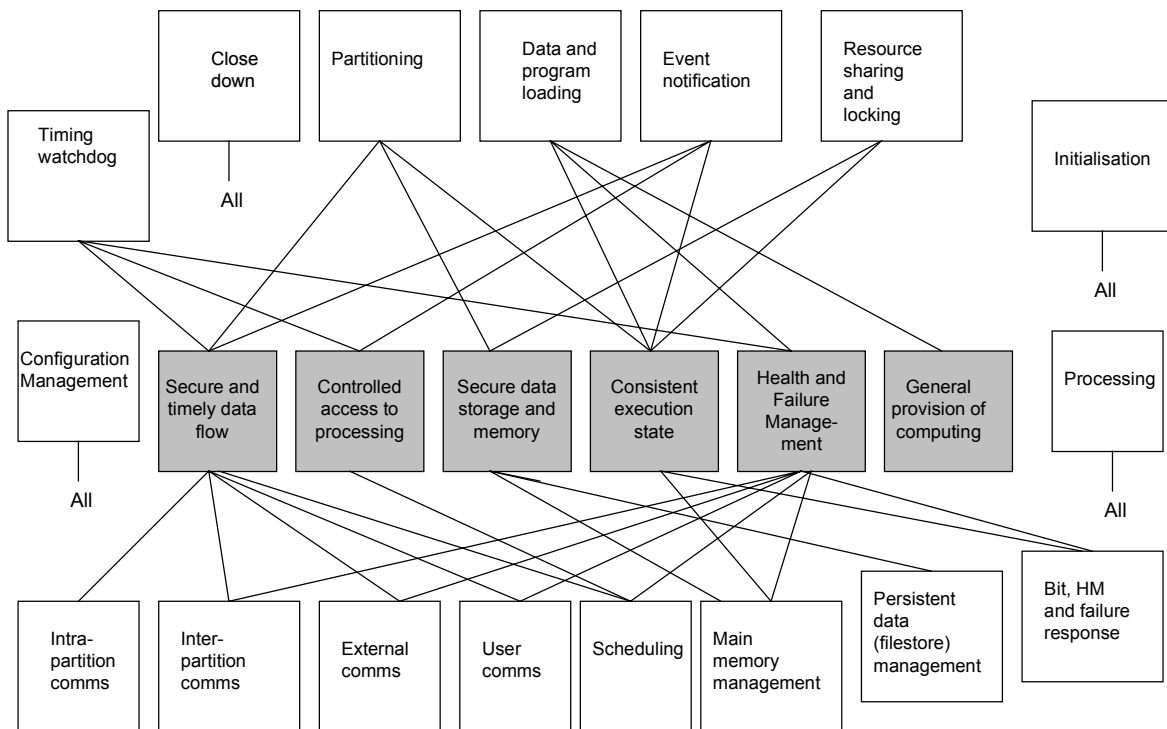
- Partitioning: facilities for allowing an application to operate independently and without unpredictable interference from others (frequently provided by some kind of process concept).
- Main memory management: facilities for allocating main memory to application programs or partitions, either statically or dynamically.
- Data and program loading: the means (whether static or dynamic) of loading application programs and configuration data for execution.
- Intra-partition communication: facilities for communicating between threads of control within a given partition.
- Inter-partition communication: facilities for communicating between different partitions.
- External communication: input/output (excluding filestore management but including any low level mass storage device access) and external communication or network facilities.
- Event notification: the mechanism for notifying application programs of events such as changes in state of the computer system or of software objects (for example, notification to one process that another process has failed).
- Scheduling: the policies and algorithms for dividing processor time between different partitions, processes or threads, and for the provision of timing services in general.
- Timing watchdog: specific facilities which allow the operating system to monitor whether real time deadlines are met by partitions or threads.
- Filestore management: the means for the organisation of persistent data files on storage devices attached to the computer, including user access protection mechanisms.
- User communication: specific features built into, or closely associated with, the operating system for the provision of textual or graphical user interfaces.
- Resource sharing and locking: features of the operating system which allow resources to be shared between applications or locked for exclusive use.
- Built in test, health monitoring and failure response: facilities for testing and reporting actual or incipient device failures.
- Initialisation: the means of creating and starting up objects provided by the operating system including partitions, threads, files, synchronisation and communication objects.

- Close-down: the means of closing down objects created by the operating system such as processes, threads, or files, in both normal operation and following failures.
- Configuration management: the means for creating the required arrangement of processes and operating system support on the hardware architecture. Also describes the facilities for the provision of information such as the available communication ports, I/O devices attached and so on. Any support provided for dynamic system reconfiguration would be placed in this service category.

The services categories are defined at a fairly abstract but recognisable level, to avoid any bias towards any particular operating system. Particular facilities of any given operating system may support more than one of these services, for example inter-partition and scheduling facilities are usually intimately connected.

Of course, not all operating systems will necessarily offer all the services listed above, or support them fully. For example, timing watchdogs and health monitoring facilities are not typically well supported by general purpose operating systems, while real time operating systems may have relatively limited facilities for user communication.

Figure 1 shows the relationship between the operating system functions and services.



**Note: relationships should only be shown where a relevant service is provided by the operating system**

Figure 1 – Operating Systems Functions and Services

## Application of SW01 Concepts to Operating System Assessment

The UK Civil Aviation Authority (CAA) Safety Regulation Group has published safety regulatory objectives for software used in safety related air traffic services equipment, as part of its general regulatory requirements CAP 670 (reference 7). The software objectives appear in a document generally known by its CAP 670 section reference number SW01. Since the CAA has sponsored the assessment model described in this paper, it was a requirement that SW01 should be used as the framework for the assessment of software integrity.

SW01 is fairly distinctive in its approach to software safety. Most widely used standards such as DO178B (reference 8) and IEC 61508 (reference 9) are “process based” in that they describe or recommend the sequence of activities which a developer should undertake in developing a safety related software system. By contrast, SW01 is “evidence-based” in that it does not prescribe any development process, but simply asks for credible evidence that the software system will be safe.

SW01 sets out five fundamental safety objectives which a safety related software system must fulfil. These are, in a slightly simplified statement:

- The software safety requirements must be valid in the system context. A pre-existing operating system is unlikely to have any specific safety requirements defined, so this objective is replaced by a requirement that the behavior of the operating system must be precisely defined.
- The safety requirements must be fully satisfied by the implementation. For an operating system, this is interpreted to mean that the operating system must meet its specification.
- The safety requirements must be traceable to the design and implementation. Traceability cannot be demanded in a software product but if it is present it will strengthen the evidence for software integrity.
- The evidence for safety must be consistent with the version of the software which is to enter service, the user documentation and other software documentation. This requirement can be interpreted to mean that the released configuration of the operating system must be internally consistent, consistent with its specification and user information such as release notes, and consistent with the evidence obtained by the operating system assessment.
- Non-safety related functions must not compromise the integrity of the safety related functions. This requirement does not apply in general for a pre-existing operating system product.

SW01 also defines the concept of the Assurance Evidence Level (AEL) to express the level of confidence that a software component will meet each requirement, on the basis of the strength and depth of the available evidence. An AEL is defined for each safety function (just as a SIL is assigned to an individual safety function in IEC 61508). An operating system assessment is expected to assess the AEL achieved by the operating system services. AELs are defined on a range from 1 (least confidence) to 5 (greatest confidence). SW01 provides guidance on the evidence requirements for each AEL.

SW01 contains a strong requirement that the behavior of a software component be defined in terms of a specific set of attributes, namely function, timing and performance, capacity, overload

tolerance, robustness, reliability and accuracy. These concepts are equally applicable to an operating system and are specifically addressed in the operating system assessment model. As noted above, SW01 uses the concept of evidence based safety assessment. Evidence that a software component meets its specification can be derived from testing, analysis or field service experience. All of these will be valid forms of evidence that an operating system behaves according to its specification.

### Outputs of an Operating System Assessment

It is envisaged that three reports will be produced from each operating system assessment:

- A precise specification of the behaviour of the operating system.
- A summary report describing the facilities provided by the operating system, and assigning AELs to each service provided by the operating system.
- A detailed report which presents the evidence, as required by SW01, for the conclusions reached in the summary report.

The Summary Assessment Report: Basic factual information will be provided under the following headings. Although much of this information will be provided in great detail in the operating system specification, a simplified version will be provided in the summary report for the convenience of the reader.

- Basic operating system identification and configuration information, including a precise definition of the hardware configurations to which the assessment applies.
- Application program interface principles (how operating system services are invoked)
- A summary of the operating system's facilities under the services taxonomy headings.
- Resource usage and capacity information.
- Robustness and overload tolerance (how the operating system responds to failures in applications, hardware or its own components).
- Undocumented features (for which no specification could be created)
- Operating system configuration facilities.

Integrity information will be provided under the following headings:

- Assessed AEL. An AEL will be assigned to individual operating system services in all cases where it is valid to do so (for example, kernel services such as process creation and scheduling may have a higher AEL than filestore management). Where it is not possible to assign different AELs to different services, an overall AEL will be assigned to the operating system.
- Known errors: any known errors will either be listed or a source of such information identified.
- Assessed failure rates. Any information on known failure rates (for example, established from field service experience) will be included in this section of the report.
- Notable uses in existing safety related systems.

The specification report: It may seem surprising that a pre-existing operating system needs any specification other than that provided by the operating system developer. In practice, however, many pre-existing operating systems are not sufficiently well defined, especially in areas such as

robustness and failure behaviour, to meet the requirements of SW01 (it is not possible to assign an AEL to a function or service unless its behaviour is precisely defined for all attributes). If, of course, the assessor finds that there is a rigorous enough specification already in existence, this will save the creation of a new specification document.

The specification should be defined in rigorous terms, using whatever specification techniques the assessor deems relevant. Mathematically formal notations are not expected, unless warranted by particular circumstances, but semi-formal notations such as entity-relationship-attribute data models, state transition diagrams and message sequence charts are encouraged. API routines should be defined in terms of function (signature, precondition and postcondition), and wherever practicable the other SW01 attributes such as timing properties should be specified.

The detailed assessment report: The contents of a detailed assessment report will depend very much on the circumstances which apply to an individual assessment. In general, three assessment strategies can be recognized:

- Examination of the supplier's development process and assessment of the available evidence under the SW01 headings (including field service experience available via the developer).
- Independent "white box" assessment without recourse to the supplier's internal development process evidence, but using the operating system source code and any available design information and field service evidence not derived directly from the supplier.
- Independent "black box" assessment without recourse to the supplier's internal development process evidence, without access to source code or design documentation, but possibly with field service evidence not derived directly from the supplier.

For Linux, it has already been established in the HSE report (reference 3) that the second of these strategies is appropriate since this operating system was developed in an ad-hoc manner but the source code and design documentation are available.

Use of the first strategy will clearly depend on the developer's willingness to co-operate with the assessment. If the supplier is uncooperative (or has gone out of business), or there is no single developer, only the second or third strategies would apply. If source code is available, a full assessment can be carried out, but if source code were not available, only "black box" testing would be possible (supplemented by field service experience where possible). In some special cases, the supplier may have already evidence of integrity, for example as part of a certification package to DO-178B, in which case the integrity assessment would be straightforward.

In all cases, the detailed assessment report will describe the method of assessment and provide justification for the conclusions reached.

### Conclusions

Operating system assessments carried out as outlined in this paper should give a much improved basis for deciding on the suitability of assessed operating systems for use in any safety related application.

### Acknowledgments

The authors wish to thank the UK CAA for commissioning the work described in this paper, and also to express their appreciation of the helpful technical and editorial input provided by Andrew Eaton and John Penny of the CAA, and by Mary Johnston of CSE.

## References

1. Bloomfield, R. E *et al*, Justifying the use of software of uncertain pedigree (SOUP) in safety related applications, HSE Contract Research Report 336/2001. London : HSE Books, 2001.
2. Bloomfield, R. E *et al*, Methods for assessing the safety integrity of safety related software of uncertain pedigree (SOUP), HSE Contract Research Report 336/2001. London : HSE Books, 2001.
3. Pierce, R. H., Preliminary Assessment of Linux for Safety Related Systems, HSE contract research report RR011/2002. London : 2002.
4. Brennan, V. Scoping study for the Operating System Integrity Evaluation of Windows, Praxis Critical Systems report S.P1170.20.6, Issue 1.1, April 2002.
5. Pygott, C. H., Assurance of safety-related applications on a COTS platform, in Proc. 11th Safety-Critical Systems Symposium, 4-6 February 2003: London: Springer, p201.
6. Conmy, P. and J. McDermid (2001). High Level Failure Analysis for Integrated Modular Avionics. 6th Australian Workshop on Industrial Experience with Safety Critical Systems and Software, Brisbane, Australia.
7. UK Civil Aviation Authority, Safety Regulation Group, Air Traffic Services Safety Requirement, Document CAP 670 section SW01 "Regulatory Objective for Software in Safety Related Air Traffic Services", Issue 6. London : October 2002.
8. ISO/IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems, International Electrotechnical Commission, 2000.
9. RTCA Inc, Software Considerations in Airborne Systems and Equipment Certification, DO 178B/ED-12B, 1992.

## Biography

R. H. Pierce, MSc., Consulting Engineer, CSE International Ltd, Glanford House, Flixborough, Scunthorpe DN15 8SN, UK. Telephone - +44 1724 862169, facsimile - +44 1724 856256, e-mail - rhp@cse-euro.com.

Mr. Pierce has extensive experience in software engineering topics (compilers, program analysis tools and software engineering methods). He has over 10 years experience in software and system safety assessment for industry domains including air traffic management and railway control and signaling systems.

M. Nicholson, PhD, Department of Computer Science The University of York, York YO10 3DD, UK. Telephone: +44 1904 430000, e-mail Mark.Nicholson@cs.york.ac.uk.

Dr. Nicholson is a teaching and research fellow in the High Integrity Systems Engineering group at York. His PhD concerned process allocation in real-time distributed systems, and he has over 7 years of experience in safety assessment of real-time systems. His current research interests include IMA certification processes and the evidence required for use of off-the-shelf operating systems in safety critical applications. He is employed on the MATISSE project (grant GR/R70590/01).

A. G. Faulkner, MSc., Consulting Engineer, CSE International Ltd, Glanford House, Flixborough, Scunthorpe DN15 8SN, UK. Telephone - +44 1724 862169, facsimile - +44 1724 856256, e-mail - agf@cse-euro.com.

Mr. Faulkner has a background in software development mainly concerned with computer based command and control systems. His current research interest is in the integrity of large scale data configured systems.