Nationaal Lucht- en Ruimtevaartlaboratorium

National Aerospace Laboratory NLR

NLR-TP-2000-010

Software partitioning for safety-critical airborne systems in practice

M.F.R. Keuning

Nationaal Lucht- en Ruimtevaartlaboratorium

National Aerospace Laboratory NLR

NLR-TP-2000-010

Software partitioning for safety-critical airborne systems in practice

M.F.R. Keuning

This report is based on a presentation held at the Twenty Fifth European Rotorcraft Forum, Rome, 14 - 16 September 1999.

Division: Information and Communication Technology
Issued: January 2000
Classification of title: Unclassified

Summary

Increasingly demanding helicopter operations lead to a higher workload on the pilot. To support these helicopter operations and relief the workload on the pilot more sophisticated avionics are incorporated. Many of the avionics' functional requirements are realised in software. This has induced a strong growth of the size and complexity of the used software.

Often this software contains safety critical components as well as less critical components. Because of economic pressure, techniques are needed to reduce the development effort of the less critical software components in these systems without compromising the functioning of the critical components and thus the airworthiness. One solution - as proposed by DO-178B: *Software Considerations in Airborne Systems and Equipment Certification* - is partitioning. A partitioning architecture is discussed which is successfully implemented and certified as a part of helicopter avionics. DO-178B categorises software components into five criticality levels. The realised avionics contains software components of three different levels including the most critical level (i.e. level A).

The developed novel partitioning architecture is based on two hardware capabilities, a Memory Management Unit (MMU) and a processor with two protection levels. The hardware of the developed system contains a compliant processor with built-in MMU that supports both segmentation as well as paging as means of memory protection. Only paging is used because with the selected processor segmentation induces up to 50% performance overhead whereas the overhead of paging stays well below 5%.

Central to the partitioning architecture is a proprietary kernel that runs in a protected environment. The proprietary kernel manages all protections within the application. Like most operating systems, protection is provided between tasks. Furthermore protection is provided between components of different software levels (i.e. intra task and inter task) which is the most important protection support concerning safety.

A formal model of avionics partitioning defines three essential requirements for the kernel's access mediation functions, i.e. complete, tamper-proof and assured. All these essential attributes are met by the developed partitioning architecture. The developed partitioning architecture can be ported to other hardware and extended, with for example: support for Integrated Modular Avionics (IMA) and support for distributed and multi-processor systems.

The implemented partitioning architecture was successfully certified as a part of the helicopter avionics. Evaluation of the partioning architecture implementation indicated a very favorable benefit to a reasonable cost. The measured processor load overhead induced by partitioning is only about 1%. Memory usage increased by approximately 5%. These resource requirements are very acceptable considering the economic advantages gained. Consequently the developed partitioning architecture is very useful and can be reused in future applications.

Abbreviations

ACR	Avionics Computer Resource
IFR	Instrument Flight Rules
IMA	Integrated Modular Avionics
MMU	Memory Management Unit
SP/DP	Single Pilot/Dual Pilot
VFR	Visual Flight Rules

Contents

1 Table

4 Figures

(18 pages in total)

1 Introduction

In the early 1980s, a new era for airborne systems and equipment began. Usage of software incorporated in these systems started to increase with a dazzling speed. The amount of software used in civil aircraft has almost been doubled every two years and given this trend it is expected to grow with a factor of about 1000 over the next 20 years [Sto96]. An industry accepted guidance - DO-178 that is currently in revision B [RTC92] - was written. This guideline has the purpose: ``to provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements".

In correspondence with FAR/JAR-25 [FAR] DO-178B [RTC92] identifies five levels of system failure conditions and associates herewith five software safety levels (hereafter simply referred to as levels). See Table 1 for the categorisation of the software safety levels.

Table 1

Software Safety Level	***Failure impact***
Level A	Catastrophic
Level B	Hazardous
Level C	Major
Level D	Minor
Level E	No Effect

The highest level software - Level A - is considered to have possible catastrophic impact if the software fails to operate according to its requirements. The lowest level - Level E - is considered to have no impact on flight safety. The identification of multiple software levels within one application imposes the problem that components of different levels may interfere with each other. This problem can be overcome by developing all components according to the highest identified level. However, as easy as this solution may seem as costly can the realisation thereof become. DO-178B [RTC92] proposes several architectural concepts to tackle this problem. One of these concepts is partitioning.

Partitioning is described by DO-178B [RTC92] as: “Partitioning is a technique for providing isolation between functionally independent software components to contain and/or isolate faults and potentially reduce the effort of the software verification process”.

The realisation of a safety critical helicopter avionics application initiated the development of a proprietary partitioning architecture that conforms to the DO-178B [RTC92] guideline. A partitioning architecture was developed whose only hardware requirements are a processor with protection modes and a memory management unit (MMU). This partitioning architecture is presented in section 2, the realisation in the avionics application is discussed in section 3.

2 A partitioning architecture for safety-critical airborne software

When using multiple levels of software - as defined by DO-178B [RTC92] - within a single hardware environment care has to be taken to prevent lower level components from influencing higher level components of the software. Faced with such a situation for a real application two options were considered:

1. To develop all software components according to the highest identified level (i.e. level A) even if a lower level would be sufficient for some components. This option was considered because the size of the components of another level than level A was relatively small.

2. To develop a partitioning scheme. Since the software is running on a processor with built-in MMU and protection mechanism (i.e. i386EX [Int87]) it is possible to implement a partitioning scheme without adding hardware requirements.

The main points of consideration were effort, flexibility and performance. Looking just at initial effort would maybe have favoured against partitioning but recurring effort because of modifications and extensions is very much in favour of partitioning. Flexibility is strongly related to the effort issues. In this sense, partitioning has a great advantage over the first option. With partitioning, it is easy to add a new extension at a lower level (e.g. new or enhanced maintenance or debug capabilities) whereas without partitioning a lot of effort could be induced. Performance however can be a big concern for partitioning. Experimental measurements on the real application pointed out that partitioning could add up to 50% execution time. This would have effectively eliminated the option to use partitioning if there were not a more efficient solution.

This performance overhead of partitioning is caused mainly by the usage of a memory management unit (MMU) which is needed to manage protection of memory and memory mapped hardware resources. The used processor has two basic approaches to memory management namely segmentation and paging (these two approaches can actually be combined into a hybrid form). Performance measurements greatly favoured paging over segmentation. Whereas segmentation adds up to 50% overhead, the overhead of paging stayed well below 5%. It should be noted that all memory allocation is static and no swapping is performed.

Paging however has a fixed granularity (4Kb for the used processor) and that can become a problem in protecting small special purpose memories (e.g. special non-volatile or dual-ported memory). However, analysis of the memory usage of the real application - especially the usage of the small special purpose memories - indicated no serious problems concerning the paging granularity.

After evaluation it was decided to develop a partitioning scheme based on paged memory management. The performance overhead of segmentation is simply too large and the potential effort increase without partitioning is unacceptable.

The rest of this section will outline the general concepts of the developed partitioning architecture. The discussion focuses mainly on the protection issues of the partitioning architecture. In addition, issues like the incorporation of tasking on top of the partitioning architecture, reusability and extendibility are discussed.

2.1 Features and goals

Following are the main features and goals of the practical general partitioning architecture:

- Compliance to the three essential attributes for an access mediation function provided by the kernel as defined by Di Vito [Vit98]:

  Complete
  There shall be no way for software running in any partition to bypass the kernel and access resources not under the kernel's control.

  Tamper-proof
  There shall be no way for software in any partition to tamper with the kernel or its data so as to subvert the kernel's control of system resources.

  Assured
  The kernel shall contain minimal functionality and shall meet all of the regulator's requirements for the criticality rating of the overall Avionics Computer Resource (ACR).

- Inter level protection support (i.e. inter and intra task).
- Multi-tasking and inter task protection support.
- Minimal extra resources requirements, notably performance overhead.
- Portability to similar hardware architectures (i.e. processor with protection levels and MMU).
- Shared memory support for inter level and inter task sharing.
- Static memory management (i.e. no swapping, no dynamic memory allocation).

Memory management is static because for dynamic memory management it is much harder to prove correct usage with respect to availability and access time. Therefore, if it is possible to use static memory management - thus no swapping and dynamic memory allocation - it is preferred above dynamic memory management for safety critical applications.

2.2 Partitioning protection architecture

The primary goal of the usage of a partitioning protection architecture in avionics is to ensure the safety of the people in the aircraft. This goal infers that it is most appropriate to use carefully chosen proven concepts for the partitioning architecture.

The partitioning protection architecture of this application is based on two concepts used in most operating systems:

1. The possibility to run partitioning management software - in the form of a kernel - in a protected environment.
2. Memory protection management through the usage of an MMU (paging).

However, the memory management of this partitioning architecture is not exactly as memory management used in other operating systems. Whereas most operating systems employ memory management mainly on a per task basis, memory management for this partitioning architecture is primarily based on a per software level basis (i.e. across tasks/threads).

Figure 1 shows the structure of the partitioning architecture. Each instantiation of a partition is assigned a level that is used to enforce access restrictions. Access to the partitions is controlled by defining execution environments with restricted access to the partitions. Six execution environments are defined: one for each of the possible software levels (i.e. A to E) and one for the kernel. Figure 2 shows the possible transitions between execution environments. How access to partitions is restricted within the six execution environments is defined as follows:

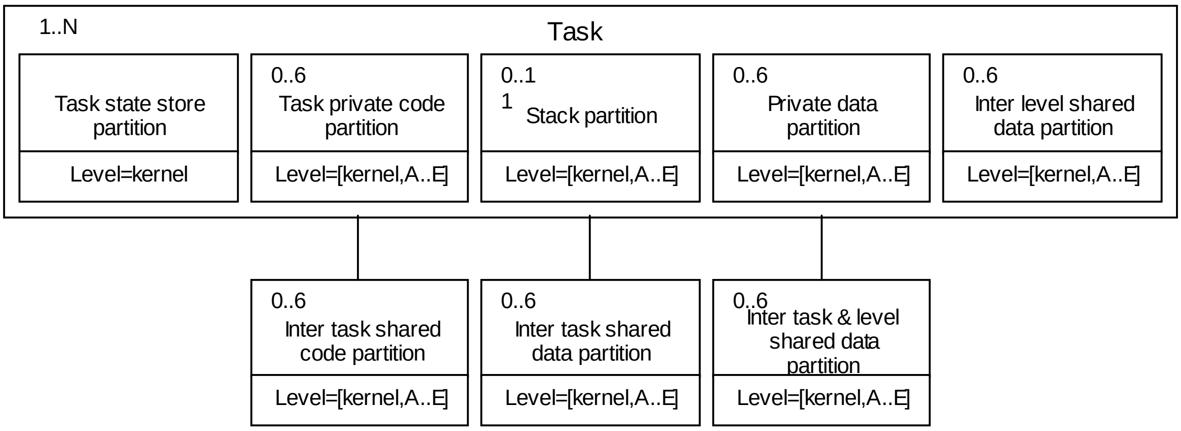

Figure 1 partitioning architecture

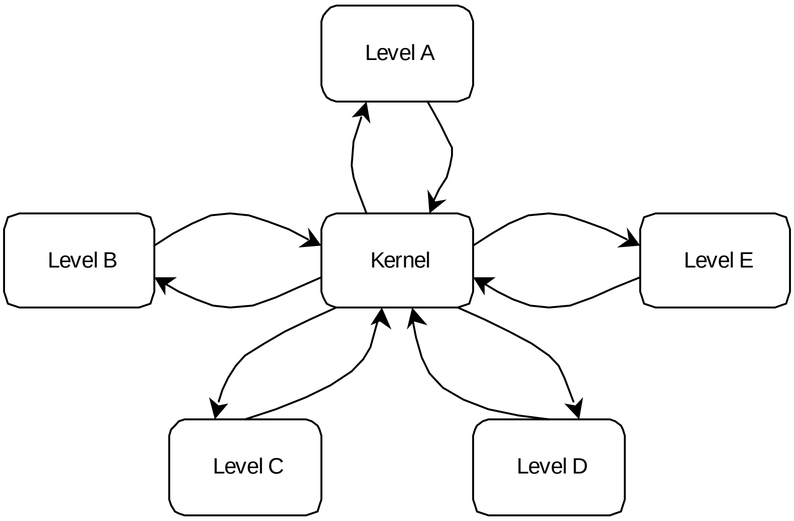

Figure 2 execution environment transitions

- Each task owns a partition to store the Task State. This partition is only readable and writable by the kernel.
- Each task owns one to eleven stack partitions. A kernel stack must always be provided. Furthermore, for each level of code executed by the task a stack partition must be provided. Finally, for each level of code involved in interrupt handling a stack partition must be assigned. Only one of the stack partitions is used at the same time by any task. Access to stack partitions other than the current stack partition is prohibited.
- Two types of code partitions exist:
  1. Task private code partitions.
  2. Inter task shared code partitions.

  Up to six code partitions of each type (task private partitions are counted per task) can exist, one for the kernel and one for each used software level. Within any execution environment, access to lower level code partitions is prohibited.
- Four types of data partitions exist:
  1. Private data partitions.
  2. Inter level shared data partitions.
  3. Inter task shared data partitions.
  4. Inter level and inter task shared data partitions.

Up to six data partitions of each type (task private partitions are counted per task) can exist, one for the kernel and one for each used software level.

This general structure can be seen as a meta-structure of which actual instantiations can be made by defining the actual instances of the partitions that are needed for an application. Implementation of a partitioning protection architecture in this way can be done by implementing a specific instance of the general meta-structure or by implementing support for the complete general architecture where an instantiation is realised through configuration. The second option will take more initial effort and will require more system resources but is flexible and far less effort is needed for reuse.

When actually implementing such a partitioning architecture an analysis should be done on what the maximum supported software level should be. The software level assigned to the partitioning management software must be at least the same as the maximum supported software level of the partitioned components. For maximum flexibility level A should be assigned to the partition management software. For the rest of this paper it is assumed that the software level assigned to the partition management software is level A.

2.3 Control coupling

Control coupling is illustrated by Figure 3. Access to other levels of software is completely controlled by the kernel, which ensures secure control coupling. The kernel assures that on a change of software level appropriate measures will be taken. The state of the current execution environment will be saved before the execution environment will be changed and control is passed to the new software level. Upon return, the execution environment will be restored and control is passed back to the requesting software level.

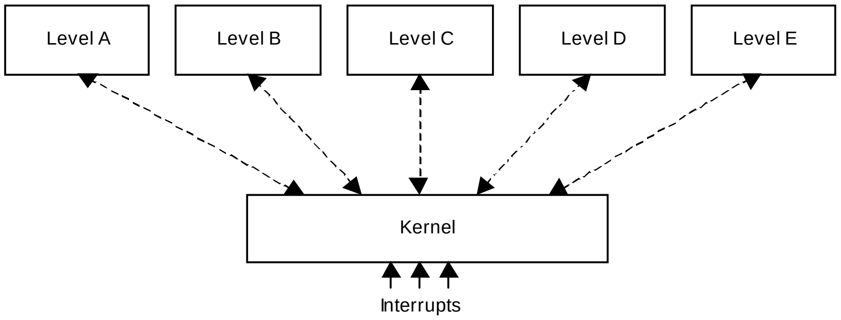

Figure 3 control coupling

The following requirements can be postulated for secure control coupling:

- Control access to a higher level is not permitted in the sense that the kernel does not allow changing execution environment and passing control to a higher level.
- Control access to a lower level requires an execution environment switch to be done before actually passing control. Thus, every control access to a lower level must be controlled by the kernel that will perform the execution environment switch.
- Under no condition may lower level code be executed in a higher-level execution environment.
- Higher level code may be run within a lower-level execution environment. In this case, no kernel intervention is needed and can be statically realised through the MMU.
- Change of control to another task is allowed only by specially assigned scheduling code, which may be either part of the kernel or implemented as user code. In either case, the scheduling code must be assigned at least the software level of the highest incorporated level of software in the application.
- Asynchronous control couplings (i.e. interrupt handling) are always handled by the kernel which however may dispatch actual handling to user level code after an appropriate environment switch.

Control couplings that require an execution environment change are serviced by the kernel by supplying so called call gates. Calls to the call gates can be completely controlled by the kernel, which can prevent any illegal control changes.

2.4 Data coupling

For data coupling between levels and tasks, some frequently used techniques can be used. The focus is on the basic concept of shared memory usage. More advanced schemes - like message passing - can be easily incorporated as an extension.

Figure 1 shows four classes of data partitions. Three of these classes are shared data partition types enabling the usage of three different kinds of sharing:

- Inter level sharing (i.e. no sharing between tasks).
- Inter task sharing (i.e. no sharing between software levels).
- Inter level and inter task sharing.

Inter level sharing is of special interest with respect to safety critical software development. Access right restrictions must be enforced for inter level sharing which is not explicitly necessary for inter task sharing.

The following restrictions must be enforced on shared data between software levels:

- Execution environments with a level lower than the level assigned to the shared data partition may not have write access to that partition.
- Execution environments with a level higher than the level assigned to the shared data partition may read from that partition but this type of access has to happen in a controlled way. Design and verification must show that usage of this data will not degrade the safety level of the functionality under any condition. To ease design and verification this kind of access could be made available only through requests to the kernel.
- Other types of access to shared data cannot degrade the safety level; thus, they can be permitted without restrictions.

2.5 Reusability and extendibility

The immense increase in quantity and complexity of software in airborne systems goes hand in hand with the increase and complexity of the hardware it runs on. Because of this growth reuse and extendibility of applied concepts in applications is also becoming increasingly important in order to save effort and cost. The partitioning architecture as discussed above is very capable of both being reusable as well as being extendable.

One easy extension is to support communication methods other than communication via shared memory. For example, a message-passing concept could be incorporated in the kernel. Messages have the advantage that they are inherently more controlled than shared memory access. This concept is used for example in a commercial operating system which is currently undergoing certification for DO-178B level B [OS98].

Current avionics development moves more and more towards the concept of Integrated Modular Avionics (IMA). The ARINC specification 653 [Aer97] specifies the baseline-operating environment for application software used within IMA. One aspect addressed by the ARINC 653 [Aer97] specification is partitioning. The developed partitioning architecture can be extended to an ARINC 653 [Aer97] compliant IMA system.

Another extension would be support for distributed systems. Take for example N processors - which are compliant with the requirements for this partitioning architecture - with communication channels to interconnect them. On each of these processors, the developed partitioning architecture could be used. The architecture could then be extended to incorporate inter processor sharing and protection.

3 Instantiation of the partitioning architecture in a realised certified application

A dedicated instantiation of the partitioning architecture as discussed in section 2 was successfully implemented in a realised and certified application. The choice to implement a dedicated instantiation of the partitioning architecture instead of a complete configurable implementation was based on factors concerning development cost and resource usage. The realised application concerns an avionics device that manages the equipment of a flight display sub-system - which the device is part of - and all the data flows in the sub-system. This sub-system is designed to operate in all possible configurations of single/dual pilot (SP/DP) and visual/instrument flight rules (VFR/IFR). For IFR the management device is duplicated and is re-configurable by the (co)pilot. In addition, some of the most critical flight data equipment are duplicated and re-configurable for IFR. All reconfigurations are also managed by the realised application.

Corruption of some of the parameters handled by the flight control display management equipment is considered a catastrophic failure condition. Therefore the software components that handle those parameters are categorised as level A; other components of the software are categorised at lower levels (i.e. B and E).

3.1 Partitioning protection architecture

Figure 4 shows the dedicated instantiation of the general structure (see Figure 1) as it is implemented in the realised application. Four instead of six execution environments are implemented; transitions are the same as in Figure 2.

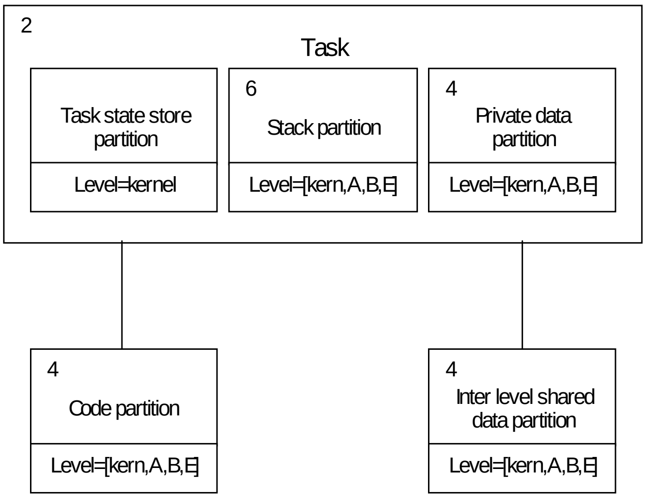

Figure 4 dedicated instantiation of the partitioning structure

3.2 Control Coupling

The implementation of the partitioning architecture in the realised application has an additional restriction concerning control coupling. This restriction states that only the highest software level (i.e. level A) is allowed to initiate control change to another level. This restriction prevents the situation where e.g. a level A component calls a level B component which in turn calls a level E component. The latter call could be very well unwanted since one of the requirements states that no level E software shall be executed when the aircraft is in the air. Note however that enforcing this restriction is not necessary because the application design should not include these cases (they are in conflict with the requirements) and the verification process should identify these cases if the software would inadvertently include one.

3.3 Data coupling

The partitioning architecture implemented in the realised application contains only one type of shared data partition namely: `Inter task & inter level shared data partition'. Access rights on shared data partitions are more restrictive then those defined for the general partitioning architecture:

- Write access is allowed only to data partitions with the same level as the level of the current execution environment.
- Read access is allowed to all data partitions. Access to data partitions of levels lower than the current execution level have been accounted for in the application design and it has been verified that invalid data cannot lead to an uncontrolled failure condition.

3.4 Overhead assessment

After the realisation of the partitioning architecture in this application, two assessments have been done concerning the introduced overhead.

1. Overhead in development effort.
2. Overhead in system resource usage.

The effort that has been put into the development of the partitioning architecture is about 1% of the total development effort put into the application. This effort is easily compensated for by the decrease in verification effort especially for modifications of the application.

The two main resources that are used by the partitioning architecture are processor capacity and memory. The partitioning architecture induces only about 1% performance overhead for the realised application. Concerning memory consumption it was determined that the usage of ROM increased by approximately 20KB on 496KB and the usage of RAM increased by about 20KB on 512KB.

4 Conclusions

This paper presented a partitioning architecture for safety critical airborne systems based on the usage of a processor with system/user protection and an MMU. With the incorporation of modern powerful processors in these systems, such a partitioning architecture can be ported to many different systems. The major benefits of this architecture are:

- Little performance overhead.
- Few memory resource requirements.
- No extra hardware requirements besides the processor protection facilities and MMU.
- Portability to a variety of systems.
- Flexibility to adjust to specific application requirements. It is even easy to adjust an existing application to incorporate this partitioning architecture.
- Easy extendibility with application specific requirements or with more general functional extensions. Among the possible extensions to this basic partitioning architecture are:
  - Support for more advanced data coupling techniques like message passing.
  - Support for Integrated Modular Avionics.
  - Support for distributed and multi-processor systems.

The partitioning architecture has been successfully implemented in a realised and certified avionics application with very reasonable resource usage. It can be concluded that this partitioning architecture is very useful and can be applied in future applications.

5 References

[Aer97] Aeronautical Radio, Inc., Annapolis, Maryland. ARINC Specification 653: Avionics Application Software Standard Interface. 1997.

[FAR] Federal Aviation Regulations/Joint Aviation Requirements 25.

[Int87] Intel Corp. 80386 Programmer’s Reference Manual. 1987. Order Number 230985-001

[OS98] Enea OSE Systems. OSE addresses safety issues in embedded systems through certification. Press release – Dallas, TX, October 1998.

[RTC92] RTCA. DO-178B; Software Considerations in Airborne Systems and Equipment Certification. 1992.

[Sto96] Neil Storey. Safety-Critical Computer Systems. Addison-Wesley, 1996.

[Vit98] Ben L. Di Vito. A Formal Model of Partitioning for Integrated Modular Avionics. Contractor Report 208703, NASA, 1998.