Exploring Modular Tickless Prioritized Preemptive RTOS for Avionics

1 Abstract

This paper explores using Modular Tickless Prioritized Preemptive RTOS (MTPPR) for avionics. Where all components are individual packaged binaries, including bootloader, kernel, drivers, and applications. A properly architected MTPPR can be certified for FAA air space and also capture the performance available to the system. MTPPR supports several goals set by Integrated Modular Avionics (IMA) architects, including: allowing multiple vendors on a CPU. The long term goal is to replace ARINC 653 with a safer, a more cost effective system, support Multiple Independent Levels of Security (MILS), and able to cover 95 percent of all avionic systems.

FAA certification is the paramount requirement. FAA's continuing mission is to provide the safest, most efficient aerospace system in the world. [1]

Implementing ARINC 653 software partitioning contains numerous invaluable lessons which can be applied back to a Modular Tickless Prioritized Preemptive RTOS targeted for FAA avionics. Although this paper is neither definitive nor authoritative, it does propose a solution to the core FAA avionic issues. This paper covers the entire avionics life cycle and attempts to dig directly to issues, which are often hidden by metrics, software abstractions, and departmental barriers.

An MTPPR solution can lower complexity, cost, and development schedule for avionic solutions by both supporting more capabilities and a simpler infrastructure. An MTPPR approach supports multiple third party hardware, software, and drivers. This can be critical to many organizations that desire to fly in FAA controlled airspace, but do not have enough resources for a complete or certifiable solution.

Generally, prioritized preemptive systems are not able to provide a deterministic solution without maintaining strict control over the entire system. However, todays hardware and tools makes it far easier to place these strict controls over the system and support multicore processors. MTPPR can be deterministic for critical applications, share a platform with applications with lesser criticality, and provide a generic environment for multiple third party vendors.

[1] `http://www.faa.gov/about/mission`

Contents

2 Identification

Author: Ty Zoerner, Infinite Delta Corp, March 13, 2016

$Id: avionics_rtos.tex 117 2016-03-13 20:46:08Z ty $
$HeadURL: svn+ssh://infinitedelta.com/svn/papers/avionics_rtos/avionics_rtos.tex $

3 Deterministic

This paper argues a prioritized preemptive system can be deterministic for critical applications. An application's dependencies determines its ability to meet requirements. Dependencies include CPU, higher priority processes, interrupts, cache, buses, memory, DMA, co-processors, multicore, or anything that can affect the application.

3.1 MTPPR Certification Tool

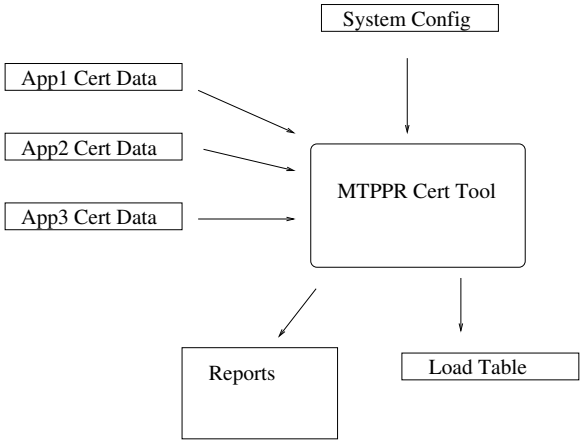
Designated Engineering Representatives (DER) are already familiar with many certification and impact analysis tools. Impact analysis shall be an integral part of an MTPPR certification tool and shall include all the dependencies. DERs and Human Factors requires it. 3.9.2

3.2 Network Example

Consider a typical situation with applications running while network data is being received into memory shared by all. The network impact varies widely between applications and between systems. The compute bound applications are not effected by the network traffic. Even the heavily memory dependant applications may only take twenty percent longer to complete under a full network load. The network traffic itself can vary considerably, and may or may not need to be time deterministic.

3.3 Quantifying Resource Impact

An application shall be subjected all possible dependency abuse and for different levels for certification. Two approaches can be taken to quantify resource impact upon applications. A general worst case approach, or a specific worst case approach. Impact of all cases are recorded and forwarded to the impact certification tool for final product certification.



$Id: certtool.fig 6 2010−08−22 20:00:59Z lloyd $

3.3.1 General Worst Case

A general worst case resource approach would include 100 percent device usage. The above network example would need to test network traffic overhead for all applications. Thereby the network needs no restrictions in this system. This is true for each dependant resource. 8.2

3.3.2 Specific Worst Case

However, many systems can not afford to assume a 100 percent network traffic overhead. The systems that can limit network traffic, a worst case scenario can be determined from a limited network traffic. Note, even for limited network traffic system, all the traffic could occur during a specific application time. So for many applications requiring small amount of CPU, the network traffic must be added. However, for long running applications, a limited network traffic may just make the difference to meet requirements.

3.3.3 Higher Priority Processes

Higher priority processes and interrupts must be included into the impact analysis for a process's ability to complete. Not only do these processes grab the CPU, but also lose precious cache data which may need reloading. Hard CPU limits for all processes and interrupts shall be imposed so that no run away process can interfere with critical programs.

Not only a does the MTPPR limit interrupt's time, the interrupt is event limited, to prevent lockup due to physical failures.

3.3.4 Network, DMA, and Bus Loading

The application shall be tested under several different levels of hardware interference issues. The impact to the application shall be recorded for system level certification purposes.

3.4 Modular and Protected

A modular design using independent binary packages allows full memory management unit (MMU) protection between all processes, devices, and kernel. Processes are now entirely protected from other processes, except where explicit memory is to be shared. A modular design allows a productive and distributed development environment. 3.9.3

Most importantly, the individual binary images are protected from most changes, minimizing regression testing. This assumes all the following are true and is an accepted process:

1. All the binaries packages making up the test suite have not changed.
2. The hardware resources being used for the test have not changed.
3. The parameters driving the test have not changed.

Accepting this process eliminates the need for source code analysis tools which many DERs find suspicious.

3.4.1 Separate Core Bootloader, Kernel, and Driver Packages

A bootloader package shall reside at the powerup boot vector, setup the basic system configuration, determine the boot state, and jump to the kernel. The kernel package is only specific to the processor, and does not contain any board specific information. The system load table is passed into the kernel from the bootloader for other packages to load to complete the system. Separate loadable drives are used instead of a single BSP containing all the hardware drivers. A minimal core must include a watchdog and interrupt enable/disable processes.

3.5 Multiple Priorities per Process

A good cooperative system can supporting process multiple priorities. The multiple priorities differs from dynamic priorities to support the required impact analysis certification tool.

3.5.1 Watchdog Timer Example

A watchdog timer is required for a given system, where if not kicked within 100 mS to CPU is reset. A cooperative watchdog time routine would be a low priority 25mS kicker with a 90mS high priority must perform setting. Thus allowing most kicks at convenient times and only jumping in when it must when the deadline looms. Very valuable for power conservation too.

3.6 Tickless

A tickless system allows processes to wake when they are required. No sub-tick CPU time wasters are needed. Much more importantly, it allows support of most deterministic IO without the use of interrupts. Hence making these drivers very deterministic.

Tickless removes and need to reduce periodic events to common factors. So there is no limits to mixing different periodic processes.

Since the kernel 3.10, Linux configured for Tickless operation has shown an order magnitude latency improvement 1.

3.6.1 Serial IO Example

A typical serial uart contains 16 bytes of buffered data. Data is lost after 8.33 mS for a read process on data at 19200 baud. Like the watch dog timer example 3.5.1 a 4 mS polling with a 8 mS must perform priority would meet full throughput. But our application only requires 40 bytes every 72.44 mS, so only three polls are required every 72.44 mS.

This serial example can be easily be added to any MTPPR system with minimal impact and therefore minimal engineering attention.

3.6.2 Phased Lock Sync Example

A triple redundant flight control system common mode failure analysis resulted in a phased lock sync requirement. This aircraft uses a naturally unstable aerodynamic design, therefore, once out of control, there is no recovery. An EMI event during a critical processor time, such as during a context switch, the only recovery is after 100mS watchdog timer and a warm start recovery period. A phased lock step system forces each of the triple redundant systems to be out of phase with each other such that there is no two processors in system critical code. 8.7

A per process delay based on the system would still allow tightly coupled IO, as in the above serial example, and meet the phased lock synch requirement. Again, this example can be easily be added to any MTPPR system with minimal impact and therefore minimal engineering attention.

3.7 Cold, Warm, and Hot Starts

Ability to warm or hot start processors and applications are critical to many avionic system, including those with the unstable aerodynamic design. 3.6.2 Although requirements vary widely for cold/warm/hot starts, the startup time is very important. Even fast cold starts are possible by allowing applications to start before the entire system checkout is completed.

3.8 Signals, Coprocessors, and Abortable Processes

Signal functions allow notification of system events and user defined events such as messages or data ready. Cold start, warm start, hot start, parity error, bus error, power fail, are some common system signals. Signals can force process aborts, and/or raise the priority of the process. Processes can be abortable, so a context does not need to be maintained. Processes can use coprocessors, such as the ALTVEC, so a larger context save may be needed when more that one process uses it.

3.9 Human Factors

Human factors are getting considerable attention in all aspects of aviation and automotive. This

section covers human factors associated with engineering avionic solutions. It specifically addresses overloading and underloading engineers.

3.9.1 Complexity

System and process complexity is the largest cause for lost oversight. Those required to over see, critic, and review do not always get the necessary information. 8.1 8.3

3.9.2 Impact Certification Tool

Suspicious source code impact analysis tools have plagued, management, engineers and DERs alike. The binary impact tool proposed here uses actual worst case data for the specific hardware under test.

3.9.3 Modular

Many different groups can work on a module, in a modular system, with zero impact on each other. Productivity jumps because the ''Need to Know'' drops.

4 Process

Although the Cert Tool is expected to be used on the early projects, it is not expected to certify a system. Full system testing will help verify the tool, but only full third party analysis can elevate the process to certification level. The initial projects will quickly determine the value of the process, and at that decide to get the necessary credentials for the cert tool.

5 Requirements

There are some specific requirements beyond those described in the Deterministic section. 3

5.1 GNU ELF Executables

The MTPPR kernel and MTPPR certification tool shall support loading GNU ELF executables configured specifically for the MTPPR kernel. GNU's binutils are GPL, which forces processing of ELF files to be GPL, so the kernel and cert tools must be GPL. Consider leading the `http://open-avionics.sourceforge.net/` effort.

5.2 MTPPR Runtime Lib

A LGPL runtime lib shall be used by all the applications and kernel. This frees up all manufactures use the system and keep their own proprietary systems in place.

5.3 Integrated Diagnostic

An integrated Diagnostic, Verification, and Certification, log shall be maintained for each process. Worst case delays and timing shall be recorded for later review.

5.4 Space Protection

MTPPR shall only support full MMU protection between all processes and the kernel. No process can affect memory not strictly owned by the process.

5.4.1 Zero-Copy Support

MTPPR shall support and encourage a zero-copy application interface for interprocess communication and devices. This directly conflicts with POSIX.

5.4.2 Inter process communication

All communication between processes and processors shall be through queues in a one way shared memory region. Allowing a one to many design.

5.4.3 Device Access

MTPPR shall support device drivers directly supplying the APIs. A general purpose POSIX API shall not be directly supported.

5.4.4 Shared Libraries

MTPPR shall support a limited shared libraries. GNU's ELF fully supports dynamic shared libraries.

5.5 Read-Only Executables

All executable enabled images shall be read only.

5.6 Policy: Zero or Minimal Interrupts

Only unexpected interrupts are encouraged. Rare events are fine, not necessarily errors. Most IO can be predicted and locked into with the MTPPR timing system.

5.7 Multiple Independent Levels of Security (MILS)

Each distributed dissemination shared memory area shall support a rotating key set. Allowing specific keys and authorizations to be updated at any time.

Modular Tickless Prioritized Preemptive RTOS (MTPPR) could considerably drop the complexity associated with Multiple Independent Levels of Security (MILS), if the Multi-Level Security (MLS) is acceptable as described in ''Multiple Independent Levels of Security: The Changing Face of Range Information Management in the 21st Century'' by G. Derrick Hinton. The tedious task to configuring disseminators 9.2 becomes automated and directly verified by the certification tool. The comes close to meeting the intent of ''Design and Verification of Secure Systems'' by John Rushby.

A federated Multiple Security Levels (MSL) scheme may no longer be needed with a properly certified MLS kernel. The hurdle here, is the validation.

6 Design

6.1 Tickless Scheduler

A simple tickless task scheduler is needed to search through a message-process list for the next process to run. Note, the message-process list is sorted, starting with the highest priority. The following pseudo code captures the core MTPPR system design:

```
now = current_time
timer_interrupt = now + 100;
for mp in message-process list do
  if mp.timeout >= now break
  if mp.last_message != message break
  if mp.timeout < timer_interrupt then
    timer_interrupt = mp.timeout
end for

set timer to timer_interrupt

context switch to mp
```

6.2 Interrupts

Interrupts shall be organized and treated like high priority processes, complete the time and event limits. The kernel when receiving an interrupt shall disable interrupts and allow these processes to run. Only after all the interrupts have been processed, will then kernel re-enable interrupts.

7 Cert Tool Verification

Verification shall include captured worst case values from a fully tested system and comparing them against the generated certification artifacts. However, only a third party analysis can elevate the process to certification level.

8 Rational and Issues

All the potential hardware, software, and system issues are outside the scope of this paper. However, some specific issues critical to this paper that need to be addressed. The following sections, listed by importance, describe these issues, including sometimes forgotten issues:

8.1 Attention to DO-178B Level A and B applications

Once an RTOS environment establishes that it can certify level A and B applications, complete with all worst case interference scenarios, only the Level A and B applications get any attention from safety. Even when lower level applications interfere, strict control keeps the interference to a known measurable quantity.

Assertion: The entire system does not need to be deterministic, only the critical applications.

8.2 Worst Case Scenarios

Only the worst case scenarios and boundary conditions are scrutinized by safety and quality. Although the entire system maybe reviewed, it is the worst case scenarios effects upon critical applications that are scrutinized by safety. If the system happens to run faster, great!, but **NEVER** miss a critical application dead line.

A large set of test scenarios are typical to certify an application, but it is the worst case values that the certification tool uses for calculating deterministic timing. So, regardless of normal application performance, it is the worst case scenario that is watched.

Assertion: The timing certification tools should only be based on worst case scenarios.

Nominal and typical timing are still important for other system aspects, but safety is concerned with worst case.

8.3 Complexity

This paper views hardware, software, and system complexity as the largest issue facing future avionic systems. Hardware MTBF drops inversely with higher hardware complexities. However both software and system complexity often leads to an exponential number of issues. Often diverting critical resources and attention way from core goals.

The total impact of requirements and processes are no longer visible to technical leads.

8.4 Thermal

Temperature can have large effects on electronics, including power, response, timings, connections, and clock crystals.

8.5 Lead-Free

Lead-Free electronic parts can short over time. Apparently whiskers grow between adjacent connections, even with the different board coatings.

8.6 Single Event Upset (SEU) and Soft Errors

Avionic computers suffers 300 times Single Event Upset (SEU) than the equivalent ground computers. [2]

8.7 Electro-Magnetic Interference (EMI) events

Large Electro-Magnetic Interference (EMI) events can effect all systems simultaneously. Non digitally filtered discretes can lead to faulty events.

8.8 Parity only buses (VME and PCI)

Double bit failures, such as through and EMI event, allows bad data to be transferred. Worse, if the address lines have a double fault, data can be written anywhere into memory.

8.9 Silos and Conflict of Interests

The increased complexity of the newer avionic systems has led to the architecture breakup. Each new group now with a narrow scope, has their own goals often in conflict with overall goals.

8.10 Obsolescence Issues

Able to perform long term support has also turned into a major issue. Example, a ten year old system may only be minimally supported do to unsupported third party kernel, whereas, a thirty year old aircraft with nearly no standard parts, but no third party obsolescence issues, can be fully supported.

Either the test and support equipment must support the future or a minimal amount of inventory is required.

8.11 Test Support

Software and hardware test support are tightly coupled with the complexity and obsolescence issues. Test requirements must press the hardware to its fullest and in different environmental conditions.

8.12 Diagnostics and Error handling

Common life cycle diagnostic and error handling techniques have been usurped by specific requirements.

8.13 Languages Ada, C, C++

C and C++ developers only need to be aware of the early often forgotten reason Ada was selected: Ada performs run time value and boundary checking, hence minimizing the effects of bad data transfers.

8.14 Priority Inversions

All systems are subject to priority inversion issues, where a higher priority task is waiting on a lower task. Only careful systems analysis or an effective certification tools catch these before integration.

[2] `http://en.wikipedia.org/wiki/Soft_error`, `http://en.wikipedia.org/wiki/Single_event_upset`

8.15 Structural Coverage

Although a complete and thorough level A structural coverage can guarantee a 95 percent certainty of a software package, both software bugs and Soft Errors 8.6 can cause infinite loops.

8.16 Degraded operations due to Hardware Failure

All single fault and many double faults must be addressed.

8.17 Cache and MMU parity

Internal CPU Cache and MMU table information shall minimally have parity to prevent a system level memory corruption. A parity capable CPU could reload the effected and continue with only a minor delay. Other registers may only effect the application running, except during a context switch. The RTOS kernel shall only minimal write access to scheduling and context switches.

8.18 Rule Enforcement

Recognizing FAA flight rules are paramount. These rules already fully support all the necessary exceptions for Military airspace, experimental vehicles, and acrobatic etc. Many issues come from attempting to add FAA rules as an after thought instead of adding specifics to an FAA approved system.

8.19 Double DMA protections

A bug or Soft Errors 8.6 can cause large scale memory corruptions regardless of the kernel selected. All DO-178B level A or B systems have bounded DMA and bus accesses so that a double failure is required to actually cause a system corruption. Data value checks must be performed for all accesses to DMA regions when received from across and parity only bus such as VME or PCI.

8.20 Real Time Requirements

Most derived real time requirements can be reworked into non critical requirements and still achieve the base requirement.

8.21 Electro-Magnetic Interference (EMI) events

A synchronized multiple CPU architectures can magnify the EMI effects, especially during critical context switches.

9 Clarifications and Terms

9.1 Abortable Tasks

A tasks that are expected to be aborted, they are always started from the entry point every time they are scheduled. No context is needed to support task resumption.

9.2 Disseminators

Disseminator is a distributed database term for the authorized data delivery to a different security level. It can include any number of translation, encryption and authentication schemes, including dynamic rotation of keys.

9.3 Mean Time Between Failures (MTBF)

The Mean Time Between Failures (MTBF) is a common avionics reliability measure, often using

units of millions of hours.

9.4 MILS

Multiple Independent Levels of Security.

9.5 MLS

Multi-Level Security.

9.6 Multipriority

Software task assigned multiple priorities. A Multipriority system is not the same as systems that support dynamic priorities. The priorities are locked at load time per a certified tool.

9.7 Preemptive

Ability to regularly suspend or abort a software task to pass control to another task.

9.8 Prioritized

Always schedule higher priority tasks before lower priority tasks.

9.9 Real-Time

The ''Real-Time'' term are systems and software subject to ''real-time constraint'' [3]. Since it applies to all the schedulers contained here, the term does not differentiate between them.

10 References

1. Linux Tickless
`http://events.linuxfoundation.org/sites/events/files/slides/LinuxCon%20-%20TicklessKernel_revc.pdf`

[3] `http://en.wikipedia.org/wiki/Real-time`