© Copyright Real-Time Consult. All rights reserved, No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
http://www.realtime-info.be
E-mail: info@realtime-info.be

REAL TIME MAGAZINE
The Dedicated Systems Developer's Reference

RTOS EVALUATION PROGRAM

Doc. Name: **What makes a good RTOS**

Doc. Version: **2.00** Doc. date: **28 February, 2000**

WHAT MAKES A GOOD RTOS

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult.

Disclaimer

Although all due care has been taken to obtain correct information and accurate test results, Real-Time Consult and Real-Time Magazine cannot be held liable for any incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided in this report, even if Real-Time Consult and Real-Time Magazine have been advised of the possibility of such damages.

http://www.realtime-info.be

E-mail: info@realtime-info.be

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult.
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
http://www.realtime-info.be
E-mail: info@realtime-info.be

EVALUATION REPORT LICENSE

This is a legal agreement between Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India, representing yourself and/or your company Accel Technologies Ltd. and the company REAL-TIME CONSULT.

1. **GRANT**. Subject to the provisions contained herein, REAL-TIME CONSULT hereby grants you a non-exclusive license to use its accompanying proprietary evaluation report for projects where you or your company are involved as major contractor or subcontractor. You are not entitled to support or telephone assistance in connection with this license.
2. **PRODUCT**. REAL-TIME CONSULT shall supply the evaluation report to you electronically via Internet. This license does not grant you any right to any enhancement or update to the document.
3. **TITLE**. Title, ownership rights, and intellectual property rights in and to the document shall be retained by REAL-TIME CONSULT and/or its suppliers or evaluated product manufacturers. The copyright laws of Belgium and all international copyright treaties protect the documents.
4. **CONTENT**. Title, ownership rights, and an intellectual property right in and to the content accessed through the document is the property of the applicable content owner and may be protected by applicable copyright or other law. This License gives you no rights over such content.
5. **YOU MAY NOT**:
   - You may not make (or allow anyone else make) copies, whether digital, printed, photographic or of another nature, except for the purposes of making a backup. The number of copies shall be limited to 2. The copies shall be exact duplicates of the original (in paper or electronic format) with all copyright notices and logos.
   - You may not post (or allow anyone else to post) the evaluation report on an electronic bulletin board or on any other form of on-line service without authorisation.
6. **INDEMNIFICATION**. You hereby agree to indemnify and hold harmless REAL-TIME CONSULT against any damages or liability of any kind arising from any use of this product other than the permitted uses specified in this agreement.
7. **DISCLAIMER OF WARRANTY**. All documents published by REAL-TIME CONSULT on the World Wide Web Server or by any other means are provided "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. This disclaimer of warranty constitutes an essential part of the agreement.
8. **LIMITATION OF LIABILITY**. Neither REAL-TIME CONSULT nor any of its directors, employees, partners or agents shall, under any circumstances, be liable to any person for any special, incidental, indirect or consequential damages, including, without limitation, damages resulting from use of OR RELIANCE ON the INFORMATION presented, loss of profits or revenues, or costs of replacement goods, even if informed in advance of the possibility of such damages.
9. **ACCURACY OF INFORMATION**. Every effort has been made to ensure the accuracy of the information presented herein. However REAL-TIME CONSULT assumes no liability for the accuracy of the information. Product information is subject to change without notice. Changes, if any, will be incorporated in new editions of these publications. REAL-TIME CONSULT may make improvements and/or changes to the products and/or the programs described in these publications at any time without notice. Mention of non-REAL-TIME CONSULT products or services is for information purposes only and constitutes neither an endorsement nor a recommendation thereof.
10. **JURISDICTION**. In the event of any disputes, the court of BRUSSELS, BELGIUM shall have exclusive jurisdiction.

Agreed by Mr Ekambaram Manickam on May 12 2001

http://www.realtime-info.be
E-mail: info@realtime-info.be

This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

© Copyright Real-Time Consult. All rights reserved.,No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

http://www.realtime-info.be
E-mail: info@realtime-info.be

This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmited in any form or by any means whatsoever without the written permission of Real-

1 Introduction

What is a Real-Time Operating System (RTOS)? There are lots of misconceptions on the topic of real time. The following text reflects the opinion of Real-time Consult on what makes a good RTOS.

We will examine techniques that can be found in General Purpose Operating Systems (GPOS) and explain why they can or cannot be used in real-time operating systems.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved.,No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

2 Real-time systems & real-time operating systems

What is a real-time system? Different definitions of real-time systems exist. Here we give just a few:

– Real-time computing is computing where system correctness depends not only on the correctness of the logical result of the computation but also on the result delivery time.
– DIN44300: The real-time operating mode is the operating mode of a computer system in which the programs for the processing of data arriving from the outside are always ready, so that their results will be available within predetermined periods of time. The arrival times of the data may be randomly distributed or may already be determined depending on the different applications.
– Koymans, Kuiper, Zijlstra – 1988: A Real-Time System is an interactive system that maintains an on-going relationship with an asynchronous environment, i.e. an environment that progresses irrespective of the RTS, in an uncooperative manner.
– Real-time (software) (IEEE 610.12 - 1990): Pertaining a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results may be used to control, monitor, or respond in a timely manner to the external process.
– Martin Timmerman: A real-time system responds in a (timely) predictable way to unpredictable external stimuli arrivals.

To build a predictable system, all its components (hardware & software) should enable this requirement to be fulfilled. Traffic on a bus for example should take place in a way allowing all events to be managed within the prescribed time limit. An RTOS should have all the features necessary to be a good building block for an RT system.

However it should not be forgotten that a good RTOS is only a building block. Using it in a wrongly designed system may lead to a malfunctioning RT system. A good RTOS can be defined as one that has a bounded (predictable) behavior under all system load scenarios (simultaneous interrupts and thread execution).

In an RT system, each individual deadline should be met. There are various types of real-time systems :

– hard real-time: missing a deadline has catastrophic results for the system;
– firm real-time: missing a deadline entails an unacceptable quality reduction as a consequence;
– soft real-time: deadlines may be missed and can be recovered from. The reduction in system quality is acceptable;
– non real-time: no deadlines have to be met.

A transactional system is NOT an RT system. Indeed, performance is defined in statistical terms as x number of average transactions per second that should be supported. If however, the requirement for such a system is more a case of x number of average transactions per second with a maximum of y fractions of a second for each transaction, then we have an RT system constraint due to the maximum time limit imposed on the transaction.

© Copyright Real-Time Consult. All rights reserved.,No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time-Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

3 System architecture

3.1 OS structures

3.1.1 Monolithic operating system

The OS is a piece of software that can be designed in different ways. 20 years ago, the OS was just one piece of code composed of different modules. One module calls another in one or more ways. This is called a monolithic OS. This type of OS has the problem of being difficult to debug. If one module has to be changed, then the impact on other modules may be substantial. If one module is fixed, other bugs in other modules may show up.

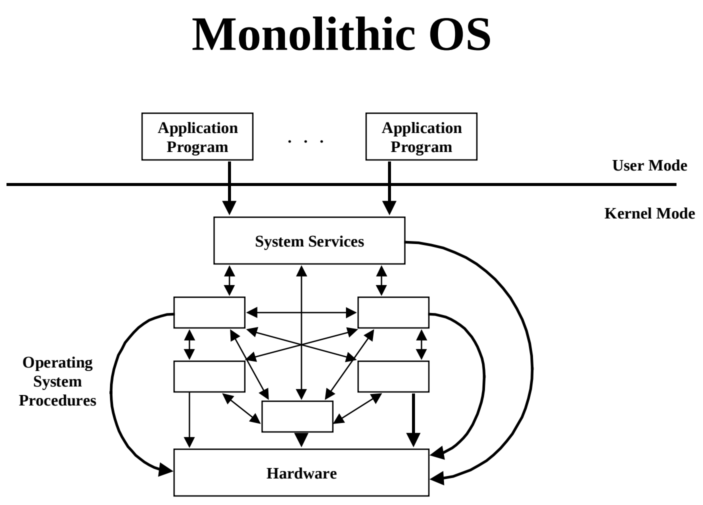

Figure 1 Monolithic operating system

The more modules there are, and the more interconnections there are between modules, the more chaotic the software becomes due to the multiple interconnections. This is sometimes referred to as “spaghetti software”. With this design, it is almost impossible to distribute the OS in one or other way on multiple processors.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

3.1.2 Layered operating system

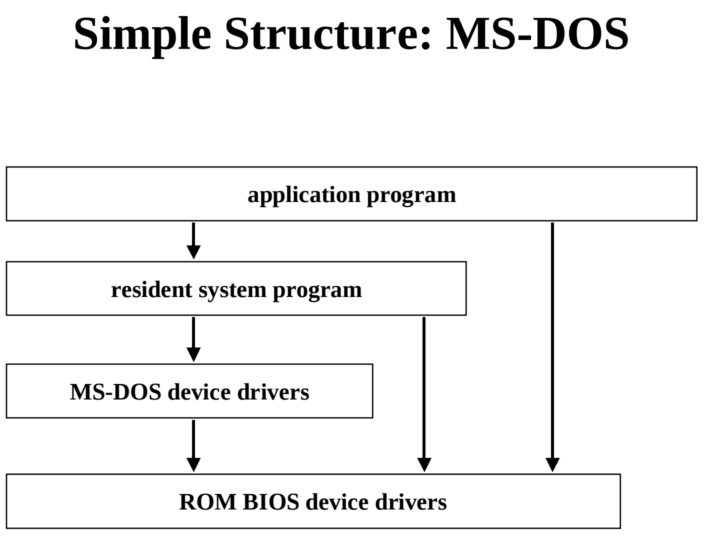

Figure 2 MS-DOS system architecture 1

A better approach is to use a “layered structure” inside the OS, such as the well-known OSI layers. However OS technology, for performance reasons, is not as orthogonal with its layers as OSI technology is. In OSI you CANNOT skip a layer. You can therefore easily replace one layer without affecting the others. This is not the case in OS technology. A system call goes directly to each individual layer. In RTOS even going directly to the hardware is desirable. In most cases the OS software is as chaotic as in the previous “monolithic” approach.

To clarify this statement the next figures show the MS-DOS approach. Figure 2 gives a representation of a well-organized OS.

However by redrawing it like Figure 3 it can be seen how things are really organized. It is indeed a layered structure, but several shortcuts exist: an application can directly access the BIOS or even the hardware.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real- This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

Simple Structure: MS-DOS (2)

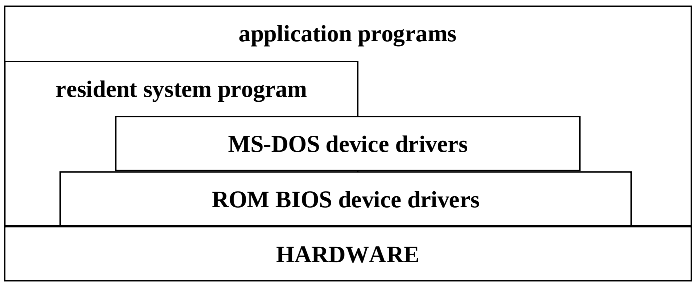

Figure 3 MS-DOS system architecture 2

RTOSs have basically been designed this way for a long time.

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

3.1.3 Client-server operating system

A new approach has been observed in the last 5 years: Client-Server technology in OS. As demonstrated in Figure 4, the fundamental idea is to limit the basics of the OS to a strict minimum (a scheduler and a synchronization primitive). The other functionality is on another level, implemented as system threads or tasks. A lot of these "server" tasks are responsible for different functions or "system calls". The applications are clients requesting services from the server via system calls.

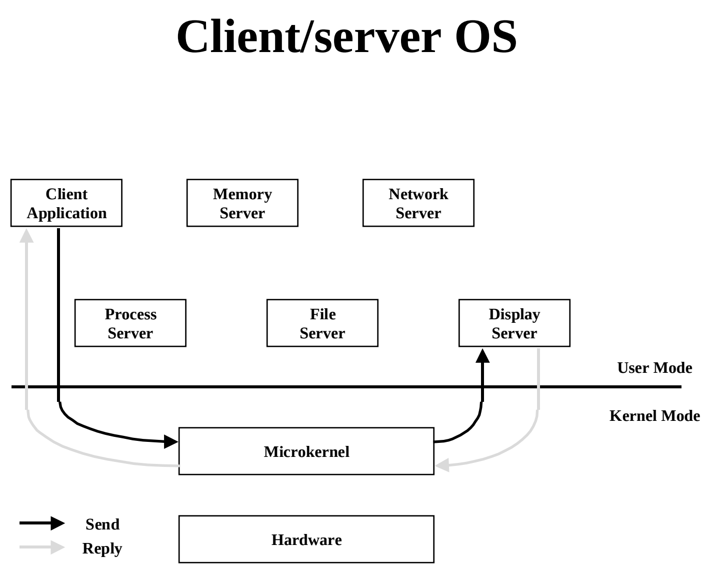

Figure 4 Client/Server operating system

This way of doing things makes it a lot easier for the OS vendor to sell a scalable OS (with a greater or lesser number of functions). It is easier to debug (each "object" remains small). Distribution over multiple processors is simple. Replacing one module does not have a "bug snowball effect". One module crash does not necessarily crash the whole system, meaning it is an investment in a more robust environment. Using this method to implement redundancy in the OS is also more conceivable. Dynamic loading and unloading of modules becomes a possibility.

The major problem with this model is the overhead due to memory protection. The server processes have to be protected. Every time a service is requested, the system has to switch from the application's memory space to the server's memory space. The time it takes to switch from one process to another will increase when the processes are protected from each other. On the other hand, if no protection is offered, a bug in the application might affect the system processes, which could compromise system stability.

3.2 Process – Thread – Task model

The first RTOS was produced more than 20 years ago by DEC for the PDP family of machines. A multitasking concept is essential if you want to develop a good real-time application. Indeed, an

http://www.realtime-info.be
E-mail: info@realtime-info.be
© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

application has to be capable of responding in a predictable way (definition of real-time) to multiple simultaneous external events (arriving in an uncontrolled way). If you only use one processor to do so, you have to introduce a sort of pseudo parallelism, called multitasking. Nowadays Rate Monotonic Scheduling (RMS) theories are helping to compute in advance the processor power you need to deal with all these simultaneous events in time.

We have one application running on a system. This application is subdivided into multiple tasks.

In the UNIX (or POSIX) world, from the outset people have talked about the different processes in the system. In these complex systems, the context or environment for each process is very heavy, (Processor, I/O, MMU, FPP, etc.) and therefore switching from one process to another is time-consuming. Two reasons changed this approach. Firstly, producing complex, distributed software requires a multitasking approach which is too heavy to implement with the process concept. Secondly, there was the POSIX effort, aimed at bringing the RT and non RT world together. The concept of a "thread" was therefore invented. It is a "sub process" or a "light-weight process". It inherits the context of a process but uses only a subset of it so that switching between threads can be done more rapidly. Also between threads, there are no security aspects because they really belong to the same environment or process.

The net result today is that it can be said that in an RT environment, a process is an application subdivided into tasks or threads.

3.3 Scheduling, Priorities & Interrupts

In a multitasking environment, you have to "schedule" from one task (or thread) to another. If more than one thread wants to use the processor simultaneously, an algorithm is needed to decide which thread will run first. A deadline-driven scheduling mechanism would be ideal. However, the current state of technology does not allow this. Pre-emptive priority scheduling offers a substitute, taking into account the existence of theories like RMS to give you a decision rule concerning which priority level to assign to each thread. Pre-emption should be used all the time to ensure that a high priority event can be dealt with before any other lower priority event. For this we not only need a pre-emptive priority scheduling mechanism: but in addition interrupt handling following different simultaneous interrupts should be handled in a pre-emptive way.

Furthermore, we have to recognise that each OS needs to disable the interrupts from time to time to execute critical code that should not be interrupted. The number of lines of code executed should be limited to a minimum to have minimum interrupt latency, but more essentially, should be bound under all circumstances.

There are different reasons to have a lot of priority levels provided in the RTOS. The first one stems from what was discussed in sections 3.1 and 3.2. In a client-server OS environment, the system itself can be viewed as one or more server applications subdivided into threads. Therefore a number of high priority levels have to be dedicated to system processes and threads.

The second reason is due to RMS scheduling theory. In a complex application with a large number of threads, it is essential to be able to place all the real-time threads on a different priority level. The non real-time threads can be placed on one level (lower than the real-time ones) and may run in a round-robin fashion. A level 0 priority or lowest level is necessary to implement the idle monitor required to measure the available processor power.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult.

4 Basic system facilities

4.1 Tasking Model

4.1.1 General

To understand the reasons for this study we need to take a look at the different phases of the system development methodology and observe where the characteristics of the RTOS emerge.

Four fundamental development phases come to light:

- Analysis: determines WHAT the system or software has to do;
- Design: HOW the system or software will satisfy these requirements;
- Implementation: DOING IT, i.e. implementing the system or software;
- Maintenance: USING IT, i.e. using the system or software.

In a waterfall model, one supposes that all these phases are consecutive. In practice, this is never possible. Most developments end up being chaotic, with all the phases being executed simultaneously. Adopting a pragmatic approach, the methodology used should just be a framework to guide producing the correct documents, whilst performing appropriate reviews and audits at the right time.

In real-time systems, both hardware and software are dealt with in what these days is called a co-design process. In such a process, the phases can be defined as follows:

- Feasibility study: how much effort will it take to build the required system;
- System analyses (SA): WHAT is the system going to do: draft refined or detailed requirements;
- System architectural design (SAD): HOW will we meet requirements by defining subsystems working in (real) parallel;
- Subsystem software analyses (SSA): WHAT is a particular subsystem going to do;
- Subsystem software architectural design (SSAD): similar to system architectural design, but here we define a pseudo-parallelism in a multitasking model;
- Software detailed design (SDD): design all the tasks in the system by subdividing them into modules
- Implementation: code writing, debugging, testing, and integration of the subsystem;
- System integration: integrating all subsystems;
- System delivery.

As can be seen, 2 important steps are concerned with architectural design. In both these steps, the OS considered as a building block is an important factor. In the SSAD – the multitasking capabilities of the OS are important. In the SAD, the capability of supporting multiple processor architectures, interconnected in different ways, is important.

In this section, attention is devoted to the multitasking model (SSAD).

We should be able to implement SSAD without knowing the RTOS used. However, commercial RTOS vendors have made certain choices and you just have to work with the possibilities and limitations of products actually available.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

All products are different in terms of the choices made. This means that an SSAD will largely depend on the RTOS chosen. This also means that porting the application to another RTOS environment is just an illusion, even if the RTOS is POSIX compliant.

4.1.2 Categories

4.1.2.1 Model

As stated in 3.2, nowadays there are various models for multitasking. Basically, an RTOS will consider the application for the system as a non-defined process, which is subdivided into tasks. Others may use a POSIX-like model with processes subdivided into threads. Nevertheless, each process should be considered as a distinct application. Not too much communication should exist between these applications and in most cases they are of a different nature: hard real-time, soft real-time and non real-time.

The use of a given model depends on the system application. In all cases, an application should first be subdivided into at least 3 subsystems: the hard real-time part, the soft real-time part and the non real-time part. Of course, the system might be so simple it just has one of these.

If the system is simple, (just hard or soft RT), then there is no need for a process/thread model. Just having tasks in the system is enough.

If however the system is complex and has at least 2 or 3 subsystems with different RT behavior, then a process/thread model is the solution.

It should be borne in mind that a simple system with just one task model would have better RT response than a more complex system with a process/thread model.

4.1.2.2 Priority levels

For the hard real-time part of the system, a method has to be found to make the system predictable under all circumstances. The best solution is to use a deadline-driven scheduler. However it will take another 10 years of development before use of such technology could be contemplated in a commercial environment.

For the time being, we have to stick to pre-emptive priority-driven schedulers. This poses a problem: how can we know the system will react in a predictable way in every case?

In 1971 Liu and Layland started working on an answer to this question. They invented what is now called Rate Monotonic Scheduling (RMS). It is a formal mathematical method for proving the conditions for having a predictable system. You need a fixed pre-emptive priority scheduling for implementing the theory. For example it can be demonstrated that a variable priority scheme is probably a better solution, but nobody has yet come up with the mathematical basis for doing this and being sure it will work in all cases.

When using RMS, you need a different priority level for each real-time task. If the system is complex, you might need a lot of priority levels. This is why we hold that at least 128 available fixed priority levels is a must.

4.1.2.3 Bounded dispatch time

When the system is not loaded, there will be just one thread waiting in a ready state to be executed. With higher loads, there might be multiple threads in the ready list. The dispatch time should be independent of the number of threads in the list.

http://www.realtime-info.be
E-mail: info@realtime-info.be

This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

In good RTS design, taking into account the thread priorities, the list is organized when an element is added to the list so that when a dispatch occurs, the first thread in the list can be taken.

4.1.2.4 Max. number of tasks (threads, process)

A task, thread or process may be considered as an OS object. Each object in the OS needs some memory space for the object definition. The more complex the object, the more attributes it will have, and the bigger the definition space will be. If there is for example an MMU in the system, the mapping tables are extra attributes for the task and more system space is needed for all this.

This definition space may be part of the system or part of the task. If it is part of the system, then in most cases the RTOS would like to reserve the maximum space it allocates to these tables. In this case, the maximum number of tasks which may coexist in the system is then a system parameter. Another approach could be full dynamic allocation of this space. The maximum number of tasks is then only limited by the available memory in the system shared among object tables, code, etc.

4.1.2.5 Scheduling policies

The scheduler is one of the basic parts of an OS. It has the function of switching from one task, thread or process to another. As this activity constitutes overhead, it should be done as quickly as possible.

To understand scheduling, it has to be understood that each task has different states. At least 3 states are needed to allow an OS to run smoothly: running, blocked and ready.

A task is running if it is using a processor to execute the task code. If it has to wait for another system resource, then the task is blocked (waiting for I/O, memory, etc.) once the missing resources that the task wants to run have been allocated to it. As different tasks probably want to run simultaneously and only as many tasks can run as there are available processors, what is needed is a “waiting for run” queue. A task in this queue is considered ready. The queue is called the “ready list”. In a symmetrical multiprocessor system, there is only one queue for all the processors. In other architectures, you have one queue per processor.

If there is more than one task in the ready queue, you need a decision-making algorithm determining which task can use the processor first. This is also called the scheduling policy.

There are probably as many policies as there are engineers inventing them. Therefore we have to limit ourselves to the ones that are actually of use in RT systems.

In all RT systems, a deadline-driven scheduling policy is required. However this is still under development and is not currently commercially available.

A pre-emptive priority scheduling policy is a minimum requirement. You cannot develop a hard predictable system without it. If you apply RMS, then each task should have a different priority level.

In more complex systems, only part of the system is hard real-time, with other parts being soft or non-real-time. The soft real-time parts should be designed like the hard RT part, with the fact that not all the needed processor power will always be available being taken into account. The same scheduling policy applies. In the non-RT part, a more general purpose OS (GPOS) approach may be desirable. In GPOS systems, the philosophy is “maximum usage of all system resources”. This philosophy is at odds with the RT requirement of being predictable.

If you want to give each task an equal share of the processor, a round robin scheme is more appropriate.

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult.
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
http://www.realtime-info.be
E-mail: info@realtime-info.be

Process states & transitions

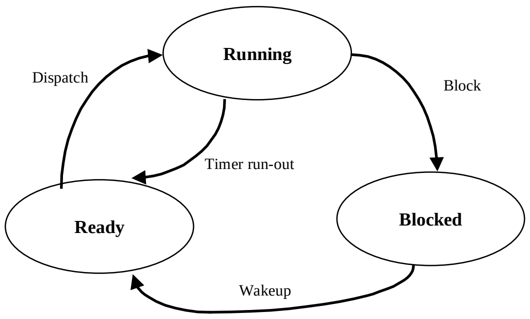

Figure 5 Process states & transitions

The non real-time part of a complex system should therefore be capable of using an RRS. Most RTOSs implement this when you put more than one task on the same priority level. Other RTOSs have an RRS explicitly defined for certain priority ranges.

Conclusion: an RTOS should always support pre-emptive priority scheduling. For complex applications, where for some parts of the system a more GPOS-oriented philosophy is needed, RRS or some other mechanisms might be useful.

4.1.2.6 Number of documented states

In the previous paragraph, we mentioned a minimum of three states per task. There is however no limit to the maximum number of states. Indeed, the blocked state can be subdivided into a number of blocked states, with the reason for the block being specified (waiting for I/O, waiting for semaphore, waiting for message send, waiting for a memory block, etc.). Making a diagram mentioning all these states provides a good graphical representation of what the OS is capable of. However this is never done in a systematic way.

Such diagrams might be very useful for debugging purposes. By animating the diagram during a reply, one could see how the system develops over time. Currently such tools are not available.

4.1.2.7 Min. RAM required per task

Memory footprint is an important issue in an embedded system despite the cost reductions in silicon and disk memory these days. The size of the OS, or the system space necessary to run the OS with all the objects defined (see 4.1.2.4), is important. A task needs to run RAM for the changing parts of the task control block (the task object definition) and for the stack and heap to be capable of executing the program (which might be in ROM or RAM). It is the RTOS vendor’s choice as to the minimum level of RAM to allocate for this. It is also up to the vendor to indicate the size of the minimum application for the intended purpose of the RTOS.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real- This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

4.1.2.8 Max. addressable memory space

Each task can address a certain memory space. Each vendor may have a different memory model, depending on whether he relies on X86 segments or not. This depends largely on the product history of the RTOS. For instance whether it was initially developed for the flat address space of Motorola processors or for the segmented X86 Intel series. The 64K segments in a 8086 processor is an important limitation for modern software. RT systems that need to be back- compatible with this type of hardware constitute a burden for the designer. On the other hand, it should be noted that this limitation is not imposed by the RTOS, making new designs possible outside the 64K space per module.

4.2 Memory

Each program needs to be held in memory to be executed. If we accept that it is not wise to develop self-modifying code, then the program may be held in ROM.

Each program using a modular approach will use subroutines and therefore needs a stack, which should be in RAM. A program makes no sense if it does not manipulate input data to produce output. These “variables” also have to be stored in RAM.

The fundamental requirement for memory in a real-time system is that its access time should be bound (or in other words predictable). As a direct consequence, the use of virtual memory is prohibited for real-time processes. This is why systems providing a virtual memory mechanism should have the ability to “lock” the process into the main memory so swapping will not occur. (Swapping is a mechanism that cannot be made predictable.)

Secondly, if paging is supported, the associative map for the pages should be part of the process context and therefore be completely loaded onto the processor or MMU. In the other case, the system is based on a statistical phenomenon which is unacceptable in hard real-time.

As a result, the story is to know if you are going to use static or dynamic memory allocation in your design.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

Combined memory protection & dynamic relocation

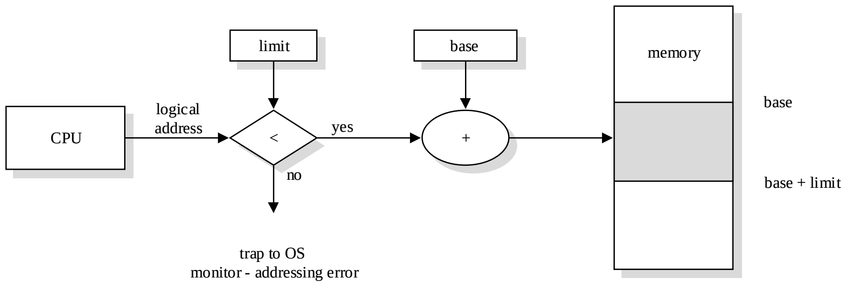

Figure 6 Memory protection & dynamic relocation

Static memory allocation means that all memory is allocated to each process or thread when the system starts up. In this case, you never have to ask for memory while a process is being executed. This however may be very costly. When hard real-time is not required, you can envisage using a dynamic allocation mechanism. This means that during runtime, a process is asking the system for a memory block of a certain size to hold a certain data structure. (You never ask memory for a piece of program in a hard RT-system). With this approach, the designer should know what to do if the memory block doesn’t become available in time. Some RTOSs support a timeout function on a memory request. You ask the OS for memory within a prescribed time limit. The task waits for the memory as long as the timeout has not expired. This feature may significantly reduce the amount of application code.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

Paging Hardware

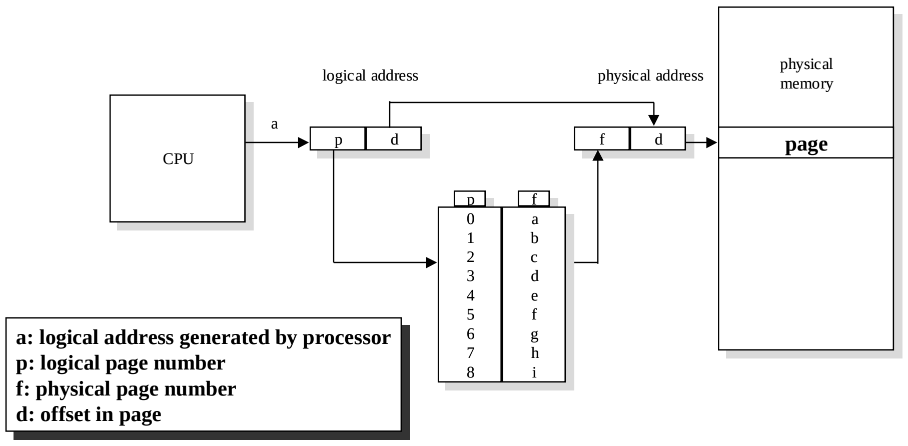

Figure 7 Paging hardware

In some circumstances, it may not be acceptable for a hardware failure to corrupt data in memory. In these instances, use of a hardware protection mechanism is recommended.

Figure 6 shows that memory relocation is possible with a change of the base address and that a protection mechanism is simple. Each logical address has to be in the range [base, base + limit] or otherwise there will be a memory protection failure.

This means that the hardware will check whether this process is allowed access to a certain memory block. This hardware protection mechanism can be found in the processor or MMU. Today’s MMUs however do much more than just protecting memory blocks. They also enable address translation, which is not needed in RT because we use cross-compilers that generate PIC code (Position Independent Code).

http://www.realtime-info.be
E-mail: info@realtime-info.be
© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

Asociated map limitations

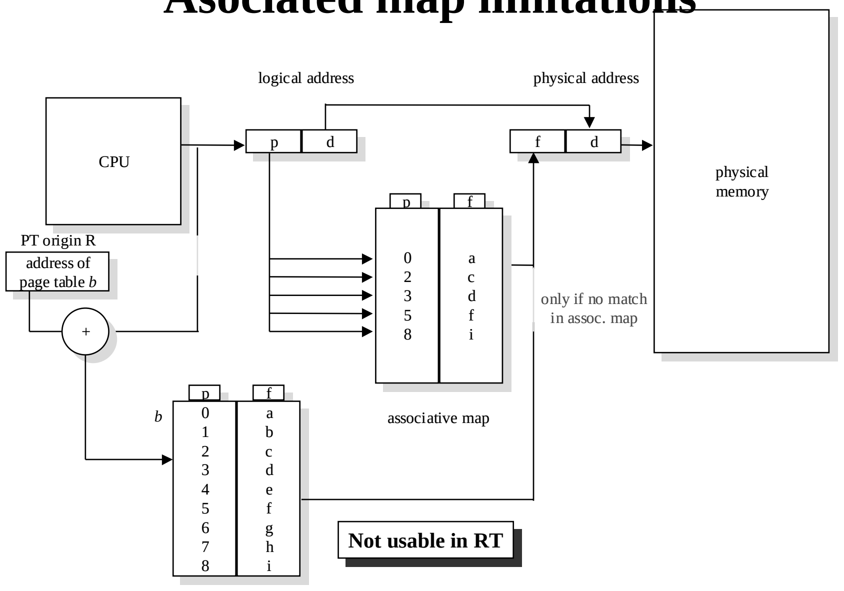

Figure 8 Associated map limitations

One potential drawback of a paged-MMU mechanism is the page size. Indeed in GPOS the page size is often 2K and the paging mechanism (see Figure 7) is connected to the memory translation mechanism (logical – physical addressing). This introduces the requirement for an associative map in the MMU. This associative map has a limited size and a sort of caching between the page tables in memory and this associative map is used to deal with this (see Figure 8). As stated before, this mechanism introduces unpredictability.

Suppose we use a paged MMU for memory protection reasons. In this case protection is linked to the pages and consequently to the page size. To make it predictable you can envisage making the associative map part of the task context. However if the page size is limited to 2 K, the number of pages needed for the whole program and the data area is too high to fit in the associative map. What you therefore need is a much larger page size.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

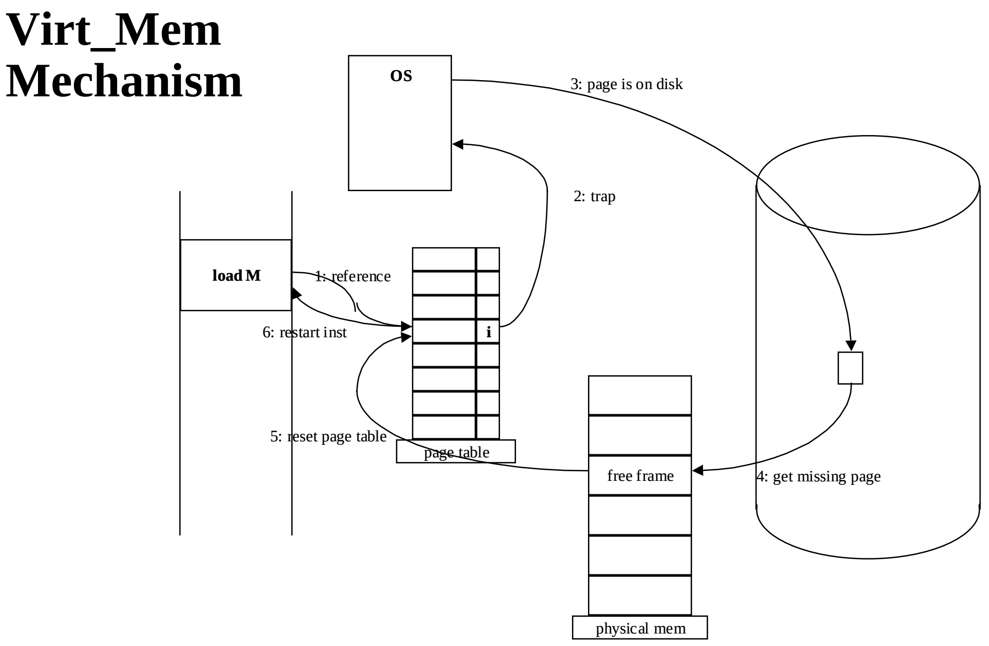

Figure 9 Virtual memory mechanism

To avoid this problem, a segmented approach can be used. Segments have variable sizes and can be much larger than 2 K, however another problem arises here. If segments with variable sizes are allocated and de-allocated, this causes external memory fragmentation. To clean up memory, some OSs use compaction of garbage collection. However, this cannot be used in an RT environment. Indeed during the compaction procedure, the tasks that are displaced cannot run, and the system becomes unpredictable. This is the major problem if object orientation is used. Consequently, as long as the compaction problem remains unsolved, C++ and JAVA are not the right choices for hard real-time tasks today.

Virtual memory (Figure 9) is another technique that cannot be made predictable, and therefore should not be used in real-time systems.

A simple solution is to allocate all memory for all objects you need during the life of the system and never de-allocate them. Another solution is always to allocate and de-allocate blocks of memory with a fixed size (introducing internal fragmentation = never using some parts of memory internal to the blocks).

From what was outlined above it can be seen that RT memory management and GP memory management have different objectives and mechanisms. It is difficult to meld them.

In simple systems that are either hard, soft or non real-time, the choice is easy. In complex systems where both hard, soft and non real-time functionalities are required, a good but expensive solution is to run each subsystem on a different processor. However these days the performance offered by readily available cheap processors is so high that you might want to place hard, soft and non real-time functionality all on the same platform.

In HRT static memory allocation is used. In SRT you have the option of dynamic memory allocation, no virtual memory, and no compaction. In non-RT you may want virtual memory and compaction.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

If we accept that non-RT tasks have a much lower priority than SRT or HRT, then the question may still be: how pre-emptable are a virtual memory mechanism and the compaction mechanism? This can only be determined by testing the RTOSs available.

Each RTOS is different in terms of its memory allocation capability. Indeed, each RTOS is targeted either towards one type of system (HRT, SRT, non-RT) or towards a mix of all of them.

A target system may not need a memory protection scheme. However, during system development, it might be interesting to have MMU support available. Indeed, having memory protection is a very nice debugging tool. It helps the designer and programmer to debug the system easily. A stack-heap overflow can be debugged in seconds with an MMU-based system, whereas in a non-protected system you might need a week to find the error. Also, it might be that the bug is there but that you never detected it during tests, and it only turns up 3 years after the system is delivered…

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

4.3 Interrupts

4.3.1 General

Remark: in this paper the word interrupt is short for "hardware interrupt". Software interrupts, together with hardware interrupts and other vectoring mechanisms provided by the processor are referred to as "exception handling" (in accordance with Motorola terminology).

An RT system is supposed to react to external events within a prescribed time limit called a deadline. All these external events are translated via the hardware as one or more bit transitions somewhere.

An initial method for detecting the occurrence of the event is to poll the event bit from the task from time to time. If the system has only one external event, and has nothing to do besides waiting for that event, then this polling mechanism is the most efficient way to go. However, in a multitasking environment, the system has to deal with more than one event and cannot afford doing busy waiting. Therefore, in the OS it was decided to detect the event via an interrupt with an associated interrupt service routine (ISR). This ISR may either be stand-alone or form part of a device driver structure, depending on the RTOS device driver model.

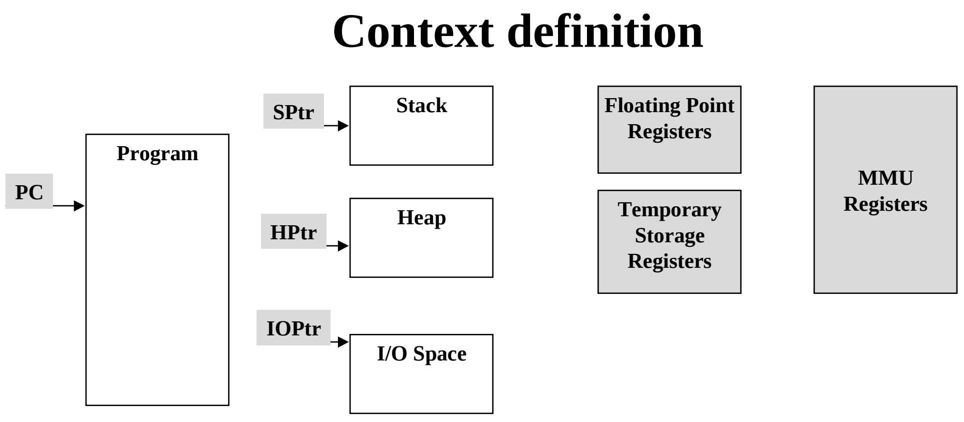

Figure 10 Context definition

Once you accept dealing with more than one event, you end up dealing with more than one simultaneous interrupt. Here again, predictability is a problem. What happens if a higher order interrupt shows up when a lower level ISR is busy? To have a quality RTOS, the lower level ISR must be pre-emptable.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

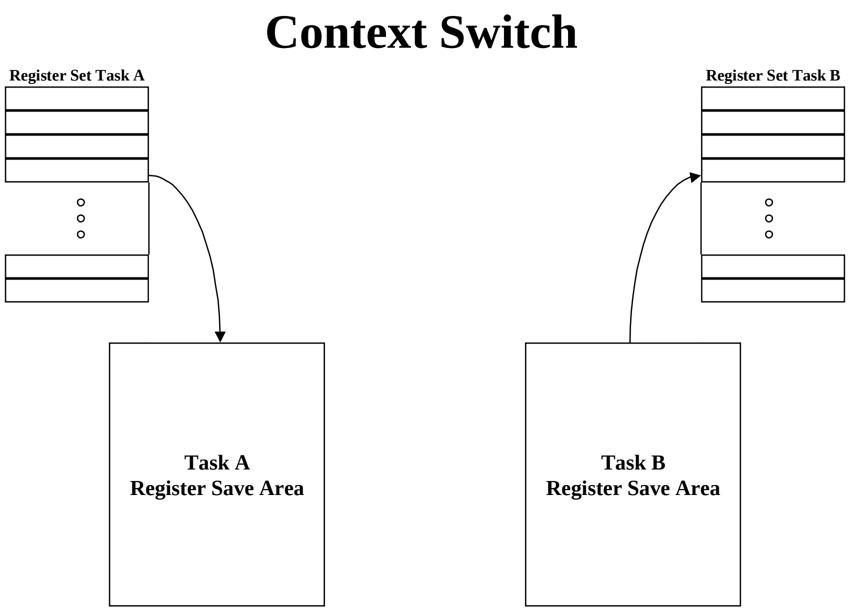

Figure 11 Context switch

Here a performance & design issue shows up. Firstly the RT system must be predictable, so there should be a set time between the interrupt and event handling. Secondly, some events need faster treatment than others (shorter deadlines).

The most straightforward method of dealing with events is to detect them via an interrupt and then start a task or thread dealing with the event. This means you need certain system calls to be executed from the ISR level and therefore you need a minimum of context (see Figure 10). The time you need, taking into account simultaneous interrupts, to go from an interrupt to starting a task is an important system parameter and should be defined (see Figure 12). Tests are needed for this because the vendors never publish comparable figures on these issues.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

Context Switch Timing

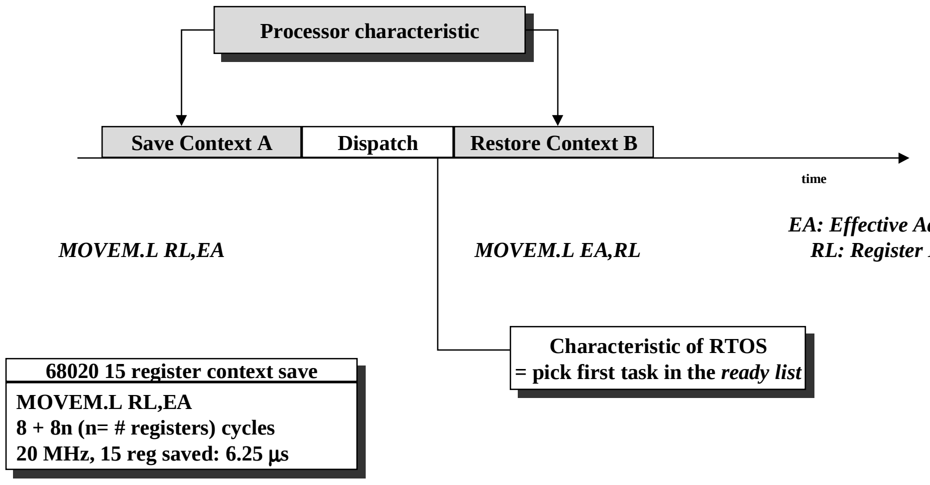

Figure 12 Context switch timing

Under some circumstances, the previous latencies cannot be accepted because the design requires shorter deadlines than the ones obtained with the previous technique. This can be solved by executing event code either in the ISR, in the 0 level ISR, in background mode, or in the device driver. This means that there are different levels where the event code can be executed. Good design requires the shortest possible ISRs and device driver, but you may decide to override this.

The various RTOSs may differ in terms of the model they use to deal with this. The simplest model has just one task level and an ISR level. The complex model has an ISR, a background ISR, a device driver and a task level. Depending on where an event is dealt with, the event handling parameters will be different (Figure 13). What matters here is the context definition for each level: the simplest for the ISR and the most complex for the process or task level. Context switching times largely depend on the context definition and the processor speed. It is the RTOS, together with the compiler, that defines the context definition on each level.

http://www.realtime-info.be
E-mail: info@realtime-info.be

This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

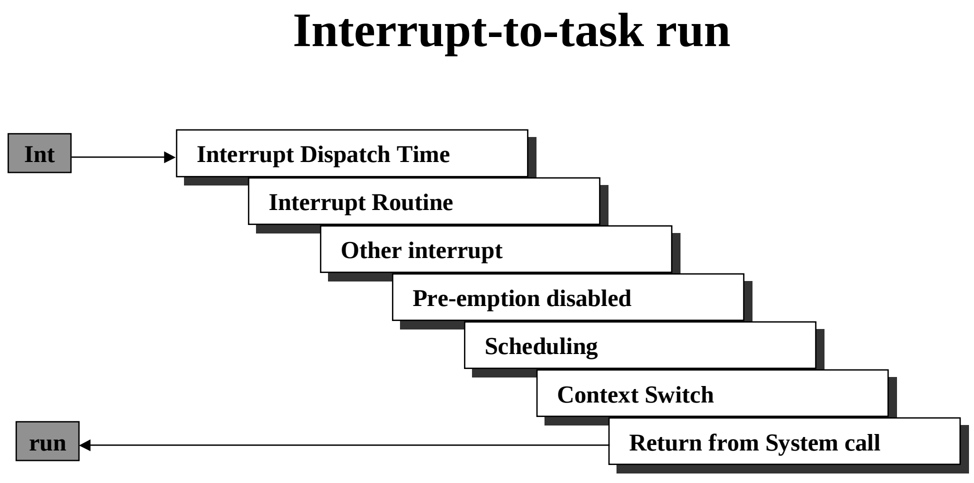

Figure 13 Interrupt-to-task run

The evaluation report should clarify which model is used by the RTOS. Figure 13 shows the various elements that are part of the total interrupt-to-task run time.

- Interrupt dispatch time: the time the hardware needs to bring the interrupt to the processor.
- Interrupt routine: the ISR execution time.
- Other interrupt: the time needed for managing each simultaneous pending interrupt.
- Pre-emption disabled: the time needed to execute critical code during which no pre-emption may happen.
- Scheduling: the time needed to make the decision on which thread to run.
- Context switch: the time to switch from one context to another.
- Return from system call: extra time needed when the interrupt occurred while a system call was being executed.

The maximum time needed to go from interrupt to run is the sum of all the potential maxima of these different latencies. The fact that more than one simultaneous interrupt is to be dealt with is important. Consequently, at the design stage you should always try to determine how many simultaneous interrupts might occur!

The test will measure the maxima for these times under all loading conditions.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmited in any form or by any means whatsoever without the written permission of Real-Time Consult.
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
http://www.realtime-info.be
E-mail: info@realtime-info.be

5 API richness

5.1 General

5.1.1 Purpose

The purpose of this section is to provide detailed consideration of all the available system calls for the OS, commonly referred to as the API or Application Program Interface.

Every system call is a software interrupt (SWI). The traditional vectoring system of each processor is used to implement the service. The number of vectors is one of the processor's limitations. The processor may have enough software vectors to implement all the system calls, but in complex OS this is rarely the case. Therefore a data passing mechanism will be used by the OS around the SWI to enable use of the same vector for all or part of the system calls and also to enable status data to be produced once the system call is executed.

The traditional approach is that the program continues executing just after the SWI instruction. This is where every programmer should implement error handling.

If an assembler is used, the programmer will clearly see the SWI and will need to know the details of the data structures and the passing mechanism of this data to the OS. When using a high level language, SWI system calls will be implemented as procedure calls in that language. The term used is a C or ADA or whatever type of interface with the OS.

The format of these procedure calls is defined by the interface library builder and may or may not match a standard like POSIX. (See 5.1.2).

The fundamental issue we want to discuss concerning the API is its richness. Indeed the number of system calls may be very limited. The minimum functionality you need in an RT multitasking environment is a scheduler and one synchronization primitive. All other synchronization and communication primitives or system calls can be built on that. The question however is, what part of the software is going to do the job, the application or the OS? The more system calls you have and the more complex they are, the fewer lines of codes the application will have. Moreover, the code executed in the OS is certainly more efficiently executed (and better debugged) in the OS than the same code in the application. So our concern is to ascertain how many system calls are supported and how complex they are. The ultimate aim is to reduce application maintenance due to limited application code length as much as possible.

Of course, a very simple application can probably get by with just a few system calls. A very complex one should have a wide choice of system calls though.

5.1.2 POSIX

Among other things, POSIX defines the syntax of the library calls that execute the SWI for the OS interface. This means that if, on the application side, in C, everybody uses the POSIX interface, then all application code should be the same. However, as will be seen, it is not as simple as that. On the implementation side, all RTOSs are different. Each product will have different performance characteristics for the "same" system call.

POSIX is very new and there are a lot of legacy applications around. Also, POSIX does not necessarily have the best system calls for efficient implementation of certain portions of code. Therefore lots of vendors not only support POSIX but also support a second set of non-POSIX system calls, which does

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

not run counter to POSIX rules. Both POSIX and non-POSIX compliant calls may co-exist on the same system. This is one of the main reasons why we do not think POSIX compliance is an essential issue today.

Another issue is that it took a very long time to standardize and the standard has been cut into different pieces: the ones everybody can agree on immediately, the ones it is difficult to agree on and the ones they will never agree on… At the time of writing, some parts of the standard were finalized, while others were still in draft form. Therefore, different products comply with different "draft revisions" of the standard which makes the actual usefulness of being compliant somewhat problematic.

Real-Time Systems range from very small embedded systems like modems to very complex hybrid systems like a satellite ground station. Today's RTOSs are not scalable enough to deal with both small and large systems. A particular RTOS is aimed at a particular target application size. The historical consequence is that a small embedded system RTOS uses a totally different API from an RT-UNIX-like RTOS. To deal with this, the POSIX committee invented profiles, as stated in POSIX 1003.13. This means that if you refer to POSIX compliance, you should specify the profile. However no one does. Another point is that these profiles do not cover all the systems around these days.

Features versus Profiles

Feature	Minimal	Control	Dedicated	Multi
Process Primitives	+	+	+	+
Process Environment	+	+	+	+
Files & Dir	-	+	-	+
IO	+	+	+	+
Device Specific	-	+	-	+
Data Base	-	-	-	+
Binary Semaphores	+	+	+	+
Memory Locking	i	i	+	+
Mapped Files	-	+	-	+
Shared Memory	i	i	+	+
Process Priority Sched	-	-	+	+
Real-time Signals	+	+	+	+
Clocks & Timers	+	+	+	+
IPC Msg Passing	+	+	+	+
Synchronised IO	+	+	+	+
Asynchronous IO	-	+	+	+
Prioritised IO	-	-	+	+
RT Files	-	+	-	+
Threads	+	+	+	c

i: implicit
c: configurable

Figure 14 Features versus profiles

The key conclusion is that POSIX compliance is not a fundamental issue. POSIX does have one advantage however: nowadays if we are talking APIs, we are talking more or less the same language, which was not the case 10 years ago.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

Problems with POSIX .13

	POSIX AEP			
	Minimal	Controler	Dedicated	Multi
Files & Dir	no	yes	no	yes
RAM Disk	no	yes	no	yes
Standard File System	no	yes	no	yes
RT File System	no	yes	no	yes
Mapped Files	no	yes	no	yes
Network	no	no	no	yes

Figure 15 Problems with POSIX 1003.13

RT application portability is a myth. Analysing and designing an RT application methodically with RT-modeling techniques is the only solution. Producing new code working from this documentation is easier than trying to port a POSIX application from one system to another.

(RTConsult) Extended POSIX .13

	Extended POSIX .13							
	Minimal	Controler			Dedicated			Multi
		Embedded	Connected	Data Acquisition	Embedded	Connected	Data Acquisition	
Files & Dir	no	**yes**	**yes**	**yes**	**no**	**yes**	**yes**	yes
RAM Disk	no	**yes**	**yes**	**yes**	**no**	**yes**	**yes**	yes
Standard File System	no	**no**	**no**	**yes**	**no**	**no**	**yes**	yes
RT File System	no	**no**	**no**	**yes**	**no**	**no**	**yes**	yes
Mapped Files	no	**no**	**no**	**yes**	**no**	**no**	**yes**	yes
Network	no	**no**	**yes**	**opt**	**no**	**yes**	**opt**	yes

Figure 16 Extended POSIX 1003.13 by RTConsult

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
http://www.realtime-info.be
E-mail: info@realtime-info.be

5.2 Categories

5.2.1 Task management

In paragraph 3.2 we explained the difference between task, threads and processes. The most complete model will support processes (= applications) subdivided in threads. Threads are the units scheduled and executed in the system. Threads can be started, stopped and resumed. When subdivided into threads, a process is neither started nor stopped because a process is more a sort of context.

When using RMS, there is no intention of changing task priority levels during execution. Therefore you don’t really need this feature. However, in some application areas it might be advantageous to do so. Priority inheritance for example is an automatic mechanism that changes the priority of a task to avoid priority inversion. It is therefore an investment for future systems to have the possibility to dynamically change the priority of a task. What is important here is to know which object may do so (another task, an ISR, etc) and under what circumstances.

5.2.2 Clock and timer

Most RT systems work with relative time today. Something happens BEFORE or AFTER some other event. In a fully event-driven system you don’t need a ticker since there is no time slicing. However if you want to time stamp some events, or if you want to introduce systems calls like “wait for one second”, you need a clock and/or a timer.

RT Synchronization today is done by blocking (or waiting) for an event. Absolute time is not used. This might change in the future. Indeed, you can also synchronize things by using absolute time. This is what humans do if they decide to start a meeting at 9:00 am. Having a precise absolute time clock in the system is therefore something we should look to in the future. Some systems need it already, especially in aerospace applications. In most cases, the software absolute clock is not precise enough for these applications. A precise hardware clock should be provided, which in turn, the RTOS should support.

5.2.3 Memory management

We discussed memory management in detail in section 4.2. In a non-POSIX environment, RTOS vendors use different names for similar objects and the same names for different objects.

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-Time Consult. This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India http://www.realtime-info.be E-mail: info@realtime-info.be

Naming in RTOSs

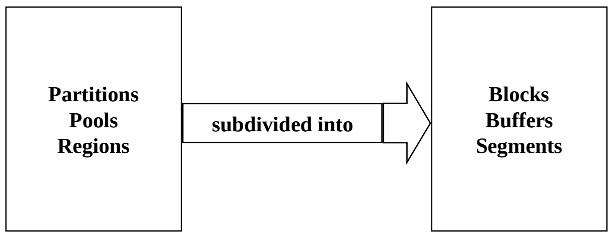

Figure 17 Naming in RTOSs

To make a good RT design, you should know whether the RTOS implements fixed or variable size blocks (buffers, segments) and if these structures are protected by an MMU mechanism.

The design should take care not to run into a fragmented memory when using dynamic allocation and de-allocation.

An important feature is the fact that memory allocation may or may not be time limited. Indeed, if the application needs memory to fulfil a deadline, then this memory should be available immediately when asked for, or at the latest within milliseconds. Some vendors support such a timeout as an attribute of the system call. This is very important in reducing the amount of application code and consequently the level of system maintenance. Indeed, if no such feature is implemented, the RTOS will return a “no available memory” error code after the allocation request. If the system can wait a while, the application should then take the effort to ask for that memory again until timed out. This makes the application code very complex.

This example clearly shows that OS calls richness is very important in reducing the number of application lines of code. This reduces maintenance and the likelihood of bugs, with the result being that the system will be more reliable.

5.2.4 Interrupt handling

An RT designer has to write his own interrupt routines (and device drivers). These modules are part of the OS. Therefore they are difficult to debug, and a mistake in these can lead to a major disaster. Some RTOSs are trying to limit the potential for such disaster by not allowing the vector table to be changed by the programmer. Others just don’t care. It is difficult to say which is the best solution. The first solution introduces some protection against programmer errors, but introduces some extra indirect jumps and therefore overhead, and a reduction in interrupt handling performance. The second solution is fast but requires the programmer to take more care.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

Another issue is to consider what system calls can be performed starting from an ISR.

5.2.5 Synchronization and exclusion objects:

Synchronization and exclusion are needed so threads can execute critical code. The objects can also be used to ensure that some threads are executed one after the other. A range variety of objects is available:

- semaphores: synchronization and exclusion;
- mutexes: exclusion;
- conditional variables (in conjunction with mutexes): exclusion depending on a condition;
- event flags: synchronization of multiple events (can contain high level logic);
- signals: asynchronous event processing and exception handling.

The rules for when to use each objects are very simple:

- How can requirements be met by producing the minimum amount of code?
- What construct can I use for flexibly handling design changes?

How to apply these rules is not the purpose of this document. However, if an RTOS does only support one or other primitive or object, these rules cannot even be applied. We hold that the richer the RTOS API, the better. At least then you will have a choice at the design stage.

5.2.6 Communication and message passing

Communication and message passing are a form of synchronization where data exchange occurs. Examples are:

- queues: multiple messages;
- mailboxes: single messages.

The rules and remarks provided in the previous section also apply here. However, when data is exchanged, another issue arises: is the data structure completely copied from the sender thread space to the receiver thread space or is just the pointer passed?

In most RTOSs, passing the full data structure is not done for performance reasons, so a pointer goes from the sender to the receiver.

Here we have another design issue: are you sure that in all circumstances the pointer is still valid when the thread is using it? This document is not the place to go into these details, but the reader should be aware of the problem.

Another issue is: “does the object accept single or multiple senders and receivers”?

By stating all these issues, one thing becomes clear: each RTOS behaves differently with what outwardly are the same objects. It is our aim to gain an understanding of this behavior by conducting tests and publishing the results in an evaluation document.

5.2.7 Waiting list length

As stated, synchronizing means blocking or waiting on a synchronization object. A very important feature in a good RTOS is that the temporal behavior of the system does not depend on the length of these waiting lists.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

Good RTOS design means that for each thread that starts pending on an object, the list is reorganized at that moment, so that the time it takes to release the object is independent of the queue list length.

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

6 Development methodology

6.1 Introduction

It is important for an RTOS to provide users with an efficient way of developing applications. Having good, efficient tools available plays an important role in the development process, but there is more. Different design philosophies exist, each with their own advantages and pitfalls. Operating systems can use different *host* and *target* configurations. The *host* is the machine on which the application is developed, while the *target* is the machine on which the application executes.

6.2 Host = Target

In this configuration, the host and target are on the same machine. The RTOS has its own development environment (compilers, debuggers) and its own command shell. With this configuration, there are no connection problems between host and target. However, the development environment is sometimes of lesser quality, since the OS vendor often does not have sufficient resources to develop both the operating system and the development environment. Furthermore, the RTOS does not have all the features available in a general purpose operating system (GPOS) which facilitates development (e.g. source code control system, backup tools, etc.).

6.3 Host ≠ Target

In this case, host and target are two different machines linked together (e.g. serial link, LAN, bus, etc.) for communication purposes. The host is a machine with a proven GPOS, which is often more suitable as a host than a machine with a dedicated RTOS. This situation often allows for a better and more complete development environment, since all the features of the host can be used.

The drawback of this configuration comes in debugging. The debugger is on the host, while the application is executed on the target. So-called debug agents are stored in the target to communicate debug information to the host.

When the host and target are different machines, the development environment should provide simulators allowing developers to execute a prototype of their application on the host by simulating the target. There are two ways the target can be simulated: by simulating the target microprocessor or by simulating the target RTOS (API).

6.4 Hybrid solutions

Hybrid solutions attempt to combine the best of both worlds. The host and target are on the same physical machine, but they run on different operating systems that communicate with each other in some way (e.g. via shared memory). The host OS is the rich and proven GPOS, while the target OS is the dedicated RTOS.

In this situation, the application can be developed using all the tools available in the GPOS, and since the RTOS and the target application code run on the same hardware, communication between the two is not a big issue.

In reality however, these hybrid solutions have their own set of problems. The same hardware resources are shared by two operating systems (the GPOS and the RTOS), which sometimes prevents the RTOS

http://www.realtime-info.be
E-mail: info@realtime-info.be
This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India
© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

from exhibiting predictable real-time behavior under all circumstances. On the other hand, if the RTOS monopolized the processor, this could keep the GPOS from performing its housekeeping chores, thus compromising the stability of the whole system. Multiprocessor architectures should improve things.

http://www.realtime-info.be
E-mail: info@realtime-info.be

This licensed copy is owned by: Mr Ekambaram Manickam, Accel Technologies Ltd., 160, Greams Road, Chennai, India, 600006, India

© Copyright Real-Time Consult. All rights reserved,.No part of the contents of this document may be reproduced or transmitted in any form or by any means whatsoever without the written permission of Real-

7 Conclusion

The purpose of this document was to explain the reasons for testing commercial RTOSs by first defining what we think are essential features for a good RTOS.

As the building block of an RT system, an RTOS should in all cases behave predictably. This means behavior bounded in time, independent of the system's load and the length of the queues in the system.

The purpose of the evaluation project is to determine if all the requirements for "a good RTOS" are met, by analyzing the documentation and by performing extensive tests.