TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

VESA VALKONEN

DEPLOYING A MODULAR APPLICATION INTO A RUNNING AUTOMATION SYSTEM

Master's thesis

Examiner: Professor Seppo Kuikka
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences on
6. May 2015

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
VALKONEN, VESA: Deploying a modular application into a running automation system
Master of Science Thesis, 55 pages
May 2015
Major: Automation Software Engineering
Examiner: Professor Seppo Kuikka

Keywords: Automation system, embedded system, real-time, performance, development, testing

Industrial systems are commonly controlled by different kinds of controllers. In advanced systems, these controllers can be computer-like devices containing an operating system, a processor, a run-time memory, a mass storage and optionally other components. The difference between controllers and computers is that the controllers usually have less available computing resources, more restricted tasks and real-time requirements which are typical for automation systems.

In this thesis some possible techniques are investigated that could speed up the development process of the applications in the controllers. When developing those applications, eventually there will come a need for testing them in a real environment. In this case, the automation system needs to be shut down for the time of deploying new applications and then it needs to be restarted in order to test the new applications. In many automation systems, the restarting process is a slow and expensive operation, which limits the amount of application combinations that can be tested and thus slows down the development process. This thesis focuses on this problem by investigating the possibility to change the used application when the system is running.

The first part of the thesis discusses theoretically the problem at hand, available techniques, implementation requirements, possible problems to appear and evaluation criteria for possible implementations. After this, the most important techniques affecting the implementation are evaluated. It is concluded that real-time operating systems provide better real-time characteristics than general operating systems patched for real-time usage. It is also noted that the characteristics of and special requirements for an automation system greatly affect when deciding which kind of an implementation technique should be used. No generally applicable solutions could be presented.

Finally one solution proposal is implemented in order to test the application reconfiguration technique in practice. Measurements done during testing give guidelines for the usability of the technique in different use cases. It is concluded that the technique could be used for development and testing of systems filling the preconditions. This technique cannot be recommended to be used in final automation systems without good reasoning and extensive focus on safety and security matters. As the final conclusion, the application reconfiguration technique is seen as a possible and beneficial feature in limited use cases.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
VALKONEN, VESA: Modulaarisen sovelluksen käyttöönotto käynnissä olevassa automaatiojärjestelmässä
Diplomityö, 55 sivua
Toukokuu 2015
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Seppo Kuikka

Avainsanat: Automaatiojärjestelmä, sulautettu järjestelmä, reaaliaika, suorituskyky, kehitys, testaus

Teollisuuden järjestelmiä ohjataan pääsääntöisesti erityyppisillä säätimillä. Kehittyneimmissä järjestelmissä säätimet voivat olla käytännössä tietokonetta vastaavia laitteita sisältäen käyttöjärjestelmän, prosessorin, työmuistin, massamuistin ja mahdollisesti muita komponentteja. Näiden säädinten erot tavallisiin tietokoneisiin ovat usein rajatummat käytettävissä olevat laskentaresurssit, rajatumpi tehtävä sekä automaatiojärjestelmille tyypilliset reaaliaikavaatimukset.

Tässä työssä tutkitaan mahdollisuutta nopeuttaa säätimissä olevien sovelluksien kehitystyötä. Sovelluksia kehittäessä jossain vaiheessa tulee tarve niiden testaamiseen oikeassa ympäristössä. Tällöin testattava järjestelmä pitää sammuttaa uuden sovelluksen järjestelmään siirtämisen ajaksi ja käynnistää uudelleen sen testaamiseksi. Useissa automaatiojärjestelmissä uudelleenkäynnistys on kuitenkin melko hidas ja kallis operaatio, mikä rajoittaa testattavien säätösovellusten yhdistelmien määrää ja hidastaa kehitystä. Tässä työssä pureudutaan tähän ongelmaan tutkimalla mahdollisuutta vaihtaa käytössä olevaa sovellusta automaatiojärjestelmän käynnissäoloaikana.

Työn alkuosa käsittelee teoreettisesti käsillä olevaa aihepiiriä, käytössä olevia tekniikoita, toteutusvaatimuksia, mahdollisia eteentulevia ongelmakohtia sekä mahdollisten toteutuksien arviointikriteerejä. Tämän jälkeen tärkeimpiä toteutukseen vaikuttavia tekniikoita vertaillaan keskenään. Todetaan että varsinaiset reaaliaikakäyttöjärjestelmät tarjoavat paremmat reaaliaikaominaisuudet kuin reaaliaikakäyttöön laajennetut yleiset käyttöjärjestelmät. Lisäksi huomioidaan, että automaatiojärjestelmän ominaisuudet ja erityisvaatimukset vaikuttavat huomattavasti toteutustekniikoiden valintaan eikä yleiskäyttöistä ratkaisua voida esittää.

Lopuksi toteutetaan yksi mahdollinen ratkaisu, jotta sovelluksien ajonaikaista käyttöönottoa voidaan testata käytännössä. Testauksen yhteydessä tehtävät mittaukset antavat osviittaa tekniikan käyttökelpoisuudesta eri käyttötapauksissa. Todetaan, että tekniikkaa voidaan hyödyntää reunaehdot täyttävissä järjestelmissä testaukseen ja sovelluskehitykseen. Lopulliseen tuotantokäyttöön tarkoitettuun automaatiojärjestelmään tätä tekniikkaa ei voida suositella ilman erityisen painavaa syytä ja suurehkoa panostusta henkilö- ja tietoturvaan. Johtopäätöksenä todetaan sovellusten ajonaikainen käyttöönotto mahdolliseksi ja hyödylliseksi ominaisuudeksi rajatuissa käyttökohteissa.

PREFACE

This is a master's thesis written for the Department of Automation Science and Engineering in Tampere University of Technology. This thesis has been conducted in the years 2014 and 2015, including the theoretical writing and the implementation of the solution described and measured. A sincere commendation belongs to Jari Kuusisto for presenting the need for this kind of research.

I want to express my special thanks to both of my thesis mentors, Prof. Seppo Kuikka (TUT) and MSc. Lassi Niemistö (Wapice Oy), for the valuable support and comments during the writing process. I also want to thank Eino Puikkonen for reviewing the thesis. I am also grateful for my employer Wapice Oy, providing supportive coworkers, facilities and tools needed for the thesis.

Vaasa 7.5.2015

Vesa Valkonen

TABLE OF CONTENTS

LYHENTEET JA MERKINNÄT

Adeos	Virtualization layer in Xenomai.
FPGA	Field-programmable gate array. Integrated circuit that can be modified after manufacturing.
FreeRTOS	A real-time operating system implementation.
I/O	Input and output.
IPC	Inter-process communication. Data transfer between the processes executed in parallel on the same computer.
ISR	Interrupt Service Routine. Usually small piece of code that quickly handles the interrupt.
Mutex	Mutual exclusion. An exclusion mechanism.
PID	Proportional-integral-derivative controller. Common feedback controller in industrial control systems.
RT_PREEMPT	Linux patch, which aims for making Linux to be fully pre-emptible
RTAI	Real-Time Application Interface. Extension for Linux.
Spinlock	Uninterruptible exclusion mechanism.
Xenomai	Real-time extension for Linux. Development has been forked from RTAI project.

1. INTRODUCTION

Distributed control systems are one solution to handle complex automation systems. They provide digital communication mechanisms such as fieldbuses between distributed controllers and a supervisor computer, thus greatly reducing the amount of wiring, improving data integrity and providing diagnostics features. However, they also need distributed devices capable of logical processing, in which case microcontrollers are commonly used. Microcontrollers can be equipped with powerful components providing capabilities similar to low-end computers. Building a control logic on top of these microcontrollers is similar to creating normal computer applications keeping in mind the real-time control requirements.

Development of control applications will eventually reach a testing phase in a real environment. Traditionally the system needs to be shut down in order to change the control applications. After the change and restart, the new application can be tested. This procedure may be inconveniently slow on some systems, which limits the amount of test cycles and therefore the amount of application combinations. It also slows down the application development process and it may especially prevent doing small improvements or optimizations to the system.

This thesis focuses on finding a solution that would be capable of switching the application on the fly, i.e. when the automation process is running. The idea of the reconfigurable applications raises many questions, which are considered in this thesis and answered if possible. Especially three questions are pointed out as research questions:

- In what kind of embedded systems are application reconfigurations recommended?
- What are the common problems when using application reconfiguration in a control system?
- How long does it take to switch the application in a real-time operating system?

In this thesis, it is expected that the readers have basic knowledge about operating systems, embedded devices and control systems in order to follow the text completely. These assumptions are made to allow focusing on the actual topic and limiting the scope of the thesis.

The thesis is started with an introduction to the concept of the reconfigurable applications. The first chapter discusses available techniques, important requirements for implementations and possible challenges that might come up when doing an implementation follow-

ing an idea of the reconfigurable application. In addition, some existing solutions are introduced and their suitability for control automation are evaluated. After this some performance metrics are introduced and discussed to help further suitability analysis and to be able to compare possible future implementations. The purpose of this chapter is to provide theoretical base for the next steps and to be able to build a usable implementation and avoid possible problematic areas.

In the next chapters, there is more detailed investigation and comparison of suitable technologies. This aims for finding the best available methods for implementing the application reconfiguration and thus getting more valuable results. A solution proposal is introduced based on the theoretical findings. That solution is then tested with various methods in order to get validation of the theoretical assumptions made previously.

In the end, results of the testing are gathered and shown in illustrative figures. Results are analyzed and compared to the expectations based on the theoretical discussion. Finally, all the findings are summarized and discussed in the conclusion chapter.

2. CONCEPTS OF RECONFIGURABLE APPLICATIONS

In this chapter, the application reconfiguration technique is discussed from conceptual and general viewpoints. The chapter starts with a general introduction of the available techniques and will then focus more on topics directly related to the application reconfiguration. At the end of this chapter, suitable metrics are introduced for performance evaluation of the technique. The discussions in this chapter give valuable knowledge of the subject, which is then used in the following chapters.

2.1 Definitions

This thesis contains some terms that are widely used but have slightly different meaning depending on the context or industry field. This chapter defines these ambiguous terms in order to prevent misunderstandings.

Application

Application software is a common term in computing that means a computer program designed to perform certain actions, such as modifying a file or showing an html formatted internet page in a human readable format. Applications are usually independent from other applications but they may also communicate with each other. The applications need an operating system to provide implementation of some operations they use.

In this thesis, an application works as described above but it is separated from operating system processes and threads. A process is an instance of a computer program and it is a basic item for operating systems. A thread is a part of a process and a process may contain multiple threads. The thread is the smallest execution item handled by the scheduler. The application means a component which is able to do a certain logical job, for example maintaining the level of a liquid in a tank by controlling the input and output flow of materials. Depending on the implementation, an application may contain multiple processes, it may contain one process with multiple threads, or it may be a thread in a large process containing multiple applications.

Dynamically reconfigurable

In the development of embedded systems the traditional workflow is that an embedded system is first defined. This includes the task it should be able to handle, running environment, available hardware and many other things. Secondly, the hardware is programmed or configured in a way that it can handle the task. Thirdly, the system is turned

on and the results of the work can be observed. If any modifications are needed, the system needs to be shut down in order to reprogram, reconfigure or attach a new hardware to it.

A dynamically reconfigurable system means that a system can be configured during its runtime. In distributed systems, this can mean a plug-and-play type of system hardware, where it is possible to add hardware modules to an existing system and the system configures itself to utilize the new modules. The term ‘dynamically reconfigurable’ can be used also when a system contains an FPGA or another programmable hardware block and the system reprograms it during its normal behavior in order to execute its tasks. [1][2][3]

This thesis focuses on dynamically reconfigurable software. Embedded devices may contain control applications that decide which output values should be based on inputs, or communication applications that communicate with other hardware modules and monitoring panels, or some other types of applications. Those applications that can be replaced with others during runtime are called in this thesis ‘dynamically reconfigurable applications’.

Control cycle

Controlling in an automation system can be continuous or event based. Continuous control means constantly measuring the process and controlling the actuators. In digital systems, all signals are discretized with both value and time, and the execution of control algorithms is periodic in nature. With very short periods, digital systems can provide seemingly continuous signals, even though it is not truly continuous control.

A control cycle means one of these periods, when control algorithms have been executed. It contains the time used for measuring process inputs and calculation of outputs. A control cycle is usually predefined amount of time and cannot be changed during execution. If the system is able to execute its algorithms faster than a control cycle, the system just waits for a new cycle to start. If the system is not able to execute its algorithms in given time, it causes misbehavior of the system. Constantly missing the execution deadlines indicates that the system does not have enough calculation power to be able to handle the execution of the control algorithms and the control unit may stuck. Occasional misses may also be a critical problem, depending on the real-time requirements of the system.

2.2 Operating system services

Modern operating systems are designed to handle execution of multiple simultaneous computer programs. A computer program instance is called a process, and an operating system can start and stop the execution of those processes one by one. With the help of the operating system, processes can use system resources without interfering each other’s execution. In the reconfigurable application point of view, it is important to know how

the processes are executed and how they can use services like inter-process communication (IPC) and how they can access the memory. Next subchapters discuss more on these services.

2.2.1 Scheduling

All commonly used operating systems have a scheduler. In a normal situation there are multiple processes running in a computer at the same time. The scheduler is responsible for giving turns for processes to use CPU, so the processes can execute their code without needing to know about the existence of other processes.

A scheduler can set a process to running state, i.e. allowing it to use CPU. Then it can wait until the process has completed its task or goes voluntarily to waiting state by doing for example an I/O-operation. This is called cooperative scheduling and in addition to guaranteeing that the process can execute its task effectively without interrupts, it causes vulnerability because one process may be using all the CPU time. Therefore, most modern operating systems use pre-emptible scheduling. Pre-emptible schedulers are able to stop a process from executing if it does not release the CPU voluntarily. This prevents the CPU intensive processes from blocking other processes. With quite short time periods given to a one process, pre-emptible schedulers are able to achieve shorter latencies after events and better fairness of the CPU time between the processes.

Schedulers try to optimize the performance of the system in many ways. They try to maximize the amount of processes that complete their tasks, minimize the response time after an event, giving execution time to processes even though the processes may have different priorities and so on. The importance of these objectives depends on the system and the user's needs. There are many algorithms which are designed to be good when using one specific set of criteria or which are designed to be quite good in most cases.

Real-time environments usually contain a requirement of hard real-time for some tasks in addition to other requirements. In hard real-time, the process should never miss its deadlines or a critical failure may occur. This issue rarely concerns PC users, so desktop schedulers do not handle the issue. In embedded real-time systems it is common to write a custom scheduling algorithm since the processes, system resources and the target are well-known. Control applications are often run periodically and each application needs to be run within that period, which common operating system schedulers do not offer and which justifies creating a custom algorithm.

2.2.2 Memory management

Operating system offers memory protection for preventing malicious processes from harming the system. Mass storages, such as hard disks and flash-drives, have a different

purpose of storing the data than the main memory, i.e. random access memory, RAM. This leads to a different requirement of memory management and protection systems.

Mass storages store data permanently even if the power is switched off. They contain all the data which computer needs for operating, typically in the form of files and folders. In a computer boot-up, a special section of the mass storage containing computer initialization operations is loaded and executed. After that, only needed files are loaded to RAM.

User access rights usually protect mass storage data, so that only the author of the file or the system administrator can access the file by default. Other users may also have access to the file, but the access has to be explicitly granted to the specific user or user group. A file can have multiple different kinds of access rights per user. It can be readable, editable or executable by a certain user or user group. There might be more fine-grained methods or file protection [4], but usually these are enough. [5]

RAM is faster to read and write than mass storages but it cannot hold its data if the power is switched off. Therefore, RAM is used as a run-time storage of process data. The operating system obfuscates the actual memory addresses of processes by giving virtual addresses of memory to them. Processes then use virtual addresses as they would use real ones and the operating system handles the mapping from a virtual address to a real one.

Virtual memory is a very powerful protection against processes misusing the memory of other processes by accident or by attack. However, this prevents processes from communicating with each other via shared RAM addresses. Communication between processes is occasionally needed and therefore operating systems provide special inter-process communication mechanisms, called IPCs. More details on different IPC mechanisms are given in chapter 3.2.3.

2.3 Requirements for reconfigurable applications

Deploying applications at run-time causes some restrictions to them. The applications should have minimal amount of dependencies on each other to allow an easier reconfiguring process. Reconfiguring process may also cause requirements, for example application internal data structures, but these requirements should be minimized.

Technically replacing the whole executable file containing all applications is possible but it is estimated to take a lot more time than replacing only one application at a time. The file containing all applications is bigger and transferring it to the control unit takes more time than a file containing only one application. Selecting only one application from a combined binary cannot be done, so it would require replacing all existing applications with the new one. Successful replacement of all applications takes more time than replacing only one and it has higher probability of causing failure, since consuming more time

generally causes more problems as described in chapter 2.3.1. Thus, the applications should be compiled to different files.

The applications are run-time reconfigurable if:

- they only have pointers and other dependencies on non-reconfigurable system components,
- no other process or thread has pointers or other direct references to them and
- all communication from and to the application is handled with an abstract communication mechanism.

These requirements are due to noting that the application is deleted from the memory and the new application may not be in the same place or it may not contain exactly the same data structures. All applications are not run-time deployable since they may contain some dependencies on others which would break the system if one of the application is removed or even replaced.

For the control software in automation systems, the basic requirement is not to cause any damage to instruments and to keep the system in a wanted, usually stable, state. In practice, the control software should provide meaningful output signals, which actuators can follow and which does not cause the process to lose its desired state. This has to be true also when doing dynamic software reconfiguration.

The most likely use-case for dynamic reconfiguration is that the control software has a misfunctionality that needs to be fixed. This may include small tuning to the control algorithm or adding a logic to handle some exceptional situation. All these are examples of cases where the set point or control algorithm is not drastically changed, so the output value of the software should also not change. In reconfiguration operation, it needs to be taken care of not to cause any transients.

Considering the generality of the reconfiguration method, it should not be restricted to only small changes. The user may want to add a completely new application to work with the existing ones, completely switch the control algorithm or delete an existing but unnecessary application. Such a big change could affect the output signal from the application and it should be taken care of so that the process stays in its limits.

2.3.1 Time used for reconfiguring an application

It is obvious that reconfiguring the system takes some time and the less is better. That is not a clearly defined statement and more valuable information can be gained by defining what is fast enough for the purpose and then measuring how long the application reconfiguration process takes.

If the control of the system is cyclic, the application reconfiguration should not take more time than there is available after each cycle, as illustrated in Figure 1. This requirement prevents the application reconfiguring process interfering the timings of the system.

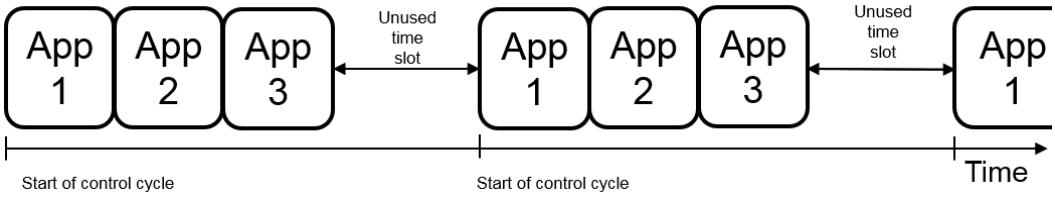


Figure 1. *The execution order and timings when the applications have the same control cycle*

Measuring the amount of unused time may not be an easy task. Operating systems usually provide the percentage of free CPU time and with the knowledge of the control cycle period, the unused time can be calculated on simple systems. However, the application execution time may vary if the real system is in a dynamic state and the application needs to perform different calculations from the last time.

Complex systems may also have more than one control cycle. This is true when the applications are handling tasks with different reaction times and usually different priorities. For example an application in a car handling traction control system needs to have a very short execution period and a high priority level but an application responsible for updating the average amount of fuel consumption does not have to be frequently executed and its priority can be lower.

Multiple time periods will decrease the amount of unused time as shown in Figure 2. It also complicates the estimation of how much unused time there is between the control cycles and more importantly it complicates the decision when the application reconfiguration should be executed.

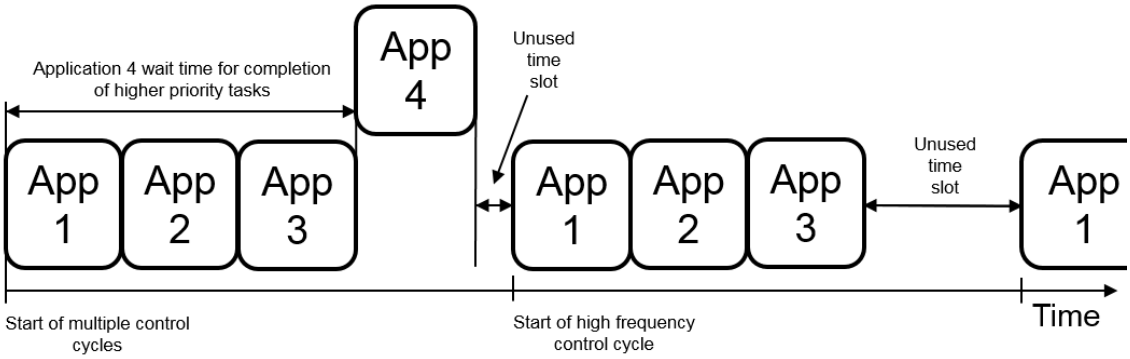


Figure 2. *The execution order and timings when the applications have different control cycles*

There are two approaches to solve this problem. Either the reconfiguration process is done fast enough to be able to be executed in the shortest time the system theoretically has or

the reconfiguration process is able to estimate when there will be a long enough time slot for it.

The application reconfiguration process does not actually need to be executed in the same priority as the applications. Depending on the implementation, it may run as a background task and fetch a new application, load it to the memory, execute possible initialization functions and so on. After everything is ready, the switching process just unregisters the previous task from the periodical scheduler and registers a new one. Again, depending on the implementation this may be only for switching a value of pointer to point to the new application function. Then after the critical part is done, the application reconfiguration task may continue to run on the background, free the memory of the old application and eventually delete it from the mass memory. This approach should be fast enough for any situation. Drawback is that the application needs to be completely independent from the previous one and the initialization functions cannot interfere with the process by for example setting the process related variables to some default value.

The application reconfiguration process may know the periods of control cycles. If it can detect also when a period is starting, it can calculate the time most suitable for the application reconfiguration. In the easiest case, the periods are multiplies of the shortest one, for example 10 ms, 20 ms, 50 ms and 100 ms. This means that there will be a time slot between two 10 ms cycles that does not contain applications from any other periods. In the easiest case all applications start at the same time, so after every 100 ms, all applications will be running within the control cycle, but it is known that in the next cycle there will be only applications with a 10 ms period. That would reduce the logic for concentrating only to monitor the highest period tasks.

2.3.2 Application data structures

Applications usually use some kind of data structures in order to calculate their output values. A data structure can be a common database for all applications, the applications may contain private data areas, or the data structure can be combination of both.

Applications may behave like functions in the functional programming paradigm meaning that with the same inputs they always produce the same output. In this case, the inputs may be read from sensors or from a common database and if the applications have data structures of their own, they never change them during runtime. This is the easiest case in the application reconfiguration point of view, since replacing such an application does not need data synchronization.

The applications that need data structures are not rare. For example, a basic PID controller needs to know the history data of the process in order to calculate its integral part. The history data may be stored into the system so the application does not need to store it, but then the system needs to know which kind of data it needs to store. That may easily lead

to a solution where system stores all possible data in order to serve all applications with unknown needs. Besides, storing an inconveniently large amount of data may slow the system when the applications need to fetch all information from the system in each control cycle.

This leads to a situation where the internal data structures of applications are not always avoidable. The data needs to be initialized in order to get the application working as it should, immediately after it has been replaced with the old one. The dangerous part is that the uninitialized data may cause peaks in the output even if the application logic is correct. On some applications, all data may be initialized with default values but that cannot be generalized to all applications. Even different states of the system may affect the default values that should be used. Therefore, some kind of application data structure synchronization is needed.

It would be wise to collect all possible data structures to one place in the memory and then the application can access that data via a pointer. It allows the possible new application to take that same data structure, which may contain valuable history data, into use by only copying one value of a pointer. This assumes that the new application needs the same data and it is able to use that kind of structuring. The applications may contain a data structure version number, which the reconfiguration system can then compare to the version number of the new application's data structure. The reconfiguration system can then decide if it can pass the data to the new application or if the new application needs to run its initialization functions.

Utilizing the data from the old application solves the data initialization problems since the data is already collected and valid. The old data structure can be used when the new application is similar to the old one and contains only small changes, for example bug fixes and optimizations. However, the old data structures cannot be utilized in every case, for example if the stored data was not enough and there is addition to that data structure, or if the new application is totally different than the old one.

Generally, the new application that needs a history data needs to collect it by itself. One possible approach would be let the applications run in parallel during the initialization of the new application. This way the new application can store the data it needs and not switching the application before all data is collected. Assumption is that the initialization function can be run without interfering the other system, for example by using the same data area as the old application, and that the new application does not need an inconveniently long history data.

2.3.3 Application output

When changing the applications at runtime, the output may also change drastically in a short period of time causing transient. This may be harmful to the automation system and too rapid changes should be avoided.

The software system, if supporting such a feature, may run an old and a new application in parallel. The control signal is taken from the output of the old application. The output of the new application is then compared to the old one and if difference of those is small enough, the new application may be taken into use. Restricting the outputs to be the same is not generally a good solution, as then it would question the whole principle of changing the application if the outputs will be nevertheless the same.

Running both the old and the new system would also solve the problem of a lacking history data in the new application, since it can collect it while running. In this case, the new application has a risk for drifting to the maximum or the minimum. A PID controller is a good example. If the new PID controller has a different set point than the previous application, the integral part starts to accumulate the error value, i.e. the difference between the set point and the process output. This causes constant increase or decrease of the application output, eventually reaching the maximum or minimum value as shown in Figure 3. That will cause a big transient in the output, if the application is then taken into use.

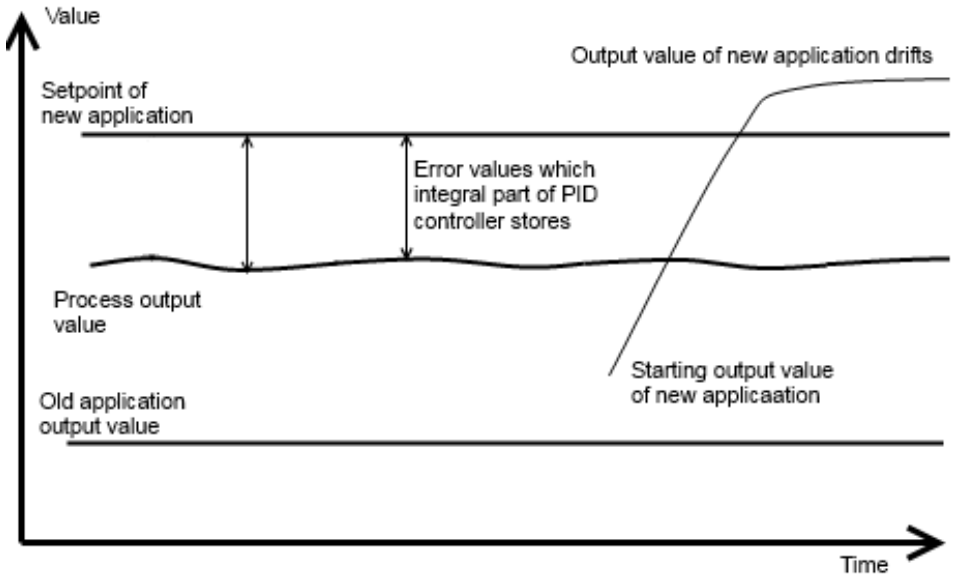


Figure 3. *The principle of the drifting application output value*

Depending on the start point of the application output, it might drift to the area accepted by the main program. If the starting point is too low, the application output will drift to the lower limit, but it is still lower than the old application. Changing the application on that time will cause transient to the system as illustrated in Figure 4. An easy solution to prevent this kind of transient is to put the lower limit to be the same as the old application

output. However, it does not fix the problem, since the set point of the new application can be lower and cause the same problem. Setting both limits to be same as the old application output can solve drifting issue but then there are problems with other types of applications, which do not collect the history data of the error value and therefore their output will never reach the accepted area.

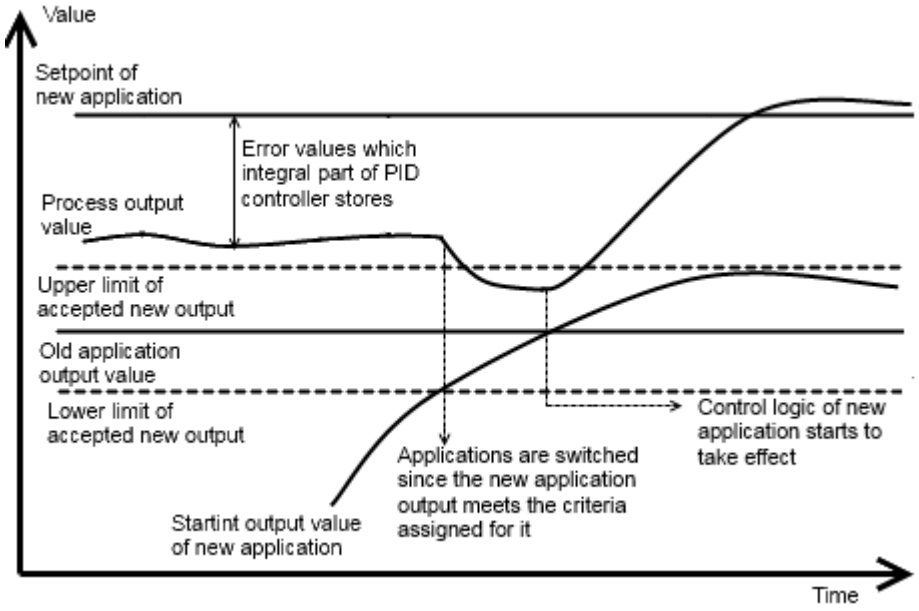


Figure 4. *Transient when changing applications*

2.4 Challenges

As mentioned in previous subchapters, there are challenges to implement the application reconfiguration to work in a generic case. There are also other kinds of challenges and drawbacks that reconfiguration technique contains. This chapter collects those challenges in order to make conclusions.

The time used for the application reconfiguration can be measured and the time available in the system can be estimated but then it is also essential to synchronize the changing operation to other events in the system. Some implementations may allow application switching during the execution of another applications but switching the application currently being executed will more likely cause troubles.

Taking a new application into use is a risk as itself. The new application may fix problems found on the old application but it also might introduce new ones, which then might cause more critical problems than the fixed ones were causing. In addition, the reconfiguration should be synchronized with events in a larger scale in the process. For example, if a power plant is increasing the electricity it is providing in response to the growing need, the normal control values of the system are in constant change. The risk is that even a good algorithm may not necessarily guarantee working process control for the system in

changing state. This can be avoided by triggering the application reconfiguration using the process operator input. The operator can then wait that the system is in a stable state before switching the applications. After the trigger, the actual application switching can be done when the software sees a suitable time slot.

Another drawback is memory handling. Since applications need to be built separately, it is estimated that they use more mass memory. That is because dynamically loadable libraries need to have header information for dynamic loading and other purposes. Also after doing multiple application reconfigurations, both the mass storage and RAM can get fragmented.

In addition, security issues need to be taken care of. Integrating process controllers to manufacturing execution systems (MES) and to enterprise resource planning (ERP) have been increased and new Internet of things (IoT) devices are constantly published. These phenomena cause process controllers to have a more likely access to the Internet, either directly or indirectly. It alone needs a great care with security issues but it is especially important in systems that use the application reconfiguration. Replacing the executable code will give the attacker an unlimited amount of ways to cause harm to the automation system. This topic is not discussed more on this thesis, since the security should be taken care of in the system also without using reconfiguration technique.

Safety issues are another thing to take into an account. The application reconfiguration operation has an impact on the system behavior and triggering it should be protected as well as possible. The main program should not be able to trigger the reconfiguration process by itself since it cannot know about the “big picture” of the system and automatic triggering will obfuscate what is happening in the system. Users also should not be able to accidentally replace essential applications with unsuitable ones. Changing essential applications can be allowed but the new application must always handle the same job. The applications may be categorized and numbered so that for example category one contains control applications, category two contains communication applications, category three contains safety related applications and so on. The applications may contain this category information and when doing the application reconfiguration, the system may check that both the old and the new application share the same category. This will decrease the possibility to change the wrong applications accidentally, which may lead the system unusable and can possibly causes damage in the real world.

2.5 Existing solutions

The reconfigurable applications are not a new thing but it is not a common practice in embedded systems either. In this chapter, previous studies on the subject are summarized.

David Stewart and his team have investigated creating a dynamically reconfigurable real-time software paradigm in order to achieve higher software development rate in embedded systems. Supporting their target, they have developed a new real-time operating system, called Chimera. Chimera has many good features like automatically integrating reconfigurable software modules, but unfortunately it runs in a very limited amount of processors. The project group focused on embedded software used in simple robots and noticed many similar advantages than in this thesis, like faster fine-tuning of an application. Their material is interesting but unfortunately outdated and the Chimera project is officially defuncted. [6][7][8]

Park et al. have built a communication middleware for home electronics. It supports plug-and-play type of adding new devices and it automatically searches the needed drivers from the Internet. This solution is both real-time and dynamically reconfigurable but it is not essential for this thesis as it lacks the hard real-time requirement. [9]

Dynamic reconfiguration is also widely used in other types of systems. Fleischmann and Otero have separately developed a system containing dynamically reconfigurable FPGA with their groups [10][11]. While this is an interesting idea, it does not share the research scope with this thesis.

2.6 Metrics

It is important to measure the performance of the created solution so it is possible to estimate if it is plausible to use in any real use case. It also reveals restrictions for using this technique if there are any. For such purpose, suitable metrics need to be defined and those are discussed more thoroughly in the next subchapters.

2.6.1 Time for application reconfiguration

Time is a key element in all control systems. However, measuring time related parameters in a pre-emptive multitasking operating system is not a trivial. Many methods will measure slightly different things and may therefore give slightly different results. It depends on the system within which measurement method is considered as the most important one. Usually it is wise to use all methods for measuring the functionality and afterwards decide if the measured technique is suitable for the used system.

In a system that contains a pre-emptive operating system, there are two fundamentally different methods to measure the time used by an application: wall clock time and CPU time. Wall clock time or real-world time means the time measured by using a wall clock or other chronometer. It measures the difference in time between starting and finishing the task at hand. In contrast, the CPU time means only the time when the task uses CPU. It does not include the time waiting for resources such as file I/O operation nor the time when another application is run. The amount of wall clock time is therefore the greater or

equal than the amount of CPU time in single core processors. On multicore machines, it is possible for a task to run in many cores simultaneously and it is therefore possible for a task to have a bigger CPU time than the wall clock time. [12]

In Linux, there is a system-wide real-time clock, monotonically increasing clock, process-specific clock and thread-specific clock [13]. Two first options are analogous as wall clocks, since they normally increase their time at the same speed as usual chronometers. A process-specific clock is created for every process and it measures the used CPU time for this process, counting all possible threads. A thread-specific clock is created for every thread and it measures the used CPU time for the thread.

In the reconfigurable application point of view, the wall clock time is more interesting since it measures how long the reconfiguration process totally takes time and it can be compared to the time available in the system. The measured CPU time can be used for estimating how CPU intensive the process is.

2.6.2 Memory consumption

Embedded devices have fewer resources than desktop computers, including less memory. A system may already use most of the available memory, which also reduces the amount of the free memory. Therefore, it is essential to measure how much memory the application reconfiguration needs.

Computers have usually two type of memory devices: a run-time memory, such as a RAM and a permanent mass storage, such as a hard disk drive or a flash chip. Typically, the mass storage content is not changed during the run-time of an embedded device, making it easier to measure. The file system of an operating system usually provides direct methods to get the used space of a file. This can be used if the applications have been built to separate libraries or otherwise the measured files do not contain any other software components. If the applications and the other components are in a single file, their memory usage can be separated by investigating the generated machine instructions, but it is not a practical solution.

Measuring the used RAM can be done programmatically for example using a *malloc_stats* system call in the Linux [14]. It prints statistics about the memory allocated by common a *malloc* system call. This gives a good estimate of the memory usage by the applications but it is not generally suitable since the processes can reserve memory by other methods as well. At least the linker uses a *mmap* system call when it needs to link a shared object loading procedure into the process [15].

Operating systems keeps track on the RAM usage per process but measuring it has the same problems than measuring the usage of mass storage, the process may contain other

components than the actual applications. The operating systems may give more detailed information from the RAM usage but it depends on the operating system.

The operating systems usually split the memory to constant sized pages. The memory pages are used especially when the virtual memory technique is used. The virtual memory means that the processes have addresses to the virtual memory, which is then translated to the real memory addresses by the operating system. The virtual memory handling improves the memory management of the operating systems but it also discretizes the RAM measurement to be a multiply of the page size. The page size can vary between systems.

In the reconfigurable application point of view, it is interesting to measure the difference of the RAM and the mass storage usage between the existing solution and the new technique. Also, the possible change in the used memory after the application reconfiguration process has been done, should be measured. All measurements should include only the memory used by the applications, if possible.

2.6.3 Switching to non-real-time mode

Control applications usually control real-world processes and therefore they should never miss their deadlines. It means that the underlying system needs to provide support for real-time software processes. The system also needs to provide non-real-time operations, such as reading a file. This leads to a situation where the system has a real-time and a non-real-time mode. A process can be in either of these modes, but not simultaneously. The process can switch between the modes depending on the operations it makes and the configuration of the system.

Dual kernel architectures, which contain a separate real-time operating system and a non-real-time operating system, usually define real-time operations to be system commands provided by the real-time operating system. This means that the system provides real-time guarantees to a process if it only uses the real-time operating system commands. It is usually not a problem in control applications as they only read input values, calculate the control variables and then send the result onwards. However, the application reconfiguration process needs more functionalities such as access to the file system in order to load the new applications into the memory. Loading the files to the memory needs to use a mass storage device. The mass storages have unspecified response latencies depending on the usage and therefore those operations are estimated be non-real-time operations.

The processes in the real-time mode have always higher priority in order to guarantee their ability to meet their deadlines. Thus if the application reconfiguration process is switched away from the real-time mode, its priority drops lower than any of the control applications. This may cause synchronization problems when switching applications, as the switch should be done when the application is not running. Therefore, it is important

to measure if the application reconfiguration process is switched to the non-real-time mode.

3. COMPARING APPLICABLE TECHNOLOGIES AND TECHNIQUES

The previous chapter contained theoretical discussion of the application reconfiguration related topics and provided basis for next phases. Before starting to implement a proof of concept solution, it is advisable to do further evaluation of technologies and techniques to find the most suitable ones. In the following subchapters, the most important parts of the system are discussed.

3.1 Operating systems

When deciding if the reconfigurable application technique can be taken into use in a system, it is necessary to check if the target system meets all requirements. Embedded systems without an operating system are left out as target systems since they are usually quite simple and their hardware is not powerful enough to meaningfully use the application reconfiguration technique. Embedded systems with an operating system, however, are usually more complicated, their hardware is more advanced and the programs running in them can be more isolated from each other to ease the use of reconfiguration technique. Operating systems also provide ready-made necessary functionalities, such as a file system and dynamic library loading.

The discussion here is limited to Linux only because it is commonly used in embedded systems (see Figure 5). According to the UMB Electronics survey, 27 % of embedded developers used Linux in their embedded projects in 2013 and 26 % was considering using it in the next year. Figure 6 shows that more than a half of the developers use embedded Linux because of its adaptability and extensibility, which are very important features also when developing the application reconfiguration technique. If any unsupported features are needed during the implementation phase, it is possible to add them. [16]

Two different Linux variants were chosen for a more detailed comparison in order to discover their characteristics and their feasibility for the application reconfiguration usage. Both of the selected systems are targeted for embedded usage. Linux with a RT_PREEMPT patch provides a pre-emptible kernel with all normal Linux functionalities. Xenomai has a different approach for providing real-time and it implements dual-kernel methodology with both a real-time kernel and Linux.

Figure 5. *"Are you considering using embedded Linux?" by UBM Electronics survey [16]*

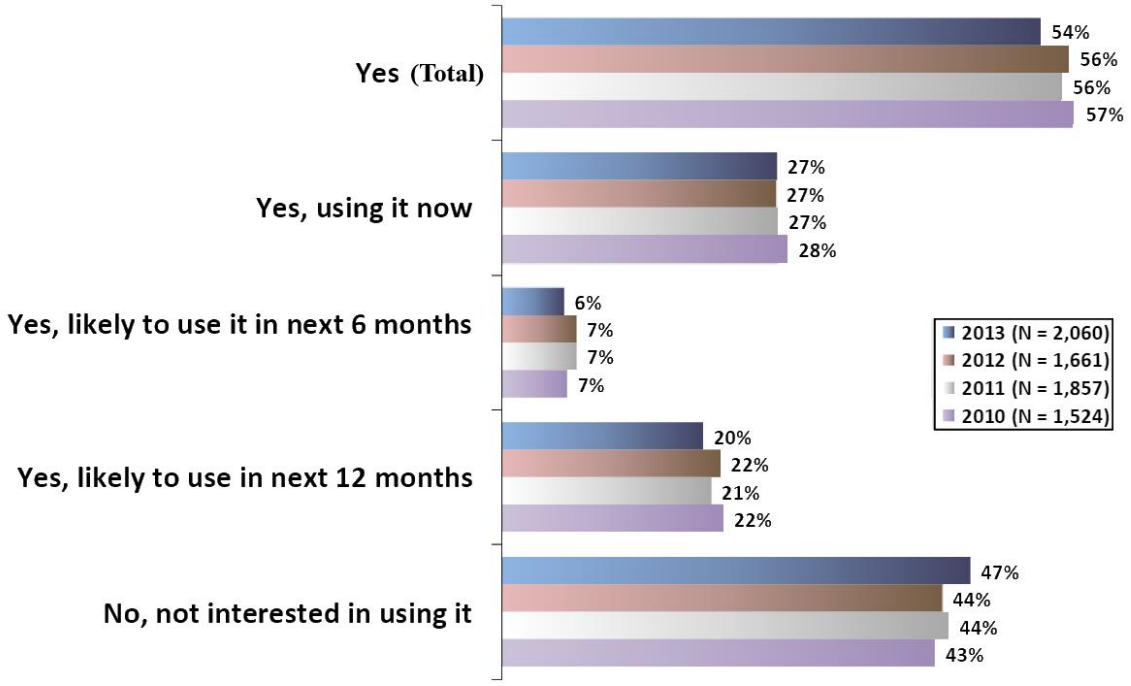
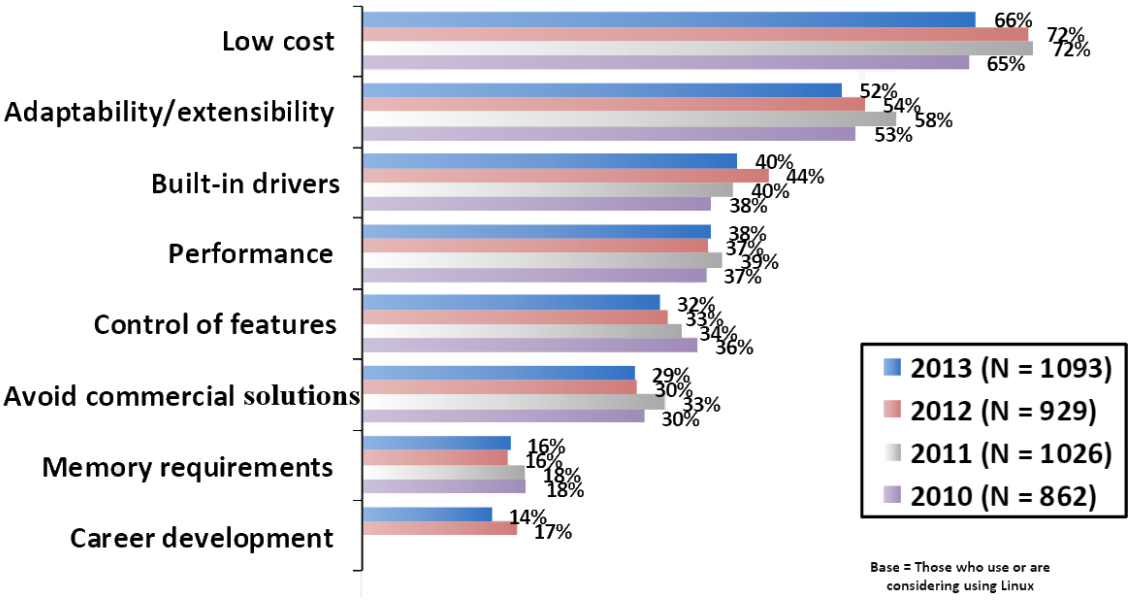


Figure 6. *"Why are you using embedded Linux?" by UBM Electronics survey [16]*



3.1.1 Linux with RT_PREEMPT patch

RT_PREEMPT patch is a development branch aiming to enhance the real-time characteristics of Linux by making the kernel completely pre-emptive. Ultimately, this patch is planned to be merged with the mainline kernel code, since these modifications are seen as welcome also for regular Linux use-cases. [17]

A regular Linux kernel code is mostly pre-emptive already, but not completely. Interrupts are handled normally right after the interrupt occurs even though there can be a high priority task running. Effectively this means that all interrupt handlers are on the highest priority even when the interrupt is related to a very low priority process. This causes Priority inversion -problem [18][19]. In addition, uninterruptible busy-waits, spinlocks, are very problematic in real-time systems since they can delay timed processes to miss their deadlines.

RT_PREEMPT patch tries to convert the whole kernel to be pre-emptive. Interrupt handlers are run in an operating system thread, so they can be interrupted and scheduled normally. This also gives a possibility to give priority for interrupts and so prevents low-priority interrupts for interrupting higher priority tasks. Spinlocks are replaced with mutexes, which makes the processes using them able to sleep without actively using valuable processor time. This makes them interruptible as well. RT_PREEMPT patch uses also higher precision clocks than the regular Linux with time intervals of 10 ms. This greatly improves the accuracy of the scheduler. [20]

Even though RT_PREEMPT improves the real-time behavior of Linux and it is planned to take it to the mainline development, it has its weaknesses. Running interrupt handlers in a kernel thread causes more work to the scheduler which then increases average response latency for an interrupt. However, in a real-time system point of view, it is more important to reduce worst-case interrupt response latency, which RT_PREEMPT patch succeeds to do [21].

Moreover, the RT_PREEMPT patch has not yet been able to modify all interrupt handlers as interruptible due to the nature of their task. Rewriting these interrupt handlers seems necessary. In addition, developing new Linux drivers is more complicated when real-time characteristics have to be taken into an account.

Other challenges that real-time causes are that used C libraries and operating system interfaces are not designed for real-time use. In addition, virtual memory management takes an unpredictable amount of time.

3.1.2 Xenomai

The Xenomai project tries to solve the real-time problems from a different angle than RT_PREEMPT patch. The point is to make different real-time operating system interfaces to be usable in a Linux based system. Xenomai does not even try to fix problems with real-time in Linux but instead it supports a dual kernel. This means that the system has actually two different operating systems, one real-time kernel and one regular Linux kernel. The Linux is run only as a low priority process in the real-time kernel, thus allowing real-time to be handled accordingly by the real-time kernel.

Along with the dual kernel, the Xenomai has the Adeos/I-pipe virtualization layer. The Adeos is a hardware virtualization layer, which allows execution of multiple operating systems in one or more processors. Operating systems get their own domain and they do not need to know that the other operating system or even the Adeos itself exists. The Adeos tries to minimize the effect of its existence in normal execution but it has to handle interrupt signals that peripheral devices send, to correct the processors. The Adeos does this using an interrupt pipe, I-pipe. Since different operating system domains can be in different priorities, I-pipe supports the domain prioritizing. In this case, all interrupts go first to the highest priority operating system and only if it is not interested in the interrupt, it goes to the next one. [22][23]

Therefore, with the Adeos it is possible to create systems where the real-time operating system is used for all time-critical processes and a normal Linux for all the rest. The used Linux version is recommended to have the RT_PREEMPT patch for example to get faster context switches. Nevertheless, with this dual kernel architecture the Xenomai has created an environment that seems to have better real-time characteristics than the RT_PREEMPT patch alone. [24]

3.1.3 Comparison of selected operating systems

Comparison of operating systems is quite a complicated task since the requirements of the embedded systems are highly dependent on the case where it is used. Different requirements mean that one system may need to have low memory usage but it might be irrelevant on some other system. In automation systems, however, maximum real-time response latency is one of the most important features and therefore the introduced operating systems are compared based on that.

The real-time response rate was tested by Haapaluoma to compare operating systems described in previous chapters, Linux with RT_PREEMPT patch in chapter 3.1.1 and Xenomai in chapter 3.1.2. Both were tested when the system was idling and under some load. The idle test was done to eliminate all other variables from affecting the results. Load test was done to get latencies that are more realistic in normal operation. The load was generated specifically for testing purposes and it was the same for both operating systems. By measuring both the idle and the under load –latencies it can also be seen how much the load affects the measured latencies. [24]

The testing setup contained the Xilinx Zynq 7000 system on a chip -board including a dual-core processor with clock frequency of 667 MHz. The first core contained the operating system under test and the second one contained a FreeRTOS operating system for time measuring purposes as shown in Figure 7. The test process was sleeping and waiting for an interrupt. The interrupt was generated externally from timer overflow, which also zeroed the timer value. The interrupt handler (ISR), which signals the test process to wake up and save the timer value, caught the interrupt. Then the test process fired a software

interrupt to notify the FreeRTOS. The FreeRTOS also caught the interrupt by the ISR and stored the timer values.

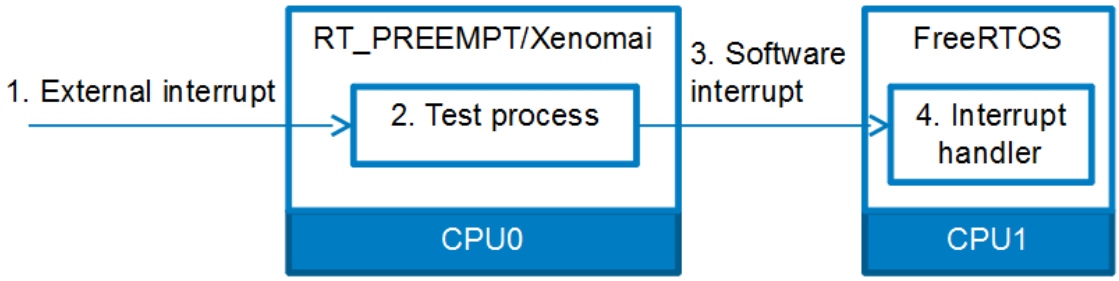


Figure 7. *Test structure in operating system latency tests [24]*

The results are shown in Table 1 and the distribution of the results in Figure 8. The ext to CPU0 rows are the most important ones, since they measure the reaction time for interrupt in the operating system under test. Interestingly the minimum latencies on the RT_PREEMPT patch were smaller on load. This is probably caused by varying latency depending on how much work the computer is currently having and how soon it jumps to the interrupt handler. The probability of higher latency is increased when operating system is under load but the minimum execution path to the interrupt handler still stays the same. For bigger amount of measurement, it is expected to eventually have smaller measured minimum latency when the system is idling.

In real-time systems, the maximum latency is the most interesting result. By measuring the maximum latency, it is possible to estimate how well the system can react to events and how well its processes can meet their deadlines in the worst case. The results show clearly that the Xenomai has better real-time response latencies than the Linux with the RT_PREEMPT patch.

Table 1. *Latency results of RT_PREEMPT patched Linux and Xenomai when idling and under some load [24]*

RT_PREEMPT IDLE

Latency	Min (us)	Max (us)	Median (us)
Ext to CPU0	22.36	54.76	28.30
CPU0 to CPU1	0.22	22.21	0.40
Ext to CPU1	22.75	55.15	28.69

Xenomai IDLE

Latency	Min (us)	Max (us)	Median (us)
Ext to CPU0	5.22	18.22	5.47
CPU0 to CPU1	0.36	0.94	0.40
Ext to CPU1	5.62	19.15	5.87

RT_PREEMPT LOAD

Latency	Min (us)	Max (us)	Median (us)
Ext to CPU0	21.67	89.93	37.33
CPU0 to CPU1	0.36	36.79	0.47
Ext to CPU1	22.07	90.43	37.83

Xenomai LOAD

Latency	Min (us)	Max (us)	Median (us)
Ext to CPU0	6.37	36.58	20.23
CPU0 to CPU1	0.36	2.63	0.61
Ext to CPU1	6.80	37.40	20.95

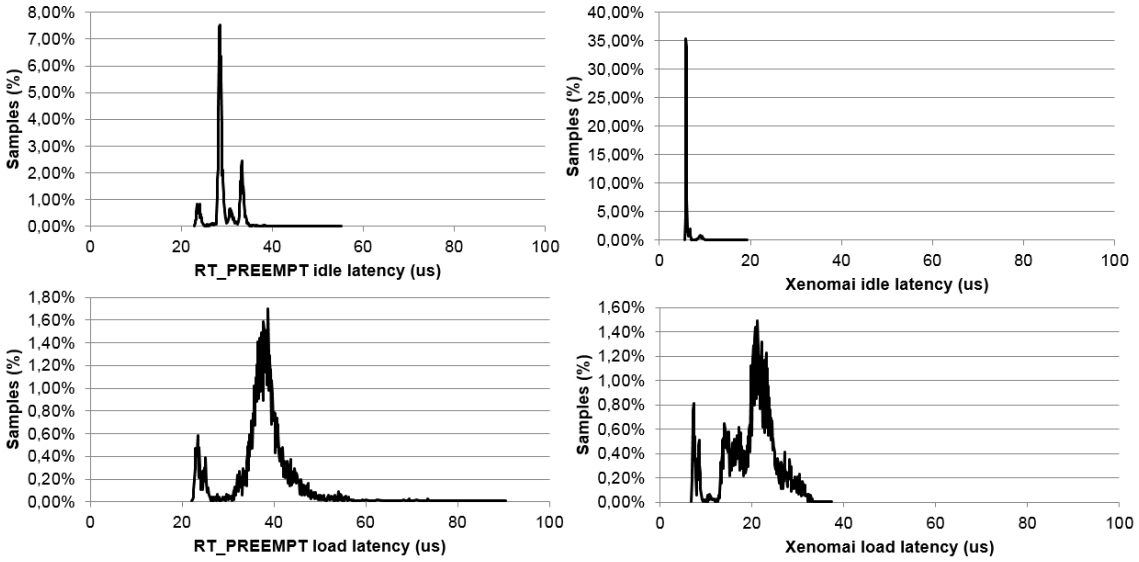


Figure 8. *Distibution graphs of latency results of RT_PREEMPT patched Linux and Xenomai when idling and under some load. Note different scaling on vertical axis [24]*

3.2 Communication mechanisms between applications

In an embedded environment, applications often need to communicate with each other to achieve the best performance when controlling an automation system. For example in process industry one may have a tank full of liquid, an application handling the surface level of the liquid and another application trying to keep the concentration of the ingredients stable. It would be useful to build a system with a concentration application handling the inputs of the tank, so it can estimate the correct blending of ingredients. However, the concentration application would need an input from the surface level application in order to decide when and how much it should increase the amount of liquid in the tank.

In distributed systems, applications that are in separate hardware modules may communicate with fieldbuses or similar techniques. Those applications can also run in the same processor and then some other communication techniques are used. Next subchapters introduce a few inter-application communication mechanisms within a hardware module and evaluate how those can be used with the application reconfiguration techniques.

3.2.1 Direct linking

Applications can be compiled together to allow them to use each other's functions and variables. Usually this means compiling different applications into static libraries and then, in the linking phase, combining those to a single executable file. One file is easier to handle e.g. when transferring the software to the hardware modules than if the software is split into multiple library files. [25]

Having all applications in a single file is not very suitable from an application reconfiguration point of view. Disabling only one application in a bigger file may be possible but the application reconfiguration technique needs to replace the application code. In theory, it is possible to modify or replace a code block within an executable file but it is inconveniently difficult to do so. Firstly, the new application needs to be the same size or smaller to fit to the code area of the old application. Then internal references to that code need to be updated in several locations within the code file or the new code needs to have similar functionality in the same memory addresses. This would greatly limit the variations of functionalities that the new application can offer.

The application switch does not need to be limited to only one application, instead the whole control executable can be replaced at once. This would avoid the new application size and memory address problems, since dependencies are solved during the compile time. Stopping all applications in the reconfiguration process and starting them again after the executable has been replaced will take more time than restarting only one application. It also has a bigger probability of failures during the application initialization, which may cause severe problems when the system is operating.

Replacing the whole application executable can be done in a normal system update, when the process is in a stopped state. However, replacing the whole executable is not very suitable for the application switching in run-time, because of the speed and reliability issues mentioned above.

3.2.2 Common database

The main program may abstract the hardware from the applications by for example storing the value of a sensor and returning that stored value instead of reading it every time when the applications ask for it. The applications have to somehow identify the data they need when asking for it from the system. This can be extended so that the applications can also store the data with an identifier not mapped to any hardware component and any application can then read the value. This arrangement represents the common database for the applications and it allows a simple value transfer between the applications.

The common database provides an application independent solution for data sharing and the number of applications using it is irrelevant, as long as the concurrency issue is handled accordingly. If the database contains a slot for every possible identifier, it will easily grow unnecessarily large. Thus creating a new entry for the database on demand is a good idea. However, the system may not know if an identifier is still in use, and multiple reconfiguration operations may lead some identifiers unused which effectively causes memory leak. Deleting unused entries from the common database can be done for example if the system counts how many applications are using an entry and the deinitialization function lowers that reference counter. Then if the reference counter reaches zero, the system can safely remove that entry.

While the common database can be a flexible way of data sharing and it decreases the communication dependencies between applications, applications may contain logical dependencies. This means that an application may read an entry assuming that it is updated by another application. After the updating application is replaced, the new application may not update the same entry again, so the reading application will not work correctly. These kinds of logical dependencies are impossible to resolve automatically, but manually added dependency flags can be used to indicate dependencies between files. These flags can then be used for automatic check of illegal dependencies.

3.2.3 Inter-process communications

Control applications can be built separately and run in separate processes instead of running all in one process. That will cause the operating system to take care of the scheduling of the applications and prevent applications from using their internal memories. Communication between applications needs to be done with inter-process communication (IPC) methods provided by the operating system.

Operating systems provide many different IPC mechanisms with different characteristics. If applications only need to synchronize themselves, they can use semaphores. Signals can be used for calling functions or asking other commands to be done. For data transfer purposes sockets, message queues or pipes can be used. Some IPC mechanisms, like pipes, can be used only for two-way communication between processes, but others can have multiple senders and receivers.

The advantage of using IPC methods is their flexibility to adjust to virtually any case, if the correct IPC method is selected. However, they have requirements for the application processes for being able to use the selected IPC. In a larger pool of applications, they may use different IPC methods for different purposes and this leads to a situation where applications need to support multiple IPC mechanisms just in case some other application will use that communication method after the next reconfiguration operation.

Shared memory can be used for inter-process communication. Communicating via it means that the applications running on different processes may access the same physical memory addresses. Using the same memory addresses with multiple processes is normally prevented by the operating system to prevent a badly behaving process from damaging other processes. If all the applications are run in a single process, then they already have access to the memory of the others and the shared memory method is not needed.

As said before, the operating system prevents processes illegally accessing memory of each other but it also provides a way to create separate shared memory areas for inter-process use. This ensures that the used memory area is free and will not corrupt other processes. However, since the memory is shared between processes, the structure of memory needs to be known in all of them.

Shared memory is a good way of sharing data but it prevents processes from accessing functions of others. Even though a process is allowed to take a pointer to the beginning of its function and store it to the shared memory, the operating system prevents other processes calling the function via the pointer since the location of the function is not inside the legal memory area of the calling process. In addition, if the operating system uses virtual memory, all processes commonly think they have the same address space e.g. starting from zero. Virtual memory handling will then calculate the real memory addresses from the virtual address process-wise and this will effectively prevent processes to access memory of each others.

3.3 Deciding when to change the application

The application reconfiguration can be done in a few variations. After a new application has been transferred to the hardware module, the application reconfiguration can be triggered by user input or the system can monitor if it has a new application file. The actual reconfiguring may be done right after the system is able to do it; the system may synchronize it to be done after completing a control cycle; or the system may be driven to some kind of safe state to avoid problems.

3.3.1 Change triggered by user input

A basic version of determining the correct change time is getting the change command by user input. User input can be thought open-mindedly in this case, it may be a command-line command to the system, creating a flag-file to a certain location that systems observe or a similar operation.

Command-line commands require some knowledge of the running applications. Operating systems support the command-line and the commands may be given remotely via SSH-connection for example if the system has Ethernet connection. In embedded systems, however, remote connections may be restricted to improving security or cannot be made due to the communication media used. It would also need the main program to handle command-line commands, which drops the handling process to user-space, if it was already running in kernel space.

The flag-file solution requires a command-line as well. It means that the user creates a file with a certain name or location, which contains the required information about the chance either in its name or content. This may include the location of the new application in the file system, the new application function name in a file, the name of the application to be replaced and other information, too, if necessary. The main program then checks if the file exists. Checking can be done periodically and sleep the thread in between checks, so it will not disturb the other applications. It is possible to synchronize the checking task so it will be done after all other tasks and therefore it has a lower probability to interfere with the other applications, if the changing happens.

3.3.2 Automated change

A more sophisticated version of changing the application is determining the need for it automatically. This does not mean analyzing the behavior of the system and determining erroneous behavior since that is nearly impossible to do to any embedded control system with its hardware. An automated change means that it detects automatically when there are newer applications available and when it is the best time to take them into use. The newer applications may be transferred to a pre-defined folder with some unspecified method. The system then observes the changes in the folder and triggers the changing process whenever there are changes.

The automated changing procedure should also determine when to actually change the applications. It may run some initialization checks first as described previously. After it decides that no errors have happened, it needs to actually make the change. If the applications running on the system are triggered by events or interrupts, it is hard to estimate the best time for changing to happen. In this case it may check that there are no processes waiting for processor time and it can execute the changing process when the processor is idle. If the applications are triggered periodically, then the safest time would be right after the applications have finished, when there is most time to execute the changing procedure before the next cycle.

Some embedded systems may have periodically executed applications, but not all applications are executed at the same period. For example some high precision control application may be executed for every 10 ms, an application responsible for communication may be executed for every 100 ms and some system diagnostic application may be executed for every 1 000 ms. In this case there is no constant amount of free time after all application cycles. It is practically impossible to find out a generically good solution for all implementations and therefore each implementation needs a solution suitable for its environment. One possibility is to check when the application with the longest period is executed and then executing the change on the next application cycle. The idea of this solution is that the most of the free time will be then, but that may not be true in all systems. Another possibility is to check when the application that is needed to be replaced is run and execute the change after it. This will cause minimal risk of misbehavior from the changing task, but it may interfere with other applications.

3.3.3 Safe state

If changing the application during normal runtime is considered too risky for example if the embedded system controls some expensive or dangerous machinery, the system may be driven to a safe state before changing the application. The safest choice would be to completely shut down the system but most of the time savings of the reconfigurable applications are lost. Target is to get the system safe for changing but still able to operate.

One possibility is to stop all applications and keep the output signal the same. This may be suitable if the controlled process is continuous or slow but it may not be suitable for batch processes or manufacturing industry. Presumption is that in slow continuous processes, the output values of the control applications are within quite a small range so the short break of updating outputs will not cause much harm and the possible error is easily fixable after the applications are working again. Batch processes have recipes that they follow and those may contain quicker changes of process states like adding a new ingredient. If the application that monitors the amount of a new ingredient is stopped just before it should stop putting more, the system ends up in a situation where it has one ingredient more than it should. Similar logic can be applied to manufacturing industry, which often has machinery with complex execution instructions. However, these statements are only rough generalizations and when deciding if the safe state is taken into use, the characteristics of the system need to be investigated individually.

Applications may have a certain execution order. This needs to be taken into account and when restarting the system, the order should be kept the same as it would be without stopping the system.

Another possibility of creating a safe state is to disable all non-essential applications. This may for example include information services. The system may still run all essential control applications and maintain safe operation of the process. This may help to find a good timeslot for the change operation but it does not prevent all problems.

4. SOLUTION PROPOSAL

One of the goals of this thesis was to provide a solution as a proof that the idea of the reconfigurable applications work in real-life. This chapter describes the made design and its background.

4.1 Background

The topic of the thesis was invented as a potential improvement for one of Wapice's customers, so it was natural to start testing the idea by developing a proof of concept with their system. Using an existing system makes it more difficult to implement a working solution as it would have been in a system specifically designed for this task. However, getting such a specifically designed system was considered impossible and it was considered out of the scope of this thesis to create one. Moreover, proving this solution to be able to taken into use in an existing industrial system would prove the idea being so flexible that it could be taken into use in other systems as well, if they meet the preconditions.

In highly automated and big manufacturing plants, it is common that the system cannot be started or stopped instantly. When the start button is pressed, it might take minutes (gas power plant [26]) or even hours (paper mill [27]) to get the process to normal running state. Normally when the system is running as planned, starting and stopping might be rare situations and those are kept to a minimum. Example shutdowns are performed in nuclear reactors once per year [28][29]. However, when developing this kind of automation systems and control algorithms for them, it is necessary to test them in practice. Changing the control algorithm on the fly would give great benefits and save a lot of time. Testing a completely new algorithm may not be a very common case, but the reconfiguration technique can be used to update the applications running the algorithm either by applying some efficiency upgrade or eliminating a found bug.

4.2 Overall design

Since the current system and workflow does not support a reconfigurable software, there is a need to modify it in many places. It was decided to do a proof of concept –type of a solution to test the design ideas and measure the effectiveness of made decisions. It was not meant to be ready for use directly for all developers nor production use. Instead, if the results show that this solution is applicable, then it needs to be generalized to work in every use case. Customer system had Xenomai operating system that has been evaluated previously in this thesis. The Xenomai has good real-time characteristics [24][30] and thus it gives a great base for building the system.

The application code itself must not require any modifications, as it was one of the goals of the solution. The applications are somewhat separate from each other but they need some way to communicate with the system and with each other. Currently all applications are built to one executable file so the applications can be compared to threads. However, it is not possible to replace only a single application with this design. Using dynamic libraries, such as Linux shared objects, it is possible to change the code under execution, thus providing basics for the application reconfiguration operation. The new architecture is illustrated in Figure 9.

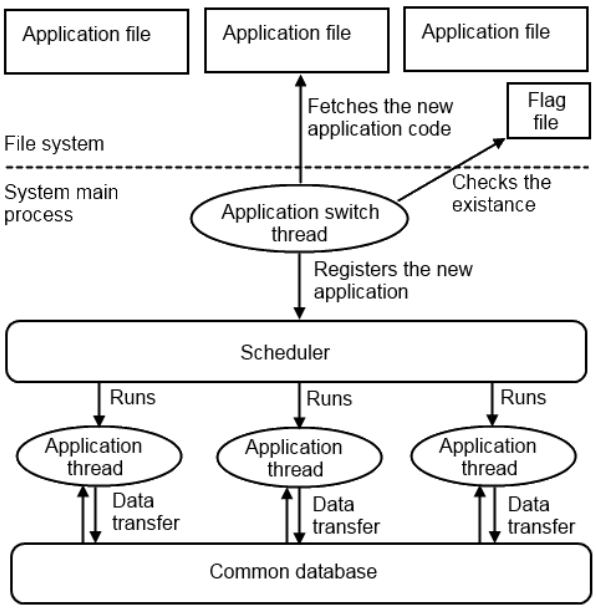


Figure 9. *The principle of the application reconfiguration structure*

The reconfiguration operation related instructions are collected to be executed in a thread. That thread checks the existence of a flag file in order to know when the reconfiguration operation has to be done. If the flag file exists, the reconfiguration thread loads the correct application file to the memory and fetches the needed functions. Then it registers the new application to the scheduler by replacing a pointer pointing to the old application code. The scheduler continues to schedule time slots to the applications normally.

The new application may not exist in the device and it needs to be transferred to it. A transferring method is considered irrelevant from the point of this thesis, as it may use the same communication protocol that was used to transfer the old application and other software components. This may be CAN, Ethernet, serial cable, USB or any other communication method.

Location of the new application in the module's mass storage, however, has some effects on the behavior. Directly replacing the file is considered risky, especially with slow communication protocols. The file may not be fully replaced when it is needed and that can

cause serious problems in execution. The problem may not arise every time since all applications are loaded to RAM memory during initialization of the hardware module and thus there should not be any need for rereading the file. To increase controllability and remove possible error situations, it was decided to transfer the new application file to a different location than the old one. This way it is also possible to do some sanity checks to the application, if necessary, before taking the application into use.

4.3 Reconfiguring process

After the new application is transferred to the module, there are a few optional things to do depending on the required reliability and functionality of the whole system. At first, the system needs to detect the new application. This also can be done in numerous ways, the solution for the thesis used an empty flag file that was manually created whenever specific application were wanted to be changed. The location and the function names were predefined and compiled into the reconfiguration process. This is not a generically applicable solution but it can be generalized by adding identification information of the old and the new application to the flag file.

When the system knows that it should replace an application, it has to open the new application file and fetch the starting points of functions. Fetching is done by using Linux system calls, such as dlopen and dlsym. The dlopen is capable of loading a dynamic library to the memory of the reconfiguration process and thus it makes the library usable by the process. Using the dlopen alone does not give much value since the reconfiguration process cannot know the structure of that new library. The dlsym system call is then used for fetching symbols from that library, such as pointers to functions or variables. With these system calls, the reconfiguration process is able to take the new application into use. For closing the old application, dlclose system call is used. It informs the operating system that the process is not using the library anymore and it can be unloaded from the RAM if no one else is using it either. [31]

The reconfiguration process in the solution included disabling the previous application and closing its library file, loading the new application into the memory and fetching the function pointers. It also included running the initialization function and registering the new application function to the scheduler. All of these operations need to be executed in generic case and they were therefore selected to be included into the solution to get more realistic results when measuring the used time. However, those operations may not need to be executed in the same order, and reordering them may save some time spent in critical part of the reconfiguration process. For example, the new application can be loaded to the memory and the new functions can be fetched before the actual application switch. This should be able to be done in all systems with sufficient amount of free memory available. Another optimization would be running the initialization tasks beforehand, but that may interfere with the old application and therefore cannot be used in all systems.

4.4 Inter-application communication

Previously it is discussed that the communication between the applications is necessary in most cases. In the solutions, all applications were running on the same process, so no inter-process communication mechanisms were needed. Architecture decisions prevented direct function calls between the applications, since the applications were located into different dynamic libraries. For these reasons, the common database technique was adopted to transfer a data between the applications.

The system already contained a common database for value transfer. It contained identifiers for accessing the data items and the actual values of the data. All applications had read and write access to all data items. This leads to a situation where for example two applications could write data to single item and cause undefined behavior of the system. The system did not protect the data items, so the applications needed to voluntarily cooperate with each other. Therefore, not all applications could coexist in the same system as they did have logical dependencies.

The common database contained only the resent value of a data item. Thus, the applications needed to collect and store all history data they needed. Data synchronization of the new applications was not implemented. If the new application needed history data, it was initialized with default values and filled with real values after the application has been taken into use, as the applications will update the values in their normal operation.

5. TEST SETUP AND MEASUREMENTS

The performance of the solution described in the previous chapter was tested by measuring some characteristics and comparing those to the original solution. This gives an impression of how well the application reconfiguration technique is able to fulfill the requirements assigned to it. In this chapter, discussion about the test environment and measurements is done.

5.1 Test environment

The test environment contained one hardware module from a bigger distributed embedded system. All modules in that system were quite independent, they had different set of applications and communication had been abstracted. Therefore testing the generated solution with only one module was considered to be enough.

The module was connected to a computer via Ethernet and a CAN bus. The Ethernet connection was used to transfer the new applications to the module. The CAN bus was used to communicate with the monitoring software running on the computer. The build system was changed to compile the new applications into shared libraries.

The module had a Freescale 8309 processor with a 400 MHz clock, 512 MB of RAM and 544 MB of flash memory. It also contained some extra hardware components that were not used in this solution, such as an FPGA.

5.2 Measuring used time

Measuring the time used for the application reconfiguration process is not an unambiguous task as discussed in chapter 2.6.1. This leads to measuring it using different methods to get more information and to ease further analysis. Each measurement method was used independently to minimize the overhead caused by the measurements themselves.

The wall clock time was measured using a monotonic system clock. This includes all the operations done by the system when measurement is ongoing, including for example possible execution of a higher priority process or interrupt handler and the process waiting for the completion of the I/O operation. From the application reconfiguration point of view, this is the most important time to measure. With it, it can be estimated if the application reconfiguration process can be completed in the available time slot.

The CPU time was measured using a process specific clock. This gives a more accurate result of how heavy and CPU intensive the application reconfiguration process actually is. In practical use cases, the consumed CPU time is not as important as the wall clock

time, but it might be interesting for example from a power consumption point of view or if the CPU load is already high.

The time-optimized solution could have been measured, which runs the initialization routines beforehand and then quickly changes the application. It was estimated that this would not give much valuable information so it was not done. Instead, both time measurements included stopping the old application and closing its file, opening the new application file, fetching the application initialization and operation functions, executing the initialization and registering the new operation function to the scheduler. These are all operations that need to be done at some point, so measuring all of them simultaneously was considered beneficial.

Opening and closing a file are I/O operations that cause the calling process to be scheduled away from the CPU. Those were included in the measurements as they were considered to provide meaningful input. Copying a new application file to a unique location and deleting the old application file was not included since they may not be included in all implementations of the reconfigurable applications. Copying and deleting are file system operations and they are not related to actual reconfiguration process; they should be handled in a separate control cycle anyway.

5.3 Measuring the memory used

The Xenomai provides many memory-profiling options to measure the memory usage from an operating system point of view. It can give the used amount of RAM and mass storage by an application. When measuring the performance of the application reconfiguration, both of those are interesting.

The Xenomai stores process related information to a */proc/[pid]/status* file, where the pid is a unique process identifier. From there it is possible to read the virtual memory size of the process, which will tell how much RAM the process has requested from the operating system. There is also information about how many pages the process has in real memory. The difference between these are that the process may reserve more memory than it actually uses, some of the used memory is swapped out or the process has opened files but not used them and the operating system therefore has never loaded those files into memory. More fine-grained memory usage is available at a */proc/[pid]/maps* file. It gives mapped memory addresses and the related path if applicable. It allows to fetch the actual amount of memory used by applications and drop the rest used by the same process. [32]

Reserved virtual memory for the whole process is measured in order to check how much extra space is needed for the reconfigurable application solution in general. Reserved memory for only the applications is also measured, as this is directly affected by taking the reconfiguration technique into use. Also used RAM pages for the whole process was measured to be able to see how much more memory the application reserves than uses.

This will give less interesting results, unless there is a huge difference, in which case the reason should be investigated.

The flash usage of the applications was also measured by comparing the size of generated binaries. Usage was expected to grow because of the overhead caused by compiling them separately into dynamic libraries. Also all commonly used variables and functions need to be included in all libraries using those, resulting bigger total size for the library based solution than when compiling all applications to one file. The flash usage was measured using *ls -l* command. The *ls* is a generally used program in Unix based systems and it lists the content of specified directory. [33]

6. TEST RESULTS

Execution of the tests discussed about in the previous chapter gave valuable information about the system and the characteristics of the application reconfiguration technique. The results are presented in this chapter and they are compared to the expectations. Further discussion and analysis of how the results affect the usability of the whole technique will be presented in the next chapter.

6.1 Application reconfiguration time

The application reconfiguration time was measured by constantly executing the reconfiguration process. Two precompiled application candidates were taken into use in turns. Totally 200 reconfigurations were performed and the results are shown in Table 2.

Table 2. *Results of the time measurements when performing the application reconfiguration*

	Min (µs)	Average (µs)	Max (µs)
Wall clock	2 103	2 623	6 689
Filtered wall clock values	2 103	2 359	3 350
CPU time	199	234	275

The results show great variation between the minimum and the maximum times used within both categories. This was expected in the wall clock case since it contains the time when the process is waiting for higher priority processes to complete. However, the worst case seems to use three times more time than optimal case. More detailed investigation shows that there are 14 measurement points that took clearly more time than others as seen in Figure 10. These exceptional measurement points are filtered out and the average and the maximum time used are recalculated and shown in Table 2.

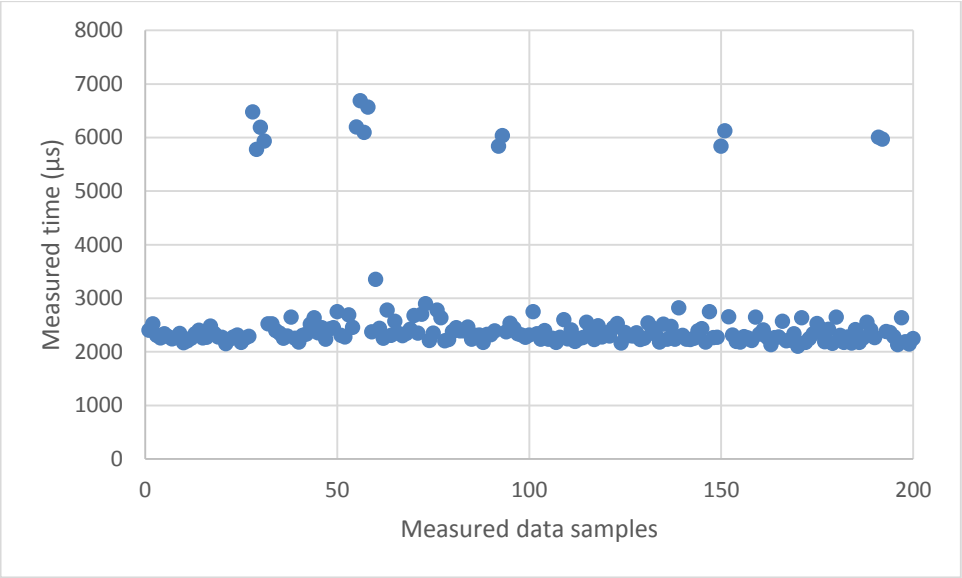


Figure 10. *The measured wall clock usage of the application reconfiguration process.*

The CPU time measurements did not vary as much as the wall clock time measurements. This was expected, since other processes should not influence the measurements. However, the CPU time varied more than expected. One reason for this could be that the application initialization functions took different amount of time, but since only two different applications were used, two clear vertical lines should be visible. Figure 11 shows the measured times used and there is able to see two lines with more measurements, one below 250 µs line and another above 200 µs line. These lines are not as clear as expected if the application initialization time explains the variance on measurements. It seems that an unknown factor is effecting the measurements, but it could not be identified.

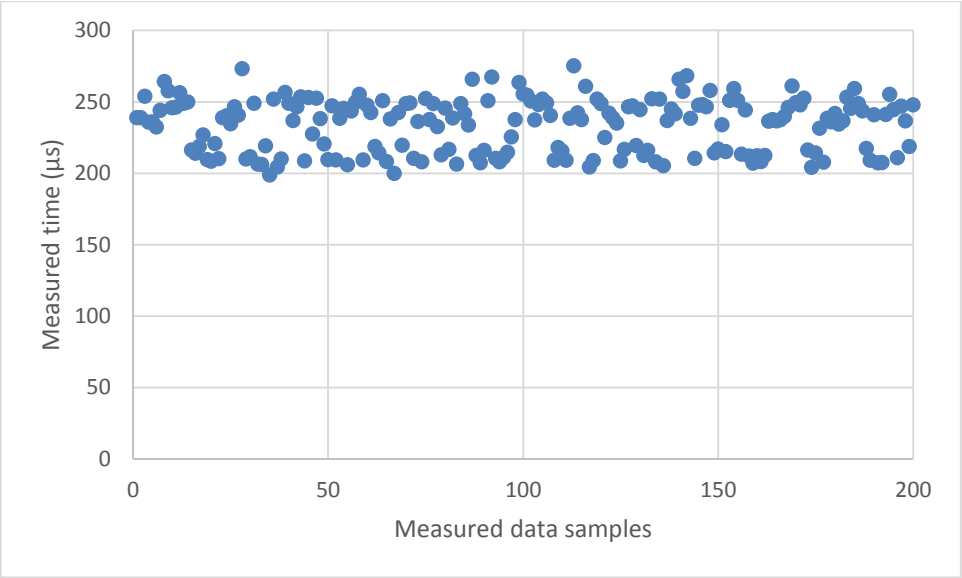


Figure 11. *The measured CPU time usage of the application reconfiguration process.*

6.2 Memory consumption

Both RAM and flash memory usages were measured, and the results are collected to Table 3. It shows the different memory usage between different architectures. The memory usage change after the reconfiguration process has been repeatedly executed is shown in Table 4.

Table 3. *Comparison of memory usage for the current architecture and the reconfigrable applications, both systems containing 5 applications*

	Current architecture (kB)	Reconfigurable applications (kB)	Difference (kB)	Difference (%)
Reserved virtual RAM for the whole process	12 056	12 268	+ 212	+ 1,8
Used real RAM pages for the whole process	9 448	9 520	+ 72	+ 0,76
Reserved virtual RAM for the applications	41	94	+ 53	+ 130
Flash memory used by the applications	50	952	+ 902	+ 1 800

Table 4. *Memory usage of the reconfigurable application before reconfiguring process and after it has been done 5 times*

	Before reconfiguration (kB)	After 5 reconfigurations (kB)	Difference (kB)	Difference (%)
Reserved virtual RAM for the whole process	12 268	12 344	+ 76	+ 0,62
Used real RAM pages for the whole process	9 520	9 532	+ 12	+ 0,13
Reserved virtual RAM for the applications	94	94	+ 0	+ 0
Flash memory used by the applications	952	952	+ 0	+ 0

The solution included different paths for the old and the new application file. This allows running initializations of the new application before closing the old one but it allows also to explicitly delete the old file and therefore it was possible to fix the memory leak issue.

Overhead memory consumption caused by compiling applications to shared objects is shown in Table 3. The process running the applications contains a lot of extra functionality, which is shown in the total consumption of RAM memory. When investigating the memory usage for applications only, it shows that the reconfigurable applications use a lot more memory than traditionally compiled ones. Increased memory usage was expected, but the amount of the increase was not estimated to be so high. After the reconfiguration process has been executed, as expected, there is no significant change in the used memory as shown in Table 4.

The reserved virtual memory for the whole process tells the amount of virtual memory the process has. The process may ask for more of it from the operating system but it also may decide not to use all available memory. This leads to differences between reserved memory and used memory pages. In addition, the application may have used more memory for example in initialization phase and then released it. The operating system may still reserve some of the virtual addresses to that process to speed up future memory reservation operations.

The reason for different reserved virtual memory usage between the current architecture and the reconfigurable applications is not clear. The reconfigurable applications technique contains more code but that only explains if the difference is same than in real memory pages used. However, the application reconfiguration reserves 212 kB more virtual RAM but uses only 72 kB more. It could be explained if the application reconfiguration process loads the new application into the RAM since the size matches closely to the

biggest application (209 kB flash). However, the reconfiguration operation was not performed and therefore it cannot be the reason. Maybe the observed extra space is needed when initializing applications for the first time.

Application specific RAM memory usage was more than estimated. It seems that the operating system reserves at least one memory page (4 kB) for the read only code section and one page for the read-write data section for every application. The theoretical maximum overhead for five applications and the reconfiguration process is then (5 + 1) * 4096 B * 2 = 49 152 kB ≈ 49 kB. The remaining difference between the two architectures may come from reserving more than enough memory in initialization phase, but that could not be measured.

The flash usage growth in the system was unexpectedly high. The application files took 165 – 209 kB flash memory space, so even the smallest application took over three times more memory than the all applications combined together.

After performing the application reconfiguration procedure, there are no significant changes in the amount of memory used. This was expected as the only theoretical reason for increasing memory would be implementation leaking memory.

During the implementation phase, draft solutions were also evaluated and one interesting phenomenon was found. If the application library is replaced in the file system and the main program does not close the old library file during the reconfiguration operation, the result is that the old application is still using flash. Even though the file was not reachable by the normal file system operations, the Xenomai did not completely delete the old library. This can be avoided by explicitly closing the unused libraries.

6.3 Switching to non-real-time mode

As discussed earlier, it would be beneficial to the reconfigurable application process to be in the real-time mode. Therefore, the possible switches between the real-time mode and the non-real-time mode were measured.

The measurements were done by using an *rt_task_inquire* system call in the Xenomai [34]. It gives various information about the specified process, including how many times the process mode has been switched. This method is useful if the process only executes the application reconfiguration related instructions. It means that the reconfiguration should be executed in different process than the actual applications. The made solution followed this architecture.

During the execution of the reconfiguration process, the process was checking from the flag file if it should perform the application reconfiguration. It caused the process to be switched to the non-real-time mode. After the check, the process was switched back to the real-time mode. This was not a valuable finding as the file system operations, such as

checking the existence of the flag file, were expected to cause a mode switch. These mode switches can be a problem if the application reconfiguration needs to be synchronized with the control cycle. The mode switch can be avoided by triggering the application switch by using a some other method, for example a semaphore.

The application reconfiguration process also opened the application file and fetched the needed functions. It caused three mode switches from the real-time mode to the non-real-time mode. These mode switches cannot be avoided since the new application needs to be loaded into the memory and the executable functions need to be fetched. This is a major problem if the applications and the reconfiguration are run in the same process in order to synchronize their executions, since it will switch also the applications to be executed in the non-real-time mode. The negative effects can be minimized by running the application reconfiguration in its own process and loading the new application into memory beforehand. Then the reconfiguration process could switch back to the real-time mode before switching the applications.

7. CONCLUSION

The application reconfiguration technique has now been discussed from multiple viewpoints, including how it would work in theory and what are the possible benefits, how an implementation can be done and how well that implementation can meet the expectations. From each chapter, valuable information has been gained to answer the research questions stated in the beginning, and in this chapter they are collected and a conclusion is formed.

7.1 Technology selections

Technologies were reviewed and their suitability was evaluated based on their performance and applicability. Operating systems were selected for comparison based on their availability, applicability and modifiability. Based on existing publications it is clear that operating systems specifically made for real-time solution have better real-time characteristics than operating systems with real-time modifications. Especially two Linux variants were evaluated and the Xenomai seemed to have clearly better real-time performance, which is a result of its dual-kernel architecture. Similar performance is expected on any specific real-time operating system.

IPC mechanisms were discussed in order to allow applications to communicate with each other even though they are used within separate processes. Different techniques have different advantages and disadvantages and therefore the used IPC mechanism needs to be selected depending on the other structure of the system.

7.2 Achievable performance

One research question was *'How long does it take to switch the application in a real-time operating system?'* This was measured with the Xenomai and it seemed that the worst-case reconfiguration time was 6,7 ms and filtered average time usage was 2,4 ms in the wall clock time. The measurements of the CPU time showed that the reconfiguration process is not very CPU intensive so the used wall clock time could be optimized by executing the time consuming file system operations beforehand or in separate thread. The measured times are low enough to allow the usage of this technique. However, the amount of used time is noticeable and it needs to be taken into an account especially on systems with short control cycle.

The performance evaluation contained also measuring the amount of RAM and flash memory used. Memory usage was unexpectedly high, especially on flash. No clear reason for this was found and this needs to be taken into account if adopting the reconfigurable applications technique.

7.3 Suitable application areas

Scheduled processes in operating systems are used in all non-embedded computers because of their obvious advantages. The reconfigurable applications technique tries to bring that idea to embedded control systems, too. According to what has been discussed in this thesis, the reconfigurable applications have some requirements for the system but otherwise it is a notable alternative.

One research question was *'What are the common problems when using application reconfiguration in a control system?'* Many problems were discusses and the solution to some of them could not be presented since the solution depends on the system in use. The most problematic technical questions were the application output verification, the initialization of the history data and the synchronization of the reconfiguration process to the control cycle. To solve these problems, some theoretical solutions were introduced, but those were not included in the implemented solutions. The measured performance could be problem on some systems, but some improvements were introduced.

Another research question was *'In what kind of embedded systems are application reconfigurations recommended?'* The reconfiguration technique is seen to suit best for development purpose as it gives fast way to evaluate made changes in applications. It removes the need for restarting the automation system, which in some cases may take a very long time. The reconfiguration technique also brings time savings to the testing phase when for example different alternative solutions are compared.

The reconfiguration technique is not recommended on final products. Usually there is not even need for this technique since the final product already contains a tested software and possible updates are rare and can cause the restarting of the product. There is always a small risk of failure when applying updates via the reconfiguration technique and depending on the system, it may cause a safety risk. In addition, if the system has security vulnerabilities, the application reconfiguration will give a dangerous amount of options to the possible attacker. Because of these matters, taking the application reconfiguration into use for final products is discouraged.

7.4 Thesis process

The thesis process has been proceeded quite well according to the original plans. The estimated time taken by each step has been more or less hold true but the concurrent working with the thesis and other work has been a bigger challenge than estimated. In addition, some unforeseen issues were found which affected the solution made and the direction of the thesis document.

Theoretical and technical study was quite laborious. The research field contains topics on a wide area of technologies and some tradeoffs had to be made in order to keep the scope

of the thesis. In addition, there is a lot of information available describing advantages of a feature in a general level but it is harder to find actual measurement results to support the presented claims. However, the results found were promising and motivated to do an experimental solution to allow further analysis of the presented topic. The solution provided essential information for verifying the assumptions. No overwhelming difficulties were found when implementing and testing the made solution.

In the end, the thesis process was successful. The thesis topics were thoroughly investigated and the questions were answered. It was possible to create an example solution to do measurements, which proves the possibility of using the technique.

7.5 Future improvements

This thesis described theoretical solution to reconfigure applications in run-time. An implementation was made but not with all mentioned features. Also testing showed some areas that could be improved.

The synchronization of the history data is seen as a very important improvement, since many control applications need to know the past values of measurements. The easiest way of doing this might be running both, the old and the new, applications in the same time, preventing the new application to effect the output. At the same time, the new application output check may be implemented. The output is not a problem in carefully developed applications but bugs could cause severe damage to the target system.

The application reconfiguration could also be synchronized with the control cycles. This would prevent errors when switching the application currently in execution. The synchronization should not be done by implementing the reconfiguration into the same process as the applications, since the reconfiguration will cause its process to be temporarily switched away from the real-time mode and it may cause severe problems in the application execution.

The application reconfiguration performance could be improved by opening the application file and fetch the needed function pointers before the actual switch. It is expected that this can be done in any system to decrease the time used in switching. Decreased time used would increase the amount of possible time slots for switching. On some systems, the application initialization function could also be done beforehand, but it may interfere with the existing application output and it is therefore not generally suitable solution.

REFERENCES

[1] Santambrogio, M.D. et al. Operating System Runtime Management of Partially Dynamically Reconfigurable Embedded Systems, 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia 2010), Scottsdale, AZ, 28-29.10.2010. Massachusetts Institution of Technology, Cambridge, MA, USA. p. 1 – 10.

[2] Wong, S. et al. Early Results from ERA – Embedded Reconfigurable Architectures, 9th IEEE International Conference on Industrial Informatics (INDIN 2011), Caparica, Lisbon, 26-29.7.2011. Delft University of Technology. Delft, Netherlands. p. 816 – 822.

[3] Singh, S.K. et al. Embedded Reconfigurable Architectures, 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC 2012), Solan, 6-8.12.2012. Uttarakhand Technical University. Roorkee, India. p. 385 – 390.

[4] Microsoft developer network: File security and Access Rights [WWW]. [accessed on 31.3.2015], Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/aa364399%28v=vs.85%29.aspx

[5] The Linux documentation project: Introduction to Linux: Chapter 3. About files and the file system [WWW]. [accessed on 4.5.2015]. Available at: http://www.tldp.org/LDP/intro-linux/html/sect_03_04.html

[6] Advanced mechatronics laboratory of Carnegie Mellon University [WWW]. [accessed on 4.4.2015]. Available at: http://www.cs.cmu.edu/~aml/aml.html

[7] Advanced mechatronics laboratory of Carnegie Mellon University: Chimera project home page [WWW]. [accessed on 4.4.2015]. Available at: http://www.cs.cmu.edu/~aml/chimera/chimera.html

[8] Stewart, D.B. et al. Dynamically reconfigurable embedded software – does it make sense? Second IEEE International conference on Engineering of Complex Computer Systems, Montreal, Que, 21-25.10.1996. Department of Electrical Engineering, Maryland University, College Park, MD, USA. p. 217 – 220.

[9] Park, J. et al. Home Network Middleware Architecture for Supporting Both Dynamic Reconfiguration and Real-Time Services, International Conference on Consumer Electronics, Digest of Technical Papers, Los Angeles, CA, USA, 13-15.6.2000. School of Electronics and Engineering, Kyungpook National University, Taegu, South-Korea. p. 372 – 373.

[10] Otero, A. et al. Dreams: A Tool for the design of Dynamically Reconfigurable Embedded and Modular Systems, International Conference on Reconfigurable Computing and FPGAs (ReConFig 2012), Cancun, 5-7.12.2012. Centro de Electrónica Industrial, Universidad Politécnica de Madrid, Spain. p. 1 – 8.

[11] Fleischmann, J. et al. A Hardware/Software Prototyping Environment for Dynamically Reconfigurable Embedded Systems, Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE ’98), Seattle, WA, 15-18.1998. Institution of Electronic Design Automation, Technical University, Munchen, Germany. p. 105 – 109.

[12] Fischer, G. et al. Linux Scheduling and Kernel Synchronization, chapter 7.4, System Clock: Of Time and Timers [WWW]. [accessed on 1.3.2015]. Available at: http://www.informit.com/articles/article.aspx?p=414983&seqNum=4

[13] Die: Linux documentation: clock_gettime [WWW]. [accessed on 6.2.2015]. Available at: http://linux.die.net/man/3/clock_gettime

[14] Linux man pages online: malloc_stats(3) [WWW]. [accessed on 6.4.2015]. Available at: http://man7.org/linux/man-pages/man3/malloc_stats.3.html

[15] Lockless Inc: Measuring memory usage [WWW]. [accessed on 6.4.2015]. Available at: http://locklessinc.com/articles/memory_usage/

[16] Modified from: UBM Electronics: Embedded Market Survey 2013 [PDF]. [accessed on 9.4.2015]. Available at: http://e.ubmelectronics.com/2013EmbeddedStudy/index.html

[17] RT_PREEMPT patch related wiki site [WWW]. [accessed on 30.10.2014]. Available at: https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

[18] Eker, J. et al. Taming Heterogeneity – The Ptolemy Approach, Proceedings of the IEEE volume 91, issue 1, January 2003. Department of Electrical Engineering and Computer Science, California University, Berkeley, CA, USA. p. 127 – 144.

[19] Helmy, T. et al. Avoidance of Priority Inversion in Real Time Systems Based on Resource Restoration, International Journal of Computer Science & Applications, Vol III, No. 1. College of Computer Science and Engineering, King Fahd University of Petroleum and Mineral, Dhahran, Kingdom of Saudi Arabia. p. 40 – 50.

[20] Linux news site: A realtime preemption overview [WWW]. [accessed on 30.10.2014]. Available at: http://lwn.net/Articles/146861/

[21] Koohwal, K. Investigating latency effects of the Linux real-time Preemption Patches (PREEMPT_RT) on AMD's GEODE LX Platform, VersaLogic Corporation [PDF]. [accessed 5.4.2015]. Available at: http://www.versalogic.com/downloads/whitepapers/real-time_linux_benchmark.pdf

[22] Yaghmour, K. Adaptive Domain Environment for Operating Systems, Opersys Inc [PDF]. [accessed on 9.4.2015]. Available at: http://www.opersys.com/ftp/pub/Adeos/adeos.pdf

[23] Xenomai documentation: Life with Adeos [WWW]. [accessed on 4.5.2015]. Available at: http://www.xenomai.org/documentation/xenomai-2.3/pdf/Life-with-Adeos-rev-B.pdf

[24] Haapaluoma, J. Xenomai and RT_PREEMPT latency tests. Wapice internal research material.

[25] The Linux documentation project: Static libraries [WWW]. [accessed on 4.5.2015]. Available at: http://tldp.org/HOWTO/Program-Library-HOWTO/static-libraries.html

[26] Wärtsilä: Gas power plant [WWW]. [accessed on 3.4.2015]. Available at: http://www.wartsila.com/en/power-plants/power-generation/gas-power-plants

[27] ABB: Regaining control [PDF]. [accessed on 3.4.2015]. Available at: http://www09.abb.com/global/scot/scot225.nsf/veritydisplay/4dea8f9d7887eb3ac12578f70028e950/$file/Green%20Bay%20Packaging%20Mill%20FeaturePPIAug2011.pdf

[28] Information site for nuclear power [WWW]. [accessed on 5.4.2015]. Available at: http://www.nucleartourist.com/operation/mtce1.htm

[29] Gregory, R. et al. Radiochemistry and Nuclear Chemistry, Third Edition, ISBN: 978-0-7506-7463-8

[30] Brown, J. et al. How fast is fast enough? Choosing between Xenomai and Linux for real-time applications, Rep Invariant Systems Inc. [PDF]. [accessed on 5.4.2015]. Available at: https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf

[31] Linux man pages online: dlopen(3) [WWW]. [accessed on 22.4.2015]. Available at: http://man7.org/linux/man-pages/man3/dlopen.3.html

[32] Linux man pages online: proc(5) [WWW]. [accessed on 6.4.2015]. Available at: http://man7.org/linux/man-pages/man5/proc.5.html

[33] GNU/Linux manual: ls [WWW]. [accessed on 22.4.2015]. Available at: http://www.gnu.org/software/coreutils/manual/html_node/ls-invocation.html

[34] Xenomai API documentation [WWW]. [accessed on 28.4.2015]. Available at: https://xenomai.org/documentation/trunk/html/api/group__task.html