Thesis basic level, 15 credits

Evaluation of OKL4

Bachelor Program in Computer Science

Mathias Bylund

Course: CDT307
Supervisor SAAB Aerotech: Martin Strand
Supervisor Mälardalens University: Moris Behnam
Examiner: Thomas Nolte
Date: 2009-04-16

This page is intentionally left empty.

Abstract

Virtualization is not a new concept in computer science. It has been used since the middle of the sixties and now software companies has interested in this technology. Virtualization is used in server side to maximize the capacity and reduce power consumption. This thesis focuses on virtualization in embedded system. The technology uses a hypervisor or a virtual machine monitor as a software layer that provide the virtual machine and to isolate the underlying hardware. One of most interesting issue is that it supports several operating system and applications running on the same hardware platform and the hypervisor has complete control of system resources. The company Open Kernel Labs is one of the leading providers of embedded systems software virtualization technology and OKL4 is one of their products, which is based on L4 family of second-generation microkernels. In this thesis, we will evaluate the kernel contains, the performance, the security and the environment of the OKL4. Finally we conclude the advantages and disadvantages of the product and technology.

Sammanfattning

Virtualisering har funnits i över 40 år men har på senare tid blivit ett hett ämne för företag. Det finns bland annat lösningar på serversidan för utnyttja dess kapacitet bättre och lagringsvirtualisering. Denna rapport handlar om virtualisering inom inbyggda system. Tekniken använder sig av hypervisor eller virtuell maskin som är ett mjukvarulager som tillhandahåller den virtuella miljön. Detta är en isolerad kopia av hårdvaran. Det som gör det intressant är man kan köra flera applikationer/operativsystem på samma hårdvaruplattform och den ska ha full koll på systemresurserna. OK Labs är en av de ledande distributörerna av inbyggda system med mjukvara och virtualiseringsteknologi. I denna rapport undersöks deras produkt OKL4. OKL4 använder sig av en mikrokärna som är baserad på L4:as mikrokärna familj. Utredningen består av vad kärnan innehåller, prestanda, dess utvecklingsmiljö och programspråk, plattformar och vilka former av support som finns. Fördelar och nackdelar att använda sig av deras produkt och hur tekniken kan komma att användas av verksamheten hos SAAB Aerotech. Avslutningsvis beskrivs hur en enkel utvecklingsmiljö skapas.

Tables of content

List of Figures

1. Introduction

1.1 Foreword

I would like to thank to Hans Sköld for introducing me at Saab Aerotech, Arboga. A special thanks to Martin Strand who has been my supervisor during the time and Fredrik Martinsson who made this project possible.

Arboga 2009-03-18

1.2 Background

The background to this thesis is that Saab Aerotech is working on real-time applications and they are interested in the virtualization technology and want to gain more knowledge about it. They would like to investigate this technology through whitepapers and Internet sources available for information. Beside evaluation of OKL4, the main issue was to see if they could use the technology in their future woks with their own products.

1.3 Purpose

The division avionics and test solution at Saab Aerotech FGZ is working on embedded real-time systems and they are interesting in knowing more about the benefits and drawbacks with the virtualization technology. Another purpose was to build and setup an environment future tests and projects. Do performance and functionality tests and run several operating systems/applications on the hardware using OKL4 and write a driver.

1.4 Limitations

Alongside the work was proceeding, the time was an issue and things from the original plan was reduced. There were no test done and nor a device driver.

2. Virtualization

There are different kinds of virtualization like process (application) virtual machine software, OS-level virtualization and the one this report is about system (hardware) virtual machine software. The technology refers to provide a software environment on which applications, including operation systems, can run on bare hardware. The software environment is called a virtual machine (VM). Figure 1 shows a virtual machine where the software layer who provides the VM environment is called hypervisor or virtual machine monitor (VMM). VMM is a soft-abstraction layer that converts a single computer hardware platform into one or more virtual machines, dynamically partitioning and sharing the hardware resources such as CPU, I/O devices and memory. The partition is completely protected and isolated; the VM represents an instance of the underlying physical machine giving the users an illusion of accessing the physical system directly. Therefore, multiple guest operating systems (OS) can run concurrently within the VMs on a single computer system. The hypervisor represents an interface that looks like hardware to the "guest" operation system. [He07]

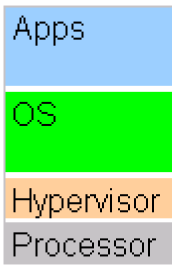

Figure 1 - The figure shows a virtual machine

There are three main characteristics for the VMM [SN05]:

- The environment should be identical or almost identical with the original machine.
- Speed should not be suffering when running in the environment.
- Complete control over system resources

All the characteristics are important to make virtualization useful in practice. The first is essential because it ensures software that runs on the real machine will run on the virtual. The second point is about the performance to ensure the efficiency of the virtualization. The last one is to assure that software cannot break out of the virtual machine. [He07]

2.1 How does it work?

The characteristics for a virtual machine are that the most instructions are executed directly on hardware. This is needed for efficiency and it requires that the virtual hardware should be similar or identical to the physical hardware. A virtual machine can have some instructions that are not supported by the physical hardware. There are several reasons for that, for example it can have an old version of the same architecture, different devices and different memory management unit. The key is that the difference is small and the instructions aren't used to often, then the virtualization can be efficient.

The instructions that cannot be directly executed, for example the resource-control requires that all instructions handles resources have to access the virtual rather than the physical resources. The virtual machine monitor must interpret these instructions, otherwise the virtualization is broken.
There are two classes of instruction that must be interpreted by the virtual machine [He07]:

- control-sensitive instructions
- behavior-sensitive instructions

The control-sensitive instructions are the instructions that modify the privileged machine state and interfere with the virtual machines control over resources. Behavior-sensitive instructions access the privileged state machine, it expose the state of the real recourses and cannot change the allocation of the resource. These two classes together are used to refer as virtualization-sensitive or sensitive instructions.

There are two basic ways to ensure if the code that is running in the virtual machine does not execute any sensitive instructions: Pure and impure virtualization. By pure virtualization it means that the sensitive instructions calls the hypervisor and do not execute within the virtual machine. In impure virtualization the sensitive instruction are removed from the virtual machine and replaced by virtualization code.

2.2 Benefits for embedded systems

In a historic perspective embedded systems had a single purpose, simple functionality, no or very simple user interface, and no or simple communication channels. The amount of software was small and loaded pre-sale, and normally there were no changes in the device. Today, the modern embedded systems are totally different and they have more complex user interface for instance in a mobile phone. The system has a user interface that has touch screen, camera, and mp3-player, Bluetooth, e-mail, games etc. There are taking more and more characteristics from general-purpose system. The software code is very large (many millions of lines) and the user can be connected to Internet and download data and games. An approach for companies is to build a test lab and test their software content in real-world environments. By using a full-system simulation with virtual hardware early on and through the whole project, they will be able to reduce the coats [He08].

2.2.1 Multiple concurrent operating systems

The main reason for use virtualization for embedded system is that it supports the execution of concurrent operating systems on the same hardware platform. For example it can be used to run a RTOS and at the same time run a high-level operating system like Windows or Linux on the same platform. In this case the RTOS uses a low-level real-time functionality as communication stack and

Windows/Linux can be used to support application code, such as user interface (see Figure 2).

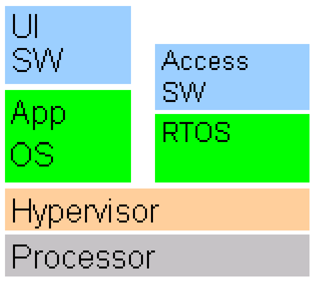

Figure 2 - Virtualization, running OS concurrently on the same hardware

It is possible to achieve the same effect using separate cores for real-time and application software stacks. The advantage of using a single core and virtualization is that there is no need for data transfer between cores and the access to other subsystems data is prevented. [Ma08]

2.2.2 Security

The most attractive feature for companies that the virtualization can offer is the security. By encapsulation of a subsystem, the other subsystem cannot be interfered with its failure. The benefit of security depends on that the guest operating system that is running de-privileged. If the guest system is running in privileged mode then the data structures of the hypervisor are not protected, and the guest can have absolute control of the machine. It is important that the real-time components can't be interfered by other operating systems or components and the access to the processor is limited and the deadlines can be meet. [Ma08]

2.2.3 Licence separation

There are many actors in developing high-level operating system. One of them is Linux and it has several benefits of using that:

- Royalty free
- Large development community
- Independent from vendors
- GPL license
- Open source

Virtualization is frequently working to segregate components subject to general public license (GPL) from proprietary code and create an execution environment for software to share the processor. Linux and the proprietary environment run in separate virtual machines. By using hypercalls the Linux driver can make requests to the real device driver via a proxy or stub. [He07, Ma08]

2.3 Limits of Virtualization

The two main issues that may limit the virtualization are the integration and granularity challenges for embedded system, not virtualization on its own. By integration it means to bring together components from subsystem into one system and ensure that the subsystems cooperate together as a system. The granularity challenge is about breaking down the system into small parts. These points will be covered more of integration and granularity:

- Software complexity
- Integration
- Security policies
- Trusted Computing Base

2.3.1 Software complexity

Software complexity is one of the issues that is related to the granularity of embedded systems. The functionality is high and gives the system more and more code. The big amount of code may lead to the case that it becomes hard to get it all correct and bugs are expected. If it's hard to get the code correct then there is a challenge for reliability of the devices. One benefit with virtualization was that the security threats could be controlled by the system but if the subsystem is failing then this can be considered as a limitation. To make it reliable, the software needs to detect faults and automatically handles and recover from them. This needs small components that are able to detect the faults. [He07]

2.3.2 Integration

To achieve the overall device functionality, subsystems are required to work together. Using virtualization on servers, many independent services can be run in their own virtual machine. In embedded systems there is a limitation because of the integration between subsystems. An example is communication between subsystems that requires integration and to make it efficient it needs high bandwidth and low latency. [He07]

Communication between subsystems is required and in order to make it efficient it needs high bandwidth and low latency. [He07]

2.3.3 Device sharing

Sharing of physical devices need some kind of policy, the integration requires that devices must be accessed by different subsystems. This approach of virtualization supports device drivers to run in their native (guest) operating system. This means that a device is locked by a particular guest, and cannot be accessed by others and the guest is trusted to drive a particular device. One issue that may cause a problem when using the virtualization in embedded system is that the device must be accessible by more than one guest. There are different kinds of solutions for sharing the devices. An approach is that if there are device drivers that are running inside the virtual machine, all drivers are ported to the hypervisor environment. Another solution is to have multiple virtual machines that share a single driver without including it in the hypervisor. To achieve this each subsystem needs a device model for which it has a driver. [He07]

2.3.4 Integrated scheduling

The scheduling in embedded systems must integrate the scheduling activities and it is done by a system wide policy. Each virtual machine cannot have an independent local policy. When a virtual machine is scheduled, the operating system of its guest schedules the thread by its own policy. If the guests don’t have any jobs to do, it schedules its idle thread. This is a special case and the hypervisor get a notification that some other virtual machine should be scheduling. In virtualization the guest OS is built with no insight on what happens in other virtual machines. The hypervisor can only associate an overall scheduling priority with each virtual machine. [He07]

2.3.5 Security policies

Virtualization on its own does not help embedded system to meet their requirements about critical security. To meet the requirements a security policy at system configuration time needs to be defined. This is for preventing code that is not trusted to get pass the policy. [He07]

2.3.6 Trusted computing base

The trusted computing base (TCB) of a computer system is the set of all hardware and software components that are critical to its security [02]. Service like encryption has to be protected. The union of code from encryption service and the code that dependent with it is called that service’s trusted computing base. Application TCB contains of all the code that runs in privileged mode of the processor. By this it means that TCB is always a part of the hypervisor or the virtual machine monitor. In the aspect of security, if the TCB is smaller, then the system will be more trustworthy. There is no formal proof of correctness of this and therefore it will contain faults as any other software. There is no support from virtualization technology to minimise the TCB. However, the virtualization can make the TCB bigger. [He08, He07]

3. Microkernel

In the early days when memory was limited, the kernels were small. As the kernel grew the number of devices also grew. The time when address space increased from 16 to 32 bits the kernel design was no longer hold back by hardware architecture and it started to increase. It was Berkley UNIX (BSD) who started the kernel era. BSD started to add a number of “virtual” devices like additional file systems, TCP/IP beside the basic system that consists of the CPU, disks and printers. The result of the growth the kernel had millions of lines of source code. The difficulties of maintain and the many bugs coming with big source code. The microkernel was design to allow easier management with code in user space. By reduce the running code in kernel mode it was increasing the security and stability. This was done and it refers as the first generations microkernels but problem was that they suffer poor inter process communication performance. The German computer scientist Jochen Liedtke started the L4 family [03] to make the performance better. The L4 family is named as the second-generation microkernels. The L4 is based on the original design and implementation made by him. Nowadays the industry sees the potential of using microkernel’s as a solution of embedded systems [He05].

The different between monolithic kernel and microkernel are [01, Figure 3]:

- Monolithic has a lot of privileged code and microkernel has little.
- Monolithic has a vertical structure and the microkernel has a horizontal.
- The microkernel is invoked by IPC and the monolithic are invoked by system calls.

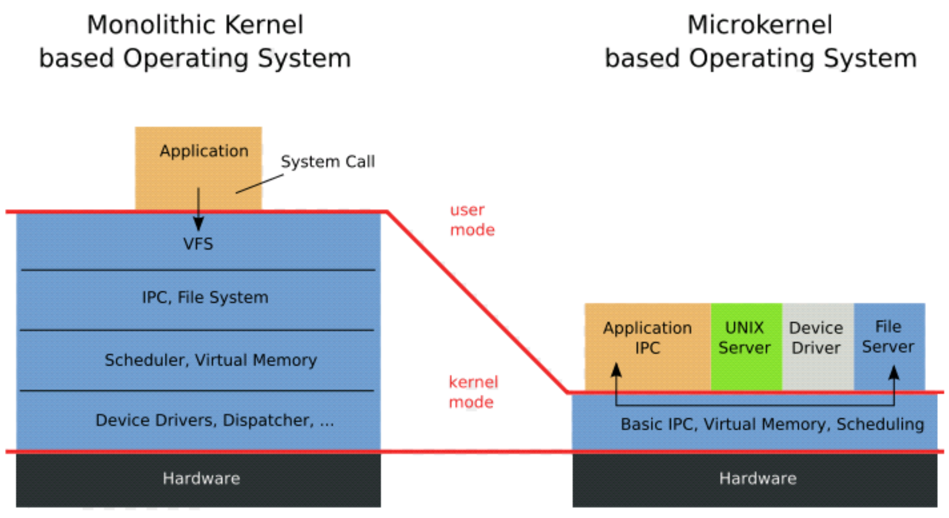
Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Figure 3 - Structure of a monolithic and microkernel based Operating system

4. OKL4

OKL4 is a system software platform built on microkernel technology. It supports system virtualization; secure execution of components in an execution environment and this with a minimal trusted computing base (TCB). Fault isolation within complex systems using lightweight protected components. It can be used as a full-featured operating system or alone. OKL4 is a result of research work from NICTA´s Embedded, Real-Time, and Operating Systems Research Program (ERTOS). OKL4 provides system virtualization that is useful for mobile devices, consumer electronics, industrial automation, medical electronics, and telecommunication/data communication infrastructure. [07]

It is an open source license and can be used in research, education and open source development projects. The open source way is to use the source code to be public inspected. There is a commercial license offered which addresses the needs of many users creating commercial products from OKL.

The OKL4 is built on the microkernel L4 and based on the Pistachio microkernel [04]. The use of microkernel has benefits in virtualization like been able to build subsystems with small trusted computing base [He05].

They have develop a product OK Linux that is a pre-built and para-virtualized version of the Linux kernel and base libraries, ready to run a guest OS under OKL4. There are two techniques for virtualization systems; full virtualization where the hypervisor is responsible for trapping and handling any privileged instructions the guest OS executes. The other is para-virtualization that OKL4 uses where the guest OS is modified through replacing all privileged instructions with explicit cause to hypervisor API known as hypercalls. Para virtualization also allows to replace multiple privileged instructions with a single hypercall reduce the numbers of context switch between privileged and non-privileged modes. These changes reduce the amount of works that the hypervisor compare to full virtualization. The benefit of that are lower virtualization overhead and the performance become closer to native.

The OKL4 Supports guest operating systems include Linux and a range of RTOSes.

The characteristics of OKL4:

- Low-overhead communication
- High performance IPC
- Efficient resource sharing
- Open Source code base
- Guest operating system executes in user-level
- Small memory footprint

- VM architecture isolates faults in guest OS
- Innovative device driver approach
- Secure Hypercell Technology
- State-of-the-art microkernel technology

4.1 About Open Kernel Labs (OK Labs)

In August 1998 OK Labs was founded as a spin-off of National Information Communication Technology Australia (NICTA). There is a partnership with NICTA and The University of South Wales (UNSW) and the combination is one of the largest research and development of microkernel. They supply software and professional services to OEM: s (Original Equipment Manufacturers) or other organizations developing and deploying intelligent devices. It’s a worldwide company with headquarters in Australia, USA and France. [07]

The Figure 4 [07] shows OK Labs community ecosystem. One of the founders is a professor at the academia (UNSW) and research is done at the NICTA. The source is open and they are a part of the L4 kernel project and family. Together with partner companies they create, validate and promote solutions.

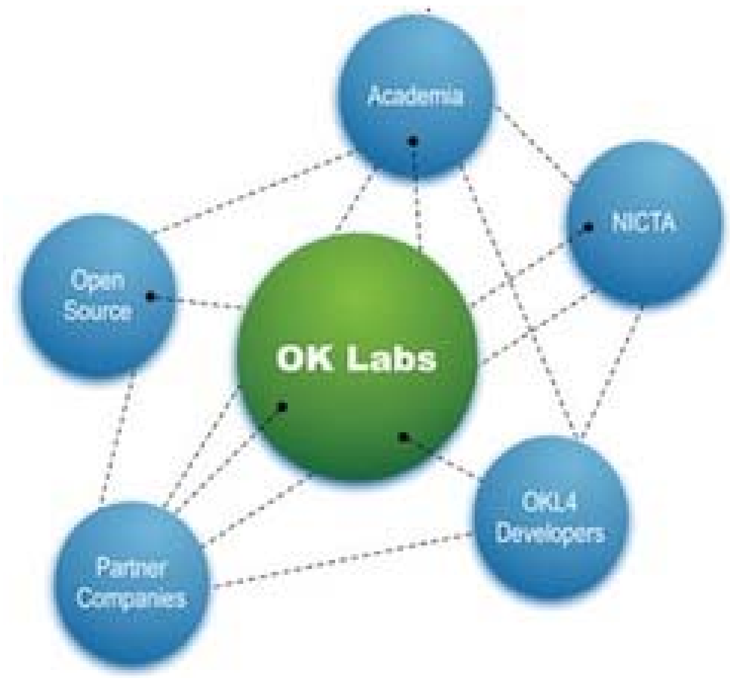

Figure 4 - OK Labs community ecosystem

4.2 Hypercell Technology ^{TM}

OK Labs uses their own architecture that they call Secure Hypercell Technology ^{TM}. It provides an environment for development and deployment of trustworthy embedded systems. As Figure 5 shows that it has different partitions (cells) and they are isolated from each other. Each cell can have a content of any system component or a complete operating system environment (virtual machine), operating system components like file system or network stacks, driver or applications. In user-level there are secure cells and they are isolated by the OKL4 microkernel who runs in privileged mode. Communications between cells are provided by the OKL4 microkernel but it's only allowed if it has been explicit authorised. The microkernel's code is the only thing that runs in privileged mode. It provides the virtualization, partitioning, and communications mechanisms (IPC) for the other system components. Since no other part of the system can access the privileged mode, the system integrity is protected of intruders or defected code within any of the cells. System components can be segregated individually at user level in secure and isolated cells. By that even at user level it can limit possible unstable or malicious code. Capabilities are used to delegate authority over resources to subsystem, more about capabilities in chapter 6.3.

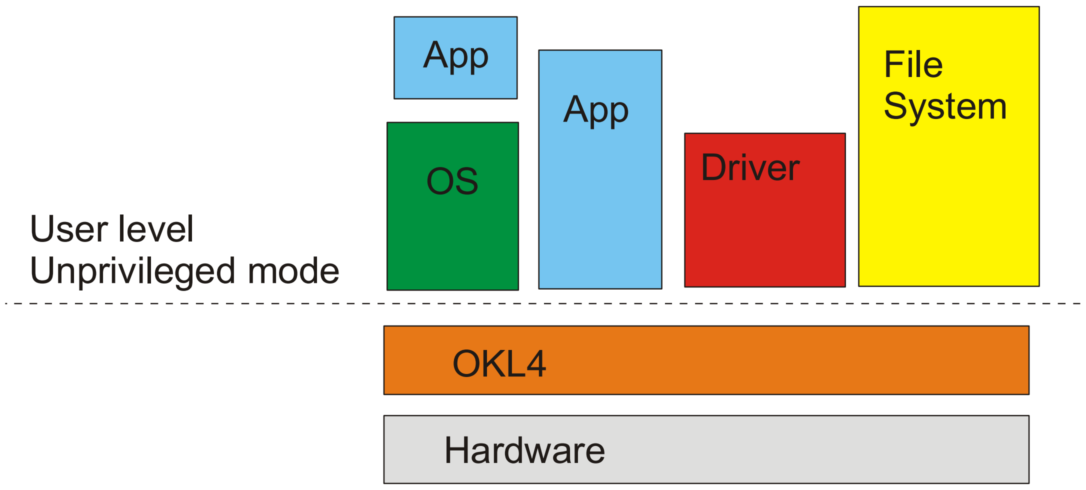

Figure 5 – Hypercell Technology

4.3 Security

The security is based on the Secure Hypercell Technology ^{TM} described in chapter 4.2. Another impotent thing is the Trusted Computing Base (TCB), and it's the part of the system that can avoid security. It runs in privileged processor mode e.g. OS kernel. User code sometimes modifies privileged data, for example Unix root process. The rule of high quality software is that it has a fault in every 1000 lines of code. In Figure 6 OKL4 the TCB is small, around 20 000 LOC compare to millions LOC in traditional embedded system where everything is TCB. [06]

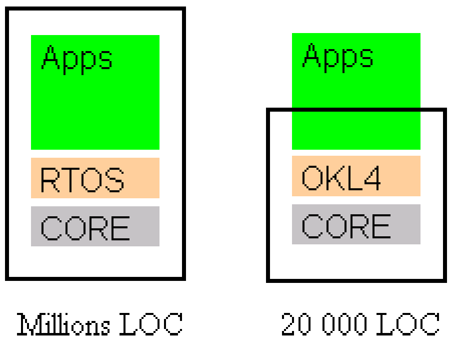

Figure 6 - TCB in a traditional embedded system versus OKL4

4.4 Hardware platforms

Today OKL4 only runs with processors that have memory management units (MMU). The security is an important requirement and not a configuration option; the use of an MMU to enforce boundaries between protected domains is central to the design of the OKL4 system. In the future, depending on the market, it could be ported to an MMU-less processor. Current supported processors that can be used are [07]:

- ARM v5(ARM9 och XScale kernel)
- ARM v6(ARM11 kernel)
- X86-32 and x86-64
- MIPS32 and MIPS64

The software downloads include OKL4 version 3.0 (microkernel and virtualization solution) and OK Linux (a virtualized Linux kernel for use with OKL4).

4.5 Support and service

4.5.1 Support

Content of available (free) support form OK Labs [07]:

- Wiki, http://wiki.ok-labs.com/
- Community, forum, http://www.ok-labs.com/community/community-portal
- Get Educated, http://wiki.ok-labs.com/GetEducated
- FAQ, http://wiki.ok-labs.com/FAQ
- Get into it, http://wiki.ok-labs.com/GetIntoIt
- Mailing list and whitepapers.

4.5.2 Professional service

Professional service and support [07]:

- In-depth technical help via email, web and voice
- Rapid turn around of technical inquiries maintenance requests
- On-site support and access to OK Labs professional services
- Jump-start installation and development assistance
- Updates and bug-fixes in development and after deployment
- Optional extended hours support for distributed development
- Long-term support for legacy software versions

4.5.3 Consulting services

List of consulting services provided by OK Labs [07]:

- OKL4 customization
- Retargeting OKL4 to new CPU architectures
- Creating board and chip support packages
- Para-virtualization of COTS and custom guest operating systems
- Developing device drivers
- Integration with in-house and 3rd party software
- Systems architecture design
- Turn-key application development

4.6 Environment and program language

C/C++ is the programming languages that are used for OKL4.

4.6.1 Tools

Iguna
It is designed for embedded systems and it provides OS functionality on top of OKL4 microkernel. It allows features as virtual memory management.

Elfweaver
It is a tool to enable the user to manipulate ELF (Executable and Linkable) files. It is used for merge multiple ELF files into a single ELF file and that means create an image.

CAmkES (component architectures for microkernel-based embedded systems)
It is a software development framework that provides:

- A language to describe component interfaces, components, and whole component-based systems

- A tool that processes these descriptions to combine programmer-provided component code with generated scaffolding and glue code to build a complete, bootable, system image

- Full integration in the OKL4 environment and build system

The framework is released under BSD license and can be downloaded from NICTA´s homepage [06].

Okl4fs
File system (okl4fs) is a module, use for inter-virtual machine communication. It allows creating bidirectional first-in-first-out (FIFO) like channel between OK Linux and another VM e.g. native OKL4 application

4.6.2 Graphics

A partner company, Fluffy Spider Technologies has a graphical stack FancyPants [04] that is a graphic engine for embedded systems. It is a small stack with a footprint of about 3MB. Figure 7 shows Linux-based OpenMoko with OKL4 and FancyPants.

Figure 7 - OKL4 and Fancy Pants in a Linux phone

4.6.3 OKL4 Source Tree

The OKL4 source tree is structured as follows [07]:

/pistachio		The OKL4 Kernel
/iguana		
	/server	The OKL4 root server
	/example	Sample OKL4 applications
	/test	Test suite for the OKL4 root server
	/v*	Virtual device servers
drivers		Device drivers
/libs		Core OKL4 libraries
	/c	The OKL4 C library
	/posix	A subset of the POSIX PSE51 profile
	/l4[e]	Userland stubs for invoking kernel primitives
	/iguana	OKL4 interface library for interacting with the OKL4 root server
	/v*	OKL4 interface library for obtaining virtual devices
/l4test		Test suite for the OKL4 kernel
/arch		Architecture-specific code
/platform		Platform-specific code
/projects		Build-system configuration for projects
/tools		Build-system tools, IDL generator (magpie) and system-configurator (elfweaver)

4.6.4 OK Linux Source Tree

OK Labs own variant of Linux used for a faster start of a project [07].

/apps			Applications that can be built along with OK Linux
/kernel-2.6.23-v2			
	/arch/l4		
		/drivers	OK Linux specific drivers
		/kernel	OK Linux kernel code
		/sys-*	Platform-specific OK Linux code (for i386, ARM, and MIPS)
	/include/asm-l4		OK Linux specific headers
/rootfs-2.6.23-v2			The OK Linux root file system
/tools			

4.7 Performance

Figure 6 shows a result of the performance between a native and a virtualized Linux on PXA22@400MHz [07]. The ratio shows that there is just a small different in performance.

Native vs. Virtualized Linux on PXA255@400MHz

ReAIM Benchmark	Linux Native	Linux on OKL4	Ratio
1 Task	45.2	43.6	0.96
2 Task	23.6	22.6	0.95
3 Task	15.8	15.3	0.96

Figure 8 - The performance different between native Linux and Linux on OKL4

On an ARM processor the IPC is performance for a one-way message passing operation less than 200 cycles. This high-performance provides efficient mechanism for setting up shared memory regions.

Figure 9 shows performance measures using Imbench latency benchmarks and the OKL4 were performed with OKL4 3.0 release and OK Linux 2.6.24. Running on an Xscale PXA255 platform at 200 MHz. Xen used a 266 MHz ARM926 processor and has different memory architecture. At native and virtual/OKL4 column the performance latencies is better if they are smaller. Relative performance column is about the degradation experience by OKL4 and Xen virtualization. Here is a higher number better. [He09]

	Latencies (μs)		Rel. Perf.	
Benchmark	Native	Virt/OKL4	OKL4	Xen
pipe	756.59	84.84	8.92	0.58
fork	6469	8742	0.74	0.29
fork+exec	59715	75515	0.79	0.30
semaphore	261.6	21.08	12.41	0.56
unix	1292.2	115.01	11.24	0.59
signal handler	14.26	54.76	0.26	0.55
null syscall	1.14	5.40	0.21	0.40
read syscall	3.34	8.45	0.40	0.52

Figure 9 - Benchmark performance of OKL4 and Xen

5. OKL4 API elements

The API (Application Programming Interface) consists of the follow interface elements:

- Address spaces and threads.
- Data types, data constructors and data fields.
- System parameters.
- Virtual register.
- System calls.
- Communication protocols.

The architecture is depended on how the elements can be accessed by a user program. [OKL08]

5.1 Threads

Threads in OKL4 are a context of execution within a program. Together with scheduling they can be used to handle the time sharing of the processor. Each thread has a unique identifier and associated with an address space. Creation and delete are made by the system call ThreadControl. Threads responsibility is to issue the system calls to the microkernel. They are the active parties during communication between different address spaces, or between user programs and the microkernel. There is an endeavour to provide time protection and an abstraction of an uninterrupted flow of time on a processing unit. This is achieved by using a pre-emptive scheduler.

The following are the different state of threads:

- Inactive
- Running
- Polling
- Sending
- Waiting to receive
- Waiting for notification
- Waiting on mutex
- Receiving

Using the system call Schedule, the current state of any thread can be obtained. [OKL08]

5.2 Address space

In OKL4 the address space is used for handling the sharing of memory-mapped resources like physical memory, device registers and I/O ports. System resources that are directly accessible through the memory management unit of a processor are considered to be memory-mapped. An address space consists of a set of mappings of virtual addresses to memory-mapped resources together with a class of permitted memory access operations and a cashing policy. By using the MapControl system call the address space mapping is constructed. In order to execute threads and access memory, address space are required. At a given point if time address space can have null, one or more threads associated with it. When an address space is associated with a thread, it cannot be deleted. The purpose of having the address space is to manage secure distribution and sharing of memory-mapped system resources. There are different types of capabilities OKL4 provides [OKL08]:

- Access Capabilities
- Physical Memory Segment Resource Capabilities
- Reply Capabilities
- Special Identifiers

5.3 Data types, data constructors and data fields

The term type is used to characterize the object stored in memory and/or virtual registers. Define the encoding of objects in term of a sequence of bits an occasionally to impose additional restrictions such a range numeric values allowed in a particular field. [OKL08]

5.4 Capabilities

Capabilities are used to control access to objects by restrict the objects access to users who have the necessary tokens or capabilities. The capability has the information of the rights that the user has to access an object. It could be copying, modifying or reading the object. Capabilities in OKL4 are used for handling the access control to system resources. In current version there are only thread objects that are protected by using capabilities. The purpose is to construct secure systems where communication between threads can be controlled using capabilities and capability spaces. An address space has a single associated capability space. Null or several objects can share a capability space. A capability space is implemented as a capability list or clist object. A capability list is a kernel object and by using the system calls CapContol it can be operated. [OKL08]

5.5 Scheduling Threads

OKL4 has pre-emptive scheduler whose responsibility is to allocate processing time to user threads. Pre-emptive means that a thread can be temporary interrupt by the system, without requiring cooperation and resume the execution of the thread in a later time of point. The algorithm is known as round robin (RR). Schedule inheritance is allowed to avoid priority inversion. Priority inversion occurs when a high-priority thread cannot run because it is waiting for an object held by a low-priority thread that is prevented by a medium priority thread.

Priorities can be used by the operating system to implement an array of scheduling principles included as for real-time applications.

OKL4 allows the user to enable or disable sceduling inheritance at compile time. When the scheduling inheritance is enabled then the microkernel will never change priority, this gives the operating system full control over the handling of the threads priority

Each thread has a content of:

- Priority level, range 0-255
- Allowed execution units
- Time slice
- Remaining time slice

The first three are configured and used by Schedule system calls. The microkernel maintains the remaining time slice internally. [OKL08]

5.6 Inter-Process Communication

This is the central component in OKL4´s system design. In OKL4 communication between threads in different address spaces and thread in same address space, are used. Most of the communication is synchronous but there is a limited form of messages that supports asynchronous delivery. The content of communication is short messages between threads and the exchange is done directly via MessageData register. Messages are transferred when both sender and receiver are ready. The message has a message tag and an optional list of message data. There is no buffering when communication is done; the microkernel copies the content from sender to receiver. This is the reason why the OKL4 microkernel has high performance.

To assure the integrity of the receiving thread, messages are exchanges synchronously to avoid that the thread is not compromised. This means that the microkernel will not deliver the message until the receiving thread is ready to get the data. There are two fundamental IPC operations:

- send, deliver message from sender to receiver
- receive, request a message from another thread

The status of a message can be:

- Blocking
- Non-blocking

The send operation must specify to which thread it wants to send. When a thread requests a send operation, the microkernel checks the destination threads capability space. If there is no capability found, the operation is aborted right away leaving an error message. The receive operation uses for a thread that request a message from another thread. [OKL08]

5.7 Mutex

To manage access to shared resources OKL4 uses mutexes:

There are two types of mutex:

- kernel mutex, to lock/unlock are system calls
- hybrid mutex, combination of library and system calls

There are three operations:

- lock, this acquire blocking
- trylock, this acquire non-blocking
- unlock, when releasing

To create a mutex the system call MutexContol is used. When the user calls, it has to specify the appropriate capability. Via calling the MutexControl system call, an existing mutex may be deleted. [OKL08]

5.8 Interrupts

The management of hardware interrupts are supported to provide an abstraction of hardware operations and also to provide access control. There is an interface to handle common interrupt operations that must be preformed on hardware. Operations like enabling and disabling interrupts are part of the OKL4 Board Support Package (BSP). To deliver hardware interrupts, it uses asynchronous inter-process communication. [OKL08]

5.9 Communication Protocols

IPC messages are used between user threads and the microkernel. When the microkernel handles a send or a receive operation e.g. received the thread are blocked or inactive during the delivery of the message. The IPC messages have the format shown in Figure 10 and are called message diagram. It describes the content of the MessageData registers used in the exchange during a microkernel-defined protocol. [OKL08]

Second word of the message							MessageData$_2$
First word of message data							MessageData$_1$
7 (label)	0 S	0 R	0 N	0 M	~	2 Count	MessageData$_0$

Figure 10 - Message diagram, a message that contains two words.

5.10 System parameters

System parameters are used to simplify the implementation on a specific architecture; the microkernel API allows a degree of flexibility in meeting specification requirements. The OKL4 API parameters are constants and the value must be documented and made available to programs at run-time [OKL08] (see appendix II for list of all system parameters).

5.11 Virtual registers

Each thread is associated with a number of virtual registers [OKL08] (see appendix III for list of all virtual registers).

5.12 System calls

A system call is presented as a function that has zero or more input parameters and providing zero or more output parameters. There are 8 privileged system calls (ThreadControl, SpaceControl, MapControl, CapControl, MutexControl, InterruptControl, CacheControl and PlatformControl) that control resources and they can only be executed by root task. The other system calls are unprivileged (ExchangeRegisters, Ipc, Schedule, ThreadSwitch, Mutex, MemoryCopy and SpaceSwitch) and they provide API to applications and can be invoked by anyone. More about the system calls in Appendix I [OKL08].

5.13 Kernel debugger

OKL has three different mechanisms for debugging and monitoring:

- Interactive
- External debugging and monitoring
- Internal debugging and monitoring

Each can independently be used and enabled depending on system configuration.

Console
If OKL4 includes a built-in console it can display run-time information and verbose booting, warning and assert messages.

Kernel Asserts
The kernel of OKL4 contains a configurable kernel assert implementation. The implementation can be disabled or configured by setting ASSERT_LEVEL to one of the level in Figure 11.

ASSERT_LEVEL	Type	Description
0	Always	Assert asserts only
1	General	Recommended production kernel
2	Normal	Recommended normal development
3	Debug	Use when debugging
4	Regression	Maximum, all asserts enabled

Figure 11 - Table of assert levels

Command Line Interface (CLI)
CLI is a built in debugger that operates on a console device. The purpose is to check system state, setting break points and other architecture purposes.

External debugging
The OKL4 kernel supports the use of external device such as in-circuit debuggers (ICD) or remote memory interface such as Firewire and host-mapped PCI.

Internal Debugging
This is for debugging and inspecting the state of the kernel from code running on the system. There are two main interfaces:

- kdebug interface
- tracebuffer interface

The kdebug interface allows the user access to services like controlling trace points, inspecting kernel memory usage information; debug console IO and maintaining thread name information.

6. OKL4 Library

It provides an interface that allows the user to create and manage cells including setting up protecting domains, managing memory and sharing memory between protection domains. This library does not support shared libraries but it is expected in a near future. [OKL08a]

6.1 Cells

Cells are an architectural concept that describes the partitioning of system resources and communication channels between OKL4 programs. There are different levels of cells; a low level can be a single application that is isolated from other programs in the system. On a complex level it can be a whole virtual machine as cell. A cell does not require more than one address space but can contain many address spaces. If it contains many address spaces then one of them will be associated with a kernel memory pool, global identifier and physical memory regions. This is called a cell manager. A cell has fix allocation of the underlying hardware and can only communicate with other cells. The cell cannot increase its access rights during run-time. [OKL08a]

6.2 Memory sections

The OKL4 is design to be flexible and small. It’s a contiguous range of virtual memory that can be backed in physical memory. [OKL08a]

6.3 OKL4 Library API

The Library API has a three-layer approach to manage functionality (Figure 12):

- ❑ Low-level
- ❑ Medium-level
- ❑ High-level

At the low-level interfaces (resource pool and allocator) the library supplies basic data structures and provides a minimal level of abstraction. It is the user responsibility to regulate the environment.

At medium-level it provides a lightweight interface of underlying kernel functionality. The execution interface accesses four interfaces in order to get to objects.

At the highest-level building blocks to create cells, access to memory section, extensions, zones and protection domains are provided

Allocator interface
There are two strategies provided by this interface to track resources:

- ❑ Range allocator
- ❑ Bitmap allocator

Resource Pools interface
The resource pool interface is consists of two interfaces:

- Kernel identifier pools (Clist ID, Thread ID, Mutex ID and Space ID)
- Memory pools

These interfaces manage the identifiers using a bitmap allocator.

Execution interface
This interface is built on top of the resource pools interface and provides the interfaces, Clist, UTCB Kspace and Kthread.

Cell interface
In highest-level the cell interface consists of the following interfaces:

- Memory section
- Protected domain
- Zone
- Extension

[OKL08a]

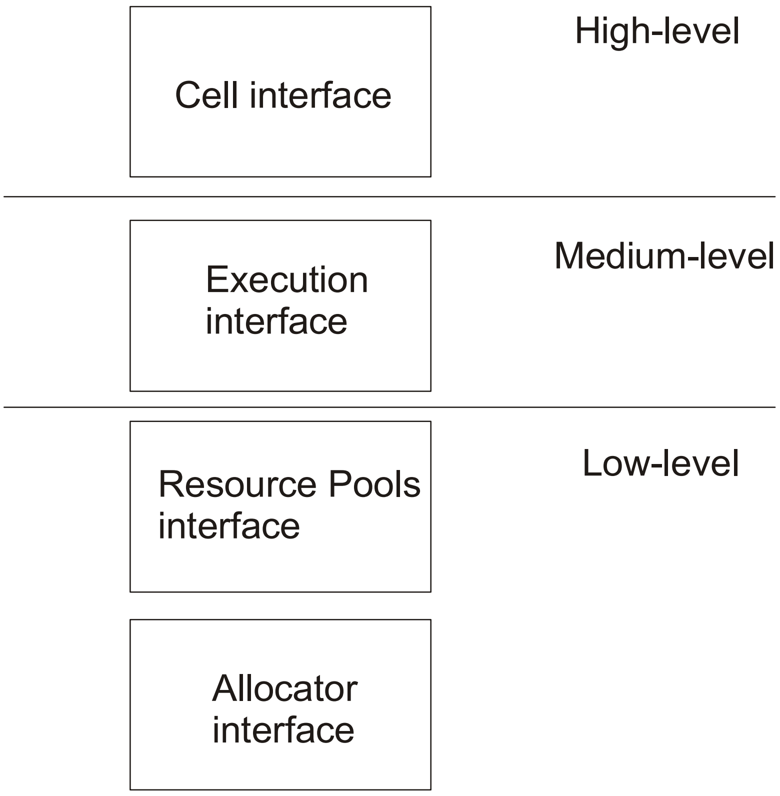

Figure 12 - Levels of abstraction

7 Driver model

In traditional operation systems, the device drivers are located inside the kernel. In that case an application must operate through the operating system to get the driver. In OKL4 the driver is taken out of the kernel (see Figure 13).

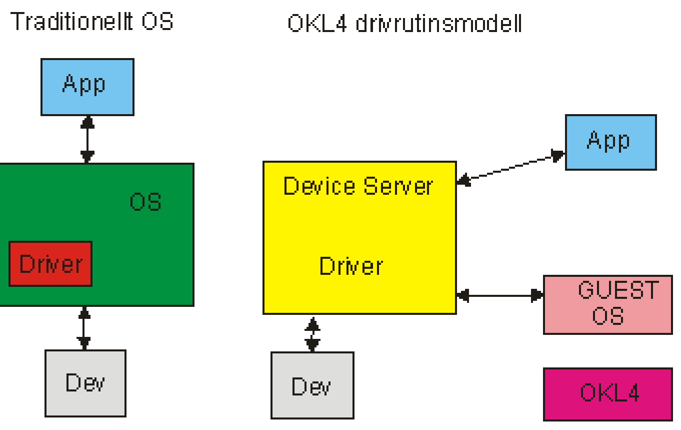

Figure 13 - Comparison between a traditional operating system and OKL4 device driver model

Benefits to have the driver outside the kernel on user-level:

- The most operation system bugs are in the device drivers. If they not are in the kernel it being smaller and more trustworthy.
- Better performance and smaller size.

Device server is created where:

- All talks indirectly with the device server to get a driver.
- Several clients can get access to drivers at the same time.
- The Device Server handles all logic.

7.1 Driver Framework

The content of the framework is components that use a client-server model. The goal is to hide complexity that is associated with device drivers from users that use the virtual device server. It allows applications to communicate with devices through a device server. The framework also hides the architecture specific complexities associated with low-level operations from the writer of the device driver. [DFW08]

7.1.1 Device Driver

OK Labs have on their homepage [10] tutorials for Writing a Device Driver and using a device driver. In the writing tutorial they use a LCD (S3C2410) from Samsung Electronics to show the process of implementing an OKL4 device driver.

All the physical characteristics of the device are specified in an XML (eXtensible Markup Language) file. The XML file has a .dx extension and contains the following tags:

- The Device Tag
- The State Tag
- The Interface Tag
- The Resource Tag
- The Block Tag
- The Register Tag
- The Field Tag

After the device has been defined all interfaces in the interface tag have to be implemented in the c-file. The driver should support two interfaces; frame buffer and device interface. These interfaces are specified in an own XML file with an extension of .di. There are four functions that the framebuffer interface supports, get_buffer (), set_buffer (), get_mode () and set_mode (). The device interface has a device_if.d file where the specification is. For testing a device driver all applications must interact with the virtual device server to get to the device. [DFW08]

7.1.2 Virtual device server

The virtual device server is visible for both the device core and user level applications and has two purposes:

- Allow OKL4 system resources to pass into the vdevice server.
- Allow access to the device driver functionality.

To prevent potential security breaches the virtual device server shields the writer of the device driver from the complexity associated with OKL4 resource management. It also shields the user level code from the underlying device driver. [DFW08]

8. Installing and running OKL4

To install and run OKL4, products from gumstix [05] was bought in. It was a motherboard connex 400xm [appendix IV], and a console-st [appendix IV] for connection with computer, and a netMMC [appendix IV] as a board for network and MMC storage. Working with OKL4 needs a Linux environment to build and compile. Communication between gumstix and the computer was made by application Ckermit and Minicom. Transferring files between desktop computer and gumstix can be done either via serial or via tftp.

A script created to connect via ckermit:

```
#!/usr/bin/kermit +
kermit -l /dev/ttyS0
set speed 115200
set reliable
fast
set carrier-watch off
set flow-control none
set prefixing all
set file type bin
set rec pack 4096
set send pack 4096
set window 5
connect
```

This appears on screen if the script succeeded:

```
*** Welcome to Gumstix ***
DRAM: 64 MB
Flash: 16 MB
SMC91C1111-0
Can't overwrite "serial#"
Can't overwrite "ethaddr"
Net: SMC91C1111-0
Hit any key to stop autoboot: 0
```

8.1 Prerequisites for OKL4

To get up and running there are three basic prerequisites [10]:

- A toolchain
- A simulator
- Phyton

8.2 Creating and simulating “hello world”

Step by step for install the toolchain:

- Download arm-linux-gnueabi-4.2.4.tar.gz
- Expand tarball and add the bin directory of the toolchain to your PATH.
- tar –xzf arm-linux-gnueabi-4.2.4.tar.gz
- PATH=$PATH:/home/ubuntu/arm-unknown-linux-gnueabi/bin/
- export $PATH
- Download skyeye-kenge-1.2.1n.tar.gz
- Add the bin directory to your PATH.
- PATH=$PATH:/home/ubuntu/kenge/bin/
- export $PATH
- Download sdk-xscale-3.0.tar.gz
- Expand the tarball
- Copy the content of the single cell directory to working directory
- Create an SDK environment variable that points to the SDK root directory by writing export OKL4_SDK_ROOT=path/to/sdk
- Make, this creates an image, image.sim in build.micro-debug/images/image.sim directory
- Simulate using the skyeye simulator by typing skyeye –c skyeye.conf –e build.micro-debug/images/image.sim

The result when simulating:

```
OKL4 - (provider: Open Kernel Labs) built on Oct 20 2008
10:49:03 using gcc version 3.4.4.
Initialized tracebuffer @ a0140000
Initializing kernel space @ f1001cf8...
Initializing kernel debugger...
Initializing interrupts...
System running with alignment exceptions enabled
Processor Id => 69052100: v5TE, Intel PXA255, rev 0
domain pairs: (0, 1)
Initialising scheduler...
Switching to idle thread
CREATE_CLIST: id=0, max_caps=1024
CREATE_SPACE: id=0, space{0:256}, clist{0:256},
mutex{0:256), max_phys_segs=7,
utcb_area{0x8000c000:2^0xc}, max_prio=255 has_kresources
1
CREATE_SEGMENT_LIST: entries=7
```

```
ASSIGN IRQ: irq=0x16
CREATE_THREAD: cap_slot=0, priroity=255, ip=0x80000000,
sp=0x8000b000, utcb_addr=0x8000c000
SETUP_SEGMENT: P:0xa0700000..0xa3800000, rwx:7,
attrib:ff
SETUP_SEGMENT: P:0xa0010000..0xa0018000, rwx:5,
attrib:ff
MAP_MEMORY: V:0x80000000 O:0x0 (P=a0010000) S:0x8000
N:0x1 A=0x3 R=0x5
SETUP_SEGMENT: P:0xa0005000..0xa0006000, rwx:6,
attrib:ff
MAP_MEMORY: V:0x80008000 O:0x0 (P=a0005000) S:0x1000
N:0x2 A=0x3 R=0x6
SETUP_SEGMENT: P:0xa0019000..0xa001a000, rwx:6,
attrib:ff
MAP_MEMORY: V:0x8000a000 O:0x0 (P=a0019000) S:0x1000
N:0x3 A=0x3 R=0x6
SETUP_SEGMENT: P:0x40100000..0x40101000, rwx:6,
attrib:ff
MAP_MEMORY: V:0x8000b000 O:0x0 (P=40100000) S:0x1000
N:0x4 A=0x0 R=0x6
SETUP_SEGMENT: P:0xa0600000..0xa0700000, rwx:6,
attrib:ff
MAP_MEMORY: V:0x80100000 O:0x0 (P=a0600000) S:0x100000
N:0x5 A=0x3 R=0x6
SETUP_SEGMENT: P:0xa0018000..0xa0019000, rwx:6,
attrib:ff
MAP_MEMORY: V:0x80009000 O:0x0 (P=a0018000) S:0x1000
N:0x6 A=0x3 R=0x6
Hello, world!
--- KD# User: L4 Rootserver exit ---
```

8.3 Transfer OKL4 to hardware

When connected to gumstix using for example the script shown earlier there are some useful commands when typing help or ? in the promt. In appendix IV there is a complete list of witch commands to use. To transfer data to gumstix there are several alternative ways to do it. In Chapter 8.3.1, we describe how to use TFTP and in 8.3.2 use serial transfer. There are also some configurations to set on the gumstix. By using print or printenv the current configuration can be shown.

8.3.1 TFTP – Trivial file transfer protocol

This needs a network card (appendix IV) to use TFTP for transferring files. Below describes a little guide to install and test TFTP in ubuntu.

Install the TFTP client and server package:

Sudo apt-get install tftp-hpa tftpd-hpa

The location for storing the TFTP files is /var/lib/tftpboot

Setup permissions of the TFTP server root directory:

```
Sudo mkdir /var/lib/tftpboot
Sudo chown nobody.nogroup /var/lib/tftpboot
Sudo chown 777 /var/lib/tftpboot
```

Edit the configuration file of the TFTP server in /etc/default/tftpd-hpa. The service shall be in deamon mode.

/etc/default/tftpd-hpa look like this:

```
#defaults for tftpd-hpa
RUN_DEAMON="yes"
OPTIONS="-1 –s /var/lib/tftpboot"
```

To start the init script

```
Sudo /etc/init.d/tftpd-hpa start
```

Check if the service has started:

```
netstat –a | grep tftp
```

If it works it should look like this:

```
udp      0      0*:tftp *:*
```

First off all the serverip variable has to the IP address of the TFTP server, set the ipaddr to assign the gumstix. Put the image boot file in the root of the TFTP server. Stop the autoboot after running the script-
To load the file use this command in prompt:

- tftp a2000000 and your boot file
- katinstall 200000 a0000000
- katload 200000 a000000; go a0000000

The first point describes the transfer of the file into RAM. Second installs the file on flash and the last one load the boot from flash.

8.3.2 Serial transfer

When connected to the gumstix files can be transferred via serial cable. The file needs to be a binary file and to apply the transfer just print:

```
loadb a2000000 filename
```

For some reason, it is common that the transfer takes timeouts, if so just disconnect and reconnect again.

9. Conclusion and Future works

9.1 Conclusion

The idea with virtualization is good and highly inevitable for company that have time and resources to implement it in their embedded systems. In early days the performance was an issue but now the performance not suffers like then. The OKL4 manages that with inter process communication where there are no buffering, the microkernel copies content from the sender to receiver. OK Labs have very good and interesting eco system with close cooperation with NICA and it seems that their technology will be in many systems in the future. NICTA is working on formal verification of the OKL4 kernel, a mathematic proof that the kernel is bugging free [He07a]. The work should be done in a near future and then they will have advantages against other products. There are many benefits of using virtualization in embedded systems. The hard part is to do the integration. My experience of reading about virtualization is that it will be a growing market for companies because they are interesting of using the benefits of virtualization technology. I have learned a lot during the weeks of this project thus it was most theoretical and not much implementing.

9.2 Future works

The future works at Saab Aerotech are that they will probably use the technology of virtualization in some project in the future. The focus is on security and performance in their systems and also testing systems with hard real-time constrains. They can build on the work I started and do further improvements to use the technology.

10. References

[DFW08] Driver Framework Manual (2008). Open Kernel Labs

[He05] Heiser G (2005). Secure Embedded Systems Need Microkernels *USENIX ;login:,* 30(6), 9–13

[He07] Heiser, G (2007). Virtualization for Embedded Systems. Technology whitepaper

[He07a] Heiser G (2007). Your System is Secure? Prove it! *USENIX ;login:* 32(6), 35–38

[He08] Heiser, G (2008). The Role of Virtualization in Embedded Systems. *1st Workshop on Isolation and Integration in Embedded Systems,* Glasgow, UK.

[He09] Heiser, G (2009). Hypervisors for Consumer Electronics

[Ma08] Matthews J (2008). Virtualization and componentization in embedded systems. Technology whitepaper

[OKL08] Open Kernel Labs (2008). OKL4 Microkernel Programming Manual

[OKL08a] Open Kernel Labs (2008). OKL4 Library Reference Manual

[SN05] Smith, J. E. & Nair, R. (2005). The Architecture Of Virtual Machines. Computer, 38(5), 32-38.

[01] http://en.wikipedia.org/wiki/Microkernel
Last visited 2009-01-30

[02] http://en.wikipedia.org/wiki/Trusted_Computing_Base
Last visited 2009-02-12

[03] http://os.inf.tu-dresden.de/L4/
The L4 µ-Kernel Family Last visited 2008-12-11

[04] http://www.fst.net/
Last visited 2009-04-03

[05] http://www.gumstix.com
Last visited 2009-03-20

[06] http://www.nicta.com.au
Last visited 2009-03-23

[07] http://www.ok-labs.com
Last visited 2009-03-31

Appendix I

System calls

CacheControl

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target address space	SPACEID	Parameter_0
$\text{Control}_{\text{IN}}$	Control word	CACHECONTROL	Parameter_1

Input registers: MessageData0 TO $\text{MessageData}_{2(\text{ControlIN.Count})-1}$

Parameter	Description	Type	Register
$\text{RegionAddress}_{\text{iIN}}$	RegionAddressiIN Location of an address range	WORD	MessageData_{2i}
$\text{RegionOp}_{\text{iIN}}$	Address range operation	CACHEREGIONOP	$\text{MessageData}_{2i+1}$

Output parameters

Parameter	Description	Type	Register
$\text{Result}_{\text{OUT}}$	The result flag	FLAG	Result_0
$\text{ErrorCode}_{\text{OUT}}$	The error code	ERRORCODE	ErrorCode

CapControl

Input parameters

Parameter	Description	Type	Register
Clist_{IN}	Target capability list	CLISTID	Parameter_0
$\text{Control}_{\text{IN}}$	Control word	CAPCONTROL	Parameter_1

Input registers MessageData0 TO $\text{MessageData}_{(\text{ControlIN.Count})-1}$

Parameter	Description	Type	Register
$\text{Parameter}_{\text{iIN}}$	Argument_i	CAPPARAMETER	MessageData_i

Output parameters

Parameter	Description	Type	Register
$\text{Result}_{\text{OUT}}$	The result flag	FLAG	Result_0
$\text{ErrorCode}_{\text{OUT}}$	The error code	ERRORCODE	ErrorCode

ExcahangeRegisters

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target thread	CAPID	Parameter_0
$\text{Control}_{\text{IN}}$	Control word	REGCONTROL	Parameter_1
$\text{StackPointer}_{\text{IN}}$	New value of StackPointer	WORD	Parameter_2
$\text{InstructionPointer}_{\text{IN}}$	New value of	CODE	Parameter_3

	InstructionPointer		
Flags_{IN}	New value of Flags	WORD	Parameter_4
$\text{UserHandle}_{\text{IN}}$	New value of UserHandle	WORD	Parameter_5

Output parameters

Parameter	Description	Type	Register
$\text{Result}_{\text{OUT}}$	The result flag	FLAG	Result_0
$\text{Control}_{\text{OUT}}$	Control word	REGCONTROL	Result_1
$\text{StackPointer}_{\text{OUT}}$	Old value of StackPointer	WORD	Result_2
$\text{InstructionPointer}_{\text{OUT}}$	Old value of InstructionPointer	CODE	Result_3
$\text{Flags}_{\text{OUT}}$	Old value of Flags	WORD	Result_4
$\text{UserHandler}_{\text{OUT}}$	Old value of UserHandle	WORD	Result_5
$\text{ErrorCode}_{\text{OUT}}$	The error code	ERRORCODE	ErrorCode

InterruptControl

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target thread	THREADID	Parameter_0
$\text{Control}_{\text{IN}}$	Control word	INTCONTROL	Parameter_1

Input registers MessageData_0 TO $\text{MessageDataControl}_{\text{IN.Count-1}}$

Parameter	Description	Type	Register
$\text{Parameter}_{\text{iIN}}$	IrqParameter Word_i	CAPPARAMETER	MessageData_i

Output parameters

Parameter	Description	Type	Register
$\text{Result}_{\text{OUT}}$	The result flag	FLAG	Result_0
$\text{ErrorCode}_{\text{OUT}}$	The error code	ERRORCODE	ErrorCode

Ipc

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target thread	THREADID	Parameter_0
$\text{Source}_{\text{IN}}$	Source thread	THREADID	Parameter_1
$\text{Acceptor}_{\text{IN}}$	Message acceptor	ACCEPTOR	Acceptor
Tag_{IN}	Message tag	MESSAGETAG	MessageData_0

Input registers MessageData_0 TO $\text{MessageDataControl}_{\text{TAGIN.Untyped}}$

Parameter	Description	Type	Register
Data_{iIN}	Message data to send	WORD	MessageData_i

Output parameters

Parameter	Description	Type	Register

Cap_{OUT}	Reply Capability	THREADID	$Result_0$
$SenderSpace_{OUT}$	The sender’s address space	SPACEID	SenderSpace
$ErrorCode_{OUT}$	The error code	ERRORCODE	ErrorCode
Tag_{OUT}	Message tag	MESSAGETAG	$MessageData_0$

Output registers $MessageData_1$ TO $MessageData_{TAGIN.Untyped}$

Parameter	Description	Type	Register
$Data_{iOUT}$	Received message data	WORD	$MessageData_i$

MapControl

Input parameters

Parameter	Description	Type	Register
$Target_{IN}$	Target address space	SPACEID	$Parameter_0$
$Control_{IN}$	Control word	MAPCONTROL	$Parameter_1$

Input registers $MessageData_0$ TO $MessageData_{3(ControlIN.Count)-2}$

Parameter	Description	Type	Register
$MapItem_{iIN}$	Input map data	MAPITEM	$MessageData_{3i}$

Output parameters

Parameter	Description	Type	Register
$Result_{OUT}$	The result flag	FLAG	$Result_0$
$ErrorCode_{OUT}$	The error code	ERRORCODE	ErrorCode

Output registers $MessageData_0$ TO $MessageData_{2(ControlIN.Count)-1}$

Parameter	Description	Type	Register
$QueryItem_{iOUT}$	Output query data	QUERYITEM	$MessageData_{3i+2}$

MemoryCopy

Input parameters

Parameter	Description	Type	Register
$Remote_{IN}$	Reply Capability	CAPID	$Parameter_0$
$Local_{IN}$	Local memory address	WORD	$Parameter_1$
$Size_{IN}$	Number of bytes to copy	WORD	$Parameter_2$
$Directiong_{IN}$	The direction of the copy	WORD	$Parameter_3$

Output parameters

Parameter	Description	Type	Register
$Result_{UT}$	The result flag	FLAG	$Result_0$
$ErrorCode_{OUT}$	The error code	ERRORCODE	ErrorCode

Mutex

Input parameters

Parameter	Description	Type	Register
$Target_{IN}$	Target mutex	MUTEXID	$Parameter_0$
$Control_{IN}$	Control word	MUTEXCONTROL	$Parameter_1$
VA_{IN}	Virtual address of hybrid mutex	ADDRESS	$Parameter_2$

Output parameters

Parameter	Description	Type	Register
$Result_{OUT}$	The result flag	FLAG	$Result_0$
$ErrorCode_{OUT}$	The error code	ERRORCODE	ErrorCode

MutexControl

Input parameters

Parameter	Description	Type	Register
$Target_{IN}$	Target mutex	MUTEXID	$Parameter_0$
$Control_{IN}$	Control word	MUTEXCONTROL	$Parameter_1$

Output parameters

Parameter	Description	Type	Register
$Result_{UT}$	The result flag	FLAG	$Result_0$
$ErrorCode_{OUT}$	The error code	ERRORCODE	ErrorCode

PlatformControl

Input parameters

Parameter	Description	Type	Register
$Control_{IN}$	Control word	PLATCONTROL	$Parameter_0$
$Param1_{IN}$	Genreal purpose word	WORD	$Parameter_1$
$Param2_{IN}$	Genreal purpose word	WORD	$Parameter_2$
$Param3_{IN}$	Genreal purpose word	WORD	$Parameter_3$

Input registers $MessageData_0$ TO $MessageData_{(ControlIN.Count)}$

Parameter	Description	Type	Register
$Argument_{iIN}$	General purpose input $argument_i$	WORD	$MessageData_i$

Output parameters

Parameter	Description	Type	Register
$Result_{UT}$	The result word	WORD	$Result_0$
$ErrorCode_{OUT}$	The error code	ERRORCODE	ErrorCode

Output registers $MessageData_0$ TO $MessageData_{(ControlIN.Count)}$

Parameter	Description	Type	Register

$\text{Argument}_{i\text{OUT}}$	General purpose output argument_i	WORD	MessageData_i

Schedule

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target thread	CAPID	Parameter_0
Slice_{IN}	Time slice length in milliseconds	WORD	Parameter_1
$\text{HwThreadMask}_{\text{IN}}$	Set of valid hardware threads	HWTHREADMASK	Parameter_2
$\text{Domain}_{\text{IN}}$	Reserved parameter	WORD_{16}	Parameter_3
$\text{Priority}_{\text{IN}}$	The priority	WORD_8	Parameter_4
$\text{Reserved}_{\text{IN}}$	Reserved parameter	WORD	Parameter_5

Output parameters

Parameter	Description	Type	Register
$\text{Result}_{\text{OUT}}$	Current state of the target thread	THREADSTATE	Result_0
$\text{Slice}_{\text{OUT}}$	Remaining time slice	WORD	Result_1
$\text{ErrorCode}_{\text{OUT}}$	The error code	ERRORCODE	ErrorCode

SpaceControl

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target address space	SPACEID	Parameter_0
$\text{Control}_{\text{IN}}$	Control word	SPACECONTROL	Parameter_1
Clist_{IN}	Capability list	CLISTID	Parameter_2
$\text{UtcbRegion}_{\text{IN}}$	Location of the UTCB region	FPAGE	Parameter_3
$\text{SpaceResources}_{\text{IN}}$	New space resources	SPACERESOURCES	Parameter_4

Output parameters

Parameter	Description	Type	Register
$\text{Result}_{\text{OUT}}$	The result flag	FLAG	Result_0
$\text{SpaceResources}_{\text{OUT}}$	Old value of the space resources	SPACERESOURCES	Result_1
$\text{ErrorCode}_{\text{OUT}}$	The error code	ERRORCODE	ErrorCode

SpaceSwitch

Input parameters

Parameter	Description	Type	Register
$\text{Target}_{\text{IN}}$	Target thread	THREADID	Parameter_0
Space_{IN}	Destination address space of the thread	SPACEID	Parameter_1
Utcb_{IN}	Address in	WORD	Parameter_2

	destination address space		

Output parameters

Parameter	Description	Type	Register
Result$_{OUT}$	The result flag	FLAG	Result$_0$
ErrorCode$_{OUT}$	The error code	ERRORCODE	ErrorCode

ThreadControl

Input parameters

Parameter	Description	Type	Register
Target$_{IN}$	Target thread	THREADID	Parameter$_0$
Space$_{IN}$	New address space of the thread	SPACEID	Parameter$_1$
Dummy$_{IN}$	Argument ignored	THREADID	Parameter$_2$
Pager$_{IN}$	New pager for the thread	THREADID	Parameter$_3$
ExceptionHandler$_{IN}$	New exception handler for the thread	THREADID	Parameter$_4$
ThreadResources$_{IN}$	Thread resources specifier	THREADRESOURCES	Parameter$_5$
Utcb$_{IN}$	UTCB entry of the new thread	WORD	Parameter$_6$

Output parameters

Parameter	Description	Type	Register
Result$_{OUT}$	The result flag	FLAG	Result$_0$
ErrorCode$_{OUT}$	The error code	ERRORCODE	ErrorCode
ThreadHandle$_{OUT}$	New thread’s handle	THREADID	MessageData$_0$

ThreadSwitch

Input parameters

Parameter	Description	Type	Register
Target$_{IN}$	Target thread or NILTHREAD	THREADID	Parameter$_0$

Appendix II

System parameters

System parameters	**Type**
MaxClists	WORD
MaxMessageData	WORD6
MaxMutexes	WORD
MaxSpaces	WORD
PageSizeMask	PAGENO
PMask	PERMISSIONS
UtcbAlignment	WORD6
UtcbRegionWidth	WORD6
UtcbMultiplier	WORD10

Appendix III

Virtual registers

Virtual register	**Type**	**Access**
Acceptor	ACCEPTOR	read/write
CopFlags	WORD_8	read/write
ErrorCode	ERRORCODE	read-only
ExceptionHandler	THREADID	not-directly-accessible
Flags	WORD	read/write
InstructionPointer	CODE	read/write
MessageData	WORD	read/write
MyThreadHandle	THREADID	read-only
NotifyFlags	WORD	read/write
NotifyMask	WORD	read/write
Pager	THREADID	read-only
Parameter	WORD	read/write
PreemptedIp	CODE	read-only
PreemptionFlags	PREEMPTIONFLAGS	read/write
PreemptCallbackIp	CODE	read/write
Priority	WORD	not-directly-accessible
ProcessingUnit	WORD	read-only
Result	WORD	read-only
SenderSpace	SPACEID	read-only
StackPointer	WORD	read/write
ThreadWord	WORD	read/write
UserHandle	WORD	read/write
Utcb	WORD	read-only

Appendix IV

Connex 400xm

The fastest motherboard in the PXA255-driven connex line. Ideal usages include keypads, terminals and telecommunications devices. Can be cased with a netCF expansion board into a netstix computer casing.

Speed:	400 MHz -(Intel XScaleÂ® PXA255)
Features:	64 MB RAM 16MB Flash memory
Connections:	60 pin Hirose I/O connector 92-pin bus header

netMMC

The sister board to the netCF expansion board, this expansion board combines network connectivity with MMC storage.

Features:	- 10-100baseT wired ethernet / RJ45 - MMC adapter supports MMC, SD and SDIO cards*
Connection:	92-pin bus header
Power:	Standard 3.5V - 6V input
	*SDIO cards do not typically have open source drivers available

Console-st

Provides console access and a power input port to verdex pro and all gumstix bluetooth-enabled motherboards. Also allows USB connection upstream to a host computer.

Features:	- 2 serial ports - USB client - mini-B socket
Connection:	60-pin I/O header
Power:	Standard 3.5V - 6V input

Appendix V

Commands on gumstix

List of commands on gumstix:

autoscr	- run script from memory
base	- print or set address offset
bdinfo	- print Board Info structure
boot	- boot default, i.e., run 'bootcmd'
bootd	- boot default, i.e., run 'bootcmd'
bootelf	- Boot from an ELF image in memory
bootm	- boot application image from memory
bootp	- boot image via network using BootP/TFTP protocol
bootvx	- Boot vxWorks from an ELF image
cmp	- memory compare
coninfo	- print console devices and information
cp	- memory copy
crc32	- checksum calculation
dcache	- enable or disable data cache
dhcp	- invoke DHCP client to obtain IP/boot params
diskboot	- boot from IDE device
echo	- echo args to console
erase	- erase FLASH memory
exit	- exit script
fatinfo	- print information about filesystem
fatload	- load binary file from a dos filesystem
fatls	- list files in a directory (default /)
flinfo	- print FLASH memory information
fsinfo	- print information about filesystems
fsload	- load binary file from a filesystem image
go	- start application at address 'addr'
help	- print online help
icache	- enable or disable instruction cache
ide	- IDE sub-system
iminfo	- print header information for application image
itest	- return true/false on integer compare
jerase	- erase FLASH memory for JFFS2
katinstall	- Load kernel to RAM from offset at top of flash
katload	- Load kernel to RAM from offset at top of flash
loadb	- load binary file over serial line (kermit mode)
loads	- load S-Record file over serial line
loady	- load binary file over serial line (ymodem mode)
loop	- infinite loop on address range
loopw	- infinite write loop on address range
ls	- list files in a directory (default /)
md	- memory display
mdc	- memory display cyclic
mm	- memory modify (auto-incrementing)

mmcinit	- init mmc card
mtest	- simple RAM test
mw	- memory write (fill)
mwc	- memory write cyclic
nfs	- boot image via network using NFS protocol
nm	- memory modify (constant address)
ping	- send ICMP ECHO_REQUEST to network host
pinit	- PCMCIA sub-system
printenv	- print environment variables
protect	- enable or disable FLASH write protection
rarpboot	- boot image via network using RARP/TFTP protocol
reset	- Perform RESET of the CPU
run	- run commands in an environment variable
saveenv	- save environment variables to persistent storage
saves	- save S-Record file over serial line
setenv	- set environment variables
sleep	- delay execution for some time
test	- minimal test like /bin/sh
tftpboot	- boot image via network using TFTP protocol
version	- print monitor version

Appendix VI

Build options

General Build Option

Option	Description	Possible values	Default
MACHINE	The machine/architecture for which to build OKL4	A valid machine name in tools/machines.py	None (required option)
PROJECT	The project to build	The name of a sub-directory inside the projects directory	None (required option)
BUILD_DIR	The sub-directory in which to build OKL4/OK Linux	Any (an existing directory will be overwritten)	Build
TOOLCHAIN	The toolchain with which to build OKL4/OK Linux	A valid toolchain in arch/{arch}/tools/toolchains.py	default_toolchain attribute in your machines.py machine definition
RUN	The simulator (or hardware) to use to execute the image following a successful build	A valid key from dict attribute run_methods in your machines.py machine definition	default_method attribute in your machines.py machine definition

Debugging options

Option	Description	Possible Values	Default
VERBOSE_STR	Controls the output of verbose build information from SCons	True or False	False
VERBOSE_INIT	Controls the output of verbose kernel initialisation information on boot	True or False	False
ENABLE_DEBUG	Enables or disables debugging for both the kernel and Iguana. This needs to be enabled for KDB, tracepoints, and tracebuffers to operate	True or False	True
DEBUG_TRACE	Sets Iguana debug output level	0 to 5 (0 being no output, 5 being highest)	0

KDB_COLOR_VT	Enables KDB to output in color mode	True or False	True
ASSERT_LEVEL	The level at which assertions are evaluated. The ASSERT macro requires a level argument - if the level for an assert is less than or equal to ASSERT_LEVEL, the assertion will be evaluated	Any integer value	3 if ENABLE_DEBUG, 5 otherwise
ENABLE_KMEM_TRACE	Allows tracing of kernel memory allocations via KDB	True or False	ENABLE_DEBUG
ENABLE_TRACEPOINTS	Enables the KDB Tracepoints feature	True or False	ENABLE_DEBUG
ENABLE_TRACEBUFFER	Enables the KDB Tracebuffer feature	True or False	ENABLE_DEBUG
TRACEBUFFER_PAGES	The number of pages to use for storage of Tracebuffer data	Any positive integer value	64
ENABLE_KDB_CONS	Enables KDB console support	True or False	ENABLE_DEBUG
ENABLE_KDB_CLI	Enables KDB command line support (interactive KDB)	True or False	ENABLE_DEBUG
KDB_BREAKIN	Enables breaking in to KDB using Ctrl-K or ESC	True or False	ENABLE_DEBUG && attribute kdb_breakin in machines.py definition
ENTER_KDB	Forces KDB to breakin upon startup, following kernel and root server initialisation	True or False	False
ENABLE_THREAD_NAMES	Enables the setting and printing of thread names	True or False	ENABLE_DEBUG
ENABLE_MUTEX_NAMES	Enables the setting and printing of mutex names	True or False	ENABLE_DEBUG
ENABLE_SPACE_NAMES	Enables the setting and printing of space names	True or False	False

Profiling Options

Option	Description	Possible Values	Default
ENABLE_PROFILING	Enables OKL4 Kernel Profiling via KDB	True or False	False
IGUANA_STATS	Allows Iguana to maintain various runtime statistics	True or False	False

Testing Options

Option	Description	Possible Values	Default
TEST_LIBS	Runs test suites provided with libraries	all, or the name of a library in libs	None
TESTING.TIMEOUT	Specifies a time limit, in seconds, for the running of a test suite	Any numeric value	30
TESTING.PRINT_LOG	Controls the printing of stdout output when running a test suite	True or False	False

OKL4 Kernel Options

Option	Description	Possible Values	Default
SCHEDULING_ALGORITHM	The scheduling algorithm to use. strict specifies that the scheduler should strictly obey priorities, with high priority threads always being selected to run over lower priority threads. inheritance specifies that priority inheritance should be used for threads in the system.	inheritance or strict	Inheritance
MAX_SPACES	The maximum number of address spaces	Any positive integer value	256

	supported		
MAX_THREADS	The maximum number of threads supported	>= 16 && <= 32768	1024
XIP	Enables OKL4 Kernel eXecution In Place	True or False	False
CONTEXT_BITMASKS	Enables the context bitmasks feature in the scheduler, which allows per-thread specification of the hardware execution units on which the thread may run	True or False	False
MUTEX_TYPE	Specifies the type of mutex primitive to expose via libmutex	user, kernel, hybrid	Hybrid
ENABLE_FASTPATHS	Enables or disables the use of all fastpaths (both C and assembly)	True or False	True
EXCEPTION_FASTPATH	Enables or disables the use of assembly fastpaths for exceptions	True or False	True && asm_fastpath_supported
IPC_FASTPATH	Enables or disables the use of assembly fastpaths for IPC	True or False	True && asm_fastpath_supported
IPC_C_FASTPATH	Enables or disables the use of C fastpaths	True or False	!asm_fastpath_support && c_fastpath_supported
ROOT_CAP_NO	Maximum number of root cap entries	>= 16 && <= 32768	4096 on IA32, 1024 otherwise
DEFAULT_CACHE_ATTR_WB	Defines whether to use write through or write back cache by default (ARM only)	True or False	True on ARM, unused otherwise
DISABLE_L2_CACHE	Disables the use of L2 Cache	True or False	False on ARM, unused otherwise

	(ARM only)		

Iguana Options

Option	Description	Possible Values	Default
EXAMPLE	An example application to build on top of Iguana	The name of a sub-directory inside iguana/example	None
EXAMPLE_ENV	The Scons environment in which to build the Iguana example application	iguana, which will build with the standard OKL4 libc, or posix, which will utilise a POSIX-enabled libc with a subset of PSE-51 support	iguana
INCLUDE_FILE	Provides the ability to embed a file into the resulting image which can then be read via the OKL4 FS library	The path to any file, relative to the OKL4 base directory	None
OKL4_FEATURE_PROFILE	Enables the use of a bundled set of OKL4 features (the EXTRA profile enables extended address spaces, zones, protected memory sections and remote memcopy)	NORMAL or EXTRA	Attribute default_feature_profile in machines.py, usually EXTRA

OK Linux Options

Option	Description	Possible Values	Default
OKLINUX_DIR	The directory path containing OK Linux, relative to OKL4 base directory	Any directory containing OK Linux	linux
LINUX_ROOTFS	The directory path containing a customised root filesystem for OK Linux	Any directory path	None
LINUX_ROOTFS_SIZE	Suggested file system image size for the customised root	Any positive integer	None

	filesystem, in blocks		
LINUX_DEVFS	The path to a customised device description file	A file path	None
LINUX_APPS	The name, or an array of names, of applications to build with OK Linux	Must correspond to a directory name of an application in the OK Linux base directory	None
ENABLE_VDSO	Use OK Linux VDSO page (sysenter) to perform Linux syscalls on ia32	True or False	False
LTP_SET	If building LTP as a OK Linux app, allows the specification of the specific LTP test suite to use	Set1, set2, set3, pthreads, signals, or all	all