Symbian OS Internals

Real-time Kernel Programming

Jane Sales

With

Andrew Rogers, Andrew Thoelke, Carlos Freitas, Corinne Dive-Reclus, Dennis May, Douglas Feather, Morgan Henry, Peter Scobie, Jasmine Strong, Jason Parker, Stefan Williams and Tony Lofthouse

And

Jon Coppeard and Martin Tasker

Reviewed by

Andrew Ford, Andrew Jordan, Andrew Thoelke, David Bachelor, Dennis May, Jason Parker, Jonathan Medhurst, Jo Stichbury, Mark Shackman, Nigel Henshaw, Peter Scobie, Richard Fitzgerald, Simon Trimmer, Tony Lofthouse, Trevor Blight and William Roberts

Symbian Press

Head of Symbian Press

Phil Northam

Managing Editor

Freddie Gjertsen

John Wiley & Sons, Ltd

Symbian OS Internals

TITLES PUBLISHED BY SYMBIAN PRESS

- Wireless Java for Symbian Devices
Jonathan Allin
0471 486841 512pp 2001 Paperback

- Symbian OS Communications Programming
Michael J Jipping
0470 844302 418pp 2002 Paperback

- Programming for the Series 60 Platform and Symbian OS
Digia
0470 849487 550pp 2002 Paperback

- Symbian OS C++ for Mobile Phones, Volume 1
Richard Harrison
0470 856114 826pp 2003 Paperback

- Programming Java 2 Micro Edition on Symbian OS
Martin de Jode
0470 092238 498pp 2004 Paperback

- Symbian OS C++ for Mobile Phones, Volume 2
Richard Harrison
0470 871083 448pp 2004 Paperback

- Symbian OS Explained
Jo Stichbury
0470 021306 448pp 2004 Paperback

- Programming PC Connectivity Applications for Symbian OS
Ian McDowall
0470 090537 480pp 2004 Paperback

- Rapid Mobile Enterprise Development for Symbian OS
Ewan Spence
0470 014857 324pp 2005 Paperback

- Symbian for Software Leaders
David Wood
0470 016833 326pp 2005 Hardback

Symbian OS Internals

Real-time Kernel Programming

Jane Sales

With

Andrew Rogers, Andrew Thoelke, Carlos Freitas, Corinne Dive-Reclus, Dennis May, Douglas Feather, Morgan Henry, Peter Scobie, Jasmine Strong, Jason Parker, Stefan Williams and Tony Lofthouse

And

Jon Coppeard and Martin Tasker

Reviewed by

Andrew Ford, Andrew Jordan, Andrew Thoelke, David Bachelor, Dennis May, Jason Parker, Jonathan Medhurst, Jo Stichbury, Mark Shackman, Nigel Henshaw, Peter Scobie, Richard Fitzgerald, Simon Trimmer, Tony Lofthouse, Trevor Blight and William Roberts

Symbian Press

Head of Symbian Press

Phil Northam

Managing Editor

Freddie Gjertsen

John Wiley & Sons, Ltd

Published by John Wiley & Sons, Ltd
The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England
Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wiley.com

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario,
Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Sales, Jane.
Symbian OS internals : real-time kernel programming / Jane Sales with Andrew Rogers [. . . et al.].
p. cm.
Includes bibliographical references and index.
ISBN-13 978-0-470-02524-6 (pbk. : alk. paper)
ISBN-10 0-470-02524-7 (pbk. : alk. paper)
1. Real-time control. I. Title.
TJ217.7.S25 2005
629.8—dc22

2005018263

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13 978-0-470-02524-6
ISBN-10 0-470-02524-7

Typeset in 10/12pt Optima by Laserwords Private Limited, Chennai, India
Printed and bound in Great Britain by Bell & Bain, Glasgow
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

Contents

Symbian Press Acknowledgements

Many people put many hours into the creation of this book, none more so than the authors and reviewers. Symbian Press would like to thank each of them without restraint for their perseverance and dedication, with a special mention for Dennis, who always seemed to be holding the short straw.

Thanks are also due to Stephen Evans and Akin Oyesola for their patience whilst some of their most vital engineers were distracted from ''real work'' for week after long week.

About this Book

The latest versions of Symbian OS are based upon Symbian's new real-time kernel. This kernel is designed to make phone-related development easier: base-porting will be easier, device driver development will be easier, software development will be easier.

Symbian OS Internals is a detailed exposition on the new real-time kernel, providing the reader with the insights of the software engineers who designed and wrote it. In the main it is an explanatory text which seeks to describe the functioning of the kernel, and, where relevant, to indicate key differences from its predecessor.

This book is invaluable for:

Those who are involved in porting Symbian OS. This book is not a base-porting manual, since this is already provided by Symbian; but it benefits the base-porting engineer by giving him or her a more solid understanding of the OS being ported.

Those writing device drivers. This book provides an in-depth explanation of how Symbian OS drivers work. Device drivers have changed considerably with the introduction of a single code, so this helps fill the knowledge gap for those converting them to the new kernel.

Those who wish to know how the Symbian OS kernel works. This book provides a detailed commentary on the internals of Symbian OS, providing information for the varied needs of readers, helping: students studying real-time operating systems, middleware programmers to understand the behavior of underlying systems, systems engineers to understand how Symbian OS compares to other similar operating systems, and, for those designing for high performance, how to achieve it.

About the Authors

Jane Sales, lead author

Jane joined Symbian (then Psion) in 1995 to lead the team whose goal was to create a new 32-bit operating system, now known as EKA1. This goal was realized two years later, when the Psion Series 5 was released. Since then, Jane has taken on a variety of roles within Symbian, including product management, systems architecture and setting up a small product-focused research group in Japan.

Jane studied at Jesus College, Oxford, where she gained an MA in mathematics in 1982. After that, she became part of a small team building a microcomputer to which they ported CCP/M. It was the start of a long-lasting love of systems programming.

In 2003, Jane moved to the South of France and began work on this book. She would like to thank her husband, Roger, for his moral and financial support over the succeeding 18 months.

Corinne Dive-Reclus

Corinne joined Symbian in 2001 after several years at Baltimore Ltd., where she designed the embedded software of ACCE (Advanced Configurable Crypto Environment), a cryptographic hardware module certified FIPS 140-1 Level 4.

In 1987, her first job was as a presales engineer at Sun Microsystems, France where she was exposed to the concept of Network Computing & Internet and to its very first remote attacks – at a time when this was still confidential. Being more interested in development than presales, she subsequently worked as a software engineer for Concept and Implicit, developing an RAD (Rapid Application Development) programming language for the creation of distributed data-oriented applications.

Since joining Symbian, Corinne has been the Platform Security System Architect, and has worked on the platform security architecture from its initial design to its implementation in Symbian v9. To ensure that all layers of the operating system will contribute to its overall security, she worked extensively with Andrew Thoelke and Dennis May to validate the security features of EKA2, design new services such as Publish&Subscribe and Capability Model, as well as defining the new behavior of the file server, known as Data Caging. For this work, Corinne received the Symbian Technical Innovation award in 2003.

Fulfilling an early passion for Prehistoric Archaeology & Computing, Corinne graduated from the Ecole Nationale Superieure de Geologie, a unique French institution combining a five-year curriculum in Engineering and Earth Science.

Douglas Feather

Douglas joined Symbian (then Psion) in 1994. He started by leading the writing of the text formatting engine for the Psion S5. He has also worked in the Web Browser and Core Apps teams where he rewrote most of the versit parser to greatly improve its performance. For five out of the last seven years he has worked mainly on the Window Server, where he added full color support, fading support, a framework to allow digital ink to be drawn over the other applications, support for multiple screens and transparent windows.

Douglas has a BSc in Mathematics from Southampton University and a PhD in Number Theory from Nottingham University. He is a committed Christian and regularly engages in vigorous theological debate, to defend the biblical truths about Jesus Christ, at Speaker's Corner (Hyde Park, London) on Sunday afternoons.

Carlos Freitas

Carlos joined Symbian in 2000, working on device and media drivers and porting the Base. His involvement with EKA2 dates from 2002 when he ported the newly released Kernel and several device drivers to a new hardware reference platform. He has since assumed responsibility for documenting, improving and maintaining the power management framework.

Carlos is a licentiate in Electronics and Telecommunications Engineering from the University of Porto, Portugal, and had a CERN bursary to research advanced optical communications. Prior to Symbian Carlos worked on both hardware and embedded software development under a number of Operating Systems such as VxWorks, μItron and WinCE.

Morgan Henry

Morgan joined Symbian (then Psion) in 1995 working in the kernel team on EKA1, where he met some amazing people. Once he regained his composure he was responsible for the kernel port for the Nokia 9210 – ''the world's first open Symbian OS phone''. During his time in the kernel team he invented the EKA1 power management framework, a cross platform DMA framework, co-developed Symbian's ROM-building tools, and worked on the partner-OS solution for single-core dual-OS phones.

Morgan is responsible for the Symbian OS kernel debug architecture for both EKA1 and EKA2, and has worked with many tools vendors, taking the project from research to production.

Before joining Symbian, Morgan dabbled in graphic design, but he now finds himself as a System Architect. He's not sure how that happened. Most recently he has been working on requirements and designs across several technology areas to shape future versions of Symbian OS.

Morgan maintains an active interest in drawing, painting and animation. He holds a BSc in Mathematics and Computer Science from Queen Mary and Westfield College, London.

Tony Lofthouse

Tony joined Symbian in 2002, having previous worked for the Santa Cruz Operation (SCO) Inc. in the Base Drivers team on the UnixWare Operating System and Veritas Inc. in the VxFS file-system team.

At Symbian he has been a lead developer on various Symbian OS hardware reference platform base ports for the Development Boards group. Tony now works in the Base department and is the technical architect for the Development Boards technology stream.

He received a BSc in Computer Science from the University of Southampton in 1996. Prior to this he had a brief career in contaminated land environmental research, but along with sometimes being cold and wet, decided this was too risky.

Dennis May

Dennis May joined Symbian (then Psion) in 1996, having previously worked on RF hardware design and software development for mobile satellite communication systems and on layer 1 software development for a GSM mobile handset.

Dennis initially worked on the development of the Psion Series 5 PDA and subsequently rewrote the Symbian maths library, worked on improvements to the EKA1 kernel and ported Symbian OS to new platforms. He went on to become the original architect and creator of the EKA2 kernel, which he designed to improve on the original kernel in the areas of real-time performance, portability and robustness.

Dennis has made several other contributions to Symbian, including work in the areas of build tools and automated testing, and he is an inventor of patented ideas relating to platform security, file systems, memory management and kernel synchronization primitives. He also had a hand in the design of the ARM architecture 6 memory management unit.

Dennis is currently working as kernel technology architect at Symbian, leading the kernel team in the continued development of the EKA2 kernel.

Dennis received a BA in Mathematics from Trinity Hall, Cambridge, in 1990 and an MSc in Communications and Signal Processing from Imperial College, London, in 1991.

Jason Parker

Jason joined Symbian (then Psion) in 1998, where he became a founding member of the Base Porting Group (BPG). He currently leads BPG and the recently formed Multimedia Porting Group (MMPG) within Symbian's Technical Consulting division.

Jason engages with Symbian semiconductor partners to ensure their SoC designs are optimized for Symbian phones. He and his teams have been instrumental in building many of Symbian's leading-edge phones. They have contributed practical technologies into the EKA2 project, including ROLF (now renamed as ROFS) and the Flash Logger. They built the first working EKA2 prototype phone and are leading the development of commercial EKA2 handsets.

Before joining Symbian, Jason developed real-time software for bespoke embedded systems. Projects included a multi-channel digital video recording system and 3D military simulators.

Jason holds a BSc in Mathematics and Physics from the University of York, and an MSc in Remote Sensing and Image Processing from Edinburgh University. Outside of Symbian, he can be found climbing the world's mountains.

Andrew Rogers

Andrew Rogers joined Symbian in 2002. He then spent time working on Bluetooth, USB, Infrared and OBEX with teams in Cambridge before joining

the base kernel team in London to work on EKA2. Whilst with the base kernel team Andrew migrated the whole OS to a new data type for TInt64 and oversaw the introduction of support for the Symbian User::Leave()/TRAP framework to be implemented in terms of C++ exceptions, including implementing the support for this on the Win32 emulator himself.

More recently, Andrew has spent the last 4 months working in Korea as a member of Symbian's technical consultancy department, but has now returned to Cambridge to work as a consultant on European phone projects.

Andrew has a BA in Computer Science from Sidney Sussex College, Cambridge. In his ''spare'' time he has been known to bring up support on EKA2 for other Win32 compilers and has also been responsible for implementing changes to the IPC mechanism to prevent problems caused by race conditions between connect and disconnect messages, whilst he is not chasing round after his two young children or writing.

Peter Scobie

Peter joined Symbian (then Psion) in 1992 after eight years working for various telecommunications and process control companies – including Dacom, Combustion Engineering and Plessey Telecommunications.

At Psion he initially led the production software team which developed software for the test systems used in the manufacture of the Psion Series 3a. Then he moved into the hand-held computer software department and was part of the original development team for the EKA1 kernel – working on EPOC Release 1 used in the Psion Series 5. During this time he designed and developed the local media sub-system and the PC Card Controller. He then worked on the development of the MultiMediaCard controller and the LFFS file system. He is still working in the base department and is now technical architect for the peripherals technology stream.

Peter has a BSc in Electrical and Electronic Engineering from Loughborough University of Technology. Outside of work Peter enjoys canoeing, sailing and coaching for his eldest son's football team.

Jasmine Strong

Jasmine Strong joined Symbian in 2003, having previously worked for Texas Instruments, designing the system-on-chip architecture of the new generation of OMAP processors. At Symbian, she has worked on performance profiling and improvements for the EKA2 kernel, using her long experience with ARM processors to exploit their special characteristics.

Jasmine has worked in embedded systems since 1998, when she started a project to produce the world's first internet-enabled walk-in freezer

cabinet. Her eclectic career has touched upon many different areas, from hydrocarbon physics to digital television.

Jasmine has been programming ARM processors since the late 1980s and is a keen motorcyclist and photographer. When she's not at work or tearing around at high speeds, Jasmine keeps a weblog.

Andrew Thoelke

Andrew joined Symbian (then Psion) in 1994 and became one of the key developers of OVAL, a rapid application development language similar to Visual Basic, for the Psion Series3a computers. He has since worked on projects throughout the lifetime of Symbian OS, and spanning many of its technology areas such as kernel, data storage, messaging, Java and platform security. He has been deeply involved in the design, development and promotion of EKA2 for the last four years, taking this project from research to production.

Today he has one of the most senior technical roles within Symbian, influencing both the technical strategy of the organization and the ongoing architectural development of Symbian OS.

He graduated from Sidney Sussex College, Cambridge with an MA in Mathematics shortly before beginning his career at Symbian.

Stefan Williams

Stefan joined Symbian in 2002 where he now works in the position of File Server Technical Lead. During his time with Symbian, Stefan has been actively involved in many aspects the Peripherals and File Server sub-systems, with most recent contributions including the design and implementation of the SDIO framework, USB Mass Storage controller and various kernel porting activities.

A graduate of Imperial College, London, Stefan has an MA in Electrical and Electronic Engineering and has previously worked on several major design commissions and commercial products, including PC-based data acquisition and signal analysis software, real-time TDR-based fault analysis systems and distributed embedded network solutions.

Stefan would like to thank his son, Derry, for not complaining too much while Dad spent his weekends writing – and promises to start being fun again!

1

Introducing EKA2

by Jane Sales with Martin Tasker

The ability to quote is a serviceable substitute for wit.
W. Somerset Maugham

1.1 The history of EKA2

Kernel design is one of the most exciting opportunities in software engineering. EKA2 is the second iteration of Symbian's 32-bit kernel architecture, and this in turn follows 8- and 16-bit kernels designed in the 1980s for Psion's personal organizers and PDAs.

Psion's Organiser, launched in 1984, was based on an 8-bit processor and supported only built-in applications. For such a device, the only kernel needed was a bootstrap loader and a small collection of system services. There was no clear requirement to differentiate the 8-bit kernel from middleware or application software.

In 1986, Psion launched the Organiser II, an 8-bit machine offering expansion based on the interpreted OPL language. The demands on the OS were slightly greater – sufficiently good memory management, for example, to support an interpreted language.

A major evolution came when, beginning in 1990, Psion launched a range of machines including a laptop, a clamshell organizer and an industrial organizer, all based on a single OS. The 16-bit EPOC kernel was tied to the Intel 8086 architecture and supported expansion, with applications written not only in OPL, but also in the native C APIs of the EPOC OS – thus opening up the OS itself to any number of aftermarket application writers.

This openness placed massive new demands on the kernel. For one thing, it had to be documented and made accessible to aftermarket programmers. Perhaps some applications would be poorly written: the kernel had to provide memory protection so a bug in one program would

not crash another – or even crash the whole OS. Applications demanded sophisticated memory management for their own working memory. A potentially limitless number of event-driven services had to execute efficiently on a highly resource-constrained machine. And all this had to be delivered on the platform of the 8086's segmented memory model, with challenges that PC programmers of the day will readily recall.

The 16-bit EPOC kernel thus had to address many of the requirements which are met by EKA2 today, because of its positioning between the embedded real-time operating systems and classic desktop operating systems such as Windows. Although it was similar to embedded RTOSes (it ran from ROM), it was bigger because it supported richer functionality and was open to aftermarket applications. Although it was similar to desktop OSes (it was open and used the 8086 architecture), it was smaller because the memory and power resources available were considerably less.

Two further evolutionary steps were necessary to arrive at EKA2.

EPOC32, released in Psion's Series 5 PDA in 1997, began life in 1994. Its kernel, retrospectively dubbed EKA1, carried over the best features of the 16-bit EPOC kernel and fixed several significant issues. Firstly, EKA1 was thoroughly 32-bit – with no relics of the awkwardness in EPOC resulting from the 8086-segmented memory architecture. Secondly, the EKA1 kernel was designed from the beginning with hardware variety and evolution in mind – unlike 16-bit EPOC, which had been tied closely to a single 80186-based chipset. Many implementation details were changed as a result of these fundamentals, but EKA1 was otherwise surprisingly similar in spirit to 16-bit EPOC.

At that time, one of the proudest moments of my career took place – in my spare bedroom! The rest of the team were out of the office, so I worked at home for a week, frantically trying to achieve the first ever boot of the kernel before they got back. And late on the Friday afternoon, the null thread finally printed out its debug message – EKA1 was born.

But EKA1 was not destined to be the end of the story. The Symbian OS system for supporting event-driven programming was efficient overall, but provided no real-time guarantees. The kernel itself was designed with robustness – key for PDAs that hold a user's personal data – as the primary goal. As Symbian OS began to address the processing needs of mobile phones, it became apparent that an OS that *could* provide real-time guarantees was really needed.

There were other influences on EKA2 too. The experience gained from real hardware porting in the context of EKA1 was beginning to demonstrate that EKA1's module boundaries were not always drawn in the right place to make porting easy. Some ports, which should have required only a driver change, in practice required the kernel to be re-built.

So a new kernel architecture was conceived and, to distinguish it from the original 32-bit EPOC kernel, it was named EKA2 (EPOC Kernel Architecture 2), with the term EKA1 being invented for the original.

EKA2 was conceived in 1998 and, little by little, brought from drawing board to market. By 2003, Symbian's lead licensees and semiconductor partners were committed to adopting EKA2 for future products.

This book was written to provide a detailed exposition on the new real-time kernel, providing the reader with the insights of the software engineers who designed and wrote it.

This chapter is designed as the foundations for that book and should give you a good understanding of the overall architecture of the new real-time kernel, and of the reasoning behind our design choices. I will also say a little about the design of the emulator, and then return to this subject in more detail a couple of times later in the book.

1.2 Basic OS concepts

I'd like to start with a basic definition of an operating system (OS):

The operating system is the fundamental software that controls the overall operation of the computer it runs on. It is responsible for the management of hardware – controlling and integrating the various hardware components in the system. The OS is also responsible for the management of software – for example, the loading of applications such as email clients and spreadsheets.

The operating system is usually the first software that is loaded into a computer's memory when that computer boots. The OS then continues the start-up process by loading device drivers and applications. These, along with all the other software on the computer, depend on the operating system to provide them with services such as disk access, memory management, task scheduling, and interfacing with the user.

Symbian OS has a design that is more modular than many other operating systems. So, for example, disk services are in the main performed by the file server, and screen and user input services by the window server. However, there is one element that you can think of as the heart of the operating system – the element that is responsible for memory management, task management and task scheduling. That element is of course the kernel, EKA2.

There are many different flavors of operating system in the world, so let's apply some adjectives to Symbian OS, and EKA2 in particular:

Symbian OS and EKA2 are **modular**. As I've already said, operating system functionality is provided in separate building blocks, not one monolithic unit. Furthermore, EKA2 is modular too, as you can see in Figure 1.1.

EKA2 is **single user**. There is no concept of multiple logins to a Symbian OS smartphone, unlike Windows, Mac OS X, UNIX or traditional mainframe operating systems.

EKA2 is **multi-tasking**. It switches CPU time between multiple threads, giving the user of the mobile phone the impression that multiple applications are running at the same time.

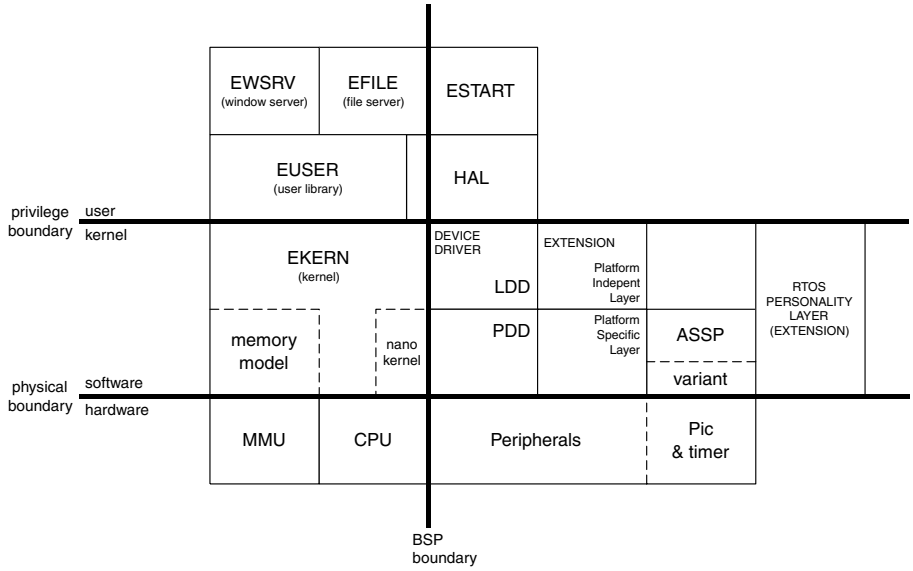

Figure 1.1 Symbian OS overview

EKA2 is a **preemptively** multi-tasking OS. EKA2 does not rely on one thread to relinquish CPU time to another, but reschedules threads perforce, from a timer tick.

EKA2 is a **priority-based** multi-tasking OS with **priority inheritance**. EKA2 allocates CPU time based on a thread's priority and minimizes the delays to a high-priority thread when a low-priority thread holds a mutex it needs.

EKA2 is **real-time**. Its services are (mostly) bounded, that is it completes them in a known amount of time.

EKA2 can be a **ROM-based** OS.

EKA2 is suitable for **open but resource-constrained** environments. We designed it for mobile phones, and so it needs less of key resources such as memory, power and hard disk than open desktop operating systems such as Windows or Linux.

1.3 Symbian OS design

1.3.1 Design goals

When creating EKA2 we set ourselves a number of design constraints. We started by deciding what we didn't want to lose from EKA1. This meant that we wanted to ensure that the new kernel was still:

1. In the embedded OS tradition
2. Suitable for resource-constrained environments

3. Modular: consisting of microkernel and user-side servers
4. Portable to a range of evolving chipsets
5. Robust against badly written user code
6. Of high integrity, ensuring the safety of user data.

Then we decided on our new goals. The key goal was that the new kernel would be real-time and have enhanced overall performance. We decided that we would meet this if we could run a GSM protocol stack on our new operating system. A side benefit, and a worthy one, would be the ability to better support high-bandwidth activities such as comms and multimedia. This goal broke down into several sub-goals and requirements:

1. Latency $\leq$ 1 ms from interrupt to user thread
2. Latency $\leq$ 500 μs from interrupt to kernel thread
3. Fast mutex operations
4. OS calls to be of determined length where possible
5. OS calls to be preemptible
6. Priority-order waiting on semaphores and mutexes
7. Timers with a finer resolution.

Then we considered how else we could improve the operating system, and we came up with the following list:

1. Ease porting – although EKA1 had been designed to be portable, we could go much further to make life easier for those porting the OS to new hardware
2. Be robust against malicious (rather than merely badly written) user code
3. Enable single-core solutions, in which embedded and user-application code run on the same processor core
4. Provide a better emulator for code development and debugging, that emulator being a closer match to real hardware
5. Simplify life for device driver writers.

And as we considered these design goals, we were aware that there was one over-riding constraint on our design. That constraint was to be backwards source compatibility with the EKA1's EUSER class library.

EUSER is the interface to the kernel for all Symbian OS applications, and there are a lot of them out there!

1.3.2 Symbian OS kernel architecture

With those design goals in mind, we designed an operating system whose architecture, at the highest level, looked like that in Figure 1.1. You can see the major building blocks of the kernel. I've also included two other key system components that are usually considered to be part of the operating system, and that I will cover in this book: the file server and the window server. I'll cover each of these building blocks and give you an idea of its basic functionality.

1.3.2.1 Nanokernel

The main function of the nanokernel is to provide simple, supervisor-mode threads, along with their scheduling and synchronization operations. We named the nanokernel as we did because the services it provides are even more primitive than those provided by most embedded real-time operating systems (RTOSes). However, we have carefully chosen those services to be sufficient to support a GSM signaling stack.

The nanokernel is the initial handler for all interrupts. It then passes the majority of them to the variant layer for dispatch. It also provides simple timing functions, such as the nanokernel timer (`NTimer`) API, which gives a callback after a specified number of ticks, and the sleep API (`NKern::Sleep`), which makes the current thread wait for a specified number of ticks.

The simple synchronization objects I mentioned earlier are the nanokernel mutex (`NFastMutex`) and the nanokernel semaphore (`NFastSemaphore`). Both of these forbid more than one thread from waiting on them.

Finally, the nanokernel provides deferred function calls (DFCs) and the oddly named immediate deferred function calls (IDFCs). If you want to find out more about these, then please turn to Chapter 6, *Interrupts and Exceptions*.

An important difference in EKA2 from EKA1 that should be noted is that neither the nanokernel nor the Symbian OS kernel link to the user library, EUSER. Instead, the nanokernel uses its own library of utility functions, and makes these available to the rest of the kernel, and device drivers too.

Another key difference from EKA1, somewhat related to the one I have just discussed, is that EKA2 does not support a kernel-side leaving mechanism. This means that errors are reported by returning an error code – or panicking the thread.

The majority of the time, the nanokernel is preemptible. Usually it runs unlocked and with interrupts enabled, but we do have to prevent

other threads from running in a few sections of code, such as thread state changes and access to the ready list. We designed these critical sections to be as short as possible and to have bounded execution times, the goal being to maintain deterministic real-time performance. We protect the critical sections in the nanokernel by disabling preemption – this is possible because these sections are very short. In general, we use a mutex known as the system lock to protect critical code in the Symbian OS kernel and memory model, but the only place where the nanokernel uses this lock is to protect the scheduler's address space switch hook on the moving memory model.

What are the limitations on the nanokernel? The main one to note is that it does not do any dynamic memory allocation; that is, it can't allocate or free memory. In all of the nanokernel's operations, it assumes that memory has been preallocated by other parts of the operating system.

1.3.2.2 Symbian OS kernel

The Symbian OS kernel provides the kernel functionality needed by Symbian OS, building on the simple threads and services provided by the nanokernel to provide more complex objects, such as user-mode threads, processes, reference-counted objects and handles, dynamically loaded libraries, inter-thread communication and more.

These objects also include a range of more sophisticated synchronization objects: Symbian OS semaphores and mutexes. Symbian OS semaphores are standard counting semaphores which support multiple waiting threads and which release waiting threads in priority order. Symbian OS mutexes are fully nestable (a thread can hold several mutexes at once, and can hold the same mutex multiple times). They also support priority inheritance: the holding thread inherits the priority of the highest priority waiting thread, if that is higher than its usual priority.

In contrast to the nanokernel, the Symbian OS kernel does allow dynamic memory allocation. It provides a kernel memory allocator – the kernel heap, which uses low-level memory services provided by an entity known as the memory model. The Symbian OS is completely MMU agnostic – we isolate all assumptions about memory to the memory model, which I describe in more detail in the next section.

The Symbian OS kernel is fully preemptible: an interrupt can cause it to reschedule at any point in its execution, even in the middle of a context switch. This means that the Symbian OS kernel can have no effect whatsoever on thread latency.

We use system lock mutex, provided by the nanokernel, to protect the most fundamental parts of the Symbian OS kernel, such as:

(i) The state of `DThread` objects. When Symbian OS threads interact with semaphores and mutexes, they undergo state transitions that are protected by the system lock

(ii) The state of most Symbian OS synchronization objects: IPC (servers and sessions), semaphores, mutexes, message queues, publish and subscribe properties

(iii) Handle arrays are valid for reading (but not writing) when the system lock is held. All the executive functions that take a handle hold the system lock while translating it – see Chapter 5, *Kernel Services*, for more on this subject.

1.3.2.3 Memory model

In EKA2, we confine our assumptions about the memory architecture of the ASIC to one module, the memory model. Thus the memory model encapsulates significant MMU differences, such as whether a cache is virtually tagged or physically tagged, and indeed, whether there is an MMU at all. In EKA1, assumptions about memory and the MMU were spread throughout the operating system, making it difficult to produce a mobile phone based on an ASIC without an MMU, for example. This has become much easier with the advent of EKA2, since the memory model allows you to model memory in different ways, and to change that decision relatively easily.

Symbian currently provides four different memory models:

1. Direct (no MMU)
2. Moving (similar to EKA1)
3. Multiple (used for ASICs with physically tagged caches such as Intel X86 and later ARM cores)
4. Emulator (used by the Symbian OS emulator that runs on Windows).

The memory model provides low-level memory management services, such as a per-process address space and memory mapping. It performs the context switch when asked to do so by the scheduler and is involved in inter-process data transfer.

The memory model also helps in the creation of processes as an instantiation of an executable image loaded by the file server, and takes part in making inter-process data transfers.

If you are interested in finding out more about the memory model, turn to Chapter 7, *Memory Models*.

1.3.2.4 Personality layer

We designed the nanokernel to provide just enough functionality to run a GSM signaling stack. The idea behind this was to allow mobile phone manufacturers to run both their signaling stacks and their personal information management (PIM) software on a single processor, providing considerable cost savings over the usual two-processor solution.

Most mobile phone manufacturers have written their signaling stacks for existing RTOSes such as Nucleus or μITRON. These signaling stacks represent a considerable investment in time and money, and it would be very time-consuming for the mobile phone manufacturers to port them to the nanokernel – not to mention the increase in defects that would probably ensue from such an exercise.

Because of this, we designed the nanokernel to allow third parties to write personality layers. A personality layer is an emulation layer over the nanokernel that provides the RTOS API to client software. The personality layer would translate an RTOS call into a call (or calls) to the nanokernel to achieve the same ends. In this way, we allow source code written for that RTOS to run under Symbian OS with little or no modification.

For a more detailed description of personality layers, and the nanokernel design decisions that support them, turn to Chapter 17, *Real Time*.

1.3.2.5 ASSP/variant extension

Typically, the CPU and the majority of hardware peripherals on mobile devices are implemented on a semiconductor device integrated circuit commonly referred to as an ASSP (Application-Specific Standard Product). To reduce both the bill of materials and the size of a phone, it is becoming common to add an increasing number of components to the ASSP. This might include stacking RAM and flash components on the same silicon package, or incorporating components into the silicon layout; for example, a DSP (digital signal processor) for audio/video processing, dedicated graphics processors and telephony baseband processors running GSM or CDMA communication stacks.

We refer to any hardware components outside the ASSP as variant-specific components. These typically include components such as flash and RAM storage technology, display devices, baseband and Bluetooth units. They are typically interfaced to the processor over semiconductor-vendor-specific buses and interconnect, or more standard communications lines such as USB and serial UARTs. ASSPs also tend to provide configurable GPIO (general purpose I/O) lines for custom functions such as MMC card detect and touch-screen pen down interrupt lines.

So, in Symbian OS, the ASSP/variant extension provides the hardware-dependent services required by the kernel – for example, timer tick interrupts and real-time clock access. In the days of EKA1, we built the ASSP into the kernel, and the separate variant layer described in the next section was mandatory. This made for unnecessary re-compilation of the kernel when porting to a new ASSP, so in EKA2 we have completely separated the ASSP from the kernel. Of course, this means that if you are porting EKA2, you no longer need to recompile the kernel every time you tweak your hardware.

1.3.2.6 Variant

In EKA2, we don't insist that you make a division between the ASSP and the variant, as we do in EKA1. You may provide one single variant DLL if you wish. Nevertheless, if you were porting the OS to a family of similar ASICs, you would probably choose to split it, putting the generic code for the family of ASICs in the ASSP extension, and the code for a particular ASIC in the variant DLL. For example, within Symbian, the Intel SA1100 ASSP has two variants, Brutus and Assabet.

1.3.2.7 Device drivers

On Symbian OS, you use device drivers to control peripherals: drivers provide the interface between those peripherals and the rest of Symbian OS. If you want, you may split your device driver in a similar way to the ASSP and variant, providing a hardware-independent logical device driver, or LDD, and a hardware-dependent physical device driver, or PDD.

Device drivers may run in the client thread or in a kernel thread: our new multi-threaded kernel design makes porting device drivers to Symbian OS from other operating systems much easier.

Symbian provides standard LDDs for a wide range of peripheral types (such as media devices, the USB controller and serial communications devices) – nevertheless, phone manufacturers will often develop their own interfaces for custom hardware.

Device drivers have changed considerably from EKA1 to EKA2. See Chapter 12, *Drivers and Extensions*, for more details.

1.3.2.8 Extensions

Extensions are merely device drivers that the kernel automatically starts at boot-time, so you can think of them as a way to extend the kernel's functionality. For example, the crash debugger is a kernel extension, allowing you to include it or exclude it from a ROM as you wish, without having to recompile the kernel.

The variant and the ASSP that I discussed earlier are important extensions that the kernel loads quite early in the boot process. After this, the kernel continues to boot until it finally starts the scheduler and enters the supervisor thread, which initializes all remaining kernel extensions. Extensions loaded at this late stage are not critical to the operation of the kernel itself, but are typically used to perform early initialization of hardware components and to provide permanently available services for devices such as the LCD, DMA, I2C and peripheral bus controllers.

The final kernel extension to be initialized is the `EXSTART` extension, which is responsible for loading the file server. I discuss system boot in more detail in Chapter 16, *Boot Processes*.