1

Introduction to Mobile Phone Systems

COPYRIGHTED MATERIAL

The phrase 'viewing the world through rose-tinted glasses' finds its origins in literature at least as far back as 1861. The phrase implies that 'viewers' have a different – usually optimistic – view of the world from the 'standard' view, as if they are seeing it through a set of nicely tinted lenses. Computer operating systems are like tinted glasses, allowing the viewer to see a collection of hardware and software – memory, disk drives, CPU chips, Bluetooth transmitters, email programs and telephony applications in an ordered and controllable way: as a set of resources that can be harnessed to accomplish various tasks. An operating system is the model through which a computer's hardware and software can work together and the structure that provides controlled access between them.

Consider the many different sets of 'tinted glasses' that are in use today for manipulating computing resources. Many of today's hardware platforms are used by multiple operating systems. For example, Intel-based hardware, such as the Pentium family of CPUs, can support several different operating systems. The Microsoft Windows family of operating systems represents a set of many different operating systems – from Windows 95 to Windows XP – that run on the same hardware platform. The Linux operating system and BeOS provide other examples. These different systems form a set of different models of resource allocation and usage that operate on the same hardware. These operating systems are very different in how they view a computer system, but they are very much the same in many respects.

This book takes a close look at the variety of operating systems with a focus on a specific type of operating system: that of mobile

phones. Mobile phone operating systems must embrace conventional system components as well as additional components crucial to mobile phones: communications and interface design. We look at each of these additional components. To be more specific, this book looks at mobile phone operating systems by examining Symbian OS. Symbian OS is an operating system that was designed from its beginnings to be implemented on mobile phones. Its design comprises conventional operating system modeling, employs a strong communications model and has a very flexible user interface model. Its origins are found in handheld computing and its usage on mobile platforms is growing dramatically. (It is predicted that, by 2008, half of all mobile phones will have a full-featured operating system, such as Symbian OS, running them.)

It is difficult to study mobile phone operating systems, even given the plethora of mobile phones, without also looking at conventional operating systems. We examine operating systems that power servers and desktop systems. We compare Symbian OS to these conventional systems, especially by comparing it to Linux.

In order to study operating systems, we must first define what an operating system is and understand the divide between an operating system and a hardware device. This chapter defines operating systems and the components that make them up. It then looks at the history of operating systems, including a history of Symbian OS. It finishes by looking at how operating systems fit onto various computing platforms.

1.1 What Is an Operating System?

There are many definitions of an operating system. All definitions agree on several points. First, an operating system is a software program. No matter where it is stored – on a hard drive, in ROM, on compact flash storage – an operating system is eventually loaded into a computer's memory and its instructions are executed just like any other software program.

Secondly, an operating system is a resource model. Operating systems are designed to present the various hardware resources of a computer to software and to a user. An operating system builds a model, a system, of how to deal with the resources of a computer. Software must work with this model to access and use those resources. The model provides a lens through which users view resources such as the communications system and the user interface.

Thirdly, an operating system binds the hardware and the software together. Because it presents the hardware to the software, an operating system is the glue that holds the two sides together. The software sees and accesses the hardware as it is presented through the operating system model. The hardware deals with the software through the same operating system model. A good operating system is based on an intuitive model that allows effective communication between the software and the hardware.

Finally, an operating system is essential. Without an operating system, a computer would not function. Its software could not be executed; its hardware would not be utilized. Any general-purpose computer has an operating system in some form. Thus, learning about operating systems means learning about an essential part of the computer.

The Operating Environment

To understand operating systems as the glue between hardware and software, let us examine these two elements and how they relate through the operating system.

Hardware is the physical part of the computer. It is the set of all the tangible components that provide the operational foundation for the software. Software is the set of programs and applications that execute their instructions on the hardware. A software program must use hardware in some way – for input, for output or to operate the hardware somehow.

Consider the example of a message manager application running on a mobile phone (see Figure 1.1). It collects text messages as they arrive, analyzes each message and responds to certain ones as the application's user has specified. The hardware receives radio signals and notifies the operating system that data is arriving. The operating system engages the sending source by working with the radio hardware to receive a text message using the appropriate data protocol. Once the complete message has arrived correctly, the operating system stores the message and notifies the message manager. The message manager application uses the operating system to access the stored message – which requires the operating system to interact with the hardware. The manager reviews the message and takes some kind of action, perhaps deleting the message or making an automatic reply. The automatic reply again requires the operating system to create a new message and to access the hardware for storage and transmission of the new message.

It is important to realize here that neither the hardware nor the software sees an operating system. The hardware is following a prescribed set of

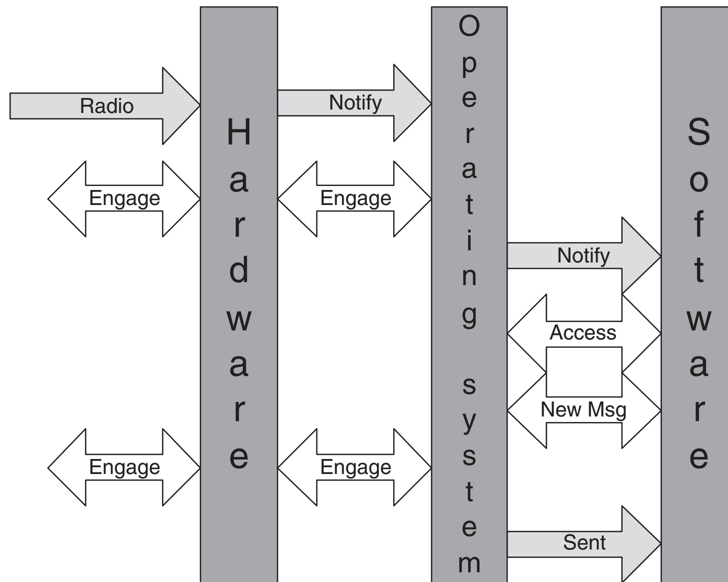

Figure 1.1 The relationship between hardware, operating system and software

instructions built into its memory. The software is using an application programming interface (API) to manipulate text from storage and to compose and send a message. Both sides see a different picture, yet both sides are drawn together and work to accomplish a joint goal. The operating system acts as the go-between and provides an operational picture to both sides.

The focus of a mobile phone is in the software that enables a user to use it. It is software that enables a user to make a phone call, send a message, set an alarm, or write on the display with electronic 'ink'. The user of the phone realizes that the hardware exists – it is in his hand, after all – but is most likely not aware of the operating system. A good operating system is transparent, allowing the user to use the software to interact with the hardware without showing its own face.

A Resource Model

The focus of an operating system is on providing ways for the software to use the hardware to do what the user wants. It is the goal of an operating system to make this happen seamlessly and transparently. Essentially, the

operating system must provide the software with an accessible model of the hardware. The hardware must become a set of resources to be operated by the software. Management of that hardware resource is the job of the operating system.

Software manipulates hardware resources through an application programming interface. APIs can be provided by the operating system designer or by a third party. Software does not usually work with hardware directly, but manipulates resources by communicating with the operating system through a function call interface. The operating system builds a model of the hardware and provides system function calls that access that hardware model in specific ways (see Figure 1.2).

Consider the previous message manager example. The operating system has many choices to make as it works with messages. It could, for example, store the message text in a file and give an application a way to find the file name and to work with that file directly. The application would have to open the file (again, through the operating system resource model) and process the raw message data. Another way to present the message would be to store the message in a file, but present an application with an abstract object called a 'text message' that the application could work with. The application would make function calls that the operating system would intercept, deriving information about the message and returning that information. The application would not be aware of where the object was stored. These are two models of message handling: one

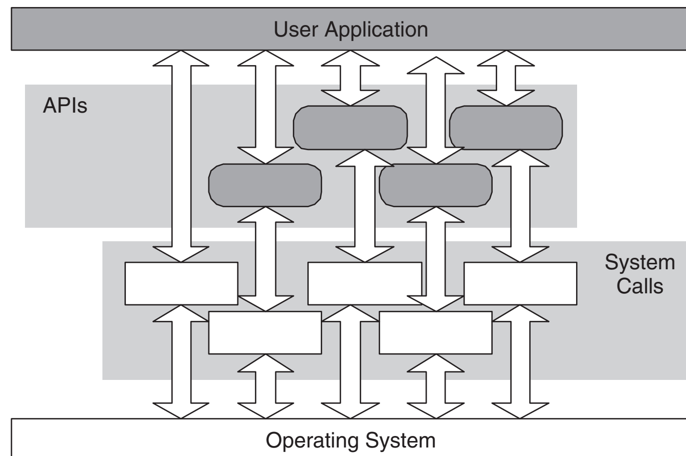

Figure 1.2 Structure of access to an operating system

more raw and direct, the other more abstract and object-oriented. The choice that the operating system makes about which one to use builds the character of the operating system.

A good system model is one that effectively and transparently provides software with an intuitive way to access system resources. System models are often based on *abstraction*. Abstraction involves the hiding of irrelevant data and the presentation of only useful, relevant information. We often label abstractions as 'objects'. For example, system resources are the abstract objects that the operating system presents to the software. They might represent a resource as a hardware object, with the detail abstracted away, or as a set of functions that can use the hardware. A file is an abstract object that represents a way to use hardware storage. A text message is an abstract object that represents a way to use both software and hardware resources to access that message. These are concepts built and supported by the operating system and provided to applications.

A good operating system has more goals than simply providing a useful model to software applications.

- *Robustness*: a good operating system is reliable and tolerates problems well. The system does not stop working due to isolated hardware or software errors and fails gracefully if it must deal with several errors at the same time. Robust operating systems provide services to software unless the hardware fails.

- *Scalability*: a good operating system incorporates resources as they are added to the system. This can be transparent to the user – the best way – or can involve some kind of user interaction. The plug-and-play concepts of Microsoft Windows – where devices are discovered and installed automatically – is an example of good scalability. On the other hand, old versions of Linux used to require recompilation of the operating system when new devices were added. This is an example of bad scalability.

- *Extensibility*: the operating system should be designed to adapt to new technologies that extend the operating system beyond the point at which it was implemented. For example, it should be able to adapt to new forms of file storage without a complete redesign of the operating system.

- *Throughput (the work that a processor can complete in a specific time period)*: an operating system must perform well and achieve

high throughput. A good operating system minimizes the time spent providing services while maximizing throughput.

- *Portability*: a good operating system should be portable, that is, able to be run on many different hardware platforms.
- *Security*: an operating system must be secure. It must prevent unauthorized users and processes from accessing stored data and system services.

Many Operating Systems Fit the Bill

Even though the list of criteria for a good operating system looks a bit daunting, many operating systems have been created over the years that meet these criteria. In addition, many operating systems did some of these very well and steered the industry in one particular area. A list of operating systems can be found at ***http://en.wikipedia.org/wiki/List_of_operating_systems***.

Many operating systems are not very portable. They are specifically designed to run on a single platform. In addition, you will note that 'popularity' is not an item on the criteria list. Most operating systems were not popular, yet were designed to address a specific system model.

1.2 History of Operating Systems

Operating systems are the heart of every general-purpose computer. Since 1957, operating systems have been an essential component of computers. This section outlines a brief history of operating systems, highlighting the history of Symbian OS.

General-Purpose Operating Systems

The earliest computers did not have operating systems. They were dedicated computing devices that performed a single task, thereby needing only one 'program' to execute. From the ancient Incas in Central America to the Difference Engine constructed by Charles Babbage in 1847 to the early days of modern computing (the ENIAC in 1946, the Mark I in 1948), early computers focused on single tasks that had direct access to hardware and no operating system.

Operating systems were invented when it became clear that access to 'the system' needed to be standardized. Until the mid-1950s, programmers wrote their own routines for accessing resources, particularly system input and output. Patterns of programming were beginning to emerge, such as repeated use of certain mathematical functions. The need for basic, standardized operating system functionality, including device drivers and execution libraries, was becoming apparent. Critical mass was reached as computer systems were designed to allow queuing of jobs, or programs, to run one after the other.

The first operating system was released in 1957. Called BESYS, this operating system was implemented by Bell Labs to handle the execution of many short programs, queued up so that the operators did not have to load each program just prior to its execution. BESYS shared CPU time between several jobs at once, thus making it the first multitasking operating system.

Operating system research and implementation moved very fast in the 1960s. Two influential examples were OS/360, released by IBM in 1964, and MULTICS, released by Bell Labs, MIT and General Electric in 1965.

OS/360 was influential because it combined a powerful command language with the ability to run many jobs at once. The command language controlled job execution and specified how each job was to access resources. In addition, OS/360 worked on various computer models; it became the standard among batch processors.

MULTICS was influential because it took a very different approach from OS/360: it allowed users to use the operating system directly. It had a unique structure – using a central core of software called a 'kernel' – and allowed users to extend the operating system through software based on the kernel. Based on the foundational ideas introduced in MULTICS, Unix was invented at Bell Labs by a man named Ken Thompson in 1972. Thompson teamed with Dennis Ritchie, the author of a programming language called 'C', to produce the source code of the Unix operating system in that language. Unix was distributed almost free of charge and, in the 1970s, it spread to many platforms.

Since the spread of Unix, there have been many developments in operating systems. One of the biggest was brought about by a development in computers: the personal computer. The ideas invented by MULTICS and honed by Unix were streamlined to fit into a personal computer with the introduction of MS-DOS in 1981. MS-DOS ran on an IBM PC using the Intel 8088 chipset. Its first version was indeed primitive, but as hardware resources were improved upon and faster processors with more memory

were packaged as desktop computers, MS-DOS evolved into Microsoft Windows and has taken on many of the foundational concepts embedded in Unix.

As we look at the evolution of operating systems, it is interesting to see the progression of computer resources that also evolved:

- computers started by running one task at a time and have progressed to running many tasks at the same time
- storage hardware has evolved from needing a large physical size for only 100 KB of data to packing 100 GB into a matchbox-sized disk
- electronic storage has made access much faster
- memory has progressed from only a few kilobytes to many gigabytes; even handheld and mobile phone platforms sport 128 MB (and larger) memories
- communication has gone from none to a large collection of possibilities: wired and wireless, serial and parallel, radio and infrared.

Operating systems have developed to take advantage of all of these aspects of computer hardware.

Symbian OS

Handheld devices were developed in the late 1980s as a way to capture the usefulness of a desktop device in a smaller, more mobile package. Although the first attempts at a handheld computer (for example, the Apple Newton) were not met with much excitement, the handheld computers developed in the mid-1990s were better tailored to the user and the way that they used computers ‘on the go’. By the turn of the 21st century, handheld computers had evolved into smartphones – a combination of computer technology and mobile phone technology. Symbian OS was developed specifically to run on the smartphone platform.

The heritage of Symbian OS begins with some of the first handheld devices. The operating system began its existence in 1988 as SIBO (an acronym for ‘16-bit organizer’). SIBO ran on computers developed by Psion Computers, which developed the operating system to run on small-footprint devices. The first computer to use SIBO, the MC laptop machine, died when it was barely out of the gate, but several successful computer models followed the MC. In 1991, Psion produced the Series 3: a small

computer with a half-VGA-sized screen that could fit into a pocket. The Series 3 was followed by the Series 3c in 1996, with additional infrared capability; the Sienna in 1996, which used a smaller screen and had more of an 'organizer' feel; and the Series 3mx in 1998, with a faster processor. Each of these SIBO machines was a great success, primarily for three reasons: SIBO had good power management, included light and effective applications, and interoperated easily with other computers, including PCs and other handheld devices. SIBO was also accessible to developers: programming was based in C, had an object-oriented design and employed application engines, a signature part of Symbian OS development. This engine approach was a powerful feature of SIBO; it made it possible to standardize an API and to use object abstraction to remove the need for the application programmer to worry about data formats.

In the mid-1990s, Psion started work on a new operating system. This was to be a 32-bit system that supported pointing devices on a touch screen, used multimedia, was more communication-rich, was more object-oriented, and was portable to different architectures and device designs. The result of Psion's effort was the introduction of EPOC Release 1. Psion built on its experience with SIBO and produced a completely new operating system. It started with many of the foundational features that set SIBO apart and built up from there.

EPOC was programmed in C++ and was designed to be object-oriented from the beginning. It used the engine approach pioneered by SIBO and expanded this design idea into a series of servers that coordinated access to system services and peripheral devices. EPOC expanded the communication possibilities, opened up the operating system to multimedia, introduced new platforms for interface items such as touch screens, and generalized the hardware interface. EPOC was further developed into two more releases: EPOC Release 3 (ER3) and EPOC Release 5 (ER5). These ran on new platforms such as the Psion Series 5 and Series 7 computers.

As EPOC was being developed, Psion was also looking to emphasize the ways that its operating system could be adapted to other hardware platforms. From mobile phones to Internet appliances, many devices could work well with EPOC. The most exciting opportunities were in the mobile phone business, where manufacturers were already searching for a new, advanced, extensible and standard operating system for its next generation of devices. To take advantage of these opportunities, Psion and the leaders in the mobile phone industry – for example, Nokia,

Ericsson, Motorola and Matsushita (Panasonic) – formed a joint venture, called Symbian, which was to take ownership of and further develop the EPOC operating system core, now called Symbian OS.

Symbian OS was explicitly targeted at several generalized platforms. It was flexible enough to meet the industry's requirements for developing a variety of advanced mobile devices and phones, while allowing manufacturers the opportunity to differentiate their products. It was also decided that Symbian OS would actively adopt current, state-of-the-art key technologies as they became available. This decision reinforced the design choices of object orientation and a client–server architecture.

1.3 Computer Systems and their Operating Systems

In addition to following computers and their history, a different way to appreciate the relationship between operating systems and hardware is to look at them from a system perspective. Each type of computer system has an operating system that was designed for it – to take advantage of its unique features.

Mainframe Systems

Mainframe systems are characterized by a large central computer with a large number and wide variety of possible peripherals. These types of computers were the first to be used to run scientific and commercial applications.

Initially, mainframe systems needed to run only a single program at a time. The operating system would accept *jobs* – packages consisting of control commands, program code and data. The control commands dictated how to compile the program, how much memory it would take, what other resources would be used, etc. Operating systems for these types of computers could be quite simple. An operating system needed to read in the job, use the control commands to configure how the program would be loaded up and executed, and manage the program's access to resources and data. When a program executed, the operating system would remain in memory, tucked away in its own section. The BESYS operating system was created in this environment.

Mainframe systems became more complex for two reasons. First, running multiple jobs in sequence became desirable. A sequence of jobs – called a *batch* – would be sorted into groups based on what

resources would be used. Often, using a resource required that the resource be on and configured in a certain way. Secondly, disk technology developed to the point where jobs could be placed on a disk drive rather than recorded on punched cards. This was a great step forward, because mistakes were easier to correct, and jobs could be submitted and processed more rapidly. Once disk access was available, an operating system could sort the jobs and choose which was most appropriate to run at a given time. This type of *job scheduling* allowed more efficient use of computer resources in addition to faster turnaround time for program execution.

In this kind of environment, idle time becomes an issue. There was a large difference between the speed of the CPU processor and the I/O speed of each device connected to the computer. Therefore, as the CPU accesses a device, much waiting is involved. This problem was exacerbated by the fact that older mainframes would run a single job at a time.

Eventually, mainframes and their operating systems came to embrace two more concepts: *multiprogramming* and *time-sharing*. To take advantage of the waiting time of a CPU, operating systems were designed to schedule multiple jobs at once. These several jobs would share the CPU: when one job caused the CPU to wait, another job took its place and executed on the CPU. This type of multiprogramming – where multiple programs ran on a single CPU – extended the idea of job scheduling to include *CPU scheduling*. This multiuse environment has several implications for memory and for I/O.

Time-sharing is an extension of CPU scheduling. If you consider a user interacting with a computer as just another job, then multiple users can interact with the computer at the same time. Time-sharing refers to the way that users share the CPU with other tasks, both other users and other jobs. OS/360 was implemented to support this kind of environment: a time-sharing, job-scheduling computing environment. Users would interact with the computer by creating jobs through terminals, saving them, then submitting them online to the computer. Output from these jobs was eventually generated and delivered to the user for consideration.

Mainframe systems shrank in size and eventually became small enough to put into a room with very little cooling equipment. The user interaction software evolved as well. The job-control program eventually became a *command shell*, a program that accepted commands interactively from a user, executed those commands and placed the output back on the screen. MULTICS was created in this environment and Unix perfected

the use of this kind of interaction. Multiprogramming was the norm in these operating systems and all 'jobs' – including the user-command shell – competed for system resources, especially the CPU. Issues that affected performance – such as which job got priority and algorithms to effectively schedule all usage of the CPU – became very important and widely discussed.

Desktop Systems

Computers continued to shrink until it was feasible to combine a monitor, a CPU and a keyboard into a single package that could occupy a desktop. These systems distributed computing power to users, rather than having users access the computing power of a single machine.

IBM constructed the first personal computer; MS-DOS was the operating system that was used for this first PC. Initially, MS-DOS was a single-job operating system. Like the old mainframes, it ran a single job (now called a *process*) at a time and the operating system made choices about which job to run and how to manage resources. Hardware systems grew faster and supported more peripherals; operating systems, like those supporting mainframes, grew and added features to support these hardware systems. MS-DOS eventually incorporated multiprogramming and could support multiple processes using the CPU. As graphical user interfaces became more widely used to interact with the computer (in the place of a command shell), MS-DOS was upgraded to become Microsoft Windows and other operating systems, such as MacOS from Apple, emerged.

Desktop systems now support multiprogramming, time-sharing, networking and many types of peripherals. These systems assume that they exist in an environment that is shared by multiple PCs and multiple users. The operating systems embrace many users at once and encourage users to venture out over networks to share resources from other computers.

Distributed Systems

A distributed system is an extension of multiple connected stand-alone systems. These systems depend on each other to varying degrees. Some distributed systems simply share a few resources – such as printers and disk drives – while others share many resources – such as CPU time and input devices. Distributed systems assume that they are connected by some sort of communication network.

There are several models of distributed systems that operating systems have taken advantage of. The *client–server* model views some computers as servers, that is, providing a service of some sort, and some computers as clients that ask for and receive a service. Web browsing is a distributed activity that is based on the client–server model. Browsers are clients that ask servers for pages. *Peer-to-peer distribution* is a model in which computers are both servers and clients, using some and being used by others. The *interdependent* model is a peer-to-peer model where peers are tightly interconnected, such that they cannot operate if other peers are not also functioning. In the interdependent model, each peer has functions that are crucial to the entire network's operations.

There are several examples of operating systems for distributed computing systems. Good examples of the client–server model are the many distributions of Linux. The appeal of Gentoo Linux is that it is solely based on the Internet for its distribution. It uses the Internet for upgrading itself, for installing itself and for updating its applications. For these uses, the operating system is a client, communicating with one of many Gentoo servers.

For an example of an interconnected distributed operating system we have to go back to the 1980s. During those years, an operating system called Domain/OS was implemented that ran on computers made by the Apollo company. Domain/OS was a version of Unix that was truly distributed between computers on a network. The execution of a command or program might occur on the local computer a user was connected to or it might occur on another computer in the network. No matter where the command was executed, the results – text or graphics – appeared on the local screen. The decision about which specific computer executed any given command was based on an algorithm, which made the location decision based on factors such as load and network performance.

Handheld Systems

As computers inexorably shrank in size, handheld devices became feasible. These computers – usually fully fledged systems with all the peripherals and issues of desktop systems – fit into and can be used with one hand. At first glance, these systems look as if they could simply take on the operating systems of their bigger siblings, but they pose some unique challenges.

First, the internal environment is more restrictive. Less memory, less storage space and slower processors all dictate that the operating system

must be tailored for a handheld environment, not just shrunk. Often, memory becomes 'disk' space: memory space is shared between a storage system and memory used by the system to run programs. The early Palm handhelds had 2 MB of memory for operating system space and file storage. In the face of these restrictions, the conventional models of operating systems change to accommodate the different environment.

Secondly, resources must be handled with more care. The resources on a handheld platform are more fragile – in the sense that a restricted environment puts more of a load on a resource. A restricted environment leaves less room for software to protect a resource. This means that an operating system must have a good model in place for dealing with resource access from multiple sources.

Thirdly, power restraints are crucial. While desktop systems are always connected to AC power, handheld systems are almost always run on batteries. Extensive running of hardware resources drain battery life dramatically. And power loss must be handled gracefully.

These considerations mean that an operating system must be written specifically for a handheld device. It faces many pressures; it must support the multiprogramming of a desktop system in a (sometimes severely) restricted environment that must sip battery power while coordinating access to many resources. This is a considerable task, but operating systems have risen to handle it. Linux has been scaled to fit on several handheld devices. Microsoft Windows has also been fitted for handheld platforms. The early versions of Symbian OS were designed for a handheld environment.

Mobile Phone Systems

As even handheld devices got smaller, it became possible to fuse a handheld device with a mobile phone. All the considerations of a handheld platform are multiplied when a handheld device becomes a communications tool. All the restrictions and issues are present while the system requirements take on communication issues as well. The resource model of the handheld platform is now augmented with communications and the functionality that comes with those communications.

On a mobile phone, the environment restrictions can be even more severe than on a handheld device. The data requirements of multimedia communication – text messages, phone calls, photographs, video clips and MP3s – are tremendous, yet must fit onto a restricted storage space. A mobile phone now has even more resources that must be carefully dealt

with. And power is even tighter than normal, as the power requirements of a mobile phone are much higher than that of a handheld device.

In the face of even tighter constraints, operating systems have risen to the challenge. Several operating systems, such as Symbian OS, have been tailored for mobile phones.

Real-time Systems

A real-time system is a special-purpose computer system where rigid time requirements have been placed on either the processor or input/output operations. These time constraints are well-defined and system failure occurs when they are not met.

Real-time systems come in two varieties. Hard real-time systems guarantee that time constraints are met. Soft real-time systems place a priority on time-critical processes. In both cases, real-time systems have a specific structure. Any time-consuming task or device is eliminated and real-time service often comes from a dedicated computer. Disk drives or slow memory cannot be tolerated. All system services – hardware or software – must be *bounded*; that is, they must have specific response-time boundaries or they cannot be used.

In a sense, some mobile phone functions are real-time functions. The service of a phone call, for example, is a real-time service. But most functions of a mobile phone can be carried out by a non-dedicated, general-purpose operating system designed for the mobile phone platform.

Symbian OS was not initially a real-time operating system but the latest versions (Symbian OS v9 onwards) are powered by a real-time kernel.

1.4 Summary

This chapter has introduced the idea of an operating system and its relationships to both hardware and software. We defined what an operating system is and discussed the modeling that an operating system does for both hardware and software. We examined the operating systems from a historical perspective and an operational perspective.

The next chapter considers the character of operating systems. It discusses some of the common features of operating systems as they exist today and makes some working definitions that we use throughout the book. We also take a much closer look at the central operating system of this book: Symbian OS.