Adrian Kosmaczewski
University of Liverpool
MSc in IT, Software Engineering

The QNX Realtime Operating System

April 4th, 2007

Contents

List of Figures

Chapter 1

Introduction

This report provides an overview of the **QNX** operating system, highlighting its main capabilities and scope, as well as some details about its inner mechanisms, particularly in the five areas of management: Process, Resource, Memory, Devices and Storage.

1.1 About QNX

QNX is a commercial, POSIX-compliant, realtime, embedded operating system, developed by QNX Software Systems from Canada. Its development started in 1982, and its latest version[1] is 6.3.2, released on September 28th, 2006 (Wikipedia, 2007).

QNX is a leading embedded operating system in several niche markets, and it is used by many major corporations and organizations (QNX, 2007*h*):

- Automotive & Transportation: Cogent, BCI, JVC
- Communications: British Telecom, WorldSpace, Wavetek
- Consumer Electronics and Domotics: Epson, Teligent
- Defense & Security: Neptec, Lockheed Martin, NASA
- Industrial Automation & Control: Motorola, US Postal Service
- Medical Devices: Burdick, Siemens

QNX is a multi-purpose operating system, that can work not only as an embedded system but also as a server or desktop operating system, as shown in figure 1.1, showing the Photon microGUI, a desktop user interface bundled with **QNX** .

[1] At the time of writing

NASA has trusted **QNX** enough to use it as the embedded operating system of choice for its first Space Shuttle mission in 2005 after the 2003 Columbia accident:

> "The LCS is a critical element of NASAs Return to Flight mission and we have to be sure it is running on the most reliable operating system available," said Iain Christie, vice president of research and development at Neptec. "Selecting the QNX Neutrino RTOS was an easy decision because we already know that the system can handle the extreme conditions found in space and that it meets our demands for ultra-reliability. We will continue to use QNX technology in all of our realtime embedded projects." (QNX, 2005)

1.2 Version History

The following table shows the release dates of different versions of QNX (OSDP, 2007):

1981	QUNIX
1984	QNX 1.0
1987	QNX 2.0
1990	QNX 4.0
1996	QNX Neutrino 1.0 (RTP)
2001 Jan.	QNX RTP 6.0, first time for private customers free of charge
2004 June	QNX Neutrino 6.3.0

1.3 Pricing & Licensing

QNX 6.2 is free for private and non-commercial use, and can be freely downloaded from the QNX Software Systems' website. Single-processor licenses for the Professional edition of **QNX** are available from USD 8'000.-, and volume discounts are available as well (OpenQNX, 2004).

1.4 Competitor Operating Systems

Among **QNX** competitors are VxWorks (http://www.windriver.com/), Integrity (http://www.ghs.com/) and Nucleus RTOS (http://www.mentor.com/).

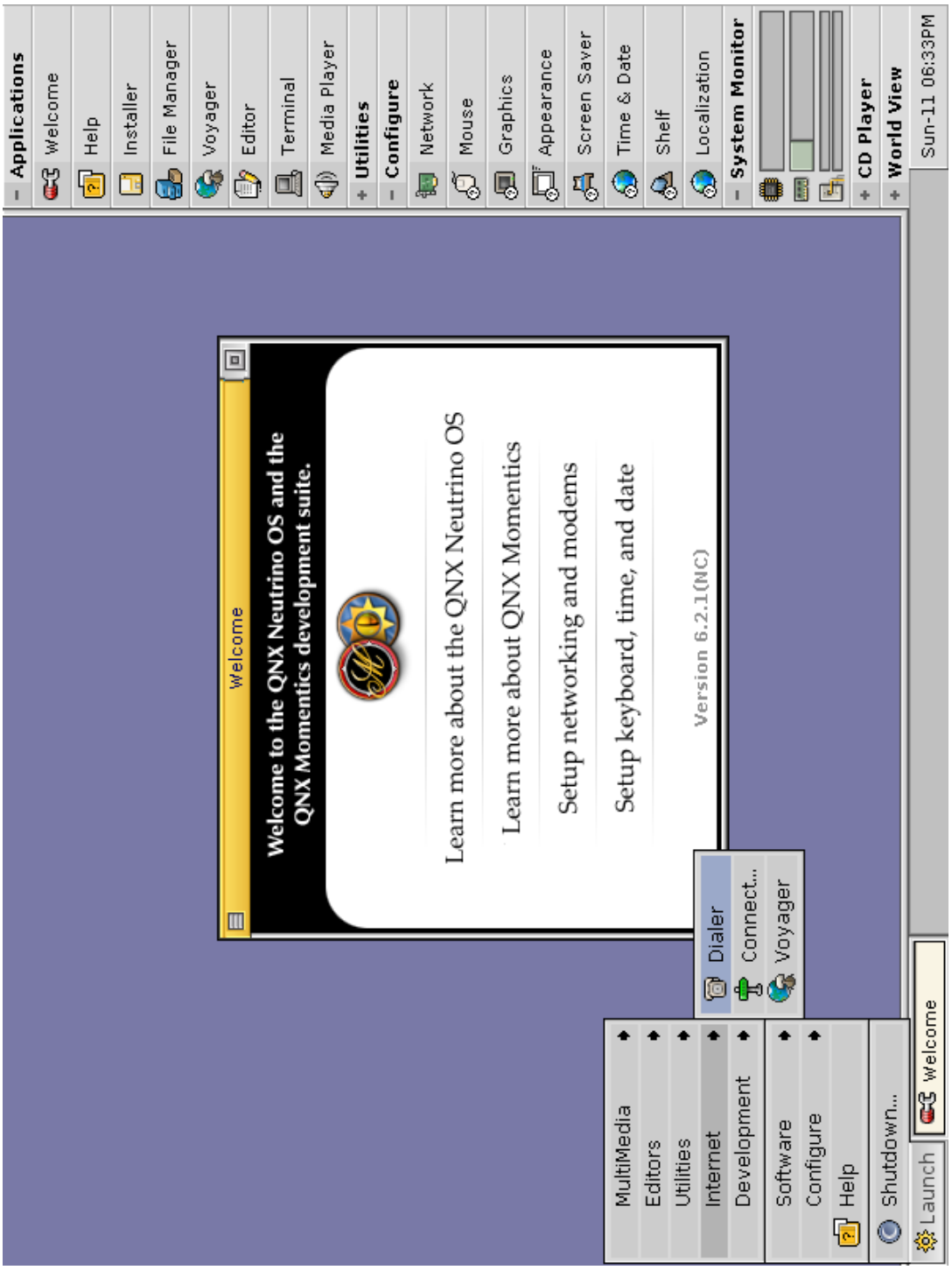

Figure 1.1: Photon microGUI Screenshot (Wikipedia, 2007)

Chapter 2

Characteristics

This chapter will provide an overview of the **QNX** realtime operating system, showcasing its major features, architecture, and major management areas.

2.1 Design Guidelines

Given its extremely high requirements in terms of availability and security, **QNX** has been designed with very specific guidelines in mind (QNX, 2007*i*):

- **Embedded**; **QNX** is designed to be used inside small hardware systems, with strong quality and uptime requirements.
- **Small footprint**; **QNX** has extremely low requirements of memory and disk space, making it ideal for embedded applications.
- **Portable**; **QNX** is available for a large range of different CPU architectures (Wikipedia, 2007) such as the x86 family (Intel, AMD), MIPS, PowerPC, SH-4, ARM, StrongARM, xScale and others.
- **Message-based Microkernel**; **QNX** is designed as a "pure" microkernel system, with a small kernel which contains only CPU scheduling, interprocess communication, interrupt redirection and timers.
- **Secure**; **QNX** is conceived to be one of the strongest and less vulnerable operating systems in the market.
- **Standard-compliant**; **QNX** makes extensive use of standard and open-source technologies, increasing its interoperability and reducing customer lock-in:
  - POSIX: **QNX** is POSIX-compliant (since version 4)
  - OpenGL: **QNX** uses this portable open-source graphics library.

  - Eclipse: the **QNX** developer toolkit is built around the Eclipse framework & platform.

- **SMP Support**; **QNX** is able to handle and leverage the most recent multiprocessor architectures, and provides a developer toolkit to help in the creation of SMP-compliant applications (QNX, 2007*e*).
- **Multimedia**; **QNX** offers advanced multimedia capabilities, making it a strong choice in the desktop market.

2.2 Architecture

Figure 2.1 shows the microkernel architecture of **QNX** :

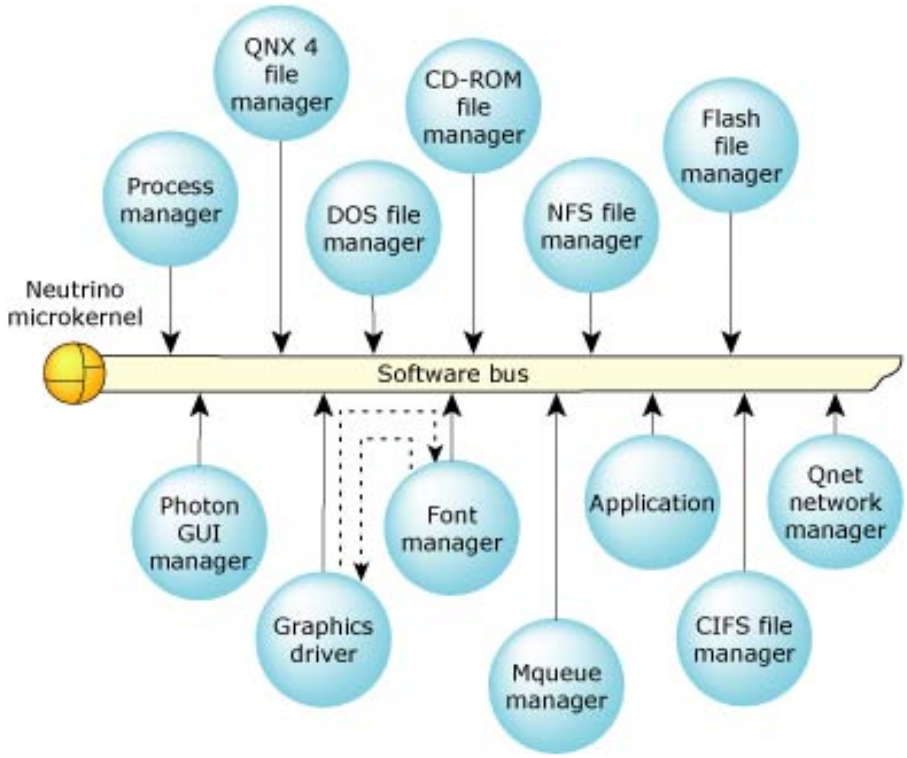

Figure 2.1: QNX Architecture (QNX, 2007*g*)

QNX has an "extreme" microkernel architecture: the kernel, also known as **Neutrino** in the **QNX** documentation, provides a small set of functionality (Ripoll, Pisa, Gai, Lanusse, Saez & Privat, 2002):

- Thread services
- Signal services
- Message-passing services: the kernel handles routing of messages throughout the entire system.
- Synchronization services
- Scheduling services: the kernel uses realtime scheduling algorithms.
- Timer services
- Process management services

A microkernel architecture,

> "structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel. There is little consensus regarding which services should remain in the kernel and which should be implemented in user space. Typically, however, microkernels provide minimal process and memory management, in addition to a communication facility." (Silberschatz, Galvin & Gagne, 2005)

Moreover, **QNX** satisfies the core principles of a realtime operating system, as specified by Silberschatz et al., (2005):

- Single purpose
- Small size
- Inexpensively mass-produced
- Specific timing requirements

All other common operating systems functions in **QNX** are provided by satellite processes running in user space (QNX, 2007*j*); this brings great stability to the system, since no other code than the above functions can work in Kernel mode, and as such these services can be relaunched without crashing the kernel. This simple yet extremely strong architecture makes **QNX** one of the most stable operating systems ever designed:

> "Like Windows or Linux, QNX's program is an operating system, the traffic cop that organizes and runs a computer's many functions. But this operating system is used mostly in highly specialized, realtime industrial applications. QNX software directs "extreme" manufacturing, such as guiding the flawless grinding of optical lenses–a process in

> which the slightest software glitch can ruin a product worth $100,000. It's also used to control facilities such as nuclear power plants and other critical installations where any software funny business could be catastrophic." (Bylinsky, 2003)

2.3 Process Management

QNX process management is based purely based in messaging (QNX, 2007*g*). Figure 2.1 shows an example of message-based collaboration among two processes (the graphics driver and the fonts manager in this case). The kernel routes messages using a single system call (MsgSend), which copies the message from one address space onto the other. Messages can be passed synchronously and asynchronously; in the case of synchronous transfers, the kernel passes the message and the control of the CPU to the receiving process at the same time, without calling the CPU scheduler kernel process. This kind of inter-process communication is unique to **QNX** and provides high performance, since receiving processes waiting for messages do not have to wait to process them. This is one of the conditions stated by Silberschatz et al., (2005) for realtime operating systems, since it helps reducing **latency**: events are served as fast as possible by the appropriate process.

The **QNX** Neutrino kernel uses several scheduling algorithms (Wang, Bo Yao & Zhu, 2001): FIFO, Round-Robin, Adaptive (which reduces the priority of processes that already executed their time slice) & Sporadic (threads are allowed to execute at periodic times). **Periodic and priority-based scheduling** are typical of realtime operating systems (Silberschatz et al., 2005), since their timing requirements are far stricter than "normal" operating systems. This capability makes **QNX** a suitable choice for management of mission-critical equipment, where processes must react immediately to hardware signals.

The **QNX** kernel also features preemptive multitasking (OSDP, 2007). **QNX** also supports POSIX threads and their related system calls APIs, making it easier to port multi-threaded applications to the **QNX** realtime operating system. However, the official documentation of **QNX** does not explicitly specify whether **QNX** features a **preemptive kernel** or not (specified as a *conditio sine qua non* by Silberschatz et al., (2005) for a realtime operating system).

QNX supports Symmetric Multiprocessing natively since 1997 (money.cnn.com, 2007). Figure 2.2 shows the SMP architecture of the **QNX** operating system, scalable up to 8 CPUs (OSDP, 2007).

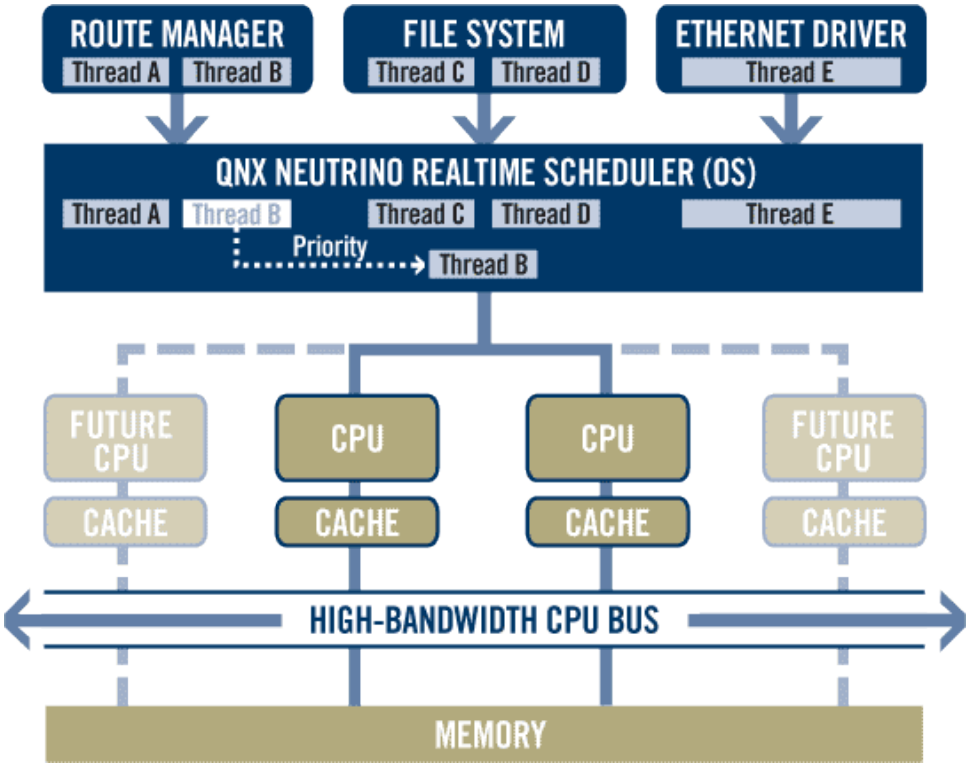

Figure 2.2: Symmetric Multiprocessing in QNX (QNX, 2007*e*)

2.4 Resource Management

Resource management in **QNX** is provided through user-space processes that interact with hardware or software resources (QNX, 2007*c*). There are no kernel-level device drivers, which provides different advantages: ease of development and debugging, increased isolation (a data corruption during a hardware operation cannot affect the kernel), and ease of management (device drivers can be stopped and started as needed by systems administrators).

2.5 Memory Management

The **QNX** Neutrino kernel offers a fully protected environment for applications (Group, 2003), where processes are completely isolated from each other and from the kernel, and no protection violation is permitted. **QNX** uses a complete POSIX model for memory management, with 16 and 32-bit addressing. For new applications, 32-bit mode is recommended, while 16-bit mode is available for porting

legacy POSIX applications to the **QNX** operating system.

2.6 Device Management

QNX does not feature kernel device drivers, but manages devices using user-space processes called "Device Resource Managers" (QNX, 2007*k*). These managers register devices in the filesystem using "pathname-space mapping", as referred to in the **QNX** documentation (QNX, 2007*c*), so that they can be accessed using POSIX-like devices, like "/dev/ser1" for example in the case of serial ports. The processes communicate with device resource managers using the standard message-based IPC mechanism provided by the Neutrino kernel.

2.7 Storage Devices and File Management

The **QNX** Neutrino kernel does not have support for file system operations; they are implemented (as in other cases) as user-space processes, that can be started, stopped, debugged and restarted as needed.

QNX supports a large range of different file systems, all of which run outside from kernel space, and can coexist simultaneously (QNX, 2007*a*). The following file systems are supported by **QNX** natively (Ripoll et al., 2002):

Embedded	**Disk**	**Special**	**Network**
Flash Memory	POSIX	Compression	NFS
RAM	ext2 (Linux)	Transactional	Microsoft CIFS
EFTS	FAT16 & FAT32 (DOS & NT)		
	CDROM (Joliet & ISO 9660)		

2.8 Collaboration Between Management Areas

The microkernel architecture of **QNX** coupled with its pure messaging paradigm provide a highly collaborative environment, where different "satellite" processes collaborate with the kernel to provide the operating system services. This, in turn, has many interesting outcomes for the **QNX** user:

- **Cooperation**: several file systems can coexist simultaneously in memory, making **QNX** highly agnostic about the underlying support formats used

in the storage devices. Moreover, compression or encryption processes can coexist with any file system, providing on-the-fly compression and decompression, or data encryption, to provide more secure or more efficient usage of storage systems. This "UNIX-like" pipelining architecture allows small processes to cooperate in order to provide more complex high-level behaviors.

- **Uptime**: **QNX** is able to achieve high uptime rates, up to 99.999%, which translate to about five minutes downtime per year (QNX, 2007*b*). This feature is called "High Availability" in the **QNX** documentation, and is highly valuable in telecommunications, health services and realtime monitoring of nuclear facilities. The microkernel architecture of **QNX** allows to guarantee such uptimes, since the failure of a single satellite service does not compromise the integrity of the kernel, and as such that of the whole system.
- **Security**: The process isolation, both in terms of memory and scheduling, create an extremely secure environment, where the failure or the interruption of a single process does not affect the kernel.
- **Performance**: Since different elements of the operating system are completely independent from the **QNX** Neutrino kernel, system administrators can taylor them to suit specific performance needs and targets: for example, **QNX** can be easily tweaked to provide high throughput (in the case of server applications), fast disk access (in the case of database applications) or small footprint (for embedded applications). The performance factor depends on the quantity and quality of the different satellite processes that run around the Neutrino kernel at any given time.

Chapter 3

Support of Distribution, Networking and Object-Orientation

3.1 Distributed Computing

QNX supports distributed computing by design (QNX, 2007*m*). The pure microkernel architecture of the system allows the kernel to proactively use the services of similar kernels distributed in a local network, providing increased performance, scalability and reliability. This is called "Transparent Distributed Processing" in the **QNX** documentation, and provides a standard framework for dynamic cooperation and sharing of resources (hardware or software) like file systems, message queues or databases in remote nodes, using **QNX** standard messaging system.

This infrastructure is available off-the-box, both to developers and systems administrators, and leverages the existing hardware investments to provide increased performance in distributed systems. Figure 3.1 shows how **QNX** is able to provide distributed computing features natively.

3.2 Networking

QNX features a unified messaging subsystem, used for IPC communication, that is also used for networking with other computers. **QNX** supports an impressive amount of standard and proprietary protocols, as shown in figure 3.2, making it able to connect to virtually any other system.

The networking features of **QNX** , as many other facilities in the system, are available as user-space processes that can be repaired and restarted independently from

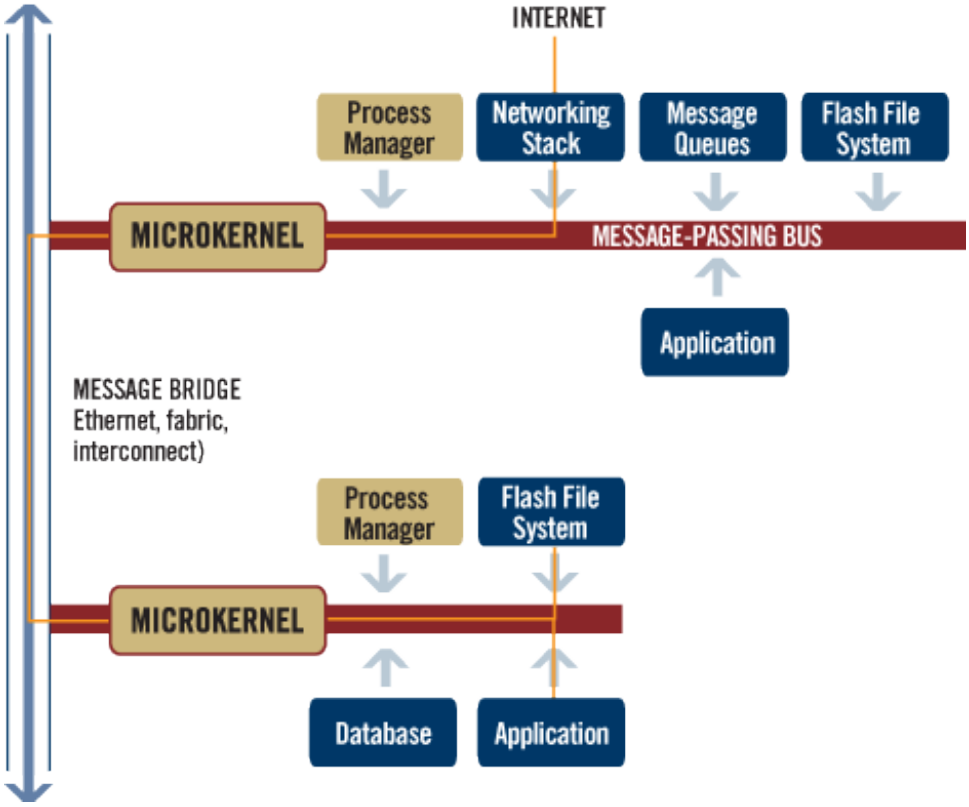

Figure 3.1: Distributed Computing(QNX, 2007*m*)

the kernel, and as such a network-based attack cannot compromise the stability of a **QNX** kernel (QNX, 2007*f*).

3.3 Object-Orientation

QNX provides a Java Runtime Environment off-the-box, allowing application developers to create and deploy Java applications, created on any other platform, leveraging IT investments and reducing time-to-market. Both Java client and server applications, as well as J2ME compliant applications are supported by **QNX** thanks to the integration with IBM WebSphere programming and deployment model (QNX, 2007*d*).

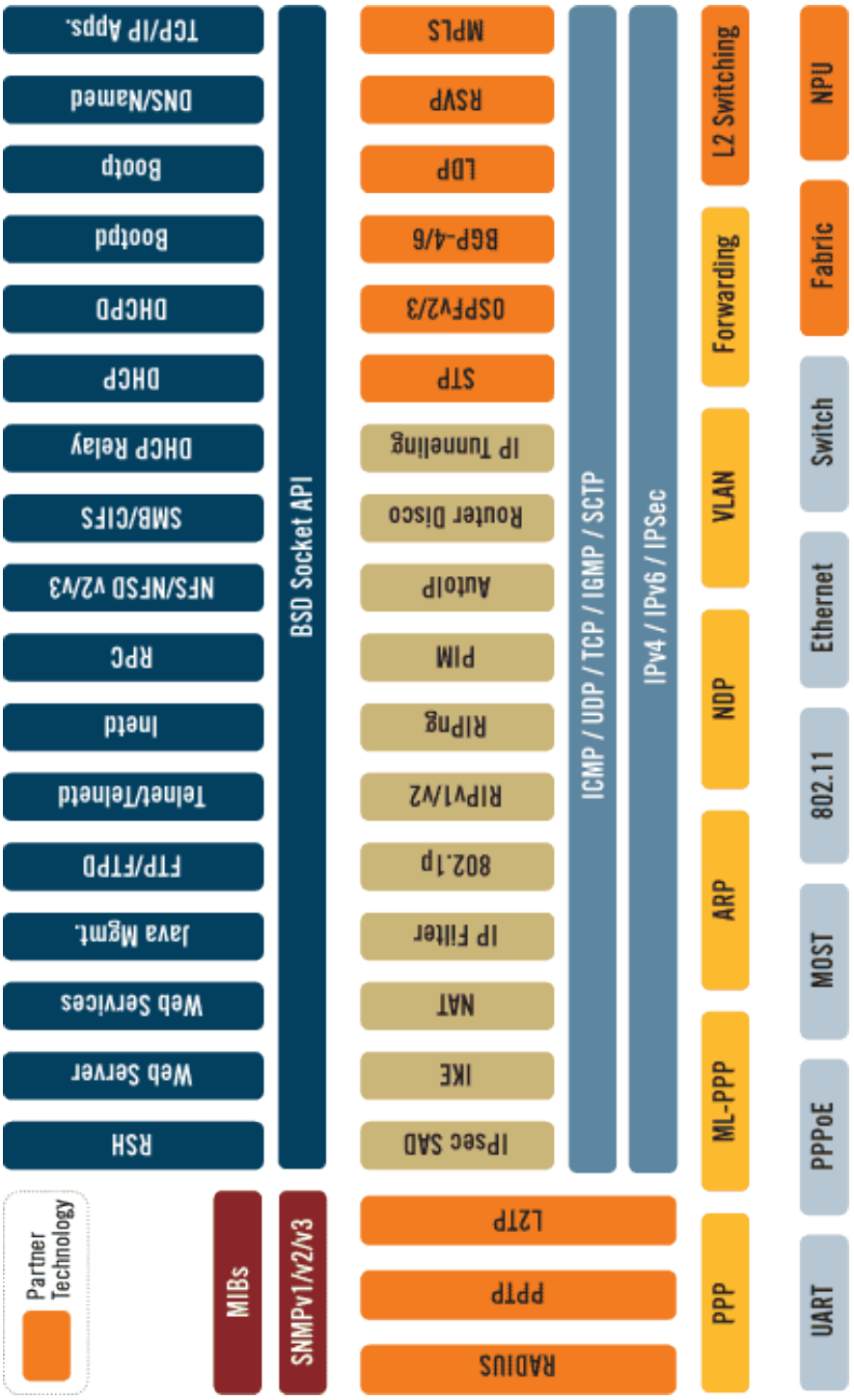

Figure 3.2: Supported Networking Protocols(QNX, 2007*f*)

Chapter 4

Security & Protection

The microkernel nature of **QNX** forces every process, including drivers or file systems, to run outside of kernel space, in a memory-protected user space. This isolation allows these individual processes to be restarted or healed individually, without affecting neither other processes nor the kernel, without needing a full restart.

Processes in **QNX** are securely isolated from each other, and being under tight control from the kernel prevents them from stealing CPU time, creating a "denial of service" situation. The kernel is able to "heal the system" automatically in case of system faults, launching ad-hoc processes that restart individual services if needed, guaranteeing the stability of the system as a whole.

The networking features of **QNX** are available as user-space processes as well, so that a network-based attack cannot compromise the stability of the kernel (QNX, 2007*l*).

The strong security architecture of **QNX** is revealed by the small number of security failures reported in the past few years, none of which was classified as critical by Secunia (Secunia, 2006):

- 2006: 1
- 2005: 4
- 2004: 1
- 2002: 4

Chapter 5

Conclusion

QNX is an impressive example of an extremely optimized and secure operating system. It provides an off-the-box features set unparalleled in other operating systems, making it a privileged choice in the embedded market. It is praised as one of the most secure and stable operating systems available, with a high degree of flexibility, and it has shown through the years that it could be extended to support new paradigms such as distribution and symmetric parallel processing.

The following quote from Money magazine shows clearly what is the reputation that **QNX** has won on the market during the last 25 years:

> Is software hopeless? Ask anyone whose computer has just confessed to an illegal operation or whose screen has locked up. Despite decades of effort, a wisecrack from the software industry's early days still stings: "If builders constructed buildings the way programmers write software, the first woodpecker to come along would cause the collapse of civilization." There's one notable exception. As far as anyone can tell, software created by a Canadian company called QNX Software Systems simply doesn't crash. QNX's software has run nonstop without mishaps at some customer sites since it was installed more than a decade ago. As a delighted user has put it, "The only way to make this software malfunction is to fire a bullet into the computer running it." (Bylinsky, 2003)

QNX will most probably continue its evolution towards the future, providing more advanced services and keeping up with the latest trends in the realtime and symmetric processing computing areas.

Bibliography

Bylinsky, G.; *"Heroes of Manufacturing"*, 2003 [Internet] http://money.cnn.com/magazines/fortune/fortune_archive/2003/03/17/339245/index.htm (Accessed April 4th, 2007)

Group, A.; *"QNX Architecture"*, 2003 [Internet] http://www.arp.harvard.edu/eng/sysman/systems/qnx/arch.html (Accessed April 4th, 2007)

money.cnn.com; *"QNX Marks Tenth Anniversary of Symmetric Multiprocessing"*, 2007 [Internet] http://money.cnn.com/news/newsfeeds/articles/prnewswire/CLTU16727032007-1.htm (Accessed April 4th, 2007)

OpenQNX; *"How much are you paying for QNX? My quote is very high!"*, 2004 [Internet] http://www.openqnx.com/PNphpBB2-viewtopic-t3047-.html (Accessed April 4th, 2007)

OSDP; *"QNX Operating System"*, 2007 [Internet] http://www.operating-system.org/betriebssystem/_english/bs-qnx.htm (Accessed April 4th, 2007)

QNX; *"QNX Gets on Board NASA's Return to Flight Mission"*, 2005 [Internet] http://www.qnx.com/news/pr_1446_4.html (Accessed April 4th, 2007)

QNX; *"File Systems"*, 2007*a* [Internet] http://www.qnx.com/products/rtos/fsys.html (Accessed April 4th, 2007)

QNX; *"High Availability"*, 2007*b* [Internet] http://www.qnx.com/developers/docs/momentics621_docs/neutrino/sys_arch/ham.html (Accessed April 4th, 2007)

QNX; *"How QNX Neutrino compares to other operating systems"*, 2007*c* [Internet] http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/user_guide/os_intro.html (Accessed April 4th, 2007)

QNX; *"Java Environments"*, 2007*d* [Internet] http://www.qnx.com/products/java/ (Accessed April 4th, 2007)

QNX; *"Multi-Core"*, 2007*e* [Internet] http://www.qnx.com/products/tech_dev_kits/smp.html (Accessed April 4th, 2007)

QNX; *"Networking Technologies"*, 2007*f* [Internet] http://www.qnx.com/products/rtos/network.html (Accessed April 4th, 2007)

QNX; *"Process model"*, 2007*g* [Internet] http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/prog/overview.html (Accessed April 4th, 2007)

QNX; *"QNX Customer Stories"*, 2007*h* [Internet] http://www.qnx.com/company/customer_stories/ (Accessed April 4th, 2007)

QNX; *"QNX Neutrino RTOS - At a Glance"*, 2007*i* [Internet] http://www.qnx.com/products/rtos/glance.html (Accessed April 4th, 2007)

QNX; *"QNX System Architecture"*, 2007*j* [Internet] http://www.qnx.com/developers/docs/momentics621_docs/neutrino/sys_arch/about.html (Accessed April 4th, 2007)

QNX; *"Resource Managers"*, 2007*k* [Internet] http://www.qnx.com/developers/docs/momentics621_docs/neutrino/sys_arch/resource.html (Accessed April 4th, 2007)

QNX; *"Secure. Realtime. Guaranteed."*, 2007*l* [Internet] http://www.qnx.com/innovation/adaptive_partitioning/index.html (Accessed April 4th, 2007)

QNX; *"Transparent Distributed Processing"*, 2007*m* [Internet] http://www.qnx.com/products/rtos/distributed.html (Accessed April 4th, 2007)

Ripoll, I., Pisa, P., Gai, P., Lanusse, A., Saez, S. & Privat, B.; *"WP1 - RTOS State of the Art Analysis"*, 2002 [Internet] http://mnis.fr/ocera_support/rtos/c2462.html (Accessed April 4th, 2007)

Secunia; *"Search results about QNX vulnerabilities in Secunia"*, 2006 [Internet] http://secunia.com/search/?search=qnx (Accessed April 4th, 2007)

Silberschatz, A., Galvin, P. B. & Gagne, G.; *"Operating System Concepts"*, John Wiley & Sons Inc, 2005, ISBN 0-471-69466-5

Wang, C. L., Bo Yao, Y. Y. & Zhu, Z.; *"A Survey of Embedded Operating System"*, 2001 [Internet] http://www.cs.ucsd.edu/classes/fa01/cse221/projects/group2.pdf (Accessed April 4th, 2007)

Wikipedia; *"QNX"*, 2007 [Internet] http://en.wikipedia.org/wiki/QNX (Accessed April 4th, 2007)