PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 46, NO. 3-4, PP. 151-161 (2002)

SOFTWARE PROBLEMS OF AN EXPERIMENTAL ROBOT CONTROLLER BASED ON QNX REAL-TIME OPERATING SYSTEMS

István OLÁH and Gábor TEVESZ

Department of Automation and Applied Informatics
Budapest University of Technology and Economics
H–1111 Budapest, Goldmann György tér 3. Hungary
Phone: (+36-1) 463-2870, Fax: (+36-1) 463-2871
e-mail: olah@aut.bme.hu; tevesz@aut.bme.hu

Received: July 1, 2003

Abstract

At the Department of Automation and Applied Informatics an experimental robot control system has been developed. The purpose of this research is to study modern robot control algorithms and their realization in a real environment. The project focuses on the problems of multiprocessor systems including the task distribution and communication. Another field of this research is to integrate a six-component force-torque sensor into the robot control system and making use of this information in new robot control algorithms. Another purpose of this study is to examine the software problems of an IBM PC-based multiprocessor system controlling a NOKIA-PUMA 560 humanoid robot arm. The features and system services of the new QNX Neutrino operating system is presented in comparison with the previously used QNX v4. The main areas of the version upgrade will be shown focusing on the interprocess communication questions. The processing components of this multiprocessor robot control system with its external interfaces will be discussed later and some further system level development possibilities will be outlined. This final part of the study gives the summary of the architectural and communication requirements of a hybrid position and force control system in the above environment.

Keywords: robot control, hybrid position and force control, multiprocessor systems, IBM PC, QNX, Neutrino, Momentics, DSP.

1. Introduction

In the middle of 90's at the Department of Automation and Applied Informatics a PC based multiprocessor robot control system was built. The host computer includes two ARC (Advanced Robot Controller) boards [1], an interface card with analogue and digital inputs and outputs as well as an Ethernet controller card. The architecture of this multiprocessor system was discussed in several studies (see References). During the years this system was extended with a communication card receiving the information from a six-component force-torque sensor through parallel connection [3], [4]. The continuous improvement both on the hardware side and, in connection with this, on the software side brought considerable changes in the resources of the robot control system. The first system had an Intel 80486 DX2 50MHz

main processor. The ARC boards had an Intel 80386EX and a Texas Instruments TMS320C31 digital signal processor on them. The system went through continuous development: the host processor is now an Intel Celeron processor working at 633MHz and the force-torque signals can be received through a PCI based interface card. *Table 1* shows the operational frequencies of the processing units and the approximate available bandwidth of the communication channels. In the early versions a highly distributed model was used to perform robot control tasks. As the host became more and more powerful, more tasks have been moved to this module and additionally, the host is now capable to run a newer version of the operating system. Most of the extended resources are available for more sophisticated robot control algorithms, however, some of them are used by the new operating system and its services [2].

Table 1. Computational and communicational resources

Processing unit	Operational frequency
Host CPU (Intel Celeron)	633 MHz
Intel 80386EX	25 MHz
TMS 320C31	33 MHz
Communication channel	Approximate Available Bandwidth
PCI bus on Host	264/132 Mbyte/sec (33MHz)
Host CPU ↔ i80368EX	5 Mbyte/sec
i80368EX ↔ TMS320C31	10 Mbyte/sec

The initial operating software ran on the QNX v4 real-time operating system. The overall computing power available now allowed upgrading it to the new version called QNX Neutriono / Momentics System (QNX v6).

The next section discusses the considerations and consequences of this upgrade process. The third section presents some problems and their solutions arose during QNX upgrade. The fourth section deals with hardware architecture related issues that brought up some other software problems more related to the low level functions and to be solved in the forthcoming software versions.

2. QNX 4 versus QNX Neutrino

In the early days there was a demand on an operating system for the host system consisting of the main board with the central microprocessor that integrates controller cards and other extension cards for various functions. This system must have been a true real-time system running the robot control software with the proper services both for developers and end users.

The QNX real-time operating system was selected because of its real-time performance, rich IPC services and capability to support complex control tasks. The project started with QNX v4 but in year 2002 the host system was ported to the QNX Neutrino (QNX v6) operating system. The new version has many changes based on the experience of the previous version; it supports several platforms and symmetric multiprocessing (SMP) and embedded solutions. Both versions are based on a microkernel architecture. In this architecture the principal operating system components run as regular processes in their own separate address space. The microkernel provides the basic services such as *process manager services*, *thread services*, *scheduling services*, *synchronisation services*, *signal services*, *message passing services* and *timer services* [5], [6].

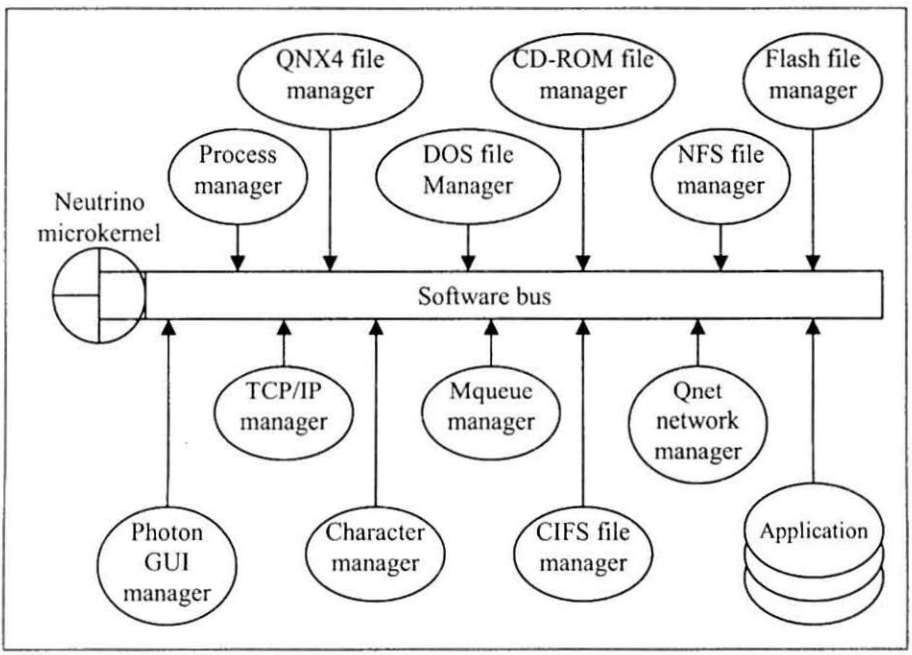

Fig. 1. The QNX architecture

Operating systems with true microkernel architecture provide the possibility of building small, efficient and many times embeddable solution especially for real-time challenges. One of the key features of these systems is that they are flexibly configurable for various tasks; the system modules can be started (loaded) or stopped depending on many factors such as hardware configuration and required services. This kind of modularity provides memory protection since all modules except the microkernel itself run in user mode, at a separate address space. This means that user written processes can act as a system service extending the functionality of the operating system in the same way as built-in services do. On the other hand, the

microkernel architecture of some operating systems brings more overhead and is not suitable for some places like office desktops.

As the previous versions of QNX, the Neutrino is a microkernel operating system developed on the experience from previous versions. *Fig. 1* shows the structure of QNX Neutrino. The big difference between QNX v4 and the present version is more POSIX compliant. QNX Neutrino is a POSIX API representation regarding the special needs for a flexible, modular, real-time operating system. Another new feature is the multi platform support, such as ARM, MIPS, etc. Since the robot control system developed at the Department of Automation and Applied Informatics includes an x86 based host system, interface extension cards, two or three Advanced Robot Controller boards that contain another one x86 processor and one Texas Digital Signal Processor on each (see *Fig. 2*), there is no new possibility to utilize this new feature. Both QNX system versions provide POSIX compliant elements, but Neutrino follows more the draft and standards. The most important ones were the following: *standard 1003.1* – defines the base functionality of an operating system therefore an application programming interface (API), *real-time extensions* – definitions for real-time services, *threads* – multiple threads in a single address space, *additional real-time extensions* – interrupt handling and *application environment profiles* – embedded system support properties. These changes were driven by a demand to provide a general environment for embedded applications and for developers who work on cross platform projects.

The QNX Neutrino kernel acts as a software bus letting software modules dynamically plugged in and out. These modules can be system modules developed by QNX, user-developed entities for system-like services or with application specific functionality. In real-time, multitasking environments the functionality relies heavily on interprocess communication (IPC). This is very important especially for modular application such as the robot control system. The largest part of the operating system version change or upgrade was the conversion of interprocess communication.

The primary form of process communication in QNX Neutrino is message passing. Since Neutrino offers POSIX API, there are some other forms of IPC, but some of them implemented using the functionality of the messages. Here are the forms of IPC (in parentheses the place where they are implemented): *message passing* (Kernel), *signals* (Kernel), *POSIX message queues* (process), *shared memory* (Process Manager), *pipes* and *FIFOs* (both in an external process). The most heavily used QNX version 4 IPC functions are message passing, proxies, signals and shared memory. The software porting project 'served' a good opportunity to review the software system from this aspect. The details of the modifications can be found in Part 3. Message passing, the basic communication form, performs data transfer between the address spaces of the processes and additional synchronisation. The process state diagram and the system functions used at both sides are shown in *Fig. 3*. The data copied has no special meaning for the operating system (binary transfer – except some special messaging services), so there is complete freedom for application developers and the speed of this transfer is only limited by the underlying hardware.

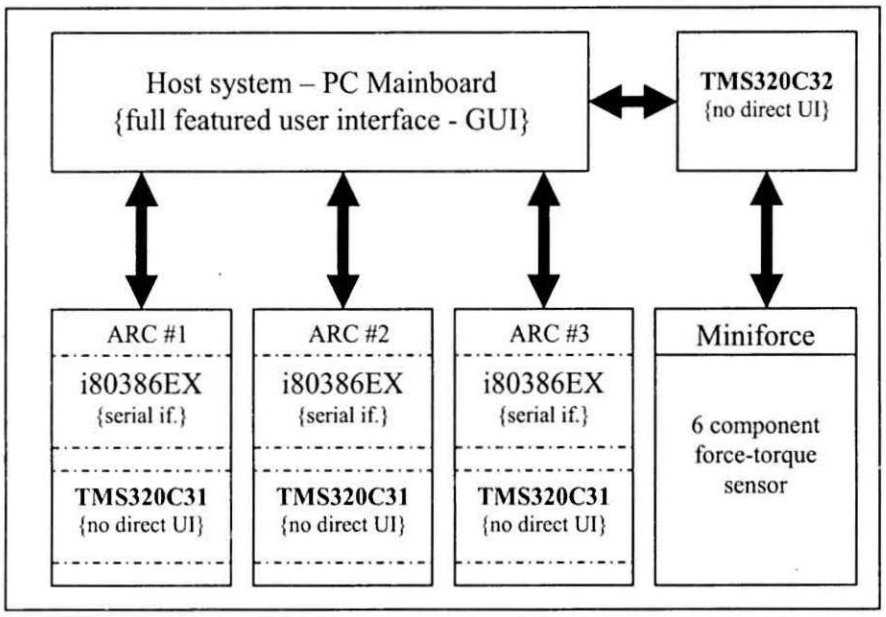

Fig. 2. Robot control system – processing units and user interfaces

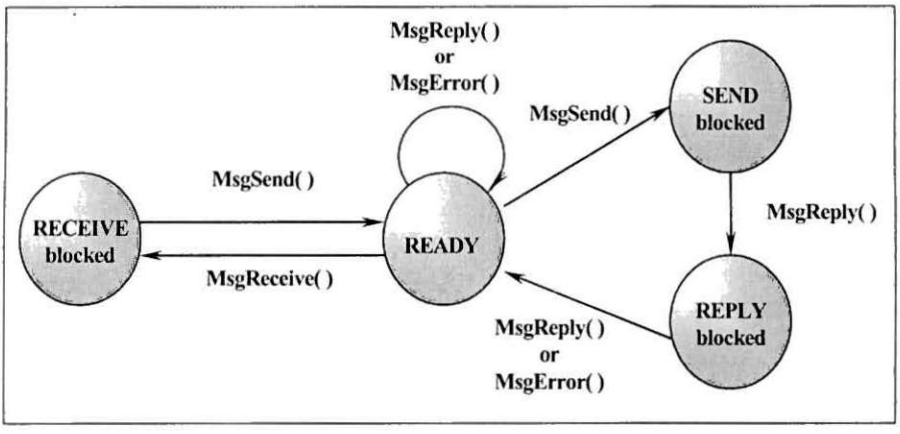

Fig. 3. Message passing in QNX Neutrino

One of the new services of QNX Neutrino that played the main role in versions change is the System Analysis Toolkit (SAT). This tool provides the monitoring of the dynamic execution of a complex software environment. The researchers and developers have the opportunity to treat many threads as a whole system examining kernel calls, message passing, handled hardware interrupts and state change of

various threads. Using SAT there is a possibility for real-time or offline system analysis. Because of the system-level view SAT can be utilized for performance analysis, optimization, real-time debugging, pinpointing deadlock and race conditions and event tracing.

There is one more point that actuated the upgrade. This is the better user environment that involves an easy to learn, easy to operate graphical user interface with integrated development environment, software debugging tolls, etc. The first appearance of the Photon graphical user interface was in QNX v4 and the new version of this desktop environment can be found in QNX Neutrino. The host system can serve both users and researchers, developers with a lot of valuable features. On one hand graphical user interface (GUI) is a common requirement for modern systems, but on the other hand it requires additional resources. The robot control system should run with high priority and that can sometimes cause poor performance in GUI handling.

3. Problems and Their Solutions during OS Upgrade

First of all the host system is an ordinary desktop system with some special features needed. The host must provide the ability to execute two QNX versions: version 4 and Neutrino (v6) together with a version of a Microsoft OS. Because of general file porting capability, the choice was Windows98. All the system use or can handle the FAT/FAT32 file system. An additional reason for installing Windows was that other development tools for x86 platforms and Texas digital signal processors can run on it.

The properties of the host subsystem in the general robot control system are the following. The general process map and the communication channels are shown in *Fig. 4* using the QNX Neutrino IPC methods. The general 'ARPS' process is the ancestor of all processes. Each process is a single thread process in the present system. After the processes started (using 'spawn()' functions) 'ARPS' executes the Advanced Robot Programming System interpreter and handles the end effector (open/close) and the digital input/output lines.

The functionality of processes (see *Fig. 4*):

- CONSOLE: Terminal window handler for user input (terminal/editor mode) and system output.
- MESSENGER: Message handler for avoiding undesired blocking.
- PENDANT: RS-232 serial line handler – teach pendant I/O functions.
- PATHGEN: Fourth order, continuous acceleration path generation.
- DYNJAC: Joint data pre-processor. Writes data into the shared memory.
- ARCPROC: General purpose Advanced Robot Controller Board handler.
- ARCRUN: ARC starter and status monitor.
- ARPS: Coordinator process.

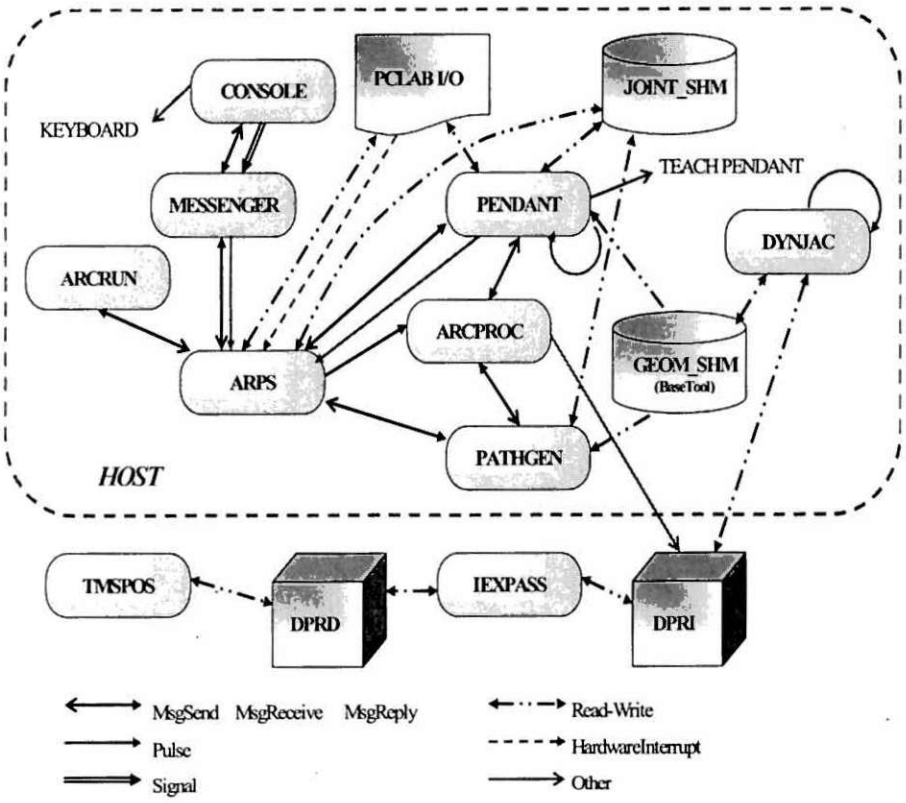

Fig. 4. The host software system – Processes and Communication

The scheduling services in QNX Neutrino have changed a lot in terms of the scheduled entities because they are realized by thread instead of processes. As a matter of fact this has very little impact on the robot control system since the processes were single-threaded. For the further development there is an interesting effect by the possibility to pre-empt a process by itself, i.e. two different threads of it. Additionally, the priority range has been extended and now it can go from 0 to 63 instead of 31.

In the previous version of the robot control software the *message passing* and the *signal* were the two ways mostly used for interprocess communication.

Message passing has been changed quite a lot in QNX Neutrino. QNX v4 used the Send(), Receive() and Reply() base functions (and there were some other forms of them, as well) while QNX v6 uses MsgSend(), MsgReceive() and MsgReply(). Not only the names have been changed but the parameters and communication channel handling are different, too. The previous version worked with processes rather than threads and the handling was simpler. During the migration, all process references must have been converted to channel reference (Process ID → Channel

ID). This method required careful overview and modification despite most of the processes were single-threaded. The further versions of the software system will more intensively utilize the new concept of the thread approach. In general QNX v4 focused more on processes and in QNX Neutrino the thread object is used for scheduling, interprocess communication, etc.

The following two examples are the drafts of the communication in QNX v4 and in QNX Neutrino respectively:

QNX v4:

```
Sending Process

Pid2 = qnx_name_locate (...);
Send ( Pid2, ...);
```

```
Answering Process

qnx_name_attach (...);

Pid1 = Receive ( 0, ...);
Reply ( Pid1, ...);
```

QNX Neutrino:

```
Sending Thread

ConID = ConnectAttach (...);
MsgSend ( ConID, ...);
```

```
Answering Thread

ChID = ChannelCreate (...);

RecID = MsgReceive ( ChID,
...);
MsgReply ( RecID, ...);
```

Signals have main role in the robot control system. The changes on this field in QNX Neutrino are as follows: there are some new signals, signals can carry data (POSIX), signals can be queued (POSIX). The implications of the new thread based approach are that threads maintain their own signal mask and it has some consequences. If more than one thread has the signal unmasked, the kernel chooses one randomly. Any consecutive signal of the same type will go to the same thread. In spite of signal mask the signal actions were maintained at the process level, so ignore or catch decisions should have been made at process level. From technical point of view, the form of the functions has not changed significantly.

The first migrated version of the robot control system to QNX Neutrino was made in 2002 and it is finished. The forthcoming development and the new versions will benefit more from the new features of QNX Neutrino operating system. These projects will be supported by the new integrated development environment and other services built into the product.

4. Processing Modules and Their Communication

The simplified architecture of the robot control system was shown in *Fig. 2*. The communication between the microprocessors is implemented through dual-port

RAMs. The approximate bandwidth between the host and the i80386EX processors is approx. 5 Mbyte/sec based on the bus architecture of the main board. The available bandwidth of the dual-port RAMs between i80386 processors and TMS320C31 digital signal processors is approx. 10 Mbyte/sec. There exists a direct memory mapped access from both sides. Some years ago there was a plan to split up the processing task between the 5 or 7 processors (2 or 3 ARCs) but the marketed PC main boards and general purpose processors advance so fast that most of the functions are accomplished by the main processor. Another important point is the multi platform development. In the following paragraphs each processing module is discussed in detail.

1. The signal processor has no operating environment; all the functionality must be programmed using the Texas development tools that run under MS DOS or MS Windows development station. (There is a version running under SUN Solaris, but not under any versions of QNX. In QNX v4 there was an emulated DOS environment, but it has not been supported for years.) As a result the TMS320C31 processors carry out the low level input handling (cross-checking, scaling, etc.) then transmitting the measured values. On the other direction the low level, multi-loop controllers for the joint drives are implemented here. Another development station is needed (with network connection) to develop software modules for this platform. There is a necessity to integrate all the low level control system functions into single Texas DSP software that can easily be configurable including architecture and parameter configuration.
2. The Intel 80386EX microprocessors have no floating point coprocessor built in, so they have significantly less computing power than the host or the Texas processors. This unit is potentially capable of doing pre-processing and it has a CAN interface, that can be used for direct communications between ARC boards or other modules. These functions are not utilized in the present version, so the modules serve as information forwarders. Taking speed into consideration, a very special software runs there and serves with four types of communication. There is a ring buffer towards the host and the TMS DSP, which is not optimized and provides character-based terminal functions where the user interface handled on the host and communicates with the built-in BIOS-like functions of the processors built on the advanced robot controller boards. There is another ring buffer for sending binary messages to the TMS (not the same as QNX message passing). This binary data can be variable in length and the delivery must not be time critical. There are two other dual-port RAM areas for mirroring. The copying of data is interrupt driven in the case of both areas and therefore the block size cannot be configured. One area is preserved for timer driven transfer; the copy takes place in every millisecond in the current system. The other is for on demand communication using the dual-port RAM interrupt capability. These two blocks are four, because the processor provides this way of communication in both directions between the host and the joint processors (TMS320C31). Since there are a very few

system services implemented, it is very hard to change the operating software. It has been developed in assembly and C, but the compiler libraries can be used with limitations (BIOS and basic OS functions implemented through software interrupts are missing).
In the future it would be beneficial to embed and run QNX Neutrino on this platform. It requires the implementation either of a full BIOS functionality or at least the necessary services for starting the embedded system. Some years ago QNX released the first version of its Embedded Kit based on QNX v4, but Neutrino contains a lot more support functions and better documentation. After the host system was ported to the new version this new field would bring more flexibility in system development and research. It seems to be better and faster to implement the fully functioning BIOS than to (re)write each required service, because there are some easily applicable solutions both from the hardware manufacturer and software developers (there are some free versions as well).

3. Last but not least, the x86 platform based host system with the most computational power performs the most of the tasks of the robot control system. It has a full featured graphical user interface – Photon v2 – and runs the integrated development environment and operating system related utilities. This is the only part of the system that can be debugged in all details. Another benefit of placing the most demanding part of the robot control system is that this block receives information from the peripherals: the analogue input signals of the reference potentiometers of each joint and the digital input/output signals use the same I/O card of the host. The built-in Ethernet interface is used for communicating with the development station of the other blocks. The six component force-torque sensor interface will be another extension card displacing the present connection channel that is too slow for this kind of real-time system. One software system for graphical simulation of the PUMA-560 manipulator has already been implemented, but it is operational only on Microsoft platform. Another direction of the development can be one QNX Neutrino based simulation environment. Seeing the growth of the resources it can be capable of real-time operations sometimes.

There is a lot of work done at the Department of Automation and Applied Informatics, Budapest University of Technology and Economics in 2002. The most important result is the porting of the robot control system. There is a lot of possibility to be utilized and there is a lot more to do on the field of research and development of modern robot control systems.

Acknowledgement

The project of studying modern robot control algorithms and their realization in a real environment is supported by Hungarian Research Fund (OTKA, grant No. T029072, grant No. T042634).

References

[1] BÉZI, I. – TEVESZ, G. Komplett aus Standard – Komponenten, *Elektronik* Feldkirchen, **1** (1996), pp. 44–48.

[2] TEVESZ, G. – BÉZI, I. – OLÁH, I., A Low-cost Robot Controller and its Software Problems, *Periodica Polytechnica Ser. El. Eng.*, **41** (3) (1997), pp. 239–249.

[3] TEVESZ, G., Architectural Problems of the Hybrid Position and Force Control System of Robots, *Periodica Polytechnica Ser. El. Eng.*, **42** (2) (1998), pp. 251–262.

[4] FODOR, G. – TEVESZ, G., Hybrid Position and Force Control Algorithm Expansion of a Robot Control System, *Periodica Polytechnica Ser. El. Eng.*, **43** (4) (1999), pp. 251–261.

[5] QNX® 4.1 Operating System – System Architecture, ©Quantum Software Systems Ltd. 1992.

[6] QNX® Neutrino® Realtime Operating System – System Architecture, ©QNX Software Systems Ltd. 2002.