Real-Time Operating Systems For Robotic Applications: A Comparative Survey

Submitted: 28th February 2015; accepted: 31th March 2015

Piotr Kmiecik, Grzegorz Granosik

DOI: 10.14313/JAMRIS_3-2015/20

Abstract:
Modern robotics brings up a variety of new challenges. Industrial-only times are gone forever. Nowadays, by performing many diversified services, robots turned out to be a part of our everyday life. There is no doubt that they should be able to interact with this new miscellaneous environment seamlessly.

Along with the number of requirements, the importance of software increases. While gaining more autonomy, besides standard motion control, a wide range of cognitive tasks has to be executed simultaneously. It is quite obvious that complex systems with a lot of independent modules and many functions to perform need flexible and reliable software solutions.

This article takes a Real-Time Operating System as an answer to the problems of the new generation robotics. By comparing selected features, it provides an evaluation of the most popular commercial and non-commercial solutions. Paper describes the key characteristics that should be taken into consideration during the design process and presents several examples of their successful robotic applications, including our own choices for decentralized controllers.

Keywords: *real-time operating system, survey*

1. Introduction

Operating System in general, can be described as the computer software that manages hardware resources and provides an environment for executing user applications. It implements various services such as multitasking, time management, interrupt service routines or device drivers, usually combined with support for many different platforms.

Real-Time Operating System (RTOS) is the similar software, but especially designed to meet the predetermined time constraints.

Depending on the consequences of exceeding the deadline they are divided into the following three categories: soft, firm and hard. The first two can be identified as best-effort systems doing their jobs just as fast as they can. Whereas for the last category, it is the worst-case that always matters - the consequences of not meeting the deadline would be fatal.

Robots, with the motion control (requiring strict time regime), navigation, obstacle avoidance and safety issues should be perceived as hard real-time systems and only those Operating Systems that provide such functionality should be an object of further examination.

2. Why the Operating System?

In case of simple, single-flow applications with limited functionality it is usually reasonable to put everything into one infinite control loop. When it comes to more complex solutions, where a lot of events and multiple flows are considered, it is often better to use some kind of operating system.

There are several reasons why to choose a market-available operating system over writing everything for scratch:

1) *Productivity:* Ready to use concepts increase the ease of development. Integrated development environments, debug, simulation and analysis tools are usually provided, which results in shortening the development cycle. The developer does no longer have to concentrate on low level issues and can put all the efforts into achieving his goals.
2) *Efficiency:* Well thought design paradigms such as semaphores, queues or mutexes are already on board. Everything is encapsulated with well-defined interfaces, hence no time is wasted on reinventing patterns or algorithms, on which someone else had probably spent much more time and which would probably end up less efficient after all.
3) *Reliability:* Based on the maturity of the projects and a substantial number of users, it can be assumed that they offer high quality piece of software. Many bugs and feature flaws should have already been discovered and fixed.
4) *Commercial/community support:* In case of any integration issues there is a huge probability that there is someone who has already encountered and resolved the same problem or just has enough knowledge to provide some help.
5) *Third-party packages:* In most cases, compatible software coming from other vendors is available as well. Incorporating it into a project can significantly influence the speed of the development process.

Switching to an operating system can also have its weaknesses. One of the most frequently mentioned is losing full control over all parts of the code. However, keeping in mind advantages such as maintainability, it still should be better to use it for larger applications.

3. Off-the-shelf RTOS

Currently, a wide range of free and proprietary products is available on the market. Systems like FreeRTOS, Nucleus, OS-9, QNX Neutriono, RTAI,

RTEMS, TI-RTOS, VxWorks are well-known, but do not exhaust the list of all available solutions. Each RTOS is tweaked for slightly different purposes and provides a variety of functionalities - from simple scheduling capabilities, through popular communications stacks, up to some advanced features such as Symmetric and Asymmetric Multiprocessing Support.

4. Selection

When it comes to picking up the most suitable RTOS an incredibly large number of factors comes into play. Sometimes it can be truly overwhelming. The system should satisfy all the functional, temporal, dependability and budget requirements.

Without any doubt, the choice will always be based on the type and the amount of tasks to handle. It will also depend on the available hardware and system architecture.

In the following sections we describe the fundamental characteristics of Real-Time Operating Systems, that are helpful when going through the decision-making process. Table 1 provides a brief overview of twelve popular RTOSes comparing their most important features.

4.1. System Architecture

The vast majority of Real-Time Operating Systems offer support for more than one platform architecture. In general, ARM and x86 compatible processors are the most popular solutions. Of course, there is also many more like MIPS, PowerPC, SuperH or PIC. It is important though, to select the RTOS that will be fully compatible with the CPU type and family used for the controller.

For various architectures, some useful functionalities such as Floating Point Support or Memory Management Unit are also available. It is vital to decide which of them could be suitable for the project and check if the considered RTOS provides support for them as well.

When dealing with advanced robotic systems, there is a tendency to replace centralized controllers with more distributed solutions. The system is no longer running on a single computing unit that all the sensors and actuators are connected to. Instead, there's a number of loosely coupled nodes connected via real-time communication network. INtime Distributed RTOS is an example of a fully scalable solution. It provides fully pre-emptive scheduling along with the ability to run multiple kernels on multi-core CPUs. Some sort of support for distributed architectures (e.g. transparent access to remote resources or fault-tolerant clusters) can also be found in QNX Neutrino and many others.

4.2. Temporal Dependencies

Even though, some robots do not require hard real-time system we refer our discussion to the most demanding option. Therefore, time dependencies are crucial and they should be estimated with the highest possible precision. The worst-case execution time (WCET) and the worst-case administrative overhead (WCAO) should be known a priori. It is also important to have a proper documentation providing the number of cycles per each system call.

Since the vast majority of systems use Rate Monotonic Scheduling, the Rate Monotonic Analysis is the most popular prediction method. For some RTOSes, specialized profilers are available. RapiTime by Rapita Systems Ltd. is an example of such tool that supports VxWorks.

4.3. Multitasking

The ability to process multiple tasks is the most fundamental feature of every operating system. Several scheduling algorithms such as round-robin, first in first out, shortest remaining time or the most popular fixed priority preemptive scheduling are commonly implemented. There are also several mechanisms to realize inter-task communication that happens to be another integral part of multitasking. Some examples are: events, message queues, semaphores, mutexes etc. It is up to the designer to decide which will fit the best.

It must be noted that most RTOSes are designed to handle real-time as well as non-real-time tasks. Not all scheduling algorithms will be appropriate for real-time purposes. From the decision-making point of view it will also be important to know the maximum number of tasks and priority levels a system can handle.

4.4. Resource Management

Resource management may be one of the most beneficial features that comes with the operating system. CPU architectures are usually well supported and we can gain an easy access to extensions such as Memory Management Unit or Floating-point Unit. Device drivers, communication stacks and file system support are just other facilities. We can easily imagine ourselves writing an I2C algorithm (even just for single robotic project), but implementing our own TCP/IP stack can cause a lot of trouble. Most of the products will offer the same standard feature set. However, it may happen that a less common component will decide of our choice.

4.5. Dependability

Although, mainly robots working in hostile environment (e.g., space, deep sea, radioactive) are seen as the most reliable, but also those operating in more friendly surroundings must follow control strictly. A robot performing its household chores can cause a serious damage or can even hurt somebody.

There are five basic features that define the quality of a fault-tolerant system [13]: reliability, maintainability, availability, safety and security. Most vendors provide some mechanisms to enhance those features. Some of them offer memory management, that allows each process to run in its own protected space. Nucleus can even restart a system when a critical failure occurs. To provide security they incorporate some cryptographic algorithms. In the age of cloud-computing, where most robotic devices operate con-

nected to a public network, this may be very significant feature. Since dependability issues are vital, it is always a good practice to check how we can benefit from proper RTOS selection.

4.6. Development

More mature, particularly commercial products, along with the compiler toolchains provide Integrated Development Environments. In most cases, it is an Eclipse plug-in, that implements some dedicated features e.g., wizards, simulators, tracers or optimization tools. Some of them like FreeRTOS can support various IDEs depending on processor architecture.

Most RTOSes are written in C or C++ and those are the commonly supported languages. However, some systems are using ADA, Basic, Fortran, Pascal or even Java.

Licensing model should also be taken into consideration. Freely available solutions are great, but with respect to engineering costs, sometimes it is better to pay to get more appropriate set of tools. If the proprietary solution is chosen, it is important to know what updates, services and support are included. The source model should also be clarified. Advanced debugging may require source code accessibility.

At the end of this check-list we would like to aware reader about the vast number of ready-to-use starter kits that can be employed in development of robotic applications. Board Support Package (BSP) is a set of software coming with the RTOS or from third-party vendors, that should bring a bootloader, device drivers, root file system and even a toolchain to it. Review of hardware possibilities is out of the scope of this paper, but since most RTOSes offer more than 50 BSPs it shouldn't be a problem to find proper solution.

5. Successful Stories

Curiosity rover has successfully landed on Mars on August 5, 2012. It's an autonomous robotic device built by NASA, which serves as a science laboratory and is supposed to provide data and analysis within eight mission objectives.

For critical tasks, e.g. landing sequence (EDL), an absolute determinism was necessary. Hence, VxWorks RTOS has been used to provide this functionality. System also supports many other tasks: communication, ground operations control or data collection.

Wind River's VxWorks is also used as an operating system for ASIMO, a humanoid robot developed by Honda. Robot is capable of performing very wide range of advanced tasks e.g., running, moving objects, objects, sounds and facial recognition.

The usability of Nucleus RTOS is presented on a simple self-balancing robot – Stella. It is a two-wheel device build upon market-available components, that benefits from kernel pre-integrated technologies such as power management and Z-Stack as well as high quality development environment.

The MITRE Centaur Robot is an autonomous vehicle used for leader-follower experiments. It utilizes RTEMS operating system for its safety-critical tasks.

Mobile robotics is not the only area of RTOS application. Nowadays, real-time operating systems are also present in many industrial devices and can be found in products made by companies, such as ABB or KUKA.

6. RobREx

RobREx is a research project aiming to produce a set of technological tools that will facilitate the development of autonomous **Rob**ots for **R**escue and **Ex**ploration. One of its objectives, being realized by the Lodz University Technology, is to provide impedance control to manipulators and grippers. The results are to be presented with two demonstrators: a multi-purpose six-axis industrial robot (Kawasaki FS003N) and especially designed 4 DOF manipulator, which combines both pneumatic and electric propulsion systems.

In the first case, a low-level control of the robot is handled by dedicated Kawasaki D70 controller. Main limitations of this solution are: closed architecture and access only to the position regulation. In order to provide impedance control we have created and external feedback employing a 6 DOF force/torque sensor and a stationary PC as shown in Fig. 1. For the second application, the control is distributed among four controllers – one for each joint, as shown in Fig. 2.

Since the impedance control requires processing a lot of external data as well as extremely short and predictable response times, all controllers had to be equipped with real-time operating system.

Regarding hardware, the external controller for Kawasaki robot is based on x86, while joint controllers for hybrid manipulator utilize ARM architecture. To simplify the development process, a solution that fits both was desired.

Moreover, both control systems were designed as distributed architectures. Some sort of communication between the nodes had to be provided – preferably CAN or real-time Ethernet.

Considering the research-oriented nature of the project, it was not certain in which direction the experiments would go in the future. Thus, the demand for open, easily expandable architecture has arisen.

Based on the above-mentioned prerequisites, Xenomai real-time linux framework has been chosen for the project. It has the common history with RTAI and provides real-time capabilities along with all the facilities a standard Linux system can offer.

The decision was also influenced by the availability of know-how and BSPs for selected electronics, as well as quite mature robotic middleware such as ROS and OROCOS, that were supposed to speed up the overall integration process.

7. Concluding Remarks

The examples above prove that real-time operating systems are applicable for robotic purposes whenever absolutely deterministic behaviour is required. It doesn't matter whether they will serve their duties in

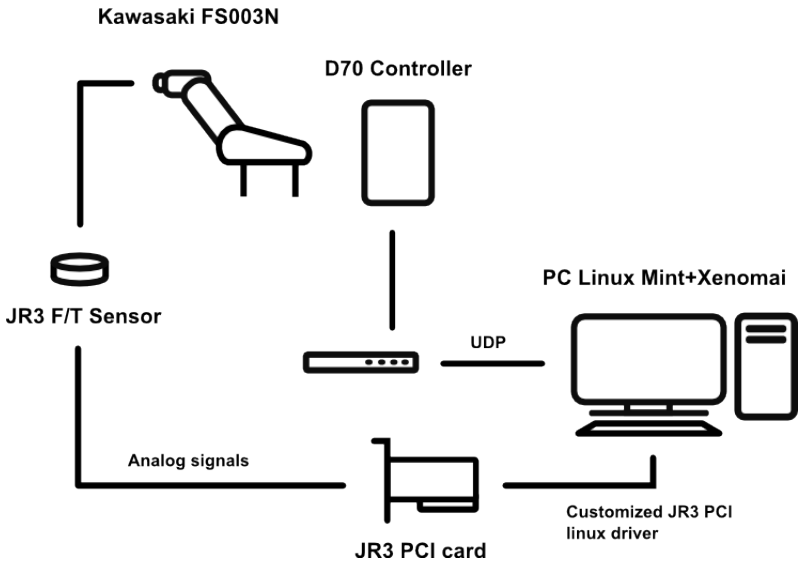

Fig. 1. Kawasaki experimental setup

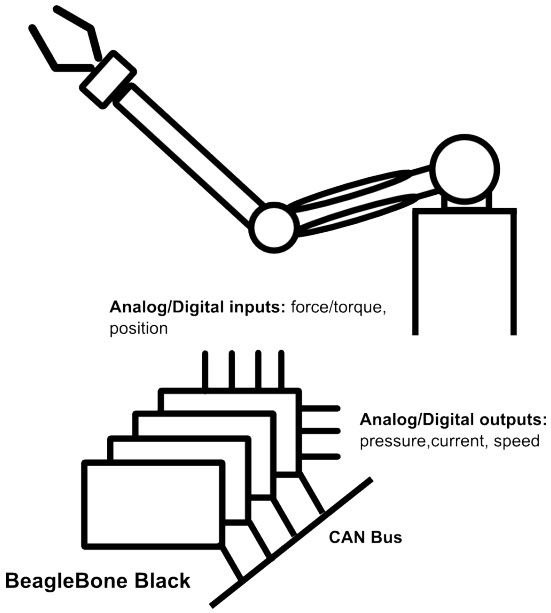

Fig. 2. 4 DOF manipulator experimental setup

very complex or very simple devices. In both cases, RTOS can significantly simplify the design.

However, due to heterogeneity of all robotic solutions, which results in the lack of standards, no operating system has yet dominated the market. That is the reason, why selection of an appropriate product is not a trivial process. Regardless of that, undertaking such an effort can pay off well in the future development. We hope that provided check-list of requirements and the table of available tools will be helpful in your research.

Tab. 1. Real-Time Operating Systems Comparison

Name	Vendor	Supported platforms
FreeRTOS [15]	Real Time Engineers Ltd.	68K: Coldfire; 8051: C8051; ARM: AT91, Cyclone V SoC, EFM32, FM3, Kinetis K, LPC1100/1700/1800/2000/4300, PSoC 5, RM48x, RZ, SAM3, SAM4/7/9/D20, SmartFusion, SmartFusion2, Stellaris, STM32, STR7/9, TM4C, TMS570, XMC1000/4000, XMC4000; AVR: AVR32 UC3; Power: 405, 440, Xylinx; PIC: dsPIC, PIC18/24/32MX/32MZ; MSP430: MSP430X; RX: RX100/200/600/62N/RX63N; x86: all; Others: 78K0R, APS3, H8/S, HCS12, Microblaze, NIOSII, RL78, SuperH, TriCore, V850, Zynq;
INTime Distributed [27, 28]	TenAsys	x86: all
LynxOS [10, 25, 26]	LynuxWorks	ARM: OMAP35x, XScale 80200; MIPS: MIPS32/64, 20Kc, RM5260/7000/7900 RISC; Power: ppc405/440, 750, 970, MPC603/603e/603ev/604/604e/604r, MPC74xx, MPC821/860 PowerQUICC, MPC8240/8245/8260 PowerQUICC II, MPC85xx PowerQUICC III, MPC8640/8641, QorIQ P1010/P4040/4080; SPARC: LEON3; x86: 386/486, Athlon, Atom, Celeron, Core 2 Duo, K6/7, Cyrix 6x86, Pentium, Pentium II/III/IV/Pro/M, MII, Xeon;
Nucleus [7, 9]	Mentor Graphics	68K: Coldfire; ARM: 922T, AM1x, AM335x, AM3x, ARM7, ARM9, ARM11, ARM926EJ-S, ARMv5TE, AT91SAM, ATSAM3U, CM1136JFS/1156T2FS/1176T2FS/720T/740T/7TDMI/920T-ETM/926EJ-S/940T/940T-ETM/946E-S, Cortex-A8, Cortex-M3, Cortex-M4, Cortex-M4F, i.MX35, LT1156T2-S, MDIRAC3, MT6217, OMAP-L1x, PXA27x, SAM7; MIPS: 32 34K, 24K, 24Kc/Kf, 4K (c,m,p), 4KE (c,m,p), 5Kc, 5Kf, 74K, M14K; Power: 603, e300/500/500v2/500mc/600, MPC500/8xx, ppc405/440/440EP, QorIQ P1; SuperH: SH-2A/3-DSP/4/4A; Others: Xtensa;
μC/OS-III [24]	Micrium	68K: Coldfire; ARM: AM18x, ARM7, ARM9, ARM11, Cortex-A8, Cortex-A9, Cortex-R4, Cortex-M0, Cortex-M1, Cortex-M3, Cortex-M4(F), F28M35x, OMAP-Lx, RM4xx, TM4C, TMS570; AVR: AVR32; MSP430: MSP430X; PIC: PIC24, PIC32; Power: 405, 440; x86: all; Others: MicroBlaze, 68HC12, DSP58K, HCS12, HCS12X, M683XX, MC9S08, FR, Lattice Mico32, XA;
OS-9 [16, 20]	Microware LP	68K: Coldfire; ARM: StrongARM, XScale; MIPS: MIPS3000, MIPS64; PowerPC, SuperH: SH-3, SH-4, SH-4A, x86,
QNX Neutrino [18, 19]	QNX Software Systems	ARM: Cortex A5, Cortex A8, Cortex A15, ARMv7 MP, ARM1136JF-S, ARM926EJ-S, ARM1136EJ-S, OMAP 2420/5905/5912, PXA 250/270, IXP 425/2351/2800, 80200, ARM 920, ARM720T; Power: ppc405/440, 460, 750, 970, ML300, ML403, ML507, QorIQ Pxxxx, MPC5xx, MPC5121/5200/5200B, MPC82xx/83xx PowerQUICC II, MPC85xx PowerQUICC III, MPC8641D; x86: all;
RTAI [3, 4]	DIAPM	68k: all; ARM: StrongARM, clps711x-family, Cirrus Logic EP7xxx, CS89712, PXA25x; PowerPC: all; x86: all;
RTEMS [1]	OAR Corporation	68K: MC68xxx, MC683xx, Coldfire, ARM; MIPS: R3000; R4000; Power: 4xx, 5xx, 6xx, 7xx, 74xx, 8xx; SuperH: SH-1/2/3/4; Others: Blackfin, H8/300, Lattice Mico32, Moxie, NIOS II, SPARC, SPARCv9;
RODOS [5, 6]	German Aerospace Center	ARM: ARM7, ARM11, STM32 32-bit ARM Cortex-M3; AVR: AVR32; Power: 405;
SYS/BIOS [12]	Texas Instruments	ARM: F28M35,TM4C, AM335x; MSP430;
VxWorks [22]	Wind River Systems	68K: 68030, 68040, 68060, 603xx, ColdFire; ARM: ARM7, ARM9, ARM11, Cortex A8, Cortex A9, Cortex A15, I.MX31, StrongARM; MIPS: BCMxxxx, 20kx, 4kx, 5kx, MIPS32/64, Octeon, RISCore3xxx, RM9xxx, RMI AU1xxx, Sb1, TX49xx, vr41xx, vr54xx, vr55xx, vr77xx, XLP, XLR, XLS; Power: 4xx, 405xx, 440xx, 5xx, 5200, 6xx, 7xx, 74xx, 82xx, 8xx, 970, MPC74xx/7xx/82xx/85xx/86xx/MPC8xx, QorIQ p10xx/p20xx/p40xx/p50xx/T4xxx; SuperH: SH-3, SH-4, SH-4A; SPARC: UltraSPARC-II; x86: all; XScale: IOP, IXP, PXA; Others: i960, Equator;

Name	API Standards	Scheduling	Number of threads	Priority levels	Inter task communication	Multicore
FreeRTOS	N/A	preemptive, cooperative, hybrid; round-robbin	unlimited	unlimited	binary/counting semaphores, mutexes, queues, recursive semaphores, notifications	N/A
INTime Distributed	N/A	preemptive; round-robbin	unlimited: 1 or 2 per core	256	mailboxes, semaphores, GOBSnet	AMP, GOBSnet
LynxOS	ABI, POSIX 1003.1-2003 PSE 53/54, ARINC 653-1 APEX	preemptive, non-preemptive; FIFO, priority quantum, round-robin	unlimited	256	semaphores, shared memory, sockets, signals, pipes, message queues, mutexes, condition variables	SMP
Nucleus	POSIX, μITRON	preemptive, priority based, multicore	unlimited	1024	counting semaphores, event flags, fixed and variable queues and pipes, mailboxes, mutexes, signals, rpmspg	AMP, SMP with rpmspg over virtIO and MCAPI, BCD
μC/OS-III	POSIX	preemptive, priority based; round-robin	unlimited	unlimited	event flags, message queues, mutexes, semaphores	N/A
OS-9	POSIX	preemptive, fixed-priority; round-robin scheduling within each priority	unlimited	65525	events, semaphores, signals, pipes, data modules	AMP, RTS Hypervisor
QNX Neutrino	POSIX	preemptive, priority based; FIFO, round-robin, sporadic	unlimited	256	atomic operations, condition variables, messages, mutexes, semaphores, signals	AMP, BMP, SMP, TDP
RTAI	POSIX 1003.1c	preemptive, priority based	2^31/2	2^31/2	mailboxes, messages, mutexes, semaphores, shared memory, signals	MUP, SMP
RTEMS	POSIX, μITRON, RTEID	preemptive, priority based; constant bandwidth server scheduling, deterministic priority, earliest deadline first, simple priority	unlimited	256	events, message queues, mutexes, semaphores, signals	AMP, SMP (under development)
RODOS	N/A	preemptive, priority based; round robin	unlimited	2^31+1	events, semaphores	N/A
SYS/BIOS	N/A	preemptive, priority based	36	66	semaphores, mutexes, mailboxes	AMP, SMP
VxWorks	POSIX PSE52	preemptive, round-robin	unlimited	256	message queues, mutual exclusion semaphores	AMP, SMP

Name	MMU	FPU	Networking	File systems	Device drivers
FreeRTOS	no	yes	ARP, DHCP, DNS, IPv4/IPv6, TCP/UDP (via add-ons)	FAT (via add-on)	UART, I2C and SPI
INTime Distributed RTOS	yes	yes	BPF, DHCP, IPv4/IPv6, TCP/UDP	FAT32	AS-Interface, CANopen, ControlNet, DeviceNet, EtherCAT, EtherNet/IP, I/O, IATA, InterBus, PCI, POWERLINK, RS-232, Modbus, Motion Control, PROFIBUS, PROFINET, RAM Disk, SATA, SERCOS, USB, VGA
LynxOS	yes	yes	ARP, BGP-4, DHCP, EGP, FTP, ICMP, IGMP, ipfw/ip6fw, IPsec/IKE/VPN, IPv4/IPv6, NAT, NTALK, NTPv3, OSPFv2, PPP, PXE Netboot, RARP, RIP, RPC, Samba, SMTP, SNMP, TCP/UDP, Telnet, TFTP, TFTP boot	Lynx Fast File, NFS, RAM disk	DRM, EIDE, IDE, Flash, PCMCIA, PTY, SCSI, SATA, UART, USB
Nucleus	yes	yes	802.1x, ARP, BOOTP, DHCP, DNS/DNSv6, Duplicate Address Detection, FTP, ICMP/ICMPv6, IGMP, IP Forwarding and Reassembly, IPSec/IKE, IP Tunneling, IPv4/IPv6, MLD, NAT, Neighbor Discovery, NUD, New Reno TCP Congestion Control, PMTU, PPP, PPPoE, RARP, SSL, SNTP, SNMP, Stateless Address Autoconfiguration, TCP/UDP, Telnet, TFTP	Nucleus SAFE, VFAT, VFS	Bluetooth, DataFlash, CAN, CANOpen, I2C, LCD, NAND, NOR, PCI/PCI-X/PCI-e, RAM Disk, SD/MMC, SDIO, SPI, Touch Panel, UART, USB, WiFi, Zigbee
μC/OS-III	yes (μC/OS-MMU)	yes	TCP/IP (via μC/TCP-IP extension)	FAT (via μC/FS extensions)	USB, CAN, Modbus, Bluetooth (via μC/OS extensions)
OS-9	yes	yes	802.11, ATM, IPsec, IPv4/IPv6, PPP, SLIP/CSLIP, SNMP, TCP/UDP	FAT, NFS, Reliance File System, TrueFFS	audio, CAN, compactPCI, EtherCAT, IDE, IEEE 1394, IrDA, microSD, Modbus, modems, parallel, PCI/PCI-X/PCI-e, PCMCIA, printers, SATA, SCSI, serial, USB, VGA
QNX Neutrino	yes	yes	802.11, ARP, DVMRP, IKE, IP filtering, IPSec, IPv4/IPv6, L2 VLAN, ML-PPP, NAT, QoS, SNMP, SSL, SSH, STP, TCP/UDP, WEP, WPA, WP2	CIFS, Ext2, FAT, NFS, ISO9660, Joliet, QNX, UDF, ro: HFS, NTFS	Bluetooth, CAN, eMMC, HDD, I2C, NAND, NOR, SDC, SDIO, Serial, SPI, USB, Zigbee
RTAI	m68k only	yes	ARP, ICMP, IPv4/IPv6, TCP/UDP, RTnet	ext3, reiserfs, journals	CAN, COMEDI, Serial, USB
RTEMS	no	yes	BOOTP, DHCP, FTP, ICMP, IPv4/IPv6, PPPD, TCP/UDP, TFTP; free add-ons: SHTTPD, SSL, SNMP	IMFS, JFFS2, FAT, MiniIMFS, NFS, RFS	Block Devices, ATA, RAM Disk, Flash Disk, SPI, SDIO
RODOS	no	yes	N/A	N/A	Ethernet, MIL, radio, RS232, S3P, SpaceWire, WiFi
SYS/BIOS	yes	yes	DHCP, HTTP, IPv4/IPv6, TCP/UDP	FatFS	EMAC, GPIO, I2C, SDSPI, SPI, UART, Watchdog, WiFi
VxWorks	yes	yes	3G, 802.1x, EAP, IGMP, IKE, IPsec, IPv4/IPv6, MACsec, MLD, NAT, PPP, SNMP, SSH, SSL, TCP/UDP, VRRP, VoIP, Web Server, WiMAX, WPA, WPA2	dosFs, HRFS, NFS	CAN/OPC, TrueFFS, USB

Name	Programming languages	Compilers	IDEs	BSPs	License	Source model
FreeRTOS	C	GNU toolchain: GCC	Eclipse, Visual Studio 2010 Express edition, IAR	4 [14]	modified GNU GPL, proprietary	open
INTime Distributed	C, C++	proprietary, included in IDE	Visual Studio 2005/2008/2010	N/A	proprietary	closed
LynxOS	Ada, C, C++	GNU toolchain: GNU 4.6.3, GDB 7.4.1	Eclipse-based Luminosity, LOCI, SpyKer	more than 50 [11]	proprietary	closed
Nucleus	Ada, C, C++	GNU C/C++, GDB	Eclipse-based Mentor Embedded Sourcery CodeBench	more than 200 [8]	proprietary	closed
μC/OS-III	C	GNU toolchain, RX Compiler, KPIT GNU	Codewarrior, IAR, High-performance Embedded Workshop, Keil, e2studio	more than 50 [23]	proprietary	source available
OS-9	BASIC, C, COBOL, Forth, Pascal	Ultra C/C++	Hawk IDE	N/A	proprietary	closed
QNX Neutrino	C, C++, Java (JamaicaVM)	GNU tool chain : GCC 4.7.3, GDB 7.5	Eclipse based QNX Momentics	more than 150 [17]	proprietary	shared
RTAI	C	GNU toolchain: GCC	N/A	N/A	GNU GPL	open
RTEMS	Ada, C, C++	GNU toolchain: GCC, GDB, GNAT	Eclipse with CDT Plugin	more than 50 [2]	modified GNU GPL	open
RODOS	C, C++, Assembly language	GNU toolchain	N/A	RaspberryPI	BSD	open
SYS/BIOS	C	proprietary, included in IDE, GCC, included in IDE	Eclipse-based Code Composer Studio IDE, Sourcery CodeBench	more than 50 [12]	proprietary	open
VxWorks	Ada, C, C++, Java	GNU toolchain, Wind River Diab Compiler, Wind River GNU Compiler, Intel C++ Compiler	Eclipse-based Workbench IDE	more than 50 [21]	proprietary	closed – available for customers

ACKNOWLEDGEMENTS

Research was partially supported by the National Centre for Research and Development under grant No. PBS1/A3/8/2012.

AUTHORS

Piotr Kmiecik* – Lodz University of Technology, Instutute of Automatic Control, Stefanowskiego 18/22, Lodz, 90-924, e-mail: kmc-85@o2.pl, www: www.robotyka.p.lodz.pl.

Grzegorz Granosik – Lodz University of Technology, Instutute of Automatic Control, Stefanowskiego 18/22, Lodz, 90-924, e-mail: granosik@p.lodz.pl, www: www.robotyka.p.lodz.pl.

*Corresponding author

REFERENCES

[1] O. Corporation. "Rtems website", feb 2015.

[2] O. Corporation. "Rtems website", feb 2015.

[3] DIAPM. "Rtai user manual", feb 2015.

[4] DIAPM. "Rtai website", feb 2015.

[5] DLR. "Rodos overview", feb 2015.

[6] DLR. "Rodos website", feb 2015.

[7] M. Embedded. "Nucleus rtos datasheet", feb 2015.

[8] M. Graphics. "Bsp list for nucleus", feb 2015.

[9] M. Graphics. "Nucleus website", feb 2015.

[10] L. Inc. "Lynxos rtos the world's most powerful, open-standards real-time os", feb 2014.

[11] L. Inc. "Bsp list for lynxos rtos", feb 2015.

[12] T. Instruments. "Sys/bios homepage", feb 2015.

[13] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*, Springer Science & Business Media, 2011.

[14] R. T. E. Ltd. "Bsp list for freertos", feb 2015.

[15] R. T. E. Ltd. "Freertos website", feb 2015.

[16] Microware. "Os-9 website", feb 2015.

[17] QNX. "Bsp list for qnx", feb 2015.

[18] QNX. "Qnx developers guide", feb 2015.

[19] QNX. "Qnx website", feb 2015.

[20] Radisys. "Os-9 datasheet", feb 2015.

[21] W. Riber. "Bsp list for vxworks", feb 2015.

[22] W. River. "Vxworks homepage", feb 2015.

[23] M. E. Software. "Bsp list for uc/os-iii", feb 2015.

[24] M. E. Software. "uc/os-iii website", feb 2015.

[25] L. S. Technologies. "Lynx rtos website", feb 2015.

[26] L. S. Technologies. "Lynxos-178 certifiable rtos for safety-critical computing", feb 2015.

[27] tenAsys. "Intime distributed rtos preliminary", feb 2015.

[28] tenAsys. "Intime distributed rtos website", feb 2015.