Predictability in Embedded Linux and Commercial Requirements

Masatoshi Iwasaki Tatsuo Nakajima
Department of Computer and Information Science
Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555 Japan
pingoo@dcl.info.waseda.ac.jp, tatsuo@dcl.info.waseda.ac.jp

Abstract

In Japan, many companies who are building complex embedded systems such as cellular phones and digital televisions are considering to adopt embedded Linux for their products. Most of these companies have been adopted the ITRON real-time operating system as their operating systems, but the functionalities are not enough to build complex embedded systems because ITRON does not provide memory protection, and file systems and network systems are provided as additional middleware components. Many future embedded systems need to access various services on the Internet, and Linux already provide a lot of middleware to reduce development cost dramatically.

Japan Embedded Linux Consortium consists of about 100 companies in Japan is now discussing on how to migrate from ITRON-based embedded systems to Linux-based embedded systems. Especially, many companies are interested in whether embedded Linux can satisfy timing constraints for their products or not.

Our paper describes requirements on timing constraints that should be satisfied in their products. We have been discussing the requirements in the Real-Time Working Group of Japan Embedded Linux Consortium, and we like to show whether current Linux can satisfy the requirements.

1 Introduction

Many manufacturers who are building embedded systems in Japan have been adopted ITRON to their products. Recently they are considering to use Linux as a replacement of ITRON. Because ITRON is lacking some major features of modern operating systems such as file system, networking, and memory protection. Also, they do not offer multimedia functionalities and various middleware.

But the migration of operating systems is not easy. The manufacturers have to review whether Linux can satisfy their requirements. They might change in development environments and hardwares if they decide to adopt Linux for their products. There are many types of embedded systems in Japan. All of them do not need Linux. Multimedia and network attached devices such as PDA and access terminals for wireless networking have been already adopted Linux. On the other hand, small and price-constrained devices will adopt ITRON and other RTOSes. All products belonging to this category have to be extremely high responsive and very inexpensive. Adopting Linux to these devices is so hard because the migration to Linux costs high and there are no need for fully functional operating systems. Just they hope is simple, fast and inexpensive. Other devices that are needed to meet their timing-constraints as same as traditional RTOSes but also wanted to be more functionalities than them are the main target of the discussions in this paper.

In the following sections, we show the various aspects of embedded systems in Japan and what their manufacturers expect to Linux. At first , we introduce Japan Embedded Linux Consortium, Emblix . In this section, we explain why Emblix are founded and what the members of Emblix are discussing. In Section 3, we describe the relationship between embedded systems and ubiquitous computing. Section 4 presents the detailed description of embedded systems in Japan. And requirements for adopting Linux to RTOS-based embedded systems are introduced in Section 5. In Section 6, we show the real-time capabilities of Linux kernel: minimizing non-preemptable sections and high resolution timer. Section 7 shows the simple experiments and results on the hardware that are really used as embedded systems, not x86-based high performance workstations but ARM, MIPS and SH. Finally, Section 8 concludes the paper.

2 Japan Embedded Linux Consortium

Japan Embedded Linux Consortium(Emblix) has been established from July of 2000 to promote Linux in embedded areas in Japan. Currently, about 100 members have been involved in our consortium. The members include Sony, Panasonic, Toshiba, IBM, NEC and Fujitsu. In our consortium, the above members have been discussed various issues to adopt Linux for their embedded products. In the section, we present a background and current activities of Emblix, and show our future plan. In the next section, we also would like to describe the relationship between ubiquitous computing and embedded systems, and our opinions to show the importance of ubiquitous computing in future embedded system communities.

2.1 Linux and ITRON

In Japan, a lot of industrial embedded products has adopted the *ITRON specification operating system(ITRON).* ITRON is not an actual operating system implementation. It specifies the kernel interface, and many vendors has been implemented the specification for their products. Also, many RTOS vendors have sold products to implement the specification. The specification contains basic functionalities such as scheduling, thread managements, and simple inter process communication. Thus, respective companies have been implemented a lot for software on the operating systems. Currently, the operating systems have been adopted by many products such as digital televisions and cellular phones that are developed in Japan.

However, most of embedded systems have became very complex now. For example, a current cellular phone in Japan contains a Web browser, a Java virtual machine, e-mail software, and camera software. On ITRON, these software modules are running on a single address space. To implement these software modules in a robust way, we need a more powerful operating system. Linux currently supports various CPU architectures, and many kernel modules can be loaded dynamically. The characteristics are very suitable for embedded systems. Therefore, many industries have considered that using Linux for their products provides big merits for them.

2.2 Motivation of Emblix

As described in the previous section, many embedded system products have adopted ITRON in Japan. Therefore, there is a big barrier to adopt Linux for their products because there are a lot of differences between the operating systems. One of the most important roles of Emblix is to identify and solve problems to migrate ITRON to embedded Linux in a smooth way. For example, most developers expect to use existing tool chains to develop embedded systems even if they adopt embedded Linux. Also, we need to reuse a lot of software developed on ITRON because we need to develop actual products very quickly, and there is no time to rewrite entire programs on Linux completely. Moreover, many people do not know issues for using Linux in embedded areas such as legal issues and worst case response time.

It is very important to discuss these issues publicly to share a lot of knowledge about using Linux for embedded systems. Emblix presents various issues to adopt embedded Linux at many embedded system forums, and defines specifications about technologies related to the migration from ITRON to embedded Linux.

2.3 Current Activities of Emblix

Currently, Emblix organizes four working groups to discuss issues about embedded Linux. The first working group discusses legal issues. The group is especially talking about issues about GPL and its impact to embedded products.

The second working group is working on the hybrid architecture. Especially, we are working on making a specification of “Linux on ITRON”. Linux on ITRON defines a specification that executes Linux on ITRON. Thus, both ITRON-based software and Linux-based software can be executed on the same system. This makes the migration from ITRON to Linux very easy because we can still use existing ITRON-based software with Linux-based software to build Linux-based embedded systems.

The third working group is working on the development environments for embedded Linux. The group defines several specifications to use ITRON-based development tools for embedded Linux.

The fourth working group is working on the real-time capabilities of embedded Linux. Especially, the group is interested in the worst case response time of embedded Linux that reports in the paper. Also, the group is working on the differences among several real-time extensions of Linux.

2.4 Future Plan of Emblix

In Japan, currently several products that adopt embedded Linux have appeared in the markets, and many products based on Linux will be shipped next year. We believe that the first phase of our role is finished. However, we need to consider many issues for building future embedded systems. For example, we need to take into account security and privacy for Internet appliances. Also, we need to take into account educational issues since traditional embedded systems use various operating system platforms, and it is not easy to use them for educational purposes. We believe that Linux may solve many educational problems due to its openness.
Currently, the most important issue in Emblix is how to establish a open source community for embedded systems. We still need many features in Linux to build future embedded systems. Also, it is very important to provide various open source middleware on embedded Linux to reduce the development cost of future embedded systems. The open source middleware enables us to build various embedded systems in research communities, and it makes it very easy to create various research prototypes and make them actual products very quickly.

3 Embedded Systems and Ubiquitous Computing

In the future, ubiquitous computing environments[4, 8] will change our lives dramatically. The vision of ubiquitous computing environments is to acquire information in our environments that are not available before by using sensor technologies[1]. Also, the environments will make it possible to control many everyday objects by embedding very small and cheap computers. One of the most important issues to realize ubiquitous computing is to integrate a real world and a cyber space in a seamless way. This makes it possible to merge bits and atoms[6]. Thus, software infrastructure for ubiquitous computing should provide a world model that provides a model of our world, which can be accessed by a program, and an application can change its behavior and change the real world by accessing the model[2]. Also, a model in a cyber space can be manipulated by a physical object[3].
In ubiquitous computing environments, we need to consider ultra heterogeneity in various aspects such as hardware platforms, application's requirements, and environmental divergence. For example, it is not easy to develop software on respective hardware platforms. Once a program is written, the program should be executed on various platforms. However, it is very difficult to achieve the goal since respective requirements are surprisingly different. We believe that it is necessary to take into account physical environments such as resource constraints, distribution, and failure explicitly when designing programs. Also, we think considering the each component's assumption and the dependencies among components is very important to build ultra portable software.

In our projects, we are currently working on developing middleware for ubiquitous computing environments. We believe that our middleware will support various future ubiquitous computing applications such as smart space applications, entertainment applications such as robotic and game, and ad-hoc interpersonal communication applications. If there is no right middleware support, it is very hard to implement these applications.

We are currently organizing the following four projects: *Autonomic Components*, *Pervasive Servers*, *Universal Interaction*, and *Smart Materials*. The *autonomic component project* provides a component framework for ultra heterogeneous environments for building ultra portable applications. The framework makes it possible to adapt a component according to application requirements, platform requirements and environment requirements by supporting dependencies among components and assumptions of each component explicitly. In the *pervasive server project*, a large number of micro servers are embedded in our environments. Some servers contain sensors to provide information about the real world, and other servers allow us to control actual objects from programs. A mobile server that is wearable by each person integrates several pervasive servers near from the mobile server in an ad-hoc way. We are developing several applications on a mobile server to show the effectiveness of our approach. The *universal interaction project* allows us to use any interaction devices to access various services. For example, a control panel can be shown at the nearest display from a user, and the panel can be navigated by various input devices such as a PDA, a cellular phone, and a game console. Also, the structure of the presentation is changed according to a user's situation. If a user can use both a large display and a small display, a video will be shown on the large display, but a control panel is appeared on the small display. Also, in the project, we are working on how several services are composed without noticing multiple services from a user. In the *smart material* project, a world model is created by using various sensor technologies. Also, the project is developing a framework to accessing the world model. In the future, various sensors are contained in everyday materials. The goal of the project is

to extract various information from everyday object without a great effort.
One of the most important issues in our projects is that ubiquitous computing middleware will become future embedded system middleware and offer additional values to current embedded systems. This means that embedded Linux is a candidate for an operating system for ubiquitous computing. Also, we need to cooperate with industries in embedded areas to achieve the vision of ubiquitous computing. For example, in our previous projects, we have implemented HAVi on Linux[5] with Fujitsu LSI Solution Limited. Our experiences show that current embedded technologies are important foundations to realize ubiquitous computing. One of future roles of Emblix is to bridge the gap between embedded technologies and ubiquitous computing technologies to make our future lives more fruitful because various future appliances and services related to embedded systems will need ubiquitous computing technologies. We believe that merging between ubiquitous computing and embedded system is one of important keys to improve Japanese economy.

4 Current Status of Embedded Systems in Japan

4.1 Features

The primary intention of Japanese embedded system manufacturers on making their product are reliable , highly responsive and low cost hardware that comes from making their products inexpensive.
Embedded systems developed in Japan were originally hardware oriented design. The functionalities accomplished by software were very small.

4.1.1 Reliability

All of manufacturers think that their embedded products must be stable and reliable. Because reliability of products has been a major factor that consumers judge whether it is fine product to buy in Japan. For examples embedded products have to reboot and recover safely even if there is a sudden stop of a power supply. There should be no or few instructions for their users when the products is in trouble. Because "easily and safely to use" means reliable for consumers in Japan.

4.1.2 Low Response Time

From the point of usability, high response is the most important factor. The products are forbidden to annoy users. To be high response is also the quality of the product. For example, electronic music instruments must meet response time of 1ms between pressing button and making sound whenever the system load is high.

4.1.3 Low Cost Hardware

In making embedded systems, mass production is a key factor for their prices, especially embedded systems for home electronics devices. To accomplish this, many companies prefer to select inexpensive hardware for their products. Because it affects the price that the consumers pays. It limits the software functionality in order to that low cost hardware decreases the available size of memories and performance of CPU compared to higher one.

4.1.4 RTOS

Because of features described above, many companies adopt RTOS to their products. Most of RTOS offer high response and low hardware spec. ITRON is the most popular RTOS in Japan. ITRON itself is a specification for building real-time operating systems for embedded systems. The specification is open and freely available. The manufacturers use this specifications for making their embedded systems.

4.2 Embedded Linux Trends in Japan

Japanese manufacturers joining Emblix are considering to use Linux as an operating system for their products. The embedded system manufacturers joined to Emblix can be divided into three groups by their positions to Linux. The members belonging to the first group are adopting Linux to their products or already adopted. They have a plan to sell Linux-based system and some are selling them now. In this group, the product that they made are mostly related multimedia and networking functionalities such as Sony CSV-E77[17] which is Linux-based PVR and Sharp SL-5500[18], Linux-based PDA. They actively use Linux to their products because ITRON and some other RTOS doesn't have multimedia and networking support by default. The problem of participants in this group is GPL. Many embedded system manufacturers say it is disadvantage for them to disclose hardware specifications and source codes in public because there are many know-hows to build embedded systems and GPL makes them widely available. The embedded systems in this category needs more than 100ms for response. That is not difficult for Linux to satisfy it.
The second group is consists of the manufacturers that are wondering to adopt Linux to their products. In fact, they want to replace RTOS that they

are using to Linux. As described in the previous sections, Linux has some advantages for supporting multimedia and networking compared to other RTOSes. They also have a plan to adopt middleware to enhance the quality of them although they can't use high performance hardware because it raises the costs of these devices. The most important problems for them is whether Linux can satisfy their timing-constraints. The embedded systems produced by the manufacturers in this group have to meet strict deadlines. These systems are expected to work with less than 1ms latency. We have mentioned the requirements of electronic music instruments as example. Recently many researches shows that the latency of Linux kernel can be minimized less than 1ms. But these results are derived by experiments examined on x86-based high performance hardware. That is not effective for the member of this group. They are working on more poor hardware environments for their products.
The embedded systems in the last group doesn't need Linux. They have to work less than a few hundreds or tens of micro seconds. In addition, they are not required complex task and there are no plan for adding some other features. A/D converter and some car navigation systems are included in this category.

5 The Real-Time Capabilities of Linux

5.1 Priority Inversion

In real-time systems, predictability is the primary factor of building reliable system. Preventing to enhance predictability is caused by priority inversion. In Linux kernel, the kernel locks and interrupts are the sources of priority inversions.
Whenever the process invokes a system call, it locks the kernel entirely until the system call completes. The kernel can't reschedule during processing a system call, even if there are processes that have higher priority than the processing one.
Priority inversion also occurs at interrupt processing. Programmers can't define the level of priorities for each interrupt device because Linux doesn't support interrupt level.

5.2 The Real-Time Enhancement of Linux kernel

We have mentioned that a big kernel lock causes priority inversion and makes the response time of the system worse. In this section, we show some approaches to improve the Linux kernel. The factors that cause the latency of the kernel can be separated three major factors .

- Timer resolution
- Scheduling jitters
- Non-preemptable sections in the kernel

The most important factor is timer resolution. If we don't avoid the latency of timer resolution, we can't see any jitters less than 10ms which is a default period of system tick [9]. Scheduling jitter is thought as less important than others when the process runs with high priority. But if there are many processes which contains real-time processes and normal processes that runs as user-interface, scheduling jitter affects the response time and the usabilities.

5.2.1 Minimizing Non-Preemptable Section

By now there are three major techniques to minimizing non-preemptable sections.

- Preemptable kernel
- Low-latency kernel
- Preemptable Lock-breaking kernel

Preemptable kernel[10] is a fully-preemptable kernel. It can reschedule processes after finishing processing interrupts if there are no spin-locks to disable rescheduling. By applying this approach, higher priority process can preempt kernel execution from lower priority process.
Low-latency kernel[11] is the approach to insert preemption points into the kernel. It can reduce non-preemptable sections.
Preemptable lock-breaking kernel is mixed way of above two approaches. Preemptable kernel can't preempt the kernel execution if there are spin-locks. This approach breaks a long spin-lock into small spin-locks. Two patches for preemptable kernel and low-latency kernel can be applied for the same kernel tree. It improves the response time of each individual patched kernel. Lock-breaking patch is based on low-latency patch[10]. This approach seems to be the best one of these[15]
These approach reduces the length of priority inversion and improves Linux response time dramatically.

5.2.2 High Resolution Timer

In a normal configuration, Linux kernel' system tick is equal to 10ms in all architectures except Alpha. It indicates that periodic executions less than 10ms are impossible if it uses the system default timer for them. Such executions can be achieved by using RTC for their timers. RTC, which is the high resolution timer, can generate signals from 2Hz to 8192Hz. The periodic processes can be notified by interrupts from /dev/rtc. RTC is also sufficient when the period is more than 10ms. As we have already mentioned, timer transaction is processed at bottom-half and it is unpredictable when it starts and how long it takes.

5.3 Other Problems

Some techniques to minimize non-preemptable sections and adopting high resolution timer is sufficient for real-time processes. But it can't work out the unpredictability of bottom-half processing. Many researches on x86-based high performance hardware shows that there is no influence of the processing time of bottom-half. But at low performance hardware, the processing time might be increased and prevent real-time and other processes. In fact some engineers indicated at a workshop of Emblix. They are worried that it occurs after they release their products and claimed by customers as the products doesn't work correctly.

6 Experiments

In this section, we measured the delay of task wake up time. We have chosen RTC for high resolution timer. The process with scheduling FIFO receives signals via /dev/rtc. It means that we did not modify the kernel source code. The processes are waked up by a periodic signal with period 1ms which is less than system ticks. Accessing RTC via /dev/rtc is one of the way to accomplish high response without any modifications to the kernel. We have adopted IOzone[14] as a background load for measuring because the disk access often causes the worst case delays[9]. This benchmarks double the size of a file that it reads and writes as the time goes. RTC generates 1024 times interrupts for 1 seconds. The target kernels for this experiment are based 2.4.18 from kernel.org[16]. We use four kernels. The first is vanilla 2.4.18. The other kernels are derived by patching to vanilla kernel: Preemption patch , Low-Latency patch and Preemption and Lock-breaking patch. We have examined these experiments on Pentium II 350 MHz with RAM 128MB.

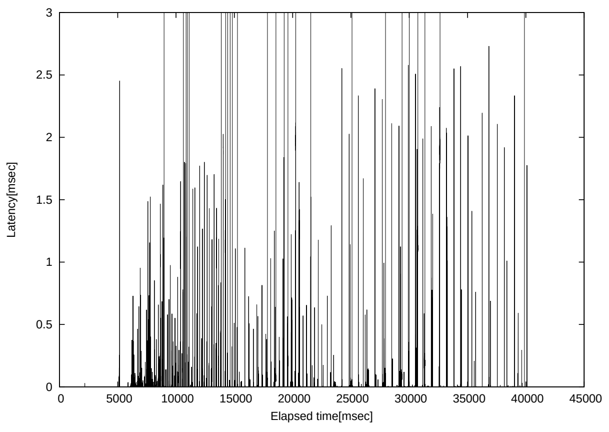

FIGURE 1: *Latency measured on 2.4.18*

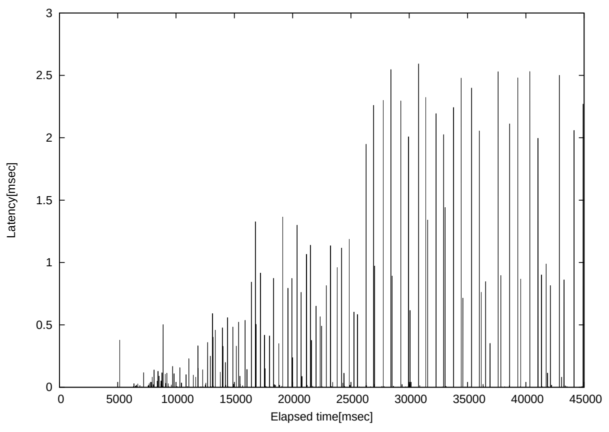

FIGURE 2: *Latency on 2.4.18 with preemption patch*

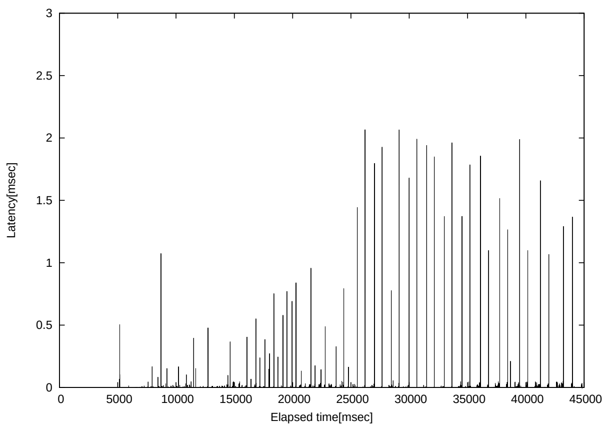

FIGURE 3: *Latency measured on 2.4.18 with low-latency patch*

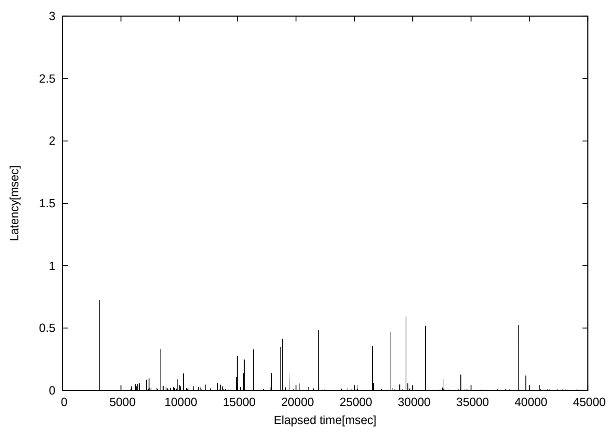

FIGURE 4: *Latency measured on 2.4.18 with preemption and lock-breaking patch*

Figure 1 shows the result of an experiment with 2.4.18 vanilla kernel. The response time should be about 1ms because of the frequency of RTC. But with the vanilla kernel, the worst case is about 180ms.
The other results are derived from real-time enhanced kernels. See Figure 2, 3, 4. The results of the preemption kernel are dramatically improved than the one of the vanilla kernel. The low-latency kernel is also superior to the vanilla kernel. The preemptable and the lock-breaking kernel got the best performance in these kernels and all latencies are less than 1 ms.

7 Discussions

In this section, we discuss the difficulties of measuring performance on non-x86 architecture. We needs some conditions to measure latency of Linux kernel with high resolution timer.

- The target must have high resolution timer such as RTC.
- The target architecture must have cycle counters such as TSC on Pentium.

Many sample boards for embedded systems meet first and third condition. But manipulating RTCs and cycle counters for each board are not easy. Because the addresses of RTC varies on each board and how to get values of cycle counters are different between each architecture. And the second conditions is much more difficult. As far as we know, there are two architectures which have device drivers of RTC that support to generate periodic signals, x86 and ARM. Device drivers included other architectures only support loading and restoring of the time. Although there are some architectures that have another high resolution timer replacing RTC, it also takes many times to setting environments of measurement.

There are also some difficulties about measuring on embedded systems. The developing environments for embedded systems are not as easy as the ones for desktop computers. Most of the boards have serial ports for console and this is the only way to manipulate the entire system. And how to load boot images varies on systems. To work out this difficulties, we can set our environment with NFS or some technique. But it also takes many times and it is better to buy commercial software for developing environment than you do it yourself.

These situations indicate that measurements on various architectures are hard tasks and take many times. On the other hand, Linux is ported many architectures and there are some requirements to making the real-time performance for each architecture public.

8 Related Work

Linux/RT developed by Timesys corporation[12] is fully-preemptable kernel for embedded systems. MontaVistaLinux[13] has preemptable lock-breaking kernels for many architecture and various boards.

9 Conclusion

In this paper we have described the requirements for embedded systems to adopt Linux and the status of embedded systems in Japan. We have also explained some approaches to improve response time of the Linux kernel. And we have shown that the latency of kernel by our experiments. As the result shows, we can say that Linux can satisfy the requirements of embedded systems.

In the future, we have a plan to measure the latency of kernel on various architectures and boards. It can reveal the performance of each architecture and engineers can decide which architecture to use for their products. It will be a good motivation for manufacturers to use Linux as an operating system for their products. Because many researchers have experimented the predictability of Linux and other operating systems such as NetBSD and FreeBSD do not provide enough information to be used in embedded systems.

References

[1] H.-W.Gellersen. A.Schmidt, and M.Beigl, “Adding Some Smartness tp Devices and Everyday Things”, In Proceedings of the Third Workshop on Mobile Computing System and Applications, 2000.

[2] A. Harter, A. Hopper, P. Steggles, A. Ward, P. Webster, “The Anatomy of a Context-Aware Application”. In “Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking”, 1999.

[3] H. Ishii, B.Ullmer, “Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms”, In Proceedings of Conference on Human Factors in Computing Systems,1997.

[4] T. Nakajima, et. al., “Technology Challenges for Building Internet-Scale Ubiquitous Computing”, In Proceedings of the Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems, 2002.

[5] T.Nakajima, “Experiences with Building Middleware for Audio and Visual Networked Home Appliances on Commodity Software”, In Proceedings of ACM Multimedia 2002, 2002.

[6] Nicholas Negroponte, ”Being Digital”, BeingVintage Books, 1996.

[7] R.Want, T.Pering, G.Danneels, M.Kumar,M.Sundar, J.Light, “The Personal Server: Changing the Way We Think About Ubiquitous Computing”, In Proceedings of Ubicomp2002.

[8] Mark Weiser, “The Computer for the 21st Century”, Scientific American, Vol. 265, No.3, 1991.

[9] Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, Jonathan Walpole “ A Measurement-Based Analysis of the Real-Time Performance of the Linux Kernel”, In Proceedings of the Real Time Technology and Applications Symposium (RTAS), Sep 2002.

[10] Robert M Love, Linux kernel patches, http://kpreempt.sourceforge.net/

[11] Andrew Morton, Linux scheduling latency, http://www.zip.com.au/ akpm/linux/schedlat.html/

[12] Timesys cooperation , TimeSys Linux/RT, http://www.timesys.com/

[13] MontaVista Software cooperation , MontaVistaLinux, http://www.mvista.com/

[14] IOzone Filesystem Benchmark, http://www.iozone.org/

[15] Clark Williams, Linux Scheduler Latency, http://www.linuxdevices.com/articles/AT8906594941.html

[16] Linux Kernel,http://www.kernel.org

[17] Sony Corporation, CSV-E77,http://www.sony.jp/products/Consumer/cocoon/

[18] Sharp Corporation, Zaurus SL-5500, http://www.sharp-usa.com/products/ModelLanding/0,1058,698,00.html