A Study on the Design of the Monitoring Architecture for Embedded Kernels based on LTT

Ji-Hye Bae*, Yoon-Young Park*, and Jung-Ho Park*

Abstract: Embedded systems are used in many fields such as home appliances, terminals, controls, communications, etc. So, to manage, control, and test these embedded systems, monitoring programs have been developed variously. In this paper, to overcome the characteristic faults of embedded systems which have resource restrictions, we implemented a development environment based on NFS and designed a monitoring tool that can evaluate and analyze kernel performance in embedded equipment by using LTT(Linux Trace Toolkit). Also, we designed a method to show monitoring data collected by using a monitoring tool, called MONETA 2.0, through the web-page.

Keywords: Monitoring tool, NFS, LTT, MONETA 2.0

1. Introduction

As an embedded system simultaneously requires characteristics of time restriction and logical accuracy, technologies related to computer science (such as semiconductor technology or communications networks) are growing rapidly and many application fields of embedded systems are variously extended.

Embedded systems are systems that contain embedded hardware and software designed to control special-purposed hardware that operate as components of a large system. Embedded softwares, unlike general softwares, are composed of system softwares and middlewares that are loaded on a microprocessor of each piece of control equipments, information equipments, and sensor equipment that requiring characteristics of real-time, high-reliability and low-power, etc.

Also, it is reqwired to use the monitoring system in an embedded system at the level of testing and customizing. Particularly, the monitoring system has various functions to execute many tasks such as debugging, testing, evaluating performance of computer programs, controlling the system and managing the whole system. Because the monitoring system collects, analyzes and manifests dynamically various status information of systems or processes, it is required to manage and test many embedded systems[5]. Applications of the monitoring system are various, we'd like to describe about system monitoring in particular.

In this paper, we applied LTT(Linux Trace Toolkit), which is the main system tracing utility to a monitor kernel an embedded system. Since LTT traces interaction between application programs and other software com-ponents, it can resolve the problems using common symbolic debugging. In addition to reconstructing the system's behavior using the data generated during a trace run, the user utilities provided with LTT allow you to extract performance data regarding the system's behavior during the trace interval[1].

In section 2, existing system monitoring tools are reviewed and described, and in section 3, we examine the method about the design of monitoring tools using LTT. In section 4, we demonstrate about the architecture and implementation “MONETA 2.0” which is the monitoring tool for embedded equipments applied NFS and SNMP, and we describe about reconstructed LTT in the Java environment. Finally, the conclusion and future work are given in section 5.

2. Related Works

In this chapter, we explain the features of existing system monitoring tools. A system monitoring tool is able to evaluate the performance of individual computers for measuring of the system status such as available disk space, CPU usage and network capability. The system performance depends on how to mediate current system resources effectively regarding requirements of many programs. Generally, most important system resources are CPU, memory and disk I/O, and because of the generalization of the internet service, the portion of the network is also important, currently.

Among monitoring systems in distributed environments, the system “Tornado” system means an integrated development environment(IDE) that has simultaneously embedded operations and both a development environment and an execution environment which are used for real-time applications. Tornado has various tools to develop and provides easy and consistent development to users by

Manuscript received September 30, 2005; accepted November 8, 2005.
This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment) (IITA-2005-C1090-0502-0031)

* Department of Computer and Information Science, Sun Moon University, Asan, Korea ({angdoo98, yypark, jhpark}@sunmoon.ac.kr)

connecting each tool. Tornado uses VxWorks as an operating system, it has a very small-sized kernel that provides multitasking environments, interprocess communications and synchronizations[9].

Qplus Esto means IDE(Integrated Development Environment) to develop application softwares executed on an embedded operating system, called Qplus. Qplus is an operating system using processes based on embedded linux, and Esto(Embedded Systems Tool) provides both the Linux system host and the Windows system host as IDE based on GUI(Graphic User Interface). Esto enables things to work (such as coding, compile, execution, debugging and monitoring) for embedded application program developers on a single host platform as an integrated environment and then enables to increment for users convenience of application programs development. Qplus Esto includes a library, target agent, target application program based on a Qplus real-time operating system at the target side. On the other hand, it is composed of application development tools such as host agent(based on Linux/Windows operating system), target builder, cross compile tool-chain, project manager, remote shell, remote debugger, remote monitor and measuring an instrument for measuring power consumption at the host side[4].

Big Brother (http://www.bb4.com) is a monitoring program that practices different access methods to the system monitoring. It can construct a Big Brother Client to various hosts as well as monitor many services (such as POP, HTTP) from central server. Big Brother Client actively monitors the measurement criteria such as disk space, CPU usage and existing process, and then, the results are reported to the Big Brother Server. Using these capabilities, we can assumptively monitor the remote server[6].

DCPI provides a detailed analysis of different processes running on a system down to pipeline stalls. In order to provide its highly detailed data, DCPI uses a very high frequency interrupt. Similarly, Morph uses the clock interrupt to gather data in order to optimize applications off-line. Both systems fail to provide their user with information on the interactions of the different processes. Neither enables the user to understand the dynamics of the observed system[3].

Path Profiler, contrary to DCPI and Morph, is an instrumentation approach to data sampling. Path Profiler is much like GProf but is much richer in detail. The problem is the overhead. On a normal running system, this is often not tolerable. Quantify uses techniques similar to Path Profiler to provide profiling information, but its capabilities remain confined to analyzing one process at a time. Moreover, its overhead is unpublished. All these profiling systems provide detailed analysis of one or many processes, but fail to provide information on system dynamics[3].

Apart from these, there are quite a few tools for measuring the kernel's performance. The most famous is probably LMbench (http://www.bitmover.com/lmbench/). LMbench is micro benchmark that measures hyper-threading effectiveness of linux API including measurements of latency and bandwidth. At these of measurements, file read, memory copy(bcopy), memory read/write, latency, pipe, context switching, networking, create/delete file system, create process, signal processing and processor clock latency are cached. LMbench checks components such as scheduler, process management, communication, networking, memory map, file system and low-level kernel primitive. LMbench, however, requires a C compiler and the Perl interpreter. It is therefore not well adapted for use in embedded systems.

Another tool for measuring kernel performance is Kernprof. Kernprof has functions such as profiling based on time, profiling based on performance counter and ACG(annotated call graph) of kernel space(http://oss.sgi.com/projects/kernprof/). Though it can generate output that can be fed to gprof, it involves the use of a kernel patch and works only for x86, ia64, sparc64, and mips64. As you can see, most embedded architectures are not supported by Kernprof.

We are left with the sample-based profiling functionality built into the kernel. This profiling system works by sampling the instruction pointer on every timer interrupt. It then increments a counter according to the instruction pointer. Over a long period of time, it is expected that the functions where the kernel spends the greatest amount of time will have a higher number of hits than other functions. Though this is a crude kernel profiling method, it is the best one available at this time for most embedded Linux systems[1].

Also, there are a few differences between existing monitoring tools and our monitoring tool in this paper. First, the Tornado system for RTOS is one of the solutions that are not free release, and users might be a little difficult to practice many application programs for development. But, our monitoring system is made up of very simple constructions and there is no cost for developing system performance. Second, Esto based on Qplus embedded linux has to be connected with the GUI application program on the host system implemented by the Java environment on the Eclipse platform, and users have to practice some application programs on Eclipse platform with the whole contents of Qplus embedded linux. On the other hand, users can easily use the monitoring system in this paper because this monitoring system is composed of very simple application programs using only web pages. Third, the above indicated system monitoring tools(like LMbench or Kernprof) are not well adapted for embedded systems because these tools have inadequate requirements for embedded systems (such as interpreter, architecture types). And in case of Path Profiler, it has an overhead problem, but our monitoring system includes LTT functions that the entire process has very little impact on the system's behavior and performance. Extensive tests have shown that the tracing infrastructure has marginal impact when not in use and an impact lower than 2.5% under some of the most stressful conditions[1].

On the above-indicated various monitoring tools, using profiling functions is well adapted for use in embedded systems. For this reason, in the next chapter, we will

describe the design of the monitoring architecture using LTT, which has many profiling functions.

3. The Design of the Embedded Kernel Monitoring Tool

In this section, we describe the 3-layer distributed monitoring system model for monitoring of embedded kernels running on distributed environments and the design of MONETA 2.0 with the architecture of LTT.

3.1 The 3-layer Distributed Monitoring System Model

The Embedded System generally doesn't exist in the form of a single equipments, but distributed system architectures comprise many kinds of embedded pieces of equipments that are connected by wire or are wireless. These distributed environments are divided into servers for monitoring, targets that are to be subjects of monitoring, and clients who can use the system. Generally speaking, targets are embedded pieces of equipment and the server is composed of one of the embedded pieces of equipment or a separate server system. In this paper, we define the monitoring architecture of a distributed system as the distributed monitoring system of 3-layer architecture, and following (Fig. 1) is showed these.

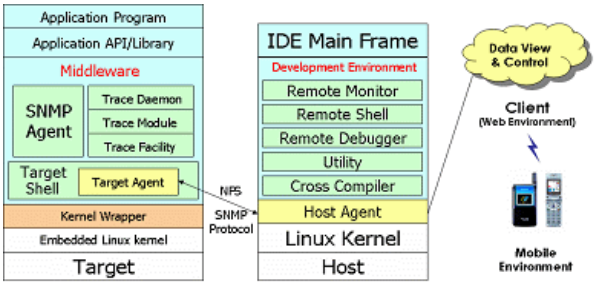


Fig. 1. The Distributed Monitoring System of 3-layer Architecture

Fig. 1 shows the distributed monitoring system of 3-layer architecture about distribution and real-time monitoring architecture in embedded system environments. The target and host are basically connected by NFS (Network File System). Furthermore, we separately installed LTT(the main system tracing utility for linux) to the target and constructed the architecture where tracing data is transmitted from the target through NFS in the LTT visualization tool of the server system. Also, we designed the architecture using SNMP(Simple Network Management Protocol), which enables us to get each kind of application information or monitoring data between the target and host by using simple SNMP commands. To graphically show information from LTT or SNMP on web pages, we constructed a web server on the host PC. Finally, we designed the mobile environment to control and manage the monitoring data.

3.2 The Design of the Kernel Monitoring Tool based on LTT

In this chapter, we'd like to describe the LTT architecture and the method of designing the monitoring tool-named MONETA-that we have implemented for this study.

3.2.1 The Architecture of LTT

The main system tracing utility for Linux is the Linux Trace Toolkit(LTT), which is used for analyzing subsets of executed processes and recording important system events. In contrast with other tracing utilities such as strace, LTT does not use the *ptrace()* mechanism to intercept applications' behavior. Instead, a kernel patch is provided with LTT that instruments key kernel subsystems. The data generated by this instrumentation is then collected by the trace subsystem and forwarded to a trace daemon to be written to disk. The entire process has very little impact on the system's behavior and performance. Extensive tests have shown that the tracing infrastructure has marginal impact when not in use and an impact lower than 2.5% under some of the most stressful conditions. In other words, the loss of system performance caused by interference status is hardly anything[1].

LTT is composed of independent software modules. *The kernel trace facility* requires tracing for the trace module and acts as a link between the trace module and the different kernel facilities. *The trace module* stores the incoming event descriptions and delivers them efficiently to the trace daemon. Also, the trace module has additional information such as time, CPU ID and sets the event mask and tracks a given PID/GID/UID. To efficiently deal with the large quantity of data generated, the trace module uses double-buffering. *The trace daemon* can retrieve additional information from the trace module and save these at given files. The trace daemon uses double-buffering like the trace module[3].

Tracing data files generated by the trace daemon, called '*.trace' and '*.proc', are shown by using a visualization tool on the host PC. To implement these LTT functions, LTT's operation is subdivided into four software components.

- Modified Kernel : It is added the macro that delivered to the kernel module about occurrence of events on each event process part of the kernel. It is the kernel instrumentation that generates the events being traced.
- Kernel Module : It sends a signal to the trace daemon after saving kernel events in the buffer. It is the tracing subsystem that collects the data generated by the kernel instrumentation into a single buffer.
- Daemon : It saves data collected by the kernel module. It is the trace daemon that writes the tracing subsystem's buffers to disk.
- Viewer : It is used on the host PC and displays tracing data in the form of various formats. It is the visualization tool that post-processes the system trace and displays it in a human-readable form.

The first two software components are implemented as a

kernel patch and the last two are separate user-space tools. While the first three software components must run on the target, the last one, the visualization tool, can run on the host[1]. To analyze tracing data, the program made up on the basis of GTK library, and Trace Visualizer is used to show the tracing data.

3.2.2 The Design of the Monitoring Tool

MONETA(Distributed **MON**itoring for **E**mbedded **TA**rget System) is a monitoring tool that shows the collection and analysis of various events issued on running of embedded equipment through web pages in a distributed environment. Between target system and server system are connected by NFS, and many events or data from the target system which is being monitored are saved in the form of files on server system[5]. MONETA 2.0 is implemented by adding LTT functions to the existing monitoring tool, called MONETA. In the following, we describe the method of design for addition of LTT functions in MONETA 2.0. The following Fig. 2 shows the design plan of MONETA 2.0 in the monitoring server system.

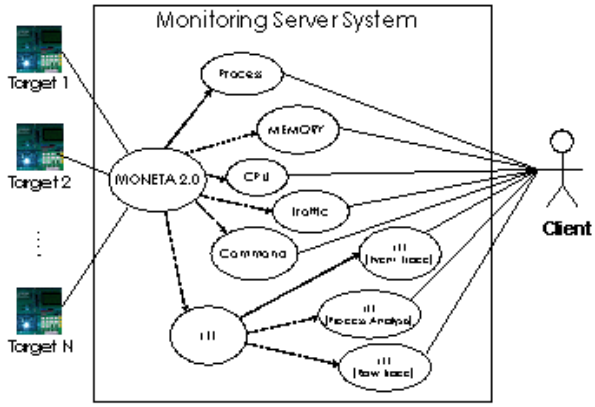


Fig. 2. The Design Plan of the Monitoring Tool in the Monitoring Server System

In Fig. 2, there are many pieces of embedded equipment (being targets) which are composed of separate objects in this monitoring tool. These embedded pieces of equipment send the monitoring information (such as process, memory, cpu, traffic, command, LTT) to web pages through MONETA 2.0. As you see the flow of arrows about each kind of information such as process, memory, etc in MONETA 2.0, each kind of information is to detailed functions of MONETA 2.0. Also, being one of the pieces of information, LTT's detailed functions are composed of event trace, process analysis, and raw trace. Eventually, the client gets these data and controls many target systems.

4. Implementation

In this chapter, we describe the architecture and the method of implementation of the monitoring tool of embedded equipments which is called MONETA 2.0 and applied analysis of tracing files created by LTT. Also, we changed the original LTT visualization tool based on the GTK library into an LTT viewer using Java swing reconstructed by Java environments, and we designed the LTT viewer in the form of a plug-in project on the Eclipse platform(Java Integrated Development Environment).

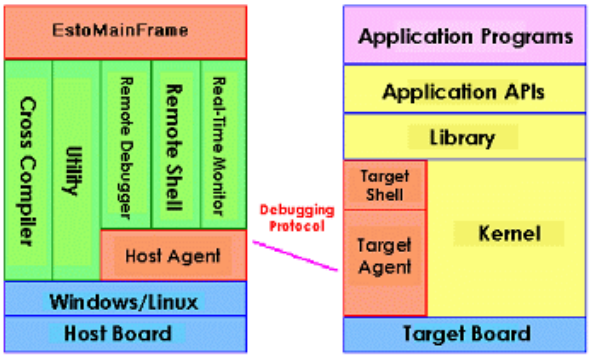


Fig. 3. The Esto Development System Architecture

The development environment using LTT is related to the real-time software development environment, called *Esto*, which is currently being developed by ETRI. (Fig. 3) shows the Esto Development Architecture[4][8]. The cross compiler is the GNU compiler for C programs, and it includes the collection of the tool-chain: gcc, make, ld, as, and binary utilities. While the cross compiler doesn't need any communication with the target, other tools need to communicate with the target.

4.1 The Monitoring Information in MONETA 2.0

In this section, we describe the monitoring information as each system resource. The system resource part is a part about information of embedded kernel performance evaluation, and it has many components including PROCESS, CPU, MEMORY, COMMAND, Kernel Trace of embedded equipments. Also, it can obtain return values about this information using the monitoring system.

① PROCESS : Information about processes running on systems

② CPU : Information about CPU usage and CPU idle time

③ MEMORY : Memory information that is used by users or daemon processes in a whole memory system

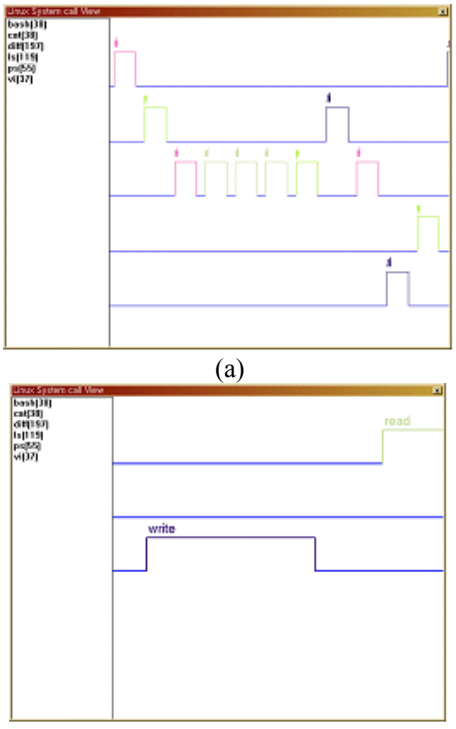
④ COMMAND : Information obtained by using remote shell. Remote shell constructs many functions which are executable commands to obtain information related to the system(such as directory, process, memory, disk, etc) as a simple menu, and then, we can get results that server sends simple commands to embedded equipments. Information obtained by using remote shell is like this:

- Directory Information : Menu that can represent files in directory. Described hiding files and detailed information such as the form of a file, usage authority, file size, etc.

- Process Information : Menu that can represent the process information running on systems. Described an environment variable information related to relevant process and hierarchical relations.
- Memory Information : Menu that can print memory status using on system as outputs.
- Disk Information : There is no disk to read data basically in embedded equipment but data is stored on a file system of a server by using NFS. So, we made a menu that can represent disk information. This menu shows detailed file information in directory as well as basic the directory information.

⑤ Kernel Trace : It is the information for kernel performance evaluation as the created information during kernel traced. In addition to each measuring performance information obtainable from embedded equipments, we implemented many application programs as the method to trace the kernel's internal system and to evaluate embedded kernel performance. Basically, these application programs are implemented based on the proc file system, and they are made up through kernel patches to create a tracing module of the kernel's internal system. The purpose for evaluating kernel performance is to effectively solve a little traffic problems or lots of interference status when various sensor information is collected by users from many pieces of embedded equipments. The component of Kernel Trace includes the following functions:

- Event Graph : It expresses processes or system behaviors that are executed by using kernel tracing information as the form of a graph. (Fig. 4) shows a graph that is presented at a μs unit about system call usage of each process in the embedded kernel. Fig. (b) is an expanding graph, which extends Fig. (a). The left menu frame presents every running process and the other side shows orderly several system calls(like read(), write(), etc) which are used by these processes. The main system tracing utility for Linux is the Linux Trace Toolkit (LTT), which is used for analyzing a subset of executed processes and recording important system events. Like (Fig. 4), LTT provides an event graph which views a list of all the processes that were active during the trace and a graph that characterizes the behavior of the system[1]. The kernel tracing tool in LTT is based on the GTK library, but we implemented the event viewer in (Fig. 4) using the basis of GDI Plus in Microsoft Inc. Because the viewer of the figure is based on Windows, we implemented to operate in web environments by using ActiveX technology.
- Process Analysis : It provides the per-process statistics and system statistics. It mainly presents CPU time, consumption time of running real application programs, I/O waiting time and information about system calls by each application program.



(a)

(b)

Fig. 4. Event Graph of Embedded Kernel System

Next Table 1 briefly shows about the monitoring information[11].

Table 1. The Monitoring Information and Description in Embedded Equipment

Resources	Components	Function	Descriptions
System Resources	PROCESS	PROC Info	Process information about USER, PID. PPID, ST, NAME, CPU, VMEM, TTY, TIME, CMD, PRI, ADDR
		PROC Time	Process analysis time
	CPU	CPU Usage Rate	CPU usage rate
		CPU Usage Time	CPU usage time (Maximum, Average, Present)
		CPU Idle Time	The amount of time spent in idle process out of the uptime of the system (Maximum, Average, Present)
		CPU Info	CPU information
	MEMORY	Total Memory Size	Total memory size installed in the system
		Cached Memory	Cached memory size in run time
		Available Memory	Available memory (Maximum, Average, Present)

		MEM Info	Memory information about page allocate, page free.
	COMMAND	System	Information about system obtained by remote shell(Such as Hostname, System, Kernel information, Date information, etc)
		General Command	Information obtained by general commands(ls, ls -al, ps -ef, free, du -a, etc) using remote shell
	Kernel Trace	Event Graph	Kernel information traced(displayed by a graph about running processes, system behaviors at tracing system time)
		Process Analysis	Kernel information traced(process statistics, system statistics)

These kinds of monitoring information are composed so that results for users can be simply understood through MONETA 2.0.

4.2 LTT Operations in MONETA 2.0

In Fig. 5, we can show the kernel information through the LTT visualization menu in MONETA 2.0 that is added LTT functions to the existing original MONETA tool.

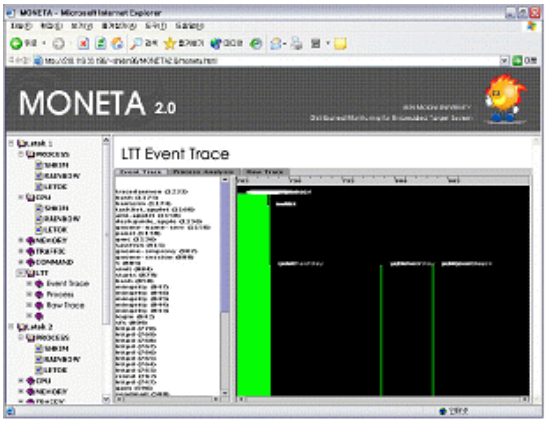


Fig. 5. Event Trace in MONETA 2.0

In Fig. 5, on the screen of MONETA 2.0, we can show the frame on the host PC regarding monitoring information of embedded equipment using LTT functions. The left side of MONETA 2.0 is the menu frame with information about individual embedded equipment and the right side shows information about event trace, which is one of the LTT functions.

The information obtained by using LTT is composed of the event trace, the process analysis and the raw trace. On the event trace frame, we can show a list of all the processes that were active during the trace and a graph that characterizes the behavior of the system. The process analysis provides the per-process statistics and system statistics. Finally, on the raw trace frame, we can easily identify applications' interaction with the rest of the system.

4.3 Reconstructed LTT in Java Environment

In this paper, we implemented the LTT visualization tool based on the GTK library into the LTT viewer using Java swing reconstructed by Java environments, and we designed LTT viewer as a form of plug-in project on the Eclipse platform(Java Integrated Development Environment).

The following Fig. 6 is the design drawing regarding analysis of tracing files created by using LTT in a Java environment.

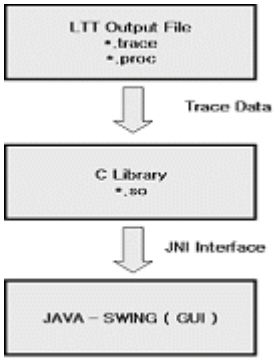


Fig. 6. LTT design to analyze in a Java environment

The explanation about Fig. 6 follows.

- Creation of tracing data in kernel which is provided by LTT patches. Then, the *.proc file and the *.trace file are created.
- Analysis of necessary tracing data from the *.proc and the *.trace files. Analysis results from the C library have to be sampled, and then, by using JNI interface, sampled data is converted to a Java environment.
- Displayed the information about extracted results by using Java swing through the graphic interface.

In this paper, we implemented the LTT visualization tool by using Java swing based on the design in (Fig. 6) and

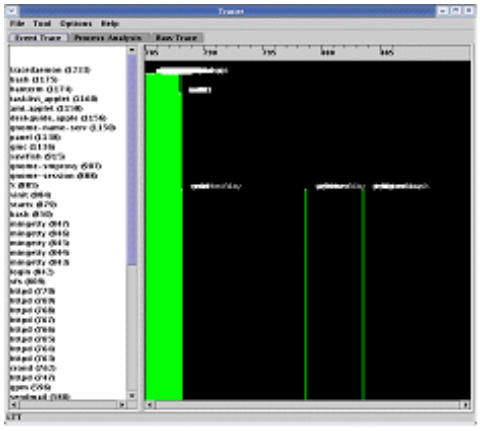


Fig. 7. LTT visualization tool re-implemented by Java swing

(Fig. 7) and it shows the frame of LTT execution in the Java application environments. The functions are the same as with the LTT visualization tool based on the GTK library provided by the LTT package, but the implementtation of Java swing is to be different. LTT visualization implemented by Java swing also includes LTT functions such as event trace, process analysis, and raw trace.

The purpose of analyzing in the Java environment is to change the LTT viewer based on the GTK library into an Eclipse plug-in project based on Java IDE as well as the simple implementation of the Java application. In this paper, we designed the Java application program as the form of the plug-in on Eclipse platform.

Fig. 8 is the frame of virtual implementation that is to be the plug-in LTT viewer on Eclipse platform. In Eclipse workbench, if you click the LTT icon and menu, all of the LTT functions are shown on the Eclipse execution frame [10].

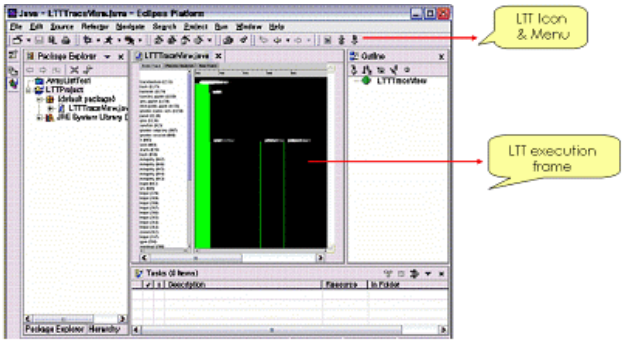


Fig. 8. The Eclipse plug-in development of LTT viewer

In this paper, the purpose of the Eclipse plug-in regarding the LTT viewer is to provide more convenient monitoring development environments to developers and users for kernel system monitoring.

5. Conclusion and Future Work

In this paper, we described the design and implementtation regarding the monitoring system of embedded equipment using LTT with system profiling functions. Also, to analyze performance of embedded equipments, we described the method of monitoring for embedded equipment using application programs with system profiling functions indicated in the monitoring system model. So, we implemented the monitoring tool with LTT functions, MONETA 2.0, which is reconstructed MONETA based on NFS and SNMP and we implemented the method to graphically show monitoring information of embedded equipment on web-pages[5].

Because MONETA 2.0 is basically the basis of the web, it has an advantage that it is enabled to use system monitoring anywhere and anytime on PCs constructed by networks. But it has no function that automatically modifies and resolves many problems or bugs of systems, and in the case of LTT, it wasn't yet implemented for a real-time monitoring system. We need to make additions to new functions that enable us to solve these problems.

References

[1] Karim Yaghmour, “Building Embedded Linux Systems”, O'Reilly.

[2] Opersys Homepage, http://www.opersys.com/LTT

[3] Karim Yaghmour and Michel R.Dagenais, “Measuring and Characterizing System Behavior using Kernel-Level Event Logging”, Proceedings of 2000 USENIX Annual Technical Conference, San Diego, California, USA, June 18-23, 2000.

[4] Embedded Software Technology Center, “Q+ Esto Manual”, ETRI, Korea.

[5] S.H.Kim, “A Study on the Monitoring System for Embedded Device based on the NFS”, Proceedings of the 21st KIPS Spring Conference, vol. 11, no. 1, pp. 899-902, May, 2004.

[6] H.J.Shin, “Unix System Construction and Management”, SungAn-Dang.

[7] Douglas R.Mauro, “Essential SNMP”, O'Reilly.

[8] S.W.Son, C.D.Lim, H.N.Kim, “Debugging Protocol for Remote Cross Development Environment”, IEEE, pp. 394~398, 2000

[9] Yoon-Young Park, “A Study on the Monitoring Model of Distributed Objects”, ETRI, Korea.

[10] Eclipse Homepage, http://www.eclipse.org

[11] Hyo-Sung Kang, Jong-Mu Choi, Jai-Hoon Kim, Young-Bae Go, “Agent-Based Embedded Monitoring System for Ubiquitous Networks Environments,” Proc. of the 2004 International Conference on Parallel and Distributed Processing Techniques and Application (PDPTA 04), Las Vegas, USA, Jun, 2004.

Ji-Hye Bae
She is a graduate student in the Department of Computer Science, Ph.D. course, the graduate school, Sun Moon University, Korea. She received her M.S degree in computer science at Sun Moon University in 2005. Her major is in embedded systems and her current research interests are ubiquitous computing and sensor network environments.
Department of Computer Science, the graduate school, Sun Moon University, Asan, Chungnam, 336-708, Korea

Yoon-Young Park
He is a professor in the Faculty of Computer and Information Sciences, Sun Moon University, Korea. He received his M.S and Ph.D. degrees in computer science from Seoul National University in 1985 and 1994, respectively. His main majors are distributed operating systems and real time systems. Recently, his research interests are sensor networks and ubiquitous computing. He is a member of KIPS(Korea Information Processing Society).
Faculty of Computer and Information Sciences, Sun Moon University, Asan, Chungnam, 336-708, Korea

Jung-Ho Park
He is a professor in the Faculty of Computer and Information Sciences, Sun Moon University, Korea. He received his M.S and Ph.D. degrees in computer science from Osaka University in 1987 and 1990, respectively. His current research interests are distributed algorithms and electronic commerce. Recently, he has an interest and takes an active part in Internet Ethics. He is a member and vice president of KIPS(Korea Information Processing Society).
Faculty of Computer and Information Sciences, Sun Moon University, Asan, Chungnam, 336-708, Korea