Kernel Tracing Toolkit for Embedded Systems[1]

NAM-SIK YOON, JI-HYE BAE, YOON-YOUNG PARK, JIN BAEK KWON
Department of Computer Science
Sunmoon University,
TangJeong-Myeon, Asan-si, Chung-Nam, 336-708
SOUTH KOREA.

Abstract - Embedded systems provide human-centric services in many fields of education, information, industry and service, and various monitoring programs have been developed for managing, controlling and testing for these embedded systems. Currently, many kernel trace toolkits are being used for monitoring. These trace toolkits are complex systems, so we present a simple and explicit embedded kernel trace toolkit for embedded systems and describe the transmission method for trace data between the embedded target system and the host system.

Key-Words: Embedded systems, Kernel trace toolkit, Trace data, ETT+

1. Introduction

Embedded systems provide human-centric services in many fields of education, information, industry and service. Thus, many people have said that the 21st century is the age of embedded systems [1].

An embedded system is the device that is manufactured for the specific purposes intended by the original producer, embedded into the microprocessor or microcontroller. The basic kernel used in embedded systems is called the *"embedded kernel"*, and the Embedded Linux Kernel has been generally used in many embedded systems. For the analysis and performance evaluation of the embedded kernel, embedded systems have used kernel trace toolkits for testing and customizing.

The kernel trace toolkit provides both kernel analysis functions and kernel performance evaluation functions for the monitoring system, which is responsible for debugging, testing, performance evaluation of application programs, system control, system management, etc. As such the kernel trace toolkit is needed to manage and test embedded systems because it dynamically collects, analyzes, and shows information on the performance or the process information of embedded systems [2].

However, the existing kernel trace toolkits [3-6] have problems such as compatibility among kernel versions, difficulty of patching kernel, complicated data analysis method, etc. Therefore, in this paper, we propose an embedded monitoring tool that extracts what is directly patched in a simple way, extracts the required specific kernel information and transmits the extracted data to the host environment based on GUI using the efficient.

The remainder of this paper is organized as follows. In the next section, we introduce related work on existing kernel trace toolkits. Then in section 3 we describe the design of our kernel monitoring system and in section 4 we present the implementation of our monitoring system. Finally, we offer our conclusions and discuss future work.

2. Related Work

The main system tracing toolkit for Linux is the Linux Trace Toolkit (LTT) [4], which is used for analyzing subsets of executed processes and recording important system events. In contrast with other tracing tools such as strace, LTT does not use the ptrace() mechanism to intercept applications' behavior. Instead, LTT provides a kernel patch to LTT that instruments key kernel subsystems within the kernel. The data generated by this instrumentation is then collected by the trace subsystem

[1] 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 Advancement) (IITA-2006-C1090-0603-0020)

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 [3, 4].

Currently, research on providing LTTng (LTT Next Generation) has recently advanced in the use of kernel trace toolkits [5]. LTTng has taken over from the previous version, known as LTT. It has the same goals of low system disturbance and architecture independence while being fully reentrant, scalable, precise, extensible and modular, but LTTng samples even more performance evaluation information of the kernel than LTT. It also provides a visualization tool, called LTTV (LTT Toolkit Viewer), which is modular architecture based on plug-ins and supports SMP (Symmetric Multi-Processing) machines. LTTng supports event types with meta-information and huge traces (10-15GB), unlike LTT [5, 6].

However, these kernel trace toolkits have a compatibility problem in that the patches are different from the kernel version respectively and it is difficult for a kernel beginner to install LTTng wholly. Also, there remains the problem that it is difficult to sample some specific information since it provides a great deal of information. To solve these problems, we present a kernel trace toolkit that enables sampling the core kernel analysis information simply and easily.

3. Design of Embedded Kernel Trace Toolkit

In this paper, we provide a kernel trace toolkit through a unique approach, called the *"ETT+ (Embedded kernel Trace Toolkit)"*. In past research [7], the previous ETT used NFS architecture to gather sampled trace data from embedded systems to a remote host and provided a single visualization tool based on Windows MFC. The new ETT+, however, has the architecture to directly transmit the sampled trace data to the host system on the assumption that there is no file system on embedded systems, and provides an integrated development visualization tool based on GTK+, which includes functions of the remote shell , a process analysis tree and kernel memory layout, etc. Because GTK+ is supplied by both Windows and Linux, the ETT+ visualization tool can be executed on both systems.

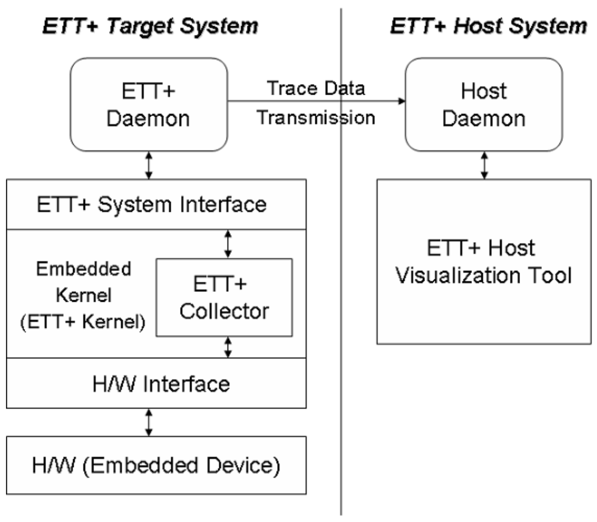


Fig.1: Overall Architecture of ETT+

Fig.1 shows the overall architecture of ETT+. The ETT+ system is divided into two parts: the target system and the host system. The ETT+ target system is responsible for the sampling and transmission of trace data and the ETT+ host system provides the visualization tool based on a GUI environment. ETT+ Collector in the ETT+ target system can get the kernel analysis data and the kernel trace data from an embedded kernel (ETT+ kernel) and transmits these to ETT+ Daemon. ETT+ Daemon enables a request for data transmission from ETT+ Kernel to ETT+ Daemon and then is responsible for transmitting the trace data to Host Daemon in the ETT+ host system. If Host Daemon collects the kernel trace data from ETT+ target system, it saves these kernel trace data on the host file system, and the ETT+ host visualization tool visualizes the kernel information graphically from the data.

The kernel trace data created by ETT+ Collector should be transmitted to the host system. The trace data include both the process information and the system call information. Fig.2 shows the overall data flow diagram in the target and the host. As shown in Fig.2, if ETT+ Daemon requests a start of trace, the ETT+ kernel triggers ETT+ Collector to begin. When the kernel trace buffer is full, the ETT+ kernel will send the signal to ETT+ Daemon. Then ETT+ Daemon receives the signal from the ETT+ kernel and requests to copy the trace data from the kernel internal memory to the user space memory.

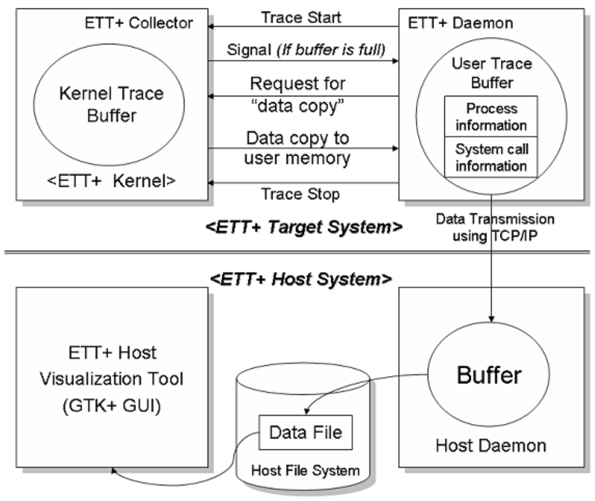


Fig.2: Trace Data Flow Diagram

If the ETT+ kernel receives a request from ETT+ Daemon, it will copy the trace data to the user space memory, and then ETT+ Daemon immediately transmits the copied trace data to Host Daemon using TCP/IP communication. When Host Daemon receives the trace data, it creates a data file using the file system on the host machine and saves the sequence trace data by appending this data file. These operations occur repeatedly until ETT+ Daemon requires a stop of trace. If the ETT+ system stops tracing by ETT+ Daemon, the ETT+ visualization tool will finally display these trace data from the data file saved on the host system.

In the next section, we present implementation issues based on the design of ETT+.

4. Implementation

In this section we introduce some methods regarding the trace data transmission between the host and the target and present some implementation issues. The trace data transmission methods are as follows:

- Using NFS: The previous ETT [7] enables saving a huge trace data file and requires a file system. The target system should be always connected to the host system by using NFS for its fault, but this type is of comparatively easier architecture than others.
- Using ETT+ Daemon: The new ETT+ presented in this paper requires a new kernel primitive (system call) to copy the trace data from ETT+ kernel space memory to user process space memory. It requires two memory copy operations to get the data from kernel to ETT+ Daemon and to send it to Host Daemon through TCP/IP connection. It also enables the control of trace data gathering period in a user level. It requires a slightly complex system, but does not need to have a file system that makes the whole system heavy.
- Directly transmitting the trace data from the ETT+ kernel: This method requires the additional data transmission code (TCP/IP communication) to be inserted in kernel source. There is no memory copy between user space and kernel space, but needs the primitive to control the trace data gathering period in a user level.

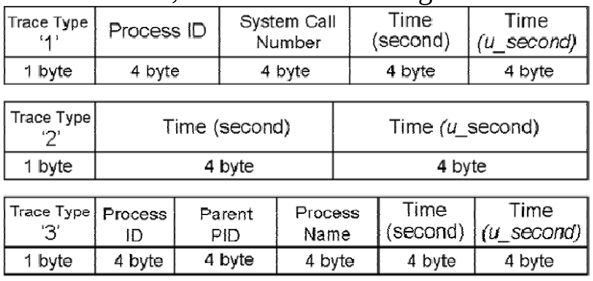
In this paper, we use the second method for an ETT+ system. It is necessary to add new primitives to request the trace data copy from ETT+ Daemon to the ETT+ kernel. In this paper, we made some kernel primitives, such as follow in Fig.3, to implement the ETT+ target system.

Primitive	Description
sys_start_trace()	Start of trace
sys_stop_trace()	Stop of trace
sys_data_copy(unsigned char *addr, unsigned long size, unsigned int pid)	Request for data copy

Fig.3: ETT+ Primitive Description

When ETT+ Daemon calls the system call "sys_start_trace()", ETT+ Collector operates and the kernel trace will begin. Then ETT+ Daemon sends the arguments to ETT+ Collector in kernel space through the system call "sys_data_copy()", and the ETT+ kernel copies the trace data to user space memory using these arguments. Arguments include three values such as target user memory address, data size to be copied and process ID of ETT+ Daemon. Finally, ETT+ Collector stops operating and tracing by using the system call "sys_stop_trace()".

ETT+ Collector is able to sample the trace data based on three trace types and each trace type constructs the trace data formats, such as follow in Fig.4.



Trace Type '1'	Process ID	System Call Number	Time (second)	Time (u_second)
1 byte	4 byte	4 byte	4 byte	4 byte

Trace Type '2'	Time (second)	Time (u_second)
1 byte	4 byte	4 byte

Trace Type '3'	Process ID	Parent PID	Process Name	Time (second)	Time (u_second)
1 byte	4 byte	4 byte	4 byte	4 byte	4 byte

Fig.4: Trace Data Format

The descriptions of the three trace types in Fig.4 are as follows:

- Trace Type '1': As the trace type for the starting of system calls, this type presents the information which is sampled about processing of the system call handler. It includes information such as process ID, system call number, system call starting time (*sec*) and system call starting time (*u_sec*).
- Trace Type '2': As the trace type which presents the ending of specific system calls, this type presents information when the system call handler finishes. It includes information such as system call endeing time (*sec*) and system call ending time (*u_sec*).
- Trace Type '3': This type saves the created process information. It includes information such as process ID, parent process ID, process name, process execute time (*sec*) and process execute time (*u_sec*).

Because the ETT+ host visualization tool has not yet been implemented completely, we introduce sampled raw data from the ETT+ target system.

Fig.5 below presents a sample of real trace data sampled from the ETT+ Collector based on these trace data formats. In the first line, as above trace data format, '3' stands for the trace type, '1' stands for a process ID, '0' means a parent process ID, 'init' means a process name, '2834470302' represents a process execute time (*sec*), and '620147' represents a process execute time (*u_sec*).

```
3   1  0   init   2834470302   620147
3   2  1   keventd   2834470302   620147
................
3      6     1      kupdated       2834470302
620147
3   43 1   syslogd   2834470302   620147
................
3   85 1    bash   2834470302   620147
3      90   85      ETT_start      2834470302
620147
1   90 174   2834470302   620147
2   2834470302   620252
................
3   92 1 dataCopy   2834470302   705905
1   92 122   2834470302   706125
2   2834470302   706138
................
```

Fig.5: Sample for Trace Data

Finally, these trace data are transmitted to Host Daemon and saved in the data file, and the ETT+ visualization tool uses this file to analyze trace outputs.

5. Conclusions

In this paper, we present the approach to develop a kernel trace data sampling and trace data transmission using our kernel trace toolkit, *"ETT+ (Embedded kernel Trace Toolkit)"*. ETT+ provides a simple patch architecture, easy user environment and a daemon environment different from the existing LTT or LTTng, which provides a modular environment and complicated installation architecture. Also, we designed the architecture that ETT+ Daemon would require to copy the trace data when the kernel trace buffer was full.

We are now considering the architecture for periodical trace data-gathering so as to make the ETT+ system operate in real time,. The host visualization tool has also not yet been implemented completely, and for the purpose of the embedded systems monitoring IDE, there are some tasks for us to implement concerning the remote shell functions and the performance evaluation information (such as interrupts, memory layout, detailed process information, etc) as our future work in this area.

References :

[1] J.H. Na, S.J. Kang, Y.I. Yoon, Y.Y. Park, S.B. Eun, H.N. Kim, *"Embedded System Programming"*, SciTech Media, 2004.

[2] Ji-Hye Bae, Yoon-Young Park, Jeong-Bae Lee, Sung-Hee Choi, Chae-Deok Lim, "A Study on the Design of the Monitoring Architecture for Embedded Kernels Based on LTT," *Proc. of 4th Asia Pacific International Symposium on Information Technology*, Gold Coast, Australia, pp.68~71, Jan., 2005.

[3] Karim Yaghmour, *"Building Embedded Linux Systems"*, O'Reilly.

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

[5] LTTng & LTTV Homepage, http://ltt.polymtl.ca

[6] Mathieu Desnoyers, Michel R.Dagenais, "The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux," *Linux Symposium*, Ottawa, Canada, Jul., 2006.

[7] Ji-Hye Bae, Hee-Kuk Kang, John Y. Kim, Yoon-Young Park, "Monitoring Systems for Embedded Equipment in Ubiquitous Environments," *International Journal of Information Processing Systems (IJIPS)*, KIPS, Vol.2, No.1, pp.58~65, Mar., 2006.

[8] Daniel P.Bovet, Marco Cesati, *"Understanding the Linux Kernel"*, 2nd Edition, O'Reilly.