19

Debugging Environment for a Large Scale Telecommunication System

E.H. Paik, Y.S. Chung, Y.J Byun, B.S. Lee
Electronics and Telecommunications Research Institute
Yusong P.O. Box 106, Taejon, 305-600, Korea. Telephone:
82-42-860-5181. email: ehpaik@sde.etri.re.kr

Abstract

Testing and debugging of real-time, embedded, concurrent software system is costly and time consuming. In order to reduce the development costs of the large-scale telecommunication system, TDX-10, we developed a CHILL Debugging Environment(CDE). CDE is a system that helps us to develop software for large-scale telecommunication systems with high productivity and reliability. It gives us effective debugging facilities such as logic tests, signal tracing for integration test, and assists us in cross debugging the software that runs on a target system. With CDE, we can not only test and debug the software on a host computer without downloading them onto a target system but also cross debug the software that runs on a target system. To simulate the software on the host, CDE provides facilities for concurrent processing that are available in the target system. This paper describes the design and implementation of CHILL debugging environment for a large-scale telecommunication system.

Keywords

Host debugging, cross debugging, environment, telecommunication, software, CHILL

1 INTRODUCTION

Today's telecommunication system has distinguished characteristics that are real-time, embedded, interactive, and distributed. Inherently, this system contains some largest and most complicated software that have been constructed. Therefore, in the development of telecommunication software, the debugging and testing activity are believed to be one of the most important processes in producing expensive and excellent-quality software [C. McDowell 1988]. A current approach to develop most of the embedded system such as telecommunication system, depends on cross development method [L.R. Collingbourne 1987]. The cross development method is a way to develop applications using cross software tools and test environments. The key for developing such a telecommunication system successfully is to provide debugging tools with programmers.

The advantages of using this environment are:

F. A. Aagesen et al. (eds.), *Information Network and Data Communication*

- Able to detect and correct program errors in telecommunication software in source-level on the host.
- Able to cross debug telecommunication software being executed on the target.
- Able to trace and monitor status of processes and behaviors of signals among processes on the host and the target.
- Able to simulate telecommunication software on the host to reduce lots of run-time errors before the system test on the target.

In this paper we described our approach and techniques to build the debugging environment in terms of source-level host debugging/cross debugging of telecommunication software on host-target environment. This environment has been developed to provide programmers who develop TDX-10 telecommunication system for an effective debugging and testing facilities. However, it can be used in wide range of embedded systems development and cross development environment as well.

2 BACKGROUND OF THE ENVIRONMENT

2.1 Environment Overview

This environment consists of debugging tools such as CHILL [CCITT 1980] host debugger [E.H.Paik, et. 1994], CHILL cross debugger [E.H.Paik, Y.S.Chung 1994], signal tracer [E.H.Lee 1992], and CRS(CHILL Run-time System) [C.H. Cho 1988]. CHILL host debugger is a tool to assist tracing and debugging programs that are being executed on the host machines. CHILL cross debugger is a tool that runs on the host to debug programs that are being executed on the target in source-level, communicating with a target debugger on the target. It needs an interactive communication between CHILL host debugger and target CHILL debugger. Target CHILL debugger is an assembly level debugger for MC68030 processors. Signal tracer is a tool for tracing and monitoring signals among processes. Using CRS that is supported by time manager, the switching software on the host can be simulated. The architecture of the debugging environment is shown in Figure 1.

This environment has been developed to provide programmers who work on TDX-10 (Time Division Exchange 10) switching system for effective debugging and testing facilities. However, it can be used in a wide range of embedded systems development and cross development environment as well.

2.2 Host-Target Configuration

In general, the most common configuration for cross development consists of a mid-range of host machine and single target system. In the development of TDX-10 software system, we have three super-mini VAX computers and a lot of SUN SPARC as host machines. All of host machines are connected via ETHERNET. We have three models of target system in which several microprocessors such as M68030 are coupled with a ring topology. Target processors are connected to host machine via a serial lines or local area network. So,

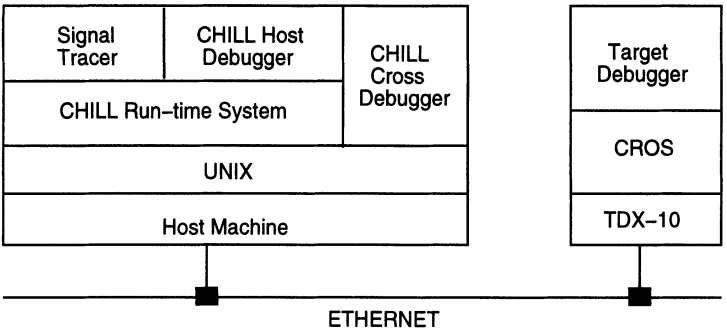

Figure 1 The Architecture of The Debugging Environment

developers can work on network terminals and access all the resources of the target on host machines. They can also access the target through the host/target configuration.

2.3 Compilation And Cross Compilation

CHILL programs are compiled and linked with the CRS for processing concurrent features of CHILL on UNIX environment. To produce debugging information, programmers should compile application programs with a debugging option. All of debugging information are generated in a executable file. CHILL host compilers [Jin P. Hong 1990] support these compilation processes in which they produce VAX assembly/machine codes and SUN assembly/machine codes in the form of UNIX/BSD. Signal tracer and host debugging tool run on a host machine using this machine code. Programs being executed on target system are compiled and linked with some CHILL run-time libraries for the target system. Cross complier supports TDX-10 run-time library. All of debugging information is generated in the executable file in the form of UNIX system V, so called common object file format(COFF).

2.4 Phase of Debugging

The debugging activities of the telecommunication software are divided into three phases. The first phase is a host debugging phase. In this phase we are able to do logic test rapidly. The second phase is to trace and monitor the CHILL signals communicated between concurrent CHILL processes for the integration test. In these two phases, we trace and monitor CHILL programs that are supported by CRS without any target systems. The third phase is to cross debug the software that runs on the target system controlled by CROS(Concurrent Real-time Operating System). In the host debugging phase, we compile the source program using a CHILL compilation system [Jin P. Hong 1990]. For the codes to be executable on the host, the compiler generates codes to call functions supported by the CRS. An executable code generated by the compiler is called a module, and the module is regarded as an imaginary process. When a module is executed, it can generate many processes. In this phase, we can debug lots of telecommunication software bugs except bugs that are occurred because of real-time features and non-determinancy.

When the telecommunication software runs on the telecommunication system, there are frequent Inter Process Communications(IPC) between processes which run on distributed processors. So, it is important to trace and monitor the IPC between processes. In the tracing CHILL signal phase, we could trace and monitor every event concerning all the processes and signals occurred during the execution of the program.

In the cross debugging phase, we can debug telecommunication software in source-level being executed on the target. A Cross debugger communicates with a target debugger resident on the target and maintains all the resources such as source code and symbolic information on the host, and controls debugging tasks with the assistance of the target. This is an efficient approach to satisfy the requirements such target programs must be executed standalone without any constraints to be compounded by the supporting tools [E.H.Paik, et. 1994, C.H. Cho 1988].

When invoking a debugging session, the debugger separates actual executable codes from the symbolic information, strips and transforms them into S-records, then, downloads the actual codes only on the target.

3 DEBUGGING ENVIRONMENT

Typically, programmers examine and debug the logic of programs by host debugger, then download and trace the execution status of programs on the real environment by the cross debugger. Both of the above debuggers are running on the host computer and support CHILL source-level debugging features.

3.1 CHILL Host Debugger

CHILL host debugger [E.H.Paik, et. 1994] is a tool to assist tracing and debugging concurrent CHILL programs that are being executed on the host. Main functions of a host debugger are to execute and trace sequential programs, and to show the status of current CHILL processes such as queues related to signal, region, buffer, event. To implement CHILL host debugger easily we used C debugger in UNIX. However, C language does not have the language concepts for various data type, the modulion (module and region) and block structure, and visibility of CHILL [CCITT 1980]. So we extended C debugger to make a CHILL host debugger which supporting these language concepts. CHILL host debugger interacts with the CRS to get CHILL host concurrent procession information. This debugger is composed of four parts such as symbol table generator, user interface, command/language evaluator, and tracer. The first part is a symbol table generator that reads symbolic information included in the executable program and creates internal symbol table for debugging. It maintains five auxiliary tables such as file, mode, line, function, and block tables. Each table is connected by linked lists and contains information such as a symbol name, language type used, location or value, class, mode, a static chain for procedure arguments, dependency about the modules/procedures/processes, and next link. After the construction of the symbol table, the debugger forks a UNIX process for debugging programs, and waits debugging commands. The second part is a user interface that parses debugging commands by using yacc and constructs abstract syntax trees(AST) for programmers' commands. This interface is similar to that of C debugger, dbx. The third part is a command/language evaluator that evaluates the AST by using the symbol

table and internal stack. It resolves names given as arguments, determines their physical attributes, and shows the resulting static information, in terms of CHILL by invoking the tracer optionally. This evaluator has a variable reference facility. In order to refer the value of variable, it calculates the locations of the variables by using symbol table. It has also a message sending facility that can send signal messages to a certain process. The forth part is a tracer. It accesses images of processes and controls the executions. It extracts or modifies the value of the symbol table and interacts with the program being debugged by using ptrace(2) in UNIX. The most important purpose of CHILL host debugger is to provide a target transparent debugging methods to programmers. In other word, CHILL host debugger is responsible for equivalent results of debugging a between host debugger and a target debugger. This debugger keeps addresses of several queues related to signal, region, buffer, event, and ready by protocol defined in the run-time system, and reads the contents of process control blocks of the run-time system during a debugging session. Each queue contains process definition names, instance numbers created by the run-time system, and the reasons why the processes are delayed. According to user requests, the debugger shows the status information in the form of process definition names(ID), instances, state and reasons optionally. For user requests such as tracing some particular process/process instances or setting breakpoints at the process/process instances, the debugger stores the commands in the command table with instance numbers together, then performs the selected process only when the control reaches predefined a process name(ID) or process instance number. This capability will lead users to set the conditional breakpoints on process instances. The debugger provides another important feature: an interactive input message generation and sending. It was designed to trace the execution of process instances by stimulating data through a input messages. During a debugging session, users will key-in the values of a signal message according to the signal mode in CHILL. For this request, the debugger generates message and sends it into predefined message queue ID in UNIX kernel. Then the run-time system conveys the message to the specified process and executes the process. We resolve this message passing scheme by using msgop(2) and msgget(2) in UNIX. Using this feature, programmers can simulate a processing situation on the host.

3.2 CHILL Cross debugger

CHILL Cross debugger is composed of three components: a host debugging part, a target debugger, and a communication link as shown in Figure 2.

Host Debugging Part

The one has similar functionality to the host debugger except that the tracer part; it consists of a symbol table generator, user interface, command/language evaluator, and cross-tracer. This one creates a symbol table for a target program being debugged, detaches the symbolic information from executable codes, strips and converts the codes to the executable object suitable for running on the target, and downloads the object into the target via the communication interface. Then it controls the debugging processes and manipulates them symbolically, interacting with source codes and the symbolic information. One important feature here is a separate downloading of the executable codes from the symbol table. The symbolic information for a program to be debugged remains on the host. The alternative of downloading over large size of the symbol table is too heavy

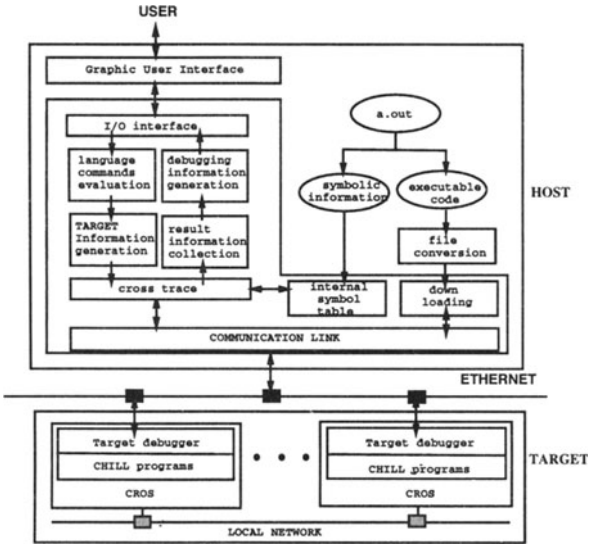

Figure 2 The Architecture of The CCD(CHILL Cross Debugger)

for the target system to completeness. Even though the host debugging part makes use of the structure of the CHILL host debugger, several problems were to be solved: easy generation and verification of the symbolic information, manipulation of a specific run-time stack frame such as MC68030, adjustment of byte-ordering between different machines, disassembly the object codes into MC68030 assembly language and so on. The first issue is a method to generate a symbolic information for the cross debugger. CHILL compilers in CSDE generate a symbolic information and the corresponding executable codes for hosts, both VAX and SUN, and the target, especially TDX-10 and MC68030. Symbolic information is originally generated in BSD/UNIX-style for the CHILL host debugger and then, it is necessary to fit it into target assembly language, SYSTEM V UNIX-style, for debugging target programs. On the other hand, since a symbol table generator of the CHILL cross debugger maintains the same structures and properties that of the CHILL host debugger. Thus, it reads the symbolic information included in the target codes and translates them again into the cross debugger's internal symbol table. Verifying whether the generated symbolic information is correct or not is more simple - it has the same information as the CHILL host debugger has.

The user interface can be achieved through the CHILL host debugger as it is. However, a user's command is translated to a sequence of low-level operations requesting to the target debugger, then they are transmitted to and executed on the target. Host debugging part receives the execution results in forms of a low-level sequence from the target and it displays those data in the CHILL-level. The result data analysis is to identify the class of received data whether the data aims for users or for internal processing of the host debugging part.

The main concept of cross debugging is implemented in "cross-tracer" that is the entry

point to the target system. It controls all the cross debugging processes with a small set of the low-level operations, including memory and register read/write and execution and breakpoints and single-step with a corresponding address and data. A cross tracer was made by xptrace that is similar to ptrace in UNIX. All operations and the results to/from the target debugger are handled through the cross-tracer via the communication link.

Target Debugger

The target debugger may be thought of as the counterpart of the cross-tracer. It is enough to provide the primitive facilities such as reading or writing contents of memory and registers, setting breakpoints in instruction level, executing a piece of codes, extracting concurrent process information, and so on. These low-level operations can be performed either by a ROM-based debugger on the target or by means of a hardware emulator including a microprocessor development system. The host debugging part transmits requests to the target debugger to perform the low-level operations. Then, control will be passed to the target debugger when breakpoints are encountered or some sorts of traps. Otherwise, the target debugger is quiescent not to alter the behavior of the running program. Much current cross developing is performed using the primitive ROM-based debugger. When debugging using a hardware emulator, the debugger can control the target indirectly through an emulator to be thought as a load-sharing processor. The advantages using this facility are that a communication overhead can be reduced by a high-speed communication interface such as a local area network if possible, and a load-sharing for cross debugging can be expected to some extent.

Since the ROM-based debugger or hardware emulator cannot usually tell about status of CHILL processes and information of synchronization primitives, we need higher-level debugger that has the capabilities showing information of concurrent processes. The "TDXDBUG" is such a debugger that has the low-level operation as well as provides concurrent processing information by requesting to the target operating system. Basically, it treats all running processes simultaneously.

Communication Link

The communication Link is a pathway between the hosts and the target systems in order to download programs, to pass the low-level debugging operation to the target, or to return the results to the host. The interface depends on a host/target network configuration. It was designed to allow a multi processor target system to be controlled from the host machines using a single interface.

It consists of three layers. The top layer defines a message that enables the host to read and write memory or register, start the program, set and clear breakpoints, reset the target. The middle layer defines some packages on the data interface. The lowest layer specifies the types of communication interfaces that may be a simple serial line such as RS-232-C, a TCP/IP-like protocol. Most of telecommunication systems don't have any high speed network boards that communicate with general purpose computers. So, we made a TCP/IP-like protocol for communication between host and target system and made an ETHERNET board that runs on target system to talk with host computers. The protocol gives programmers transmission line open/close, making ETHERNET packets, packets send/receive and so on. Currently, TCP/IP-like protocol has been implemented and the combination of a remote procedure call(RPC) over LAN and IPC protocol is being considered.

3.3 CHILL Signal Tracer

There are various methods for concurrent programming in CHILL. The concurrent facility supports EVENT for synchronization, BUFFER and SIGNAL for communication, and REGION for mutual exclusion among the CHILL processes. Buffers use common memory for the communication between processes, however, signals use IPC rather than common memory. The TDX-10 software is composed of concurrent programs executed on the multiple processors as a form of processes communicating each other by signals. So, we need to correct that a signal arrives at destination process actually via a described path. The correctness of concurrent programs depends on the contents and sequence of messages transmitted between processes. For these motives, we made use of a technique that displays post-mortem views of signal transmissions after monitoring the CHILL processes in our CHILL signal tracer. Some special run-time routines are used to monitor all the events and messages concerning signals and processes that occurred during the execution of the program. Because these libraries exist in CRS or CROS, programs to monitor need only be re-linked, and not recompile. After executing the program, CHILL signal tracer is invoked to analyze and display the result of signal flow graphically.

CHILL signal monitoring library

The signal monitoring library records every event history concerning all the processes and signals occurred during the execution of the program. There are SEND and RECEIVE-CASE action statements concerning signal transmissions in CHILL. When these action statements are executed, the monitoring sequence is following.

1) If execution control meets SEND and RECEIVECASE statements during the execution of the program, the control is passed to _sendsig and _rcvsigcase primitives of the CRS or CROS

2) In the case of that _sendsig library is called, before calling the kernel the library records historically the status of processes sending/receiving signal and message information. In the case of that _rcvsigcase library is called, after the execution control is passed from the kernel to the library again, the library records historically the status of processes that sent/received signal actually and received message information.

Analysis and displaying CHILL signal event

The CHILL signal tracer is invoked graphically to display CHILL signal transmission history and process event history on SUN workstation window environments. CHILL signal tracer analyzes the event history relating to processes and signals and builds an internal database to use later. According to a user interface, the CHILL signal tracer transforms the data to graphical data and displays them on a graphic window. Then users can trace signals and processes with mouse manipulation. The user interface is composed of two views largely. One is displaying the signal transmission situation in terms of processes. The other is displaying the signal transmission path among its processes.

3.4 CHILL Run-time System

A major difference between sequential and concurrent programs exists in the definition of program state in each case. The state of a sequential program can be characterized simply by a program counter value and memory image of its data. The state of a concurrent

program requires such information to be considered for each of its processes. The ability to represent program state is a fundamental capability of any debugging tool. However, if concurrent programs are executed on multiple computers, making it a distributed program, communication delays make it difficult to capture its entire state at any given time. Conventional debugging tools are of little use in this situation since they are typically oriented toward monitoring the operation of what would only be a single process of a distributed program. Various concurrent processing features in CHILL are:

- Process definition
- Process creation and kill
- Region based on monitor concept
- Event for synchronizing processes
- Message passing by using buffer
- Signal send/receive

The CRS includes all the above concurrent processing features defined in CHILL. It realizes the interface to programs produced by the compiler. It provides run-time environment on the host machines, which are currently Ultrix 4.2, SunOS2.x and Solaris 2.3, before application programs are running under target operating system. CHILL host Compiler expands each concurrent statement to one or more sequential statements in terms of CHILL syntax and semantics. The interface between application program and CRS may be written in terms of specification of procedures. CRS creates its own processes without relying on the host operating system. It allocates run-time stacks for starting processes and performs process management. It regards a process of host operating system as a virtual processor to be run CHILL processes, and accordingly it manages those processes in its context. CRS consists of kernel and CHILL primitive handlers. The kernel provides the functions such as context telecommunication, scheduling, allocation of run-time stacks for processes, and synchronization routines invoked by the primitive handlers, which are considered of the most machine and operating system dependent part. The interface between application programs and CRS is accomplished via the primitive handlers. Since they are realized by procedures invoked by processes, they perform their own functions on the process contexts requesting services. However, when they call the kernel routines, they become to run on the kernel context. When the program starts, CRS kernel initiates the program status, creates outer-most process and allocates the real-time stack for processes.

4 CONCLUSION

We have introduced a debugging environment for a large-scale telecommunication system. This environment consists of several tools such as a host debugger, a cross debugger, a CRS, a signal tracer. These tools are composed of the well-defined interface and they are easily retargetable as well as rehostable. They run on the VAX and SPARC workstations and support general MC68030 processor applications as the target system. We made this environment to develop TDX-10 telecommunication system software. TDX-10 is a large-scale digital telecommunication system being developed by ETRI(Electronics and Telecommunications Research Institute) since 1987 funded by KT(KOREA Telecommunication). By using this environment, programmers could reduce lots of test and trial

run-time errors before the system test on the target system or field test. To be more useful in real-time embedded systems development, it is now being extended debugging environment to the extent of fully supporting multi-programs within a single processor and a multi-processor system. Furthermore, a window based debugging environment is being developed to give programmers debugging window on workstation with powerful window system.

REFERENCES

CCITT (1980) CCITT High Level Language(CHILL) Recommendation Z.200, IXth Plenary Assembly.

C.H. Cho, S.H. Kim, and J.P. Hong (1988), The Implementation of Portable CHILL Run-Time System, Proceedings of '88 KISS Spring Conference, 441-4.

C. McDowell and D. Helmbold (1989), Debugging Concurrent Programs, ACM Computing Surveys, 593-622.

E.H.Lee, et al (1992), The development of CHILL Signal Tracer, Proceedings of KISS, 433-6.

E.H.Paik, et al (1994), Debugging and Simulation Environment for a switching system, proceedings of Pacific Telecommunications Conference, Hawaii, 642-6.

E.H.Paik, Y.S.Chung, Wan Choi, C.W.Yoo, H.B.Song (1994), CHDM: A Distributed CHILL Program Debugger, Proceedings of KISS,1994. 105-8.

Jin P. Hong, D.G. Lee, K.S. Park, and H.G. Bahk (1990), A CHILL Programming Environment, Proceedings of 5th CHILL Conference.

L.R. Collingbourne (1987), A Practical Approach to developing Real-Time Ada Programs For Embedded Systems, Proceedings of the International Workshop on Real-Time Ada Issues, 15-17.

BIOGRAPHY

Eui-Hyun Paik received the M.S. degree in Computer science from Soongsil University in 1987. He joined ETRI in 1987. he participated in several projects for developing CHILL Compiling system and CHILL debugging environment. Now, he is working as a senior researcher of software environments section.

Young-Sik Chung received the M.S. degree in Computer Science from Hannam University in 1992. He joined ETRI in 1983. he participated in several projects for developing CHILL Compiling system and CHILL debugging environment. Now, he is working as a senior researcher of software environments section.

Young-Jun Byun received the M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology in 1993. He joined ETRI as a member of research staff in 1993 and is currently working on the design and development of the CHILL debugger.

Byung-Sun Lee received the M.S. degree in Computer Science from Dongguk University in 1982. He joined ETRI in 1982, and is currently a head of software environments section. His interests include requirements specification, formal methods, and software testing, especially for telecommunication networks and switching systems.