

High-Level CHILL Debugging System in Cross-Development Environments

YoungJoon Byun, Young-Sik Chung, Byung Sun Lee

Software Environments Section
Electronics and Telecommunications Research Institute
Yusong P. O. Box 106, Taejon, 305-600, KOREA
{byun, yschung, bslee}@etri.re.kr

Abstract

CHILL is a concurrent programming language, especially for implementing telecommunications software. ETRI has used the language for the development of switching software. Generally, switching software is characterized by real-time execution, parallel and distributed processing, large scale source code, and high complexity. The software is also developed on cross development environments. In this paper, we present a CHILL cross debugging system, in which the system can test and debug the programs on remote system from local system. The system provides the traditional debugging features such as executing and tracing a program, listing source code, setting breakpoints, examining and setting program locations, and single-stepping. It also provides parallel and distributed real-time debugging, a powerful command language, signal sending capability, and graphical user interface for switching software.

1. Introduction

CHILL, a general purpose high-level programming language recommended by ITU-T, is mainly used for the development of telecommunications software, such as TDX, ISDN, and ATM [1][2]. This language provides concurrent execution, modular and structured programming, separation compilation, strong type checking, and data abstraction features. Telecommunications software is generally characterized by real-time and concurrent execution, very large source code, high complexity, and high performance requirements [3]. CHILL is therefore considered as a suitable language for switching systems with these characteristics.

ETRI is developing a large-scale ATM switching system [4]. The development of the ATM switching soft-

ware is carried out on cross development environments. In this environment, the software compiled at a host system executes and operates at a target system. However, the target system has limited resources and functional constraints, so it is difficult to use the testing and debugging tools designed for general-purpose computers at the target system directly. To cope with these problems, a cross debugging system has been developed, where a debugger in the host system can control and monitor the execution codes in the target system through a high speed communication line with the support of the target operating system [5].

This paper discusses the CHILL cross debugging system, which has been developed jointly by ETRI and Kvatro Telecom AS, a Norwegian company. The system is intended to provide an integrated testing environment that makes it possible to perform debugging and testing of CHILL switching software in a convenient and efficient way. For this purpose, the system provides parallel and distributed real-time debugging, sending signals, debugger command language, and graphical user interface (GUI). The paper is organized as follows. At first, we give an architecture of our debugging system, in which we show a basic concept of cross debugging and the target system configuration. Then we introduce the features of the debugging system in section 3. Section 4 deals with the debugging target software, in which we show how to debug the target software. We also present important components of the system: a CHILL debugger, debugging information, and a communication protocol. We conclude the paper with a summary of the system and the further work.

2. Cross debugging

The cross debugging, sometimes called remote de-

bugging or tele-testing, is based on the client-server model and used for the debugging of embedded systems such as a switching system [5][6]. This debugging method makes it possible to debug programs in a target system remotely from a host system. A debugger on the host system needs a small agent, a debugger server, on the target system that provides primitive debugging support. The debugger, instead of creating debuggees and expecting the operating system to give notification of debugging events, sends and receives messages via a serial line or a network to the target system.

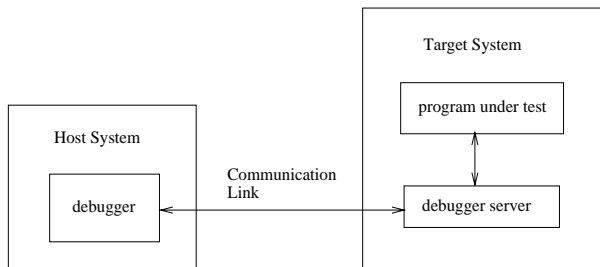


Figure 1: Basic concept of cross debugging

Figure 1 shows the basic concept of current debugging system. The debugger can control and monitor the program under test (PUT) with the support of the debugger server, residing in the target operating system. The debugger and the debugger server exchange messages with specially designed communication protocol.

2.1. System architecture

Figure 2 shows the system architecture of the CHILL cross debugging system. The system is largely composed of three parts: a host system, a target system, and a communication link. The host system is a general purpose computer system. The users can access the debugger by means of GUI to debug the CHILL processes created from a PUT. When a new debugging session starts, the debugger connects to the PUT and exchanges debugging messages with the debugger server. The debugger requires source code and debugging information to control the PUT at source-level.

The scalable real-time operating system (SROS), a distributed real-time operating system, is used as an execution platform for the target ATM system [7]. It is based on a micro-kernel architecture and mainly composed of a micro-kernel and a group of system servers. The micro-kernel provides the basic system control functions and a real-time parallel processing function. The system servers provide a wide variety of services. They comprise I/O server, inter-process

communication server, time server, file system server, Ethernet server, and debugger server. Among these, the Ethernet server and the debugging server play critical roles for cross debugging. The former takes charge of communications with the debugger, while the latter provides the basic debugging functions. The debugger server receives requests from the debugger and executes the primitive debugging functions such as reading or writing contents of memory or registers, setting traps in instruction level, executing a piece of code, and so on. The communication link connects the debugger and the target system, downloads the target object code, and passes the requests and results for the debugging operations. The link is constructed on Ethernet and UDP/IP.

2.2. Target system

The target, ATM switching system in our case, has a distributed real-time architecture for supporting large subscribers [4][8]. It consists of several subsystems as shown in figure 3. It has one ATM central switching subsystem for connecting of each subscriber and many ATM local switching subsystems for the interface of subscribers. Each subsystem consists of one on-board processor and one network module. The processors are duplicated for fault-tolerance. The central subsystem takes a role of administration and maintenance, such as charging and statistics. It has disk to keep the information for the management of switching systems. The processor in a central subsystem is called an operation and maintenance processor (OMP). The main function of a local subsystem is the call processing of broadband subscribers and trunks, so the processors in the local subsystems are called a call and connection control processor (CCCP). Each subsystem uses SROS as an operating system and the developers can test and debug their applications running on subsystems from a host system.

3. Debugging features

The debugging system allows programmers to run and debug programs symbolically in terms of CHILL [9]. It provides the features of traditional sequential debuggers such as executing and tracing a program, listing source code, setting breakpoints, examining and setting program locations, and single-stepping. It also provides a powerful macro mechanism and a general purpose debugging language, called a pilot command language (PCL), consequently making it possible to prepare a variety of test scripts. The scripts are very

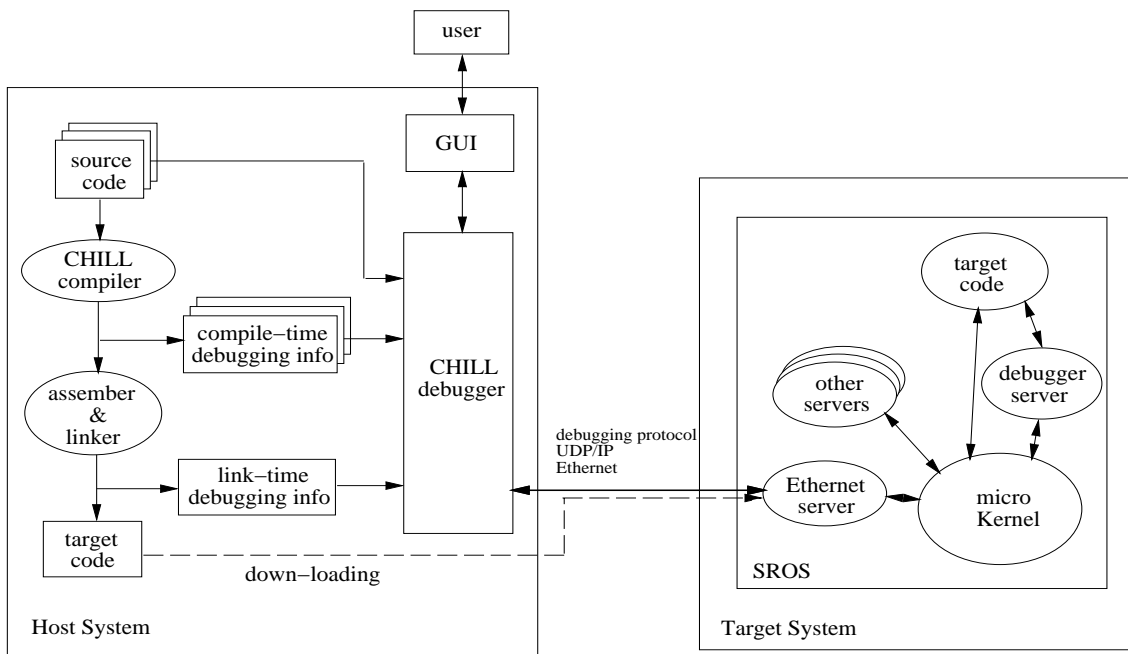


Figure 2: Debugging system architecture

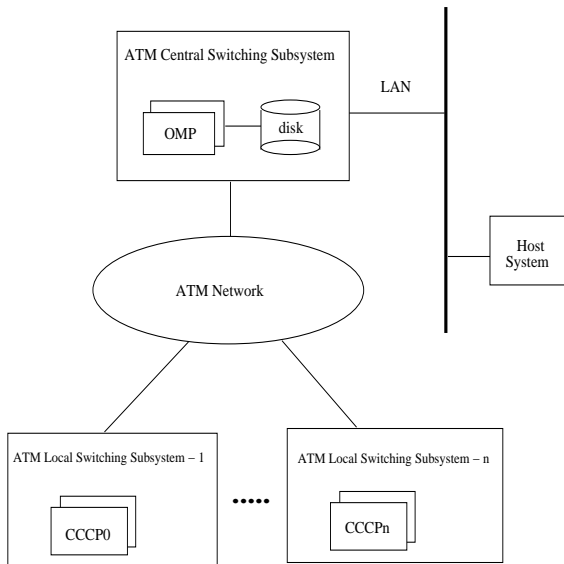


Figure 3: Target system configuration

useful for the regression testing of large-scale switching software. In addition to these sequential features, the system can control and monitor the execution states of individual processes as well as the states of a whole testing program.

For the proper execution of a software unit in switching system, it should communicate and exchange information with related software units. The switching sys-

tem uses CHILL signals for synchronization and information passing among units. The unit generally waits for signals from other units. After the receipt of the wanted signal, it can continue the execution. If a programmer intends to conduct a unit testing in the state where development of other related units has not yet been completed, it is necessary for the unit to receive the required signals from the outside. The system provides the signal sending feature for the above case. It is, therefore, possible to debug and test a unit, regardless of other related units. In addition to the feature, the system will be strengthened by tracing signals, with which users can analyze the dynamic behaviors of processes and signals after the execution.

Simple usage is an important goal of a debugging system. In this debugging system, the users can access the system with GUI.

The following is a brief description of the major features of the debugging system discussed in this section:

- Execution
 - connecting and disconnecting to a remote program under test
 - starting and reloading a program under test
 - accessing source files
 - traps with conditional statements
 - various stepping on source code
 - examining stack frames
 - forced procedure calls

- Parallel Programs Debugging
 - showing the states of processes and programs
 - execution and trace of a particular process
 - multiple contexts
 - displaying current state of target system
- Sending and Tracing Signals
 - sending a signal to the destination process or processor
 - post-mortem analysis of signal behaviors
 - displaying signals arrived at the specified process
- Programmability
 - PCL script language and powerful macros
 - debugger locations and user-defined locations
 - logging the testing result
- Evaluation of Expressions
 - accessing and modification the locations
 - showing the mode of locations
 - CHILL-like PCL expressions and evaluation
- Graphical User Interface

4. Debugging target software

When a new debugging session starts, the debugging system needs several steps. First, the CHILL cross compiler on a host system compiles and links a program being executed on the target system and generates a target code, an execution file running on the target system. During compilation, the compiler generates debugging information for the source program. The debugger requires the information to get information about a source code and its symbol. After compilation, the target code is loaded into the target system, and then the debugger tries a connection to the target code. Figure 4 shows the successful connection step-up between host and target system. The upper window is the host system, which show that the target code, named main process, is in suspended state on target OS. The user can start the debugging of the target code by clicking the start button. The below window is for the target system, which show that a user downloads the target code from the host system and that the connection information.

The following summarizes the cross debugging steps described above:

1. cross compilation for a target code
2. downloading the target code
3. connection setup between the debugger and the target code

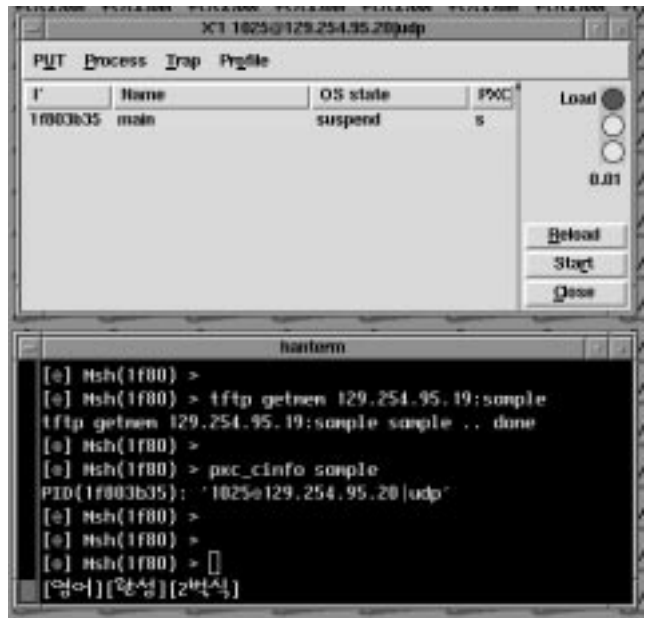


Figure 4: Connection setting

4. debugging session
5. disconnection

4.1. CHILL debugger

The CHILL debugger is a kind of remote debugger. It can monitor the remote programs under test with the support from a debugger server. The debugger has many similar features as ordinary breakpoint based source-level debuggers. But unlike ordinary debuggers, the debugger has unusual features such as asynchronous operation, programmability, and multiple sessions [10].

The debugger operates in parallel with the programs under test. That is, the debugger and debugees execute asynchronously. So, the debugger operations can be performed without affecting program execution at all. The debugger is also designed to guarantee that the suspension of a particular process does not affect other irrelevant processes. Through such design, the suspension of a process causes minimum impact on the execution of entire programs. The program, therefore, can execute at full speed, even if some processes are being manipulated by the debugger. Besides, the processes in the debugger server execute with low execution priority, compared with processes in programs under test. This causes minimum impact on the execution of real-time programs.

Test script language is very important for large-scale software testing and debugging. The users can write

several kinds of testing scripts with PCL, which provide many capabilities to help complete programmability. It makes the users evaluate the CHILL expressions in scripts and provide powerful macros. It also includes CHILL constructs such as loop and conditional statements. Furthermore, the user can define trap body with a sequence of commands, which define what will the trap do when a process hits the trap. Below shows a simple script file.

```

- '-' means comment
- start a log file for post-mortem analysis
LOG $sthis FILE F"test.result";
- load the testing block
LOAD $xthis;
- set a trap for an inspection of a location value
TRAP (where)
  IF value = 100 THEN
    $test_result := TRUE;
  ELSE
    $test_result := FALSE;
  value := 100;
  FI;
  RESUME;
END;
- start the testing block
RESUME;
- end of log file
CLOSE LOG

```

In a traditional sequential debugger, there is only one debugging session. The debugger provides several sessions at the same time. This is essential for parallel debugging. The figure 5 shows the debugging of two processes.

4.2. Debugging information

To describe the properties of a program under test to the debugger, the CHILL compiler generates debugging information about the program. The compiler uses the debug information language (DIL) to represent it [11]. DIL represents symbolic information about the program in the form of LISP style lists, which is human readable and makes the implementor of DIL generator to develop it easily. A DIL file includes two kinds of information: one is for the structure of the program and another is for the symbols in the program. Program structure information includes the source file name, line numbers, block information, and so on. Symbol information includes mode definition, mode and address of the symbols, and so on.

The debugging information for a program is composed of DIL compile information (ci) and DIL link information (li). DIL ci is the set of debugging infor-

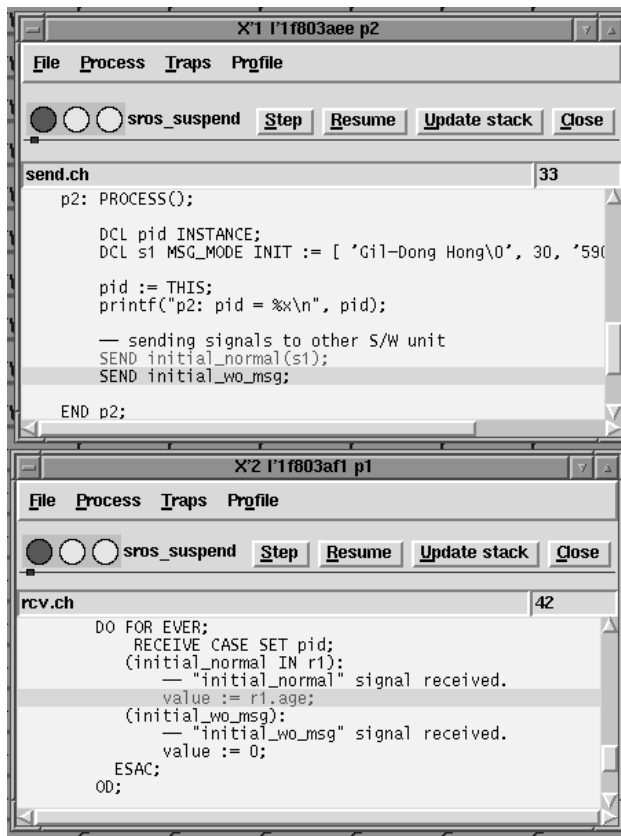


Figure 5: Snapshot of two processes debugging

mation produced for one corresponding CHILL compilation unit. A DIL li holds the key information about the compilation units constituting a program. The DIL li file is a catalog that can be read relatively fast at connection time. With the DIL li file, DIL ci files can be read incrementally by the debugger during a debugging session. The debugger uses the DIL files and source texts to control the program at source level.

4.3. Communication protocol

A communication protocol is used for communications between the debugger and debugger server. It is composed of CHILL signals called “program execution control services interface signals”. The signals define a number of operations such as reading and writing locations, single-stepping of each process, forcing procedure calls, suspending and resuming processes or programs, and defining traps [6]. The protocol is asynchronous, so that multiple operations can proceed simultaneously.

The following are important interface signals:

- setting up a debugging connection
- terminating a connection

- passing values
- controlling the program state
- handling processes
- handling locations
- trap handling
- stepping
- status reporting

5. Conclusions

In this paper, we described the CHILL cross debugging system for large-scale switching software. First of all, we introduced the basic concept and system architecture of the system. We also presented the features of the system and each component in detail. The system provides parallel and distributed real-time debugging, a signal sending, debugging command language, and GUI. We are now using the system for ATM switching software.

As further research, we are developing a host CHILL debugging system. In cross development environments, it is impossible to conduct testing and debugging of developed switching software if the hardware and operating system of the target system are not completed yet. To cope with these problems, the host debugging system will conduct the testing and debugging of switching software in the host environment. By using the system, it is possible to develop software within a short span of time, regardless of the implementation of target hardware and operating system. It is also possible to test most functional requirements before loading the switching software into the target system, thereby making it possible to locate errors at an early stage.

Acknowledgments

The authors would like to express their gratitude to the SDE and OS members for the assistance to this project, and to the members of Kvatro Telecom AS for the helpful discussions.

References

- [1] ITU-T. CCITT HIGH LEVEL LANGUAGE (CHILL), Recommendation Z.200, ISO/IEC 9496. Geneva, Swiss, 1996.
- [2] DongGill Lee, JoonKyung Lee, Wan Choi, Byung Sun Lee, and Chimoon Han. "A New Integrated Software Development Environment Based on SDL, MSC, and CHILL for Large-scale Switching Systems". *ETRI Journal*, 18(4):265–286, January 1997.
- [3] Kristen Rekdal. "CHILL - the International Standard Language for Telecommunications Programming". *Teletronikk*, 89(2/3):5–10, 1993.
- [4] Young Boo Kim, Soon Seok Lee, Chang Hwan Oh, Young Sun Kim, Chimoon Han, and Chu Hwan Yim. "An Architecture of Scalable ATM Switching System and Its Call Processing Capacity Estimation". *ETRI Journal*, 18(3):107–125, October 1996.
- [5] Jonathan B. Rosenberg. *How Debuggers Work: Algorithms, Data Structures, and Architecture*, chapter 11, pages 195–196. Wiley Computer Publishing, 1996.
- [6] Trond Borsting. "A CHILL Run-time System Support for Teletesting". In *Proceedings of the 5th CHILL Conference*, pages 96–103, Rio de Janeiro, Brazil, March 1990.
- [7] Boo-Keum Jung, Young-Jun Cha, Hyeoungkyu Chang, Eun-Hyang Lee, Hyung-Hwan Kim, Sung-Ik Jun, Ju-Hyun Cho, and Wan Choi. "SROS: Scalable Real-Time Operating System for ATM Switching Systems". In *Proceedings of the Second Conference on Communication Software*, pages 425–429, Seo-rak, Korea, July 1997.
- [8] Yong-Ik Yoon, Yoo-Mi Park, Mi-Kyong Han, and Seung-Sun Lee. "Scalable Distributed Real-time Database Management System for Switching Systems". In *Proceedings of ISS'97: World Telecommunications Congress*, pages 539–545, Toronto, Canada, September 1997.
- [9] Kvatro Telecom AS. *Pilot User's Manual*, December 1996. Revision 3.3.
- [10] Jon E. Stromme. "Integrated Testing and Debugging of Concurrent Software Systems". In *Proceedings of Information Network and Data Communication*, pages 273–286, Trondheim, Norway, June 1996.
- [11] Staale Deraas, Petter Moe, and Trond Borsting. DIL - Debug Information Language. Technical report, Kvatro Telecom AS, December 1996. Revision 6.