교환기 IPC를 이용한 CHILL 교차 디버거, LGDB의 구현

윤기창, 문정석, 김병철, 권경인, 조시철
{kcyun; jsmoon; kimbc; kikwon; sccho}@rex.lgic.co.kr
LG 정보통신

Implementing LGDB, CHILL Cross Debugger Using IPC of Switch System

Ki-Chang Yun, Jeong-Seok Moon, Byung-Chul Kim, Kyoung-In Kwon, Si-Cheol Cho
LG Information & Communications, Ltd,

요 약

교환기 소프트웨어 개발에 많은 언어가 사용되고 있으나, 현재 우리나라의 교환기 제조업체 대부분은 ITU-T에서 통신 소프트웨어 개발 언어로 권고하는 CHILL 언어를 사용하고 있다. CHILL은 엄격한 타입 검사, 구조적 프로그래밍, 모듈화, 병렬 처리 기능을 특징으로 하고 있다. LG 정보통신에서는 이러한 CHILL 프로그램의 시험 및 디버깅을 위하여 GNU Debugger (GDB)를 기반으로 CHILL 교차 디버거인 LGDB (LG Debugger)를 개발하였다. LGDB는 현재 사용 중인 대부분의 마이크로 프로세서를 지원하며, 호스트에서 개발한 프로그램들이 교환기 시스템과 같은 목적 시스템 상에서 정상적으로 실행되는지 검사하고, 만약 오류가 발생하였을 경우 이를 추적하여 수정하는 기능을 제공한다. 점차 대용량화, 고성능화가 요구되고 있는 교환기 개발 프로젝트의 추세를 감안할 때, LGDB는 교환기 소프트웨어의 품질 및 생산성 향상에 매우 높은 기여를 할 것이다.

1. 개 요

교환기 소프트웨어를 개발하는 데에 많은 언어가 사용되고 있으며, 우리나라의 교환기 제조업체에서는 주로 CHILL 언어를 사용하고 있다. CHILL 언어는 ITU-T에서 정보통신 소프트웨어 개발 언어로서 권고하고 있는 프로그래밍 언어로, 엄격한 타입 검사, 구조적 프로그래밍, 모듈화, 병렬 처리 기능을 특징으로 하고 있다.

교환기 시스템과 같은 내장형 시스템 상에서 실행되는 소프트웨어 개발은 일반적으로 Solaris 2.x를 운영체제로 하는 Sun SPARC 시스템에서 코딩과 컴파일 작업을 한 후, 교환기 전용 운영 체제인 CROS (Concurrent Real-time Operating System) / EROS (Enhanced Real-time Operating System)에 의해 운영되는 M68k / PowerPC 시스템에 로딩하여 시험 운용하는 방법을 이용한다. 이 기법을 사용할 경우, 호스트에서 개발한 프로그램들이 목적 시스템에서 정상적으로 실행되는지 검사하고, 만약 오류가 발생하였을 경우 이를 추적하여 수정하는 기능을 제공하는 도구가 필요하다. 이러한 목적을 위해 CHILL 교차 디버거 (LGDB)를 개발하였다.

LGDB는 목적 시스템에서 실행되는 교환기 소프트웨어를 network을 이용하여 호스트에서 디버깅할 수 있는 source level 디버거로서 오류가 예상되는 블록의 소스 라인에 breakpoint를 설정한 후 블록을 실행시키면 breakpoint가 설정된 곳에 프로그램 제어가 도달하는지 여부를 확인할 수 있고, 프로그램 제어가 일시 정지된 곳에서 소스 라인별로 실행시키면서 데이터 값을 검색할 수 있다. 또한, LGDB를 이용하여 호스트 상에서 직접 교환기의 shell 명령을 수행시킬 수도 있고, 목적 시스템으로 파일을 download하거나 호스트 시스템으로 가져올 수도 있다.

본 논문은 CHILL 교차 디버거의 설계 및 구현에 대해 설명한다. 제 2장에서는 LGDB의 전반적인 구조에 대해 서술하고, 제 3장에서는 각각의 구성 요소의 내부 구조에 대해 기술한다. 제 4장에서는 디버깅 정보 변환기에 대해 설명하고, 제 5장에서는 LGDB가 기존의 다른 디버거들과 비교되는 특징들을 나열한다. 마지막 제 6장은 결론과 앞으로의 과제이다.

2. LGDB의 구조

교차 디버깅 시스템의 일반적인 구조는 호스트 시스템에서 통신 link를 이용하여 교환기 시스템과 같은 목적 시스템 상에서 실행되는 프로그램의 수행을 제어할 수 있도록 구성된다. 호스트 시스템 상의 디버깅 client는 사용자로부터 breakpoint를 설정한다든지 데이터 값을 검색하는 것과 같은 디버깅 명령을 입력받은 뒤 이를 목적 시스템의 디버깅 server로 전송한다. 목적 시스템 상의 디버깅 server는 목적 시스템의 운영 체제의 기능을 이용하여 필요한 정보를 얻은 후 이를 다시 호스트 상의 client로 전송한다.

LGDB의 전체적인 구조는 그림 1에서와 같이 호스트 시스템 상의 LGDB client, 목적 시스템 상의 LGDB server 및 통신 link의 세 가지 요소로 구성된다. 새로운 디버깅 session을 시작하면 사용자는 목적 시스템으로 실행 파일을 download하고 LGDB server와 client-server 형태로 디버깅 packet을 주고 받는다. LGDB client는 breakpoint 설정과 같은 디버깅 요구를 LGDB server에게 보내고 응답을 기다리며, LGDB server는 통신 protocol에 따라 client가 보내온 메시지에 대한 응답을 전송한다. 응답을 받으면 LGDB client는 디버깅을 마칠 때까지 디버깅 요구를 보내고 응답을 받는 과정을 계속한다. 이 때 LGDB client는 source level 디버깅을 위하여 디버깅 정보[5]를 가지고 있는 실행 파일 및 소스 파일을 필요로 하며, 이들을 이용하여 소스 코드의 리스트를 보여줄 수 있다. 디버깅 정보는 LGDB client가 내부적으로 symbol table을 구성하는 데 사용된다.

목적 시스템용 실행 파일이 수행을 시작하면 CROS / EROS는 실행 코드로부터 CHILL 프로세스를 생성한다. LGDB server는 CROS / EROS의 기능을 이용하여 디버깅하고자 하는 프로세스를 제어하며 breakpoint 설정 또는 메모리나 레지스터 내용의 참조 / 변경 및 코드 일부의 수행과 같은 LGDB client의 요구에 대한 처리를 수행하고, 그 결과를 client에게 응답으로 보내준다.

LGDB client와 LGDB server간의 통신을 위한 통신 link는 호스트 시스템과 목적 시스템 사이에서 목적 시스템용 실행 파일을 download하거나 LGDB server에게 디버깅 요구를 전송하고 그 결과를 되돌려 받는데 사용되며, CROS / EROS에서 제공하는 교환기 IPC (InterProcess Communication) [2]를 이용한다. LGDB client가 교환기 내의 프로세스와 교환기 IPC로 통신하기 위해서는 workstation에 IPC server [2]가 있어야 하며, LGDB client는 이를 이용할 수 있어야 한다. 교환기 IPC는 교환기 프로세스들간에 정보를 주고받는 기본적인 mechanism으로, 이를 사용함으로써 교환기 내의 모든 프로세스들과 안정적으로 통신할 수 있으며, IPC server는 UNIX 상에서 교환기 IPC를 사용할 수 있도록 해 주는 프로세스이다.

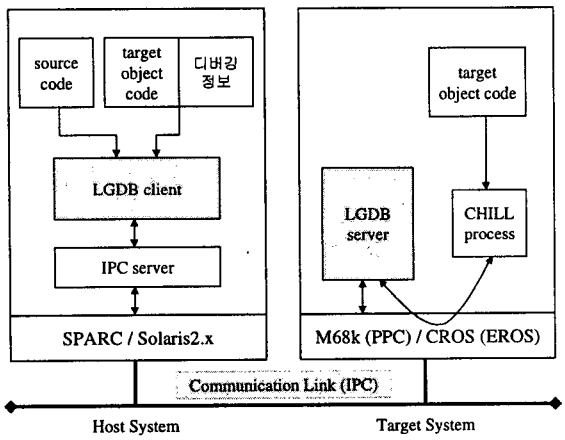

그림 1 LGDB의 전체적 구조

3. LGDB 구성 요소의 설계 및 구현

3.1. LGDB client

LGDB client는 User Interface, Expression Parser, Command Handler, Object file Reader, Communication Handler로 구성되며[3], 기본적으로 GDB [1]가 제공하는 기능들을 그대로 이용하고, 이 중 Command Handler 및 Communication Handler를 수정하여 구현한다.

User Interface는 사용자에게 명령어를 입력하는 인터페이스를 제공하고, Expression Parser는 디버깅 명령어에 포함되어 있는 CHILL 수식을 parsing한다. Command Handler는 사용자가 입력한 디버깅 명령어를 분석하여 해당 subroutine으로 제어를 넘기는 부분으로 명령어 table과 해당 명령어가 수행할 subroutine 및 관련 정보를 관리한다. 교환기 shell 명령의 수행 및 목적 시스템과의 파일 up / downloading을 위하여 해당 기능을 수행하는 subroutine이 Command Handler에 추가되었다. Object file Reader는 목적 시스템용 실행 파일에 포함되어 있는 디버깅 정보를 읽어들여 내부적으로 symbol table을 구성, 관리하는 부분으로, LGDB에서 사용되는 object file format은 COFF (Common Object File Format) [4], ELF (Executable and Linking Format) [6] 등이 있다. Communication Handler는 LGDB server와 통신하기 위한 인터페이스를 제공하는데, 이를 위해 기존의 교환기 IPC 상에서 UDP protocol을 emulation하여 Berkeley socket 인터페이스를 구현하였다. 이에 대해서는 3.3.에서 자세히 기술한다.

3.2. LGDB server

LGDB client가 디버깅 명령을 LGDB server로 전송하면 명령을 받은 LGDB server는 목적 시스템의 운영 체제의 기능을 이용하여 필요한 정보를 얻은 후 이를 다시 LGDB client로 보내준다. 이를 위해 CROS / EROS는 Solaris 2.x의 ptrace () [6]와 유사한 기능을 수행하는 primitive들을 제공하여 LGDB server가 이를 이용하여 디버깅을 수행할 수 있도록 하였다. LGDB server는 GDB에서 remote 디버깅을 위해 제공되는 gdbserver [1]를 교환기 시스템에 맞도록 수정하여 구현하며, 그림 2에서와 같이 Server-Main, Server-Child, Exception Handler 및 Communication Handler로 구성된다.

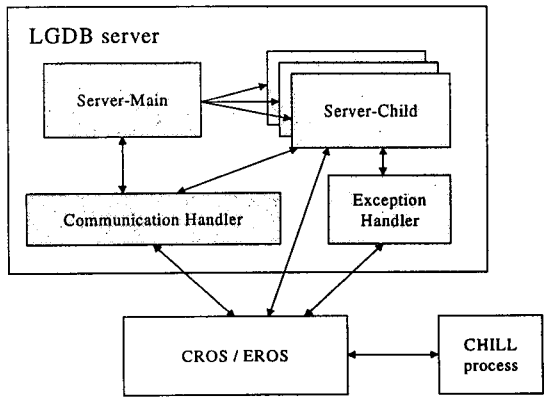

그림 2 LGDB server의 구조

Server-Main은 LGDB server의 전체를 관리하는 부분으로, 새로운 디버깅 session이 시작되어 LGDB client가 디버깅 대상의 설정을 요구하는 메시지를 보내면, Server-Main은 그 메시지로부터 디버깅할 프로세스에 대한 정보와 메시지를 보낸 LGDB client의 정보를 알아낸 다음, Server-Child를 생성하여 Server-Child가 디버깅할 프로세스를 제어하도록 한다. 또한, 디버깅을 위해 지나치게 많은 자원이 사용되는 것을 방지하기 위해 Server-Child 프로세스의 개수에 제한을 두어 각 프로세스에 대한 정보를 관리한다.

Server-Child는 LGDB server에서 중심적인 역할을 수행하는 부분으로, 목적 시스템상에서의 프로세스의 수행을 제어하며 메모리와 레지스터의 내용을 검색하고 수정한다. 즉, Server-Child는 Server-Main으로부터 생성될 때 디버깅할 프로세스에 대한 정보와 디버깅을 요구한 LGDB client에 대한 정보를 넘겨받아 이를 이용하여 디버깅 대상의 설정 (attach)을 시도하고, 이후 LGDB client가 보내오는 디버깅 메시지를 받아서 CROS / EROS가 제공하는 기능을 이용하여 각각의 경우에 알맞은 처리를 한 뒤, 그 결과를 LGDB client로 다시 전송하여 준다. 특히, LGDB client로부터 breakpoint 설정 요구가 들어오면 목적 시스템용 실행 파일의 해당 address의 instruction을 trap instruction으로 대체하고, 프로세스의 수행이 breakpoint를 설정한 지점에 도달하면 LGDB client에게 이러한 정보를 전송한 다음 trap instruction을 원래의 instruction으로 복원하고 breakpoint의 해제 요구가 있을 때까지 관련 정보를 유지, 관리한다.

Exception Handler는 프로그램의 수행 중에 발생하는 bus error 또는 segmentation violation 등을 감지하여 관련 정보를 LGDB client에게 전

송하며, Communication Handler는 LGDB client와의 통신을 담당하는 부분으로 디버깅 명령이 포함된 packet을 받아들여 해당 요구를 수행한 결과를 다시 packet으로 구성하여 LGDB client로 전송한다. 이 데이터 packet들은 GDB에서 정의한 protocol을 기반으로 설계한 통신 protocol을 사용하여 전송되며, 이에 대해서는 다음 절에서 기술한다.

3.3. 통신 인터페이스 및 프로토콜

LGDB client와 server간 통신을 위해 CROS / EROS에서 제공하는 교환기 IPC 상에서 UDP protocol을 emulation하여 Berkeley socket 인터페이스를 구현하고, 이를 TDX-IPC socket이라 명명하였다. TDX-IPC socket을 이용하여 LGDB에서뿐만 아니라, 호스트 시스템과 목적 시스템 사이의 통신을 필요로 하는 다른 프로그램들, 이를테면 remote shell 프로그램 또는 remote file loading 프로그램에서도 Berkeley socket과 동일한 인터페이스로 통신할 수 있다. 또한, 여러 개의 프로세스들이 동시에 서로 다른 port 번호를 가지고 통신할 수 있도록 workstation 및 CROS / EROS의 shell에 port 번호를 관리하고 TDX-IPC socket datagram을 적절한 프로세스로 전달하는 기능을 담당하는 port manager (이하 portman)라는 프로세스를 구현하였다. Portman은 다음과 같은 기능을 수행한다.

- TDX-IPC socket client에 적절한 port 번호를 bind해 준다.
- 더 이상 사용되지 않는 port를 다른 client가 bind할 수 있도록 release한다.
- Sender측 client로부터 datagram을 받아 buffering한다.
- Receiver측 client에게 buffering된 datagram을 전달한다.

프로그램들은 상대방의 프로세서 식별자 (processor identifier)와 port 번호를 알고 있는 상태에서만 통신이 가능하며, 이를 위해 모든 프로그램들은 datagram을 주고 받기 전에 그들의 TDX-IPC socket에 자신의 port 번호를 bind해야 한다. 그림 3은 portman이 LGDB client와 LGDB server간의 통신을 중재하는 과정을 도식화한 것이다.

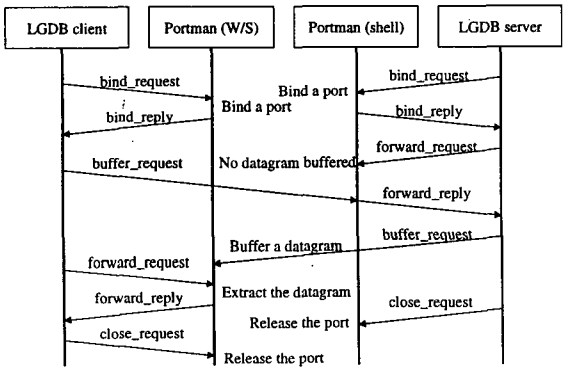

그림 3 port manager의 기능

LGDB client와 LGDB server간에 데이터를 전송하기 위해서는 통신 link와 함께 쌍방간에 정의된 통신 프로토콜이 필요하다. LGDB에서는 client와 server간 신뢰성이 높은 통신을 하기 위해 remote protocol을 정의하여 사용하는데, 이는 GDB에서 정의한 remote protocol [1]을 바탕으로 구성하며, **'$data#checksum'**과 같은 형식의 frame을 하나의 packet으로 하여 디버깅을 위한 데이터를 전송한다.

4. 디버깅 정보 변환기

당 연구소에서 교환기 소프트웨어를 개발하는 데 사용하는 CHILL 컴파일러 (shc / chc)는 디버깅 정보를 stab [5] 형식으로 생성한다. 그런데, 현재 목적 시스템용 운영 체제인 CROS는 COFF [4] 형식을 지원하므로, shc / chc에서는 stab 형식의 디버깅 정보를 COFF 형식으로 변환하는 작업을 bsdcoff라는 프로그램이 하게 된다. 하지만, bsdcoff가 생성한 COFF 형식의 디버깅 정보는 기본 COFF 형식을 따르고 있지 않아 LGDB는 이를 정확히 이해할 수 없다. 따라서, shc / chc가 생성한 stab 형식의 디버깅 정보를 기본 COFF 형식으로 변환함으로써 LGDB가 디버깅 정보를 정확히 인식할 수 있도록 하는 디버깅 정보 변환기 (stab2coff)를 구현하였다.

5. LGDB의 특징

LGDB는 기존의 다른 CHILL 디버거들에 비해 다음과 같은 특징들을 가진다.

- GDB가 가지고 있는 대부분의 기능들을 지원할 뿐만 아니라, 디버깅을 하면서 교환기의 shell 명령을 수행하여 프로세스의 수행 상태를 직접 확인할 수 있고, 목적 시스템과의 파일 up / downloading도 LGDB 상에서 가능하다.
- 여러 사용자가 동시에 서로 다른 프로세스를 독립적으로 디버깅할 수 있다.
- 한 프로세스에서의 메모리 변경이 다른 프로세스들에게 영향을 미치지 않는다. 즉, breakpoint를 설정하기 위해 특정 프로세스의 메모리 내용을 변경하는 경우, 그 프로세스와 text 영역을 공유하는 다른 child 프로세스들이 다같이 수행을 정지하게 되지 않도록 CROS / EROS는 디버깅 대상 프로세스로의 context switch가 일어날 때 해당 address의 instruction을 trap instruction으로 대체하고, 다른 프로세스로 context switch가 일어나면 변경된 메모리의 내용을 원래의 내용으로 복구시킨다. 이렇게 함으로써 디버깅과 상관없는 프로세스들은 정상적으로 수행이 가능하다.
- LGDB client와 server간의 통신은 교환기 IPC를 사용함으로써 통신의 신뢰도를 높게 하였고, TDX-IPC socket 인터페이스를 구현함으로써 호스트 시스템과 목적 시스템 사이의 통신을 필요로 하는 다른 프로그램들에서도 이를 이용할 수 있도록 하였다.

6. 결론 및 향후 과제

본 논문은 교환기 소프트웨어를 테스트하고 디버깅할 수 있는 도구인 CHILL 교차 디버거의 설계와 구현에 대해 기술하였다. 본 논문에서 소개된 LGDB는 소프트웨어 디버깅 도구로 널리 사용되고 있는 GDB를 기반으로 설계하였으며, 실제로 STAREX-TX1A 교환기 개발에 적용하여 사용할 수 있다.

Debug packet 및 ACK의 유실 가능성에 대비하기 위해 통신 protocol의 보완을 고려 중에 있으며, 시그널 정보 생성 및 시그널 송수신 기능의 구현, realtime 디버깅 지원 등이 앞으로의 과제이다.

참고 문헌

[1] Richard M. Stallman, *Debugging with GDB, The GNU Source- Level Debugger*, Free Software Foundation, 1994

[2] 조용인 외 2명, *STAREX-TX1 OS 사용자 설명서*, LG 정보통신, 1998

[3] YoungJoon Byun 외 3명, *The Implementation of CHILL Cross Debugging System for ATM Switching Software*, 한국 전자통신 연구소

[4] Gintaras R. Gircys, *Understanding and Using COFF*, O'Rielly & Associates, Inc., 1988

[5] Julia Menapace, *The "stabs" Debug Format*, Cygnus Support, 1992

[6] SunSoft, *SunOS Reference Manual – Solaris 2.6*, Sun Microsystems, 1997