임베디드 ARM 리눅스를 위한 KGDB 구현*

°이재호 김선자
한국전자통신연구원 인터넷정보가전연구부

email: {bigleap, sunjakim}@etri.re.kr

The Implementation of KGDB for Embedded ARM-LINUX

Jae-Ho Lee°, Sun-Ja Kim
Internet Appliance Technology Dept. ,ETRI

요 약

KGDB 는 리눅스 커널을 위한 소스 레벨 디버거로서, GDB 와 함께 동작하여 커널 개발자가 리눅스 커널을 응용 프로그램처럼 디버깅할 수 있는 기능을 제공한다. 현재 KGDB 는 X86 계열의 하드웨어를 위한 리눅스에 오픈 프로젝트로서 커널 버전 2.4.18 까지 개발되어있다. 본 논문에서는 X86 용 리눅스만을 위한 KGDB 를 확장하여, 내장형 시스템에 널리 사용되는 ARM 프로세서 기반의 하드웨어에서 Linux 커널을 개발할 때에도 KGDB 를 이용할 수 있도록 프로세서 의존적인 부분을 구현하고, 이를 커널에 통합하여 삼성에서 개발된 ARM920T 기반의 S3C2400 보드에서 동작실험을 하였다

1. 서론

리눅스 커널 디버깅은 일반적인 응용 프로그램의 오류를 추적하여 수정하는 일처럼 쉽지않다. 이는 일반 응용 프로그램이 특정한 프로세서에서 수행되는 반면, 커널 자체를 디버깅하기 위한 잘못된 접근은 시스템이 다운되거나 crash 될 수 있기 때문이다. 특히 임베디드 시스템 개발에 사용 되는 커널은 최종적으로 타겟 시스템에 다운로딩되어 수행될 때 예상치 못한 오류를 발생하거나, 수정된 오류부분이 새로운 문제를 일으킬 수 있다. 이러한 환경에서의 디버깅은 개발자 경험에 매우 의존적이며, 반복적인 수정과 테스트(trial and error process)가 요구되므로 효율적인 커널 디버깅을 위한 도구가 필요하다. 현재 ARM Linux 커널 디버거는 Lenio 의 Embedix SDK 나 RedHat 에서 솔루션을 내놓긴 하였으나 소스가 공개되지 않아, 많은 사용료를 지불하고 사용해야 하는 실정이다. 본 논문의 2 장에서 리눅스 커널을 디버깅 하기위한 관련 기술들을 간략히 기술하고, 특히 오픈 프로젝트로 진행중인 KGDB 의 동작 원리와 프로세스 의존적인 부분을 분석한다. 3 장에서는 앞서 분석한 내용을 바탕으로 ARM-Linux 를 위한 KGDB 를 구현하고, 4 장에서 ARM 프로세서를 사용하는 실제 하드웨어에서 리눅스 커널 포팅 및 코어 모듈을 개발하는 환경에서 KGDB 동작을 검증하였다.

2. 관련연구

2.1 리눅스 커널 디버깅 기술

디버깅 기술은 커널 자체에 포함된 코드를 이용하거나 독립적인 외부 디버거 프로그램을 사용할 수 있다. 전자의 방법으로 커널에서 지원하는 "printk" 함수를 넣어 문제되는 부분을 출력하거나, "\proc" 파일 시스템 인터페이스(ps, top, uptime etc..)를 통해 시스템에 관한 유용한 정보를 모을 수 있다. 간단한 오류는 사용자 영역의 프로그램에서 시스템 호출이 제대로 이루어 지는가 "strace" 프로그램을 사용하여 프로그램의 흐름을 추적할 수 있다. 또한 대부분의 시스템 폴트는 커널이 다운되기 전에 Oops 메시지 출력함으로 문제를 분석할 수 있다. 그러나 이들은 정적인 디버깅 방법으로 반복적인 커널 코드 수정과 테스트를 요하는 임베디드 시스템 개발환경에는 적합치 못하다. 후자의 방법으로 인터렉티브 디버깅 도구로서 커널에 내장된 KDB(Built-in Kernel Debugger)가 있다. 이는 커널의 많은 부분을 수정한 패치를 적용하고 커널과 디버거를 하나의 이미지로 컴파일 한 뒤, 하나의 머신에 다운로딩되어 동작되므로 별도의 개발 호스트가 필요 없는 장점이 있는 반면, PC 와 입출력 인터페이스가 다른 임베디드 시스템 개발도구로 여러 가지 단점을

지닌다. LKCD (Linux Kernel Crash Dump) 분석방법은 커널에서 Ooops 메시지를 발생했을 때 시스템의 상태를 덤프파일에 저장해 두었다가, 나중에 LCRASH 프로그램을 이용하여 덤프파일을 요약 분석하여 문제를 해결한다. 현재 이러한 방법은 인텔의 32 비트 CPU 만을 지원하며, 시스템 덤프를 위한 SCSI 스왑 공간을 요구하는 하드웨어 제약사항을 지닌다. 본 논문에서 다루는 KGDB 는 정식 커널을 최소로 수정하면서도, GDB 사용에 익숙한 개발자에게 커널도 마치 응용프로그램처럼 인터렉티브하게 디버깅할 수 있는 편리함을 제공한다. 또한 GDB 의 프론트엔드 GUI 로서 DDD(Data Display Debugger) 나 Source Insight 프로그램을 사용할 수 있는 장점이 있으며, 개발자 호스트와 타겟 시스템이 분리되어 있는 환경이 임베디드 시스템 개발에 적합한 디버깅 도구로 유용하다.

2.2 X86 프로세서를 위한 KGDB

현재 KGDB 오픈 프로젝트로 linux-2.4.18-kgdb-1.5.patch 까지 개발되어 있으며, 유일하게 X86 계열의 CPU 만 지원한다. KGB 가 갖는 기본적인 특성과 기능은 아래와 같다.

Kernel Asserts: 커널 코드 내에 Assert 코드를 넣어 조건을 검사하고 오류가 있다면 gdbstub 처리 루틴으로 제어를 넘긴다.

Source level debugging: KGDB 가 포함된 리눅스 커널과 코어 모듈을 GDB 프로토콜을 이용하여 소스레벨에서 디버깅이 가능하다.

Support for threads in a kernel: "info thread"와 같은 명령을 통해 커널 내의 수행중인 thread 정보를 알아내고, 이들을 추적할 수 있는 기능을 제공한다.

IA32 hardware debugging supports: 수행 메모리 이미지를 고치지 않고 디버깅 레지스터를 별도로 두어 하드웨어적으로 breakpoint 를 발생시킬 수 있다.

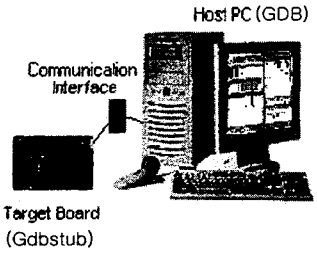
Console output through GDB: 개발 호스트의 터미널과 키보드를 공유하여 타겟 보드에서 발생된 커널 메시지와 디버깅 관련 메시지를 한곳에서 보는 것이 가능하다.

KGDB 의 사용은 리눅스 커널을 패치하고 커널 설정시 커널 디버깅 옵션을 Enable 시킴으로써 사용할 수 있으며, KGDB 실제 구현 부분은 아래와 같이 세 부분으로 요약된다.

Gdbstub : 커널 디버깅의 핵심요소로서 개발 호스트의 GDB 로부터의 요청 메시지들을 처리하기 위해 타겟 보드 위에서 GDB 서버로서 동작한다. 커널이 디버깅 모드로 동작하면 gdbstub 처리루틴에서 모든 프로세스를 제어할 수 있다.

Fault handling: 커널이 예상치 못한 오류에 대해 패닉을 발생 시키고 시스템이 바로 다운되는 대신 gdbstub 처리루틴으로 제어를 넘겨 오류 발생원인을 분석할 수 있다.

Serial Interface: KGDB 를 사용하기 위해서는 타겟 보드와 개발 호스트로 사용될 두 개의 machine 이 필요하며 이들 간의 통신을 위해 시리얼 인터페이스를 사용한다. 이는 네트웍을 통한 통신을 사용하지 않으므로 커널의 네트웍 구현 부분에 대해서도 디버깅이 가능함을 의미한다. [그림 1]은 두개의 machine 이 시리얼 인터페이스를 통하여 KGDB 를 이용하여 임베디드 시스템을 개발하는 환경을 보여준다.



[그림 1] KGDB 를 이용한 Embedded LINUX 개발 환경

3. ARM-Linux 커널을 위한 KGDB 구현

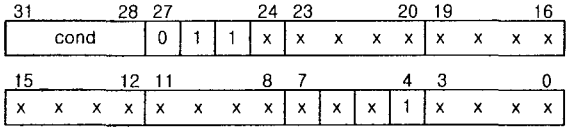
3.1 ARM 프로세스를 위한 디버깅 모드 진입방법

X86 프로세스의 디버깅 모드 진입은 개발 호스트 상에서 시리얼을 통한 "ctl-c" 키 입력이나 타겟에서 수행되는 커널의 예외상황 발생시 BREAKPOINT()를 수행함으로써 이루어진다. 이는 다음과 같이 정의되어 인터럽트를 호출하고, hook 함수로 등록된 gdbstub 처리루틴으로 제어를 넘긴다.

```
#define BREAKPOINT          asm(" int $3");
```

이와 유사한 방법으로 ARM 프로세서에서는 X86 프로세스의 "INT $3" 을 호출하는 대신, 다음 수행될 메모리 번지에 undefined 명령어를 넣어 예외 상황을 만든다. ARM 프로세스의 undefined 명령어 포맷은 [그림 2]와 같으며, 이를 이용하여 BREAKPOINT()를 정의한다.

```
#define BREAKPOINT asm(" .word 0xE7FFDEFE");
```



[그림 2] ARM 프로세서의 undefined 명령어 포맷

3.2 Fault 처리 부분 수정

커널에서 오류를 발생하고 시스템 동작을 멈추기 전에 마지막으로 처리되는 부분이 "arch/arm/kernel/traps.c" 에 있는 코드들이다. 커널은 Ooops 메시지와 함께 시스템이 다운되기 전에 메모리와 스택 영역, CPU 모드, 예외 처리, 미 정의된 명령어, data abort 등을 검사한 뒤 커널의 최종상태를 출력한다. 따라서 구현된 KGDB stub 코드에서는 커널이 다운되기 전에 gdbstub 처리 루틴으로 점프할 수 있도록 코드를 수정하였다.

3.3 Gdbstub 처리 루틴의 기본 기능

호스트의 GDB 의 요청 메시지를 처리하기 위한 gdbstub 루틴의 기본기능은 [표 1]와 같다.

[표 1] Gdbstub 처리루틴의 기본 기능

기 능	구현 함수	설 명
[1] CPU Read/Write	regs_to_gdbregs gdbregs_to_regs	- ARM CPU 내부 레지스터 값들을 Read - ARM CPU 내부 레지스터에 지정한 값을 Write
[2] Memory Read/Write	mem2hex hex2mem	- 타겟 메모리 내의 특정 어드레스로 부터 값을 Read - 타겟 메모리 내의 특정 어드레스에 지정한 값을 Write
[3] Serial Read/Write	putpacket getpacket	- 개발 host로 메시지를 전송 - 개발 host로 부터 메시지 수신

3.4 Linux 과 KGDB 의 컴파일

최종 타겟 이미지를 생성하기 위한 커널 컴파일 전단계에서 KGDB 사용여부를 개발자에게 질의하기 위해 "config.in" 파일에 "CONFIG_KGDB_xxx" 라는 이름의 환경 변수를 추가하여 커널 설정 과정에서 반영되도록 하였다.

4. 동작 검증

4.1 임베디드 리눅스 개발을 위한 KGDB 사용환경

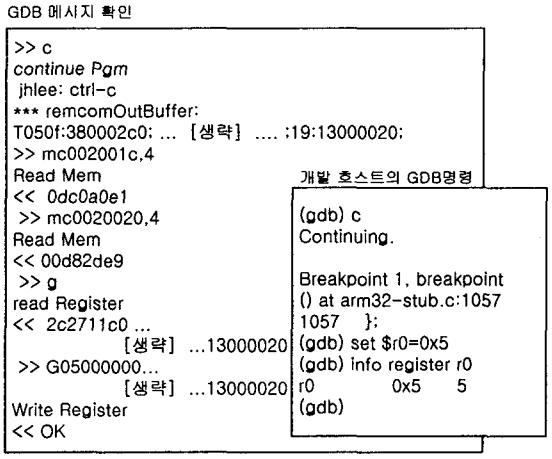
구현된 KGDB 를 사용하여 임베디드 시스템에 탑재되는 리눅스 커널이나 모듈을 개발하는 환경은 [그림 1]과 같으며, 세부적으로 요구되는 사양은 아래와 같다.

* 개발 호스트: GDB-5.0, RedHat 7.2 (ver. 2.4.18) on Pentium-III
* 타겟 시스템: kernel 2.4.17 + patch-2.4.17-rmk5 (ARM patch)

ARM920T based S3C2400 (made in Samsung)

4.2 KGDB 동작 검증

KGDB 는 3.3 절에서 기술한 기본동작을 바탕으로 동작하기 때문에 gdbstub 처리루틴이 검증은 개발 호스트의 GDB 요청 메시지를 제대로 수신하여 해석하고, 처리된 결과가 GDB 프로토콜에 맞도록 수신되는가 확인한다. 이를 위하여 KGDB 가 사용하는 인터페이스 외에 모니터링을 위한 시리얼 드라이버를 추가하고 터미날 프로그램을 통하여 송수신 메시지를 캡쳐하여 동작 검증하였다.



[그림 3] GDB 프로토콜을 통한 KGDB 실행화면

4.3 결론 및 향후 연구과제

현재 오픈 프로젝트로 진행중인 리눅스 커널 디버거인 KGDB 는 X86 계열의 CPU 만을 위해 개발 되어졌다. X86 기반의 프로세스를 사용하지않는 수 많은 embedded 시스템 분야에서 전통적인 RTOS 를 대신하여 리눅스가 가진 여러 가지 장점들로 이들을 대체하고 있는 실정이다. 본 논문에서는 임베디드 시스템 개발에 널리 쓰이는 ARM 프로세스군을 위해 리눅스 커널 및 코어 모듈을 개발할 때에도 KGDB 를 유용하게 사용할 수 있도록 하드웨어 의존적인 부분을 구현하고 실제 하드웨어에서 동작을 검증하였다.

향후 과제로서 리눅스 커널 내에서 패닉(panic)을 일으킬 수 있는 상황을 좀더 면밀히 연구하고, 타겟 시스템의 동작이 멈추기 전에 좀더 많은 예외 상황을 검사할 수 있도록 커널 내 kernel assert 코드를 위치시키는 것이 필요하다. 아울러 여기서 다루고 있지 않는 KGDB 를 통한 동적 모듈들에 대한 디버깅 기술들에 대한 추가 연구가 필요할 것이다.

5. 참고 문헌 및 웹 사이트

[1] KGDB 개발사이트, http://kgdb.sourceforge.net
[2] KDB 개발사이트, http://oss.sgi.com/projects/kdb/
[3] ARM Linux 개발자 사이트, http://www.arm.linux.org.uk
[4] http://www.lineo.com
[5] GDB 매뉴얼, http://sources.redhat.com/gdb/
[6] Alessandro , "Linux Device Drivers 2nd Edition", O'reilly, June 2001