정보가전용 내장형 소프트웨어 개발을 위한 원격 디버거의 설계 및 구현

이광용, 김창갑, 김흥남
한국전자통신연구원 소프트웨어공학연구부
e-mail : {kylee,changkap,hnkim}@etri.re.kr

A Design and Implementation of a Remote Debugger for Embedded Internet Software

Kwang-Yong Lee*, ChangKap Kim, Heung-Nam Kim
Software Engineering Department, ETRI

요 약

현재, 정보기기의 급속한 발전과 많은 수요로 인해 실시간 OS 에 대한 기술 개발의 필요성이 대두되었고, 많은 실시간 OS 제품이나 그 개발도구들이 개발되어 있으나, 기존의 상용 제품들은 산업용 실시간 OS 에서 파생된 제품이거나 PC 나 Workstation 용 OS 로부터 다운사이징한 제품들이고, 대부분 외국 제품들이어서 그 실시간 OS 들을 사용함으로써 부담하는 기술료는 국내 가전용 제품의 경쟁력을 급속히 약화시킬 우려가 있다. 이에 본논문에서는 본 연구소에서 자체 개발한 Q+(QPlus) 정보가전용 실시간 OS 와 이와 연동하는 커널 원격 디버깅 환경인 Q+Esto 디버깅 환경의 구현기술에 대해 소개한다. Q+Esto 원격 디버깅 환경은 사용하기 편리한 사용자인터페이스 제공, 모듈화 및 계층화를 통한 디버깅 기능의 확장성 제고, 그리고 리모트 디버깅 인터페이스와 같은 원격 통신 모듈을 통한 타겟정보 접근시간의 축소등의 장점을 갖고 있다.

1. 서론

최근 정보기기의 급속한 발전과 많은 수요로 인해서 실시간 OS 에 대한 기술 개발의 필요성이 대두되었고, 많은 실시간 OS 제품이나 개발도구가 개발되어 있다[1-6]. 그러나, 기존의 제품들은 주로 VRTX, pSOS, VxWorks 등과 같은 산업용 실시간 OS 에서 파생되거나 Windows CE 나 QNX 와 같이 기존의 PC 나 Workstation OS 로부터 다운사이징한 제품들이며, 이는 가전용 제품에 적합한 조립형 실시간 OS 로써는 최적의 OS 라 할 수 없다. 그리고, 최근 몇몇 업체와 학교, 연구소 등에서 실시간 OS 개발을 시도하고 있으나 국내의 실시간 OS 기술은 연구개발이 초기 단계에 있고, 외국 실시간 OS 는 범용성을 중심으로 지원하기 때문에 가전용품에 대한 전문적인 지원에는 한계가 있다. 또한, 국내 업계는 점차적으로 가전용 제품의 국산화율을 높이고자 노력하고 있는데 현재 가전용 제품의 추세로 보아 조만간 많은 제품들이 실시간 OS 를 필요로하다. 그러나, 외국 제품의 실시간 OS 들을 사용함으로써 기술료를 부담한다는 것은 국내 가전용 제품의 경쟁력을 급속히 약화시킬 우려가 있다.

이에, 본 실험실에서는 지난 한해동안 조립형 실시간 OS 개발 과제에서 자체 개발한 Q+ 실시간 OS 및 이와 연동하는 원격 디버깅 환경인 Q+Esto 디버거를 개발하였으며 올해는 디버거 기능을 순차프로그램용 디버거에서 멀티태스크용 디버거로 그 기능 확장할 계획에 있다.[7]

2. Q+Esto 원격 디버깅 환경

Q+Esto 원격 디버깅을 위한 교차(cross-development) 환경은 다음 그림 1 에서와 같다. 이 그림에서와 같이 커널 교차 디버거가 수행되는 호스트 시스템과, 디버깅 모니터를 수행하는 타겟 시스템(여기에서는 SA110 을 탑재한 EBSA21285 보드를 지칭함)[8], 그리고 이들을 연결하는 시리얼 라인 혹은 이더넷 라인으로 구성된다. 대부분의 원격 디버깅 환경에서는 시리얼라인과 이더넷라인을 동시에 활용한다. 이의 구현 형태를 좀더 자세히 살펴보면 호스트 시스템은 Windows NT/95/98 을 플랫폼으로 하는 범용 컴퓨터로 구성되어 있으며, 이 호스트 컴퓨터에서 사용하는 원격 디버거는 Cygnus 에서 제공하는 GNU gdb 를 수정/보완하여 사용한다. 타겟시스템은 EBSA21285 하드웨어를 기반으로 구성되어 있으며, 타겟위에서 하

드웨어 시스템 초기화 루틴들과 커널 시스템을 적재시키기 위한 단순한 로직들로 구성된 커널 모니터링 시스템으로 구성되어 있다. 이 커널 모니터는 GNU gdb 와 연동하여 동작한다. 여기에서 GNU gdb 를 호스트용 교차 디버거로 주로 사용하는 이유는 이 gdb 는 공개된 소프트웨어로 로열티에 문제가 없으며, 다양한 타겟을 지원 가능한 형태로 구조화되어 있고, 디버그 자체 버그를 그동안 많이 수정/보완이 되었기 때문이다.

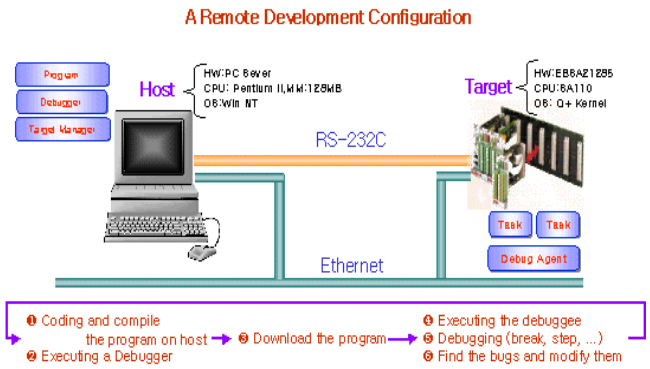


그림 1. 커널 교차 디버깅 환경

그림 1 과 같은 구성에서 커널 교차 디버깅은 호스트 시스템에서 타겟 시스템용으로 교차 컴파일된 실행파일을 타겟보드로 전송한 후 호스트에서 타겟에 적재된 실행파일을 제어하면서 디버깅을 수행한다. 이때, 호스트에 있는 커널 디버거는 타겟과의 통신 트랙픽을 최소화하기 위해 디버깅 정보가 포함된 실행파일과 이것의 소스 프로그램을 필요로 한다. 교차 디버거는 이 실행파일에 포함된 디버깅 정보를 이용하여 내부 심볼 테이블을 구축하고 소스프로그램의 열람, 변수의 모든 출력등 타겟 시스템에서 직접적인 수행을 필요로 하지 않는 기능을 목적 시스템과의 통신 없이 사용자에 알려준다.

이상과 같은 교차디버깅 방법을 순서적으로 기술하면 다음과 같다.

단계 1. 그림 1 과같이 시리얼 라인 연결, 호스트-타겟 시스템 시동 등의 초기실행환경을 구성한다.
단계 2. 호스트 시스템에서 교차 컴파일러를 사용하여 타겟용 실행파일을 생성한다.
단계 3. 생성된 실행파일을 타겟시스템으로 다운로딩 한다.
단계 4. 타겟의 실행파일을 실행시킨다.
단계 5. 호스트 시스템에서 디버깅 명령을 이용하여 타겟 시스템의 실행을 관찰하면서 버그를 발견해 낸다.
단계 6. 버그 수정 후 단계 2 부터 다시 시작한다.

3. 원격 디버거 구현 기술

정보가전 제품에서와 같은 내장형 시스템은 프로그램 크기는 작을지 모르나 개발자의 실수를 눈감아 주지 않는 개발하기 힘든 시스템이며, 현재, 정보가전 제품의 기능 변화속도에 맞추어 효과적으로 품질이 좋은 시스템을 개발할 수 있는 지원도구들이 요구되고 있다[9-11]. 본 실험실에서 개발한 원격디버거는 이러한 요구사항들을 만족시키기 위해 다음과 같은 설계 원리에 따라 개발이 되었다.

- 확장성 및 이해성 증진을 위한 계층화된 아키텍쳐의 제공
- 재사용성을 위한 객체기반 모듈화
- 디버깅 성능향상을 위한 lightweight 호스트-타겟 연결

3.1 Q+Esto 원격 디버거

Q+Esto 디버거는 GNU 의 소스 레벨 디버거인 GDB 를 기반으로 개발되었다. GDB 는 풍부한 디버깅 기능, 소스 프로그램 확보의 용이성, 높은 이식성 등으로 인하여 소프트웨어 디버깅 도구로 널리 이용되고 있다. 또한, GDB 는 새로운 언어의 확장이 용이한 형태로 모듈화 된 구조로 되어 있다(그림 2 참조). 현재 개발된 gdb 교차디버거의 주요기능은 다음과 같다.

- 실행 파일의 수행, 정지 및 재수행
- 소스 프로그램 열람 기능
- 정지점 설정 및 해제 기능
- 한 명령어 단위의 수행
- 특정 변수 내용의 조회 및 변경
- 스택 참조 기능

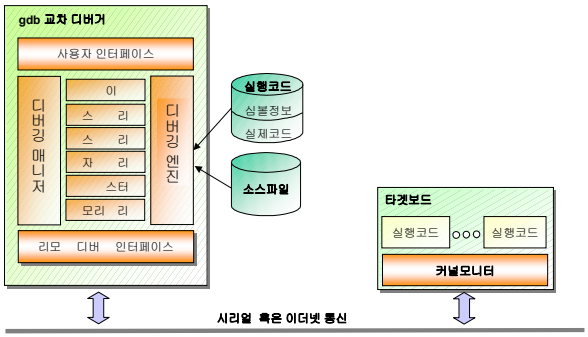


그림 2. Q+Esto 원격 디버거의 시스템 아키텍쳐

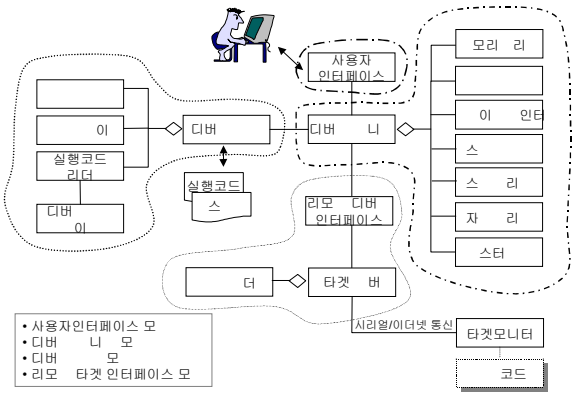


그림 3. Q+Esto 원격 디버거의 내부 구조

그림 3 은 Q+Esto 원격 디버거의 내부 구조를 보여준다. 여기에서 디버깅엔진은 디버깅 정보를 읽어 내부 심볼테이블을 구성하고 사용자와의 인터페이스를 통하여 소스 파일을 디버깅 한다. gdb 교차 디버거의 내부는 사용자인터페이스 모듈, 디버깅 매니저 모듈, 디버깅 엔진 모듈, 그리고 리모트 디버깅 인터페이스 모듈로 구성한다. 첫 째, 사용자인터페이스 모듈은(그림 4 참조) 사용자로부터 디버깅 명령을 입력으로 받아 명령의 종류별로 디버깅 매니저에 보고하고 디버깅 매니저의 처리결과를 받아 사용자에게 전시하는 역할을 담당한다. 이 모듈에는 제어관점, 소스관점, 스택관점, 정지점관점, CPU 및 메모리관점, 조사 및 평가관점, 그리고 변수관점을 표시하는 기능들로 구성되어 있다. 두 번째, 디버깅 매니저 모듈은 사용자 인터페이스 모듈로부터의 명령을 해석하고 이를 실행엔진인 디버거 엔진에게 전달하여 디버깅 명령을 수행하고 그 결과를 사용자 인터페이스 모듈에 전달하는 중간 브리지 역할을 담당하는 객체들로 구성되어 있다. 특히, 디버깅 엔진의 실행 상태를 관리하기 위해 상태 핸들러를 독립적인 객체로 두고 있기 때문에 사용자 인터페이스의 디버깅 과정을 통제할 수 있도록 되어 있다. 이 모듈에는 사용자인터페이스 모듈의 여러 관점들을 지원하기 위한 특별한 객체들로 구성된다. 세 번째, 디버깅 엔진 모듈은 디버깅 매니저로부터의 통제에 따라 실제 디버깅 명령을 수행하는 모듈들로 구성되어 있으며 GNU 디버거 기능을 그대로 수용한다. 구성은 명령어 파서 객체, 명령어 테이블, 실행코드 리더, 디버거 심볼테이블 들로 구성되어 있다. 구체적으로 이 모듈은 다음 그림 5 에서 처럼 디버깅 디렉토리, 소스파일, 명령어 테이블, 그리고 심볼테이블등을 엔진 시작과 동시에 초기화하고나서 실제 디버깅 명령을 입력으로하여 필요한 기능을 수행하는 구조로 되어 있다. 마지막으로, 리모트 디버깅 인터페이스 모듈은 교차 디버거의 상위 모듈과 타겟 모니터와의 연결을 담당하는 모듈로써 디버깅 매니저로부터의 디버깅 명령을 받아 타겟 모니터에 전달하고 그 수행결과를 수집하여 상위모듈들로 전달하는 역할을 담당한다.

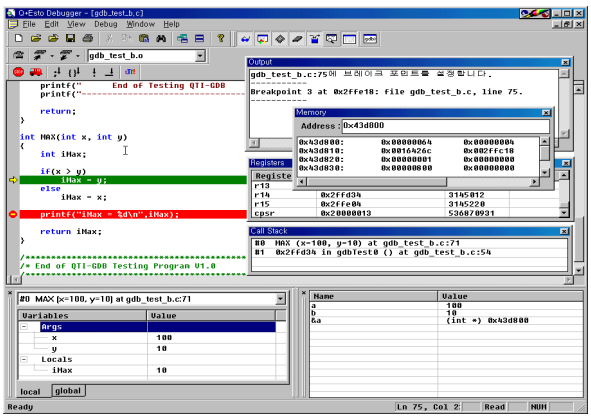


그림 4. Q+Esto 디버거 사용자인터페이스

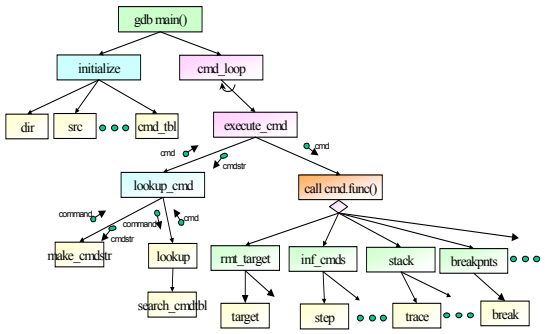


그림 5. 디버깅 엔진 기능

3.2 kmon 타겟모니터 구현

타겟 모니터 시스템은 타게보드 power-on 과 동시에 동작하며 호스트로부터 프로그램을 다운로드 받아 수행시킨다. 이 시스템은 그림 6 과같이 구성되며, 프로그램 실행제어, 정지점 관리, 메모리/레지스터 관리, 그리고 통신 관리 등의 크게 4 가지 모듈로 이루어져 있다. 이를 좀더 자세히 살펴보면 프로그램 실행제어 모듈은 타겟 모니터의 핵심 부분으로 타게시스템에 적재된 실행파일을 제어하고 메모리, 레지스터 등의 내용을 조회 또는 변경하는 기능을 수행한다. 정지점 관리 모듈은 실행파일의 특정 소스라인에 정지점을 설정하여 프로그램의 실행을 강제로 중단시키는 기능을 제공한다. 이를 위하여 교차 디버거로부터 정지점 설정 요청이 오면 하드웨어에서 제공하는 트랩을 이용하여 해당 메모리 주소의 명령어를 정지점 트랩 명령어와 대치한다. 그리고 일단 정지점에 도달하면 다시 원래 명령어로 환원하여 실행하고 정지점 설정에 대한 정보는 해제 요청을 받을 때까지 관리한다. 메모리/레지스터 관리 블록은 프로세스 실행과 관련해서 변경된 값을 호스트의 요청에 의해 제공 혹은 변경하는 기능을 수행한다. 그리고 통신관리 모듈은 교차디버거로부터 시리얼 라인을 통해 패킷을 전달받고 이를 분석하여 프로그램 실행 제어 모듈로 전송하거나 타겟모니터의 실행결과를 패킷으로 구성하여 호스트 시스템으로 전송한다. 이 패킷은 gdb 에서 정의된 리모트 디버깅 통신 프로토콜을 이용하여 전송된다.

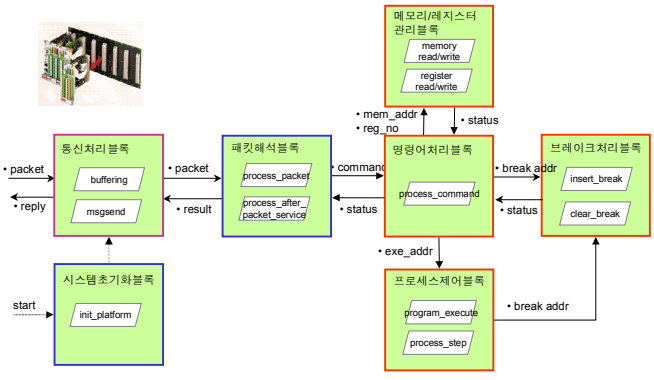


그림 6. kmon 타겟 모니터의 구조

3.3 리모트 디버깅 인터페이스 모듈 및 통신 프로토콜 구현

본 Q+Esto 원격 디버거에서는 kmon 타겟모니터와 통신하기 위하여 아래와 같은 11 가지 리모트 디버깅 인터페이스 함수들을 사용한다.

- msgsend
- wait_for_debug_message
- kmon_RDI_open, kmon_RDI_close
- kmon_RDI_read, kmon_RDI_write
- kmon_RDI_CPUread, kmon_RDI_CPUwrite
- kmon_RDI_setbreak, kmon_RDI_clearbreak
- kmon_RDI_ExecuteOrStep

여기에서, msgsend 함수와 wati_for_debug_messge 함수는 호스트 혹은 타겟으로부터의 명령 혹은 응답을 패킷형태로 구성하여 서로 송/수신함수로서 아래와 같은 패킷 구조를 이용하여 호스트-타겟간의 명령어 채널을 구성한다. 하나의 패킷은 시작문자(STX)로 부터 종결문자(ETX) 까지의 문자열로 구성되어 있으며, 아래 구조에서 보듯이 차례데로 패킷의 종류를 나타내는 Type 필드 1 바이트와 Contents 의 길이를 표현하는 Length 필드 2 바이트, 그리고 패킷의 오류 점검을 위한 CRC 필드 4 바이트로 구성된다. 그리고, msgsend 함수에서 이 패킷을 구성하는데 Length 값은 Channel 필드 4 바이트를 포함해서 계산한다.

STX (1)	Type (1)	Length (2)	Channel (4)	Contents (#Length)	CRC (4)	ETX (1)

kmon_RDI_open 에서는 ADP_ParamNegotiate 와 ADP_Reset 메시지를 kmon 타겟모니터에게 전송하고 kmon 타겟모니터에서는 이 메시지에 반응하여 시스템을 초기화하고 그 상태를 호스트에 전달한다. kmon_RDI_close 함수에서는 ADP_End 메시지를 타겟에 전달하고 kmon 타겟 모니터의 응용프로그램 모니터링 상태를 종결하도록 한다.

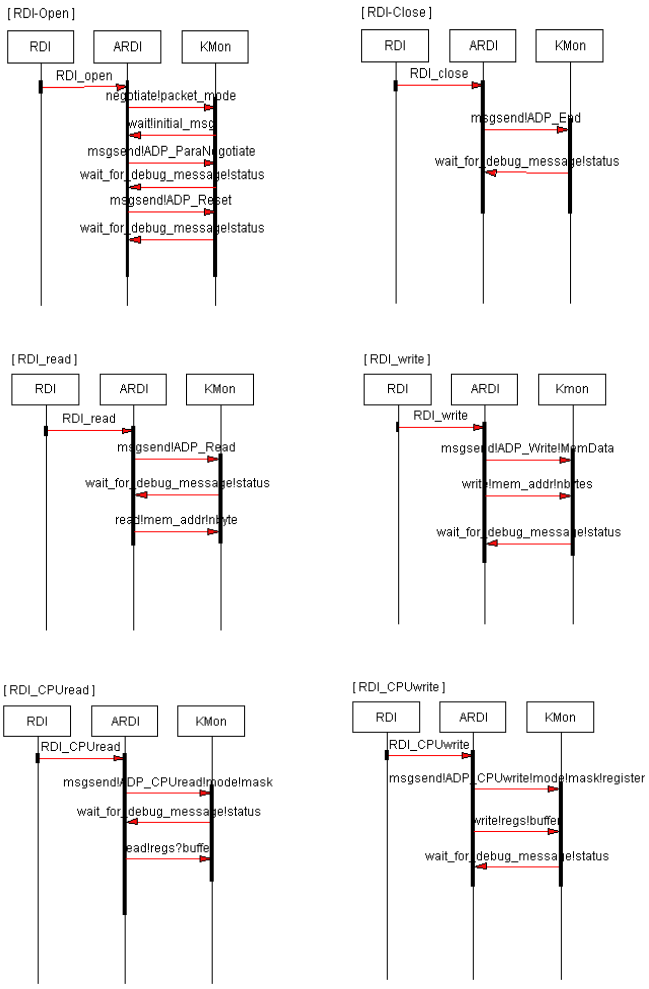
kmon_RDI_read 와 kmon_RDI_write 함수는 타겟의 특정 메모리 어드레스에 존재하는 메모리값을 읽기/쓰기 하는 함수들로 여기에서는 ADP_Read 와 ADP_Write 메시지를 구성하여 타겟에 전달하고 그 상태를 확인한다.

kmon_RDI_CPUread 와 kmon_RDI_CPUwrite 함수는 타겟의 레지스터들에서 값을 읽어오고 쓰기위한 함수로 ADP_CPUread 와 ADP_CPUwrite 메시지를 사용한다.

kmon_RDI_setbreak 와 kmon_RDI_clearbreak 함수는 정지점 설정/해제를 위한 것으로써 ADP_SetBreak 와 ADP_ClearBreak 메시지를 이용해서 타겟모니터에게 명령을 전달한다.

kmon_RDI_ExecuteOrStep 는 gdb 의 run 명령어에 반응하여 디버깅 프로그램을 시작시점으로부터 실행시키기 위한 기능과 step 혹은 continue 명령어에 반응하여 프로그램을 특정 브레이크포인트 지점까지 프로그램을 수행시키는 기능으로 구성되어 있다. 이 때 사용하는 메시가 ADP_Execute 이며 run, step, continue 의 명령어에 따라 특정 ADP_ExecuteRun, ADP_ExecuteStep, ADP_ExecuteContinue 로 달리하여 타겟모니터에 전달이 된다.

이상의 리모트 디버깅 인터페이스 기능의 호스트-타겟간의 프로토콜을 메시지 시퀀스 챠트로 표현하면 다음 그림 7 과 같다. 이 그림들에서 보듯이 하나의 명령어에 대한 송/수신 프로토콜은 msgsend 함수에 의해 명령어에 해당하는 특정 메시지를 타겟모니터에 전달하고 타겟모니터는 그 메시지에 반응하여 적절한 행위를 수행하고 그 결과에 대하여 호스트 쪽에 리턴 wait_for_debug_message 함수를 통해 입수한다. 일단, 명령어 채널이 형성이 되면 kmon_RDI_CPUread / kmon_RDI_CPUwrite 혹은 kmon_RDI_read / kmon_RDI_write 함수에서 처럼 경우에 따라서는 데이타 송수신을 통해 필요로하는 자료를 서로 주고 받는다. 이 통신 프로토콜에서 볼 때, RDI_read 혹은 RDI_CPUread 통신 프로토콜의 경우 호스트-타겟간에 명령어 채널이 완전이 형성 되었음이 보증이 된 후에 정보를 읽어오는 처리를 수행하고, RDI_write 와 RDI_CPUwrite 와 같이 명령어 채널과 동시에 정보도 동시 보내는 형태를 취하고 있음을 알 수 있다.



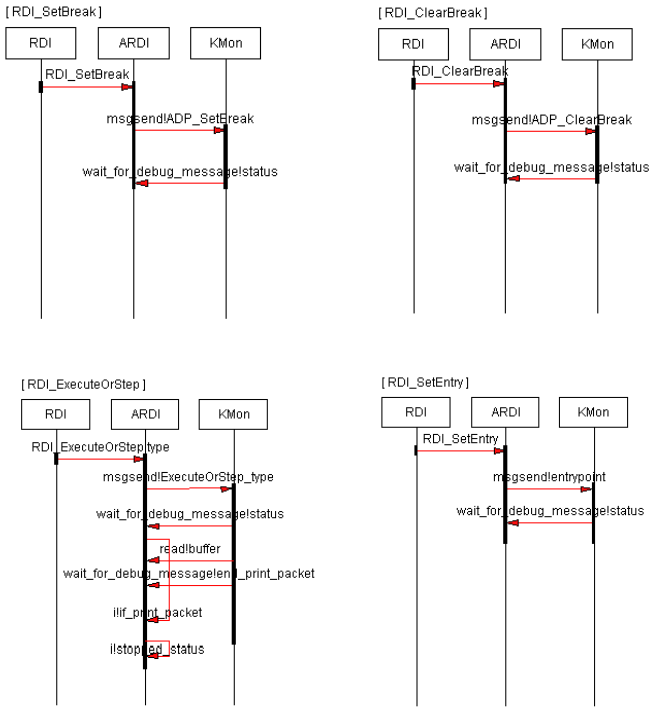


그림 7. 디버깅 메시지 통신 프로토콜

4. 결론 및 향후연구방향

본 논문에서는 C 언어 원격 디버깅용 교차 디버거 기술에 대한 내용을 서술하였다. 이의 세부 사항으로 Q+Esto 원격 디버깅 환경을 기술하였으며 또한 gdb 교차디버거, kmon 커널 모니터, 리모트 디버깅 인터페이스 모듈 및 통신 프로토콜에 대하여 기술하였다.

본 디버거는 gdb 를 기반으로 구현되며 정보가전용 내장형 시스템과 같이 프로그램 크기는 작지만 사소한 실수에도 응용 프로그램 자체가 제기능을 다하지 못하는 시스템을 디버깅 하기 위하여 만들어졌다. 현재 gdb 를 활용하여 소스수준에서 GUI 를 통해 디버깅 가능한 상태이나, 향후 연구에서는 다중 프로세서의 디버깅을 위한 기능 설계가 이루어지고 있다. 이를 위하여 원격 디버거 내부에서 프로세스 처리 부분에 대한 연구가 이루어지고 있으며 멀티 태스크 지원 [12,13]을 위한 커널 모니터의 기능 보강에 대하여 논의 되고 있다.

참고문헌

[1] Hideyuki Tokuda and Makoto Kotera, “A Real-Time Tool Set for the ARTS Kernel,” *Proceedings of Real-Time Systems Symposium*, 1988.
[2] WindRiver, *Tornado User's Guide,* 1995.
[3] WindRiver, *Tornado API Guide 1.0.1*, 1997.
[4] Microtec, *Spectra Boot and VRTX Real-Time OS*, 1996.
[5] Eun-Hyang Lee, et al., “A Cross Debugging Architecture for Switching Software,” *Proceedings of International Conference on Communication Technology* (ICCT), 1996.
[6] YoungJoon Byun, et al., “High-Level CHILL Debugging System in Cross-Development Environments,” *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, 1998.
[7] 김흥남, ”사용자개발도구연구”,정보가전용 실시간 OS 컨퍼런스(RTOS’99) 자료집, pp.178-196, Nov. 17, 1999
[8] Intel, *StrongARM EBSA-285 Evaluation Board*, Oct. 1998.
[9] Andrew S. Tanenbaum, *Modern Operating Systems*, pp.84-85, Prentice-Hall International, 1992.
[10] Jack G. Ganssle, “Debuggers for Modern Embedded Systems,” *Embedded Systems Programming*, Nov. 1998.
[11] Eldad Maniv, “New Trends in Real-Time Software Debugging,” Real-Time Magazine 99-2 (http://www.realtime-info.com), pp.23-25, 1999.
[12] Jonathan B. Rosenberg, *How Debuggers Work*, John Wiley & Sons, 1996.
[13] T. Yasuda, K. Ueki, "A Debugging Technique Using Event History", *Proc. of the Conference on Real-Time Computing Systems and Applications*, pp. 137-141, 1994.