Q+P Esto의 원격 개발을 지원하는 타겟에이전트

임형택, 심현철, 손승우, 김흥남, 김채규
한국전자통신연구원, 컴퓨터소프트웨어기술연구소
인터넷정보가전연구부, 내장형SW연구팀
email : {htlim, shc63039, swson, hnkim, kyu}@etri.re.kr

Target Agent to Support Remote Development in Q+P Esto

Hyung-Taek Lim, Hyun-Chul Sim, Seung-Woo Son, Heung-Nam Kim, Chae-Kyu Kim
Embedded S/W Research Team
Dept. of Internet Appliance Technology
ETRI- Computer & Software Technology Laboratory

요 약

Q+P Esto는 정보가전용 RTOS인 Q+P를 위한 원격 개발 환경이다. 타겟에이전트는 타겟에서 실행되면서 호스트에 있는 디버거와 원격쉘, 그리고 자원모니터 같은 Q+P Esto 도구들이 원격 개발을 하는데 필요한 기본적인 기능들을 제공한다. 본 논문은 GNU gdb 5.0에 있는 gdbserver를 이용하여 타겟에서 실행 중인 응용 프로세스를 감시 및 제어하는 구조를 제안한다. 디버깅을 위하여 정지점(breakpoint)에 걸린 응용 프로세스를 continue 시켰을 때 응용 프로세스가 무한 루프에 빠지더라도 타겟에이전트는 다른 도구의 요청을 계속 처리할 수 있다. 응용 프로세스를 제어하는 자세한 메커니즘은 gdbserver가 담당한다. 따라서, 타겟에이전트는 단지 gdb remote protocol만 사용하여 gdbserver와 통신하면 되므로 구현이 용이하다.

1. 서론

Q+[4]는 한국전자통신연구원의 내장형SW연구팀에서 개발한 정보가전을 위한 내장형 운영체제로써 Q+T와 Q+P의 두 가지가 있다. Q+T는 VxWorks[3]처럼 단일한 메모리 공간에서 커널과 응용 태스크(또는 쓰레드)들이 동작하는 운영체제이다. Q+P는 리눅스 기반으로써 커널과 응용이 서로 다른 주소공간에서 실행되며 프로세스들 간의 주소공간이 분리되어 있는 운영체제이다.

Q+ Esto는 Q+ 환경에서 원격 교차 개발을 지원하는 통합 개발 도구이다[6][8][13]. 그림 1처럼 호스트의 Esto는 컴파일러, 타겟 설정자(target configurator), 원격쉘[15], 원격 모니터, 디버거[10][14] 등의 여러 가지 도구들로 구성된다. 이 도구들의 원격 개발을 위해서는 타겟에서 데몬 프로세스(또는 태스크)로 실행되는 프로그램이 필요하다. 그런데, 각 도구마다 이 프로그램이 따로 존재하는 경우 다음과 같은 단점이 있다.

- 각 도구의 원격 개발을 지원하는 타겟의 프로그램들은 서로 공통적인 기능들을 갖고 있다. 즉, 동일한 기능이 불필요하게 중복되어 전체적인 타겟의 크기를 증가시킨다.
- 호스트에 새로운 도구가 추가될 때마다 이 도구의 원격 개발을 타겟에서 지원하는 프로그램을 처음부터 작성해야 한다.

타겟에이전트(TA 또는 Target Agent)는 호스트의 여러 도구들이 필요로 하는 기능을 제공하는 단일한 프로그램이다. 공통적인 기능이 중복되지 않으므로 타겟의 크기를 줄일 수 있다. 호스트에 새로운 도구를 추가한 경우에도 타겟에이전트에 없는 새로운 기능만 추가하면 되므로 개발 기간을 단축할 수 있다.

호스트에는 다양한 도구들의 요청을 타겟에이전트에게 중개해주는 호스트에이전트(HA 또는 Host Agent)가 있다. 호스트에이전트는 도구들이 필요로 하는 기

본적인 인터페이스를 제공함으로써 새로운 도구를 Esto 환경에 추가하기 쉽고(확장성) 도구들이 타겟의 운영체제나 아키텍처에 영향을 받지 않게 한다(이식성). 즉, 다양한 도구들을 지원하여 호스트에서 개발한 프로그램을 타겟에 전송하여 실행시키거나, 타겟을 감시 및 제어할 수 있도록 호스트에는 호스트에이전트가 있고 타겟에는 타겟에이전트가 존재한다.

Q+T 커널에서 타겟에이전트[5][7][9][11][12]는 태스크로 동작하며 일부 기능은 훅 함수와 인터럽트 서비스 루틴으로 실행되는 반면 Q+P 커널에서는 데몬 프로세스로 동작한다. 본 논문에서는 Q+P 용 타겟에이전트의 구조를 설명하고 구현한다. 타겟에이전트가 제공해야 하는 기본적인 기능에 대하여 2장에서 소개하고, 이 기능들을 Q+P 커널에서 지원하기 위한 타겟에이전트의 구조를 3장에서 설명한다. 4장에서 타겟에이전트를 구현하고 실험한 결과를 소개한 후 5장에서 논문을 마친다.

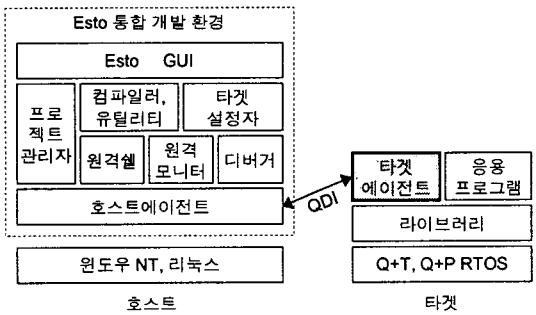

그림 1. Esto 통합 개발 환경의 구조

2. 타겟에이전트의 기능

타겟에이전트는 호스트에이전트에게 QDI(Q+ Debug Interface)를 제공한다. QDI는 표 1과 같이 원격 개발을 위하여 타겟을 제어 및 감시하는데 필요한 기본적인 인터페이스들로 구성된다.

표 1. QDI의 종류

프로토콜 이름	기능
QDI_TARGET_PING	타겟에이전트가 살아있는지 검사
QDI_TARGET_CONNECT	호스트에이전트와 타겟에이전트 간의 연결 설정
QDI_TARGET_DISCONNECT	연결을 끊음
QDI_MEM_READ	타겟의 메모리 읽기
QDI_MEM_WRITE	타겟의 메모리에 쓰기
QDI_CONTEXT_CREATE	타겟에 프로세스를 생성
QDI_CONTEXT_KILL	타겟의 프로세스를 종료시킴
QDI_CONTEXT_SUSPEND	타겟의 프로세스를 일시중지시킴
QDI_CONTEXT_RESUME	일시중지된 프로세스를 계속 실행시킴
QDI_REGS_GET	프로세스의 레지스터 읽기
QDI_REGS_SET	프로세스의 레지스터에 쓰기
QDI_EVENTPOINT_ADD	이벤트 등록
QDI_EVENTPOINT_DELETE	이벤트 제거
QDI_EVENT_GET	타겟에서 발생한 이벤트에 대한 정보를 알려줌
QDI_CONTEXT_CONT	정지점에 걸린 프로세스를 계속 실행시킴
QDI_CONTEXT_STEP	정지점에 걸린 프로세스를 한 스텝만 실행시킴
QDI_DIRECTORY_ENTRY	타겟의 디렉토리 정보를 알려줌
QDI_FILE_PUT	호스트에서 타겟으로 파일을 전송
QDI_FILE_REMOVE	타겟에 있는 파일을 제거

3. 타겟에이전트의 구조

Q+T에서 타겟에이전트는 타겟의 모든 메모리에 접근할 수 있으므로 수행 중인 태스크의 메모리나 레지스터에 직접 읽기 쓰기를 할 수 있다. 반면, Q+P에서 타겟에이전트는 프로세스이므로 다른 응용 프로세스의 메모리나 레지스터를 직접 읽거나 쓸 수 없다. 타겟에이전트가 응용 프로세스에 접근하기 위해서는 ptrace() 시스템콜을 이용해야 한다. ptrace()는 유닉스에서 한 프로세스가 다른 프로세스의 실행을 감시 및 제어할 수 있게 해주기 위하여 제공되며 주로 디버거에서 많이 사용된다[2].

gdbserver[1]는 타겟에서 ptrace()를 이용하여 응용 프로세스의 메모리나 레지스터에 접근하고 응용 프로세스를 감시 및 제어하는 작은 프로그램이며 gdb에 시리얼 통신이나 이더넷으로 연결되어 원격 디버깅을 지원한다.

3.1 절에서 타겟에이전트가 ptrace()를 직접 이용하는 구조의 문제점을 기술한 후 3.2 절에서 타겟에이전트가 gdbserver를 이용하는 구조를 제시한다.

3.1. ptrace()를 직접 이용하는 구조

그림 2처럼 타겟에이전트는 직접 ptrace() 시스템콜을 이용하여 응용 프로세스를 제어할 수 있다. 이 구조에서 응용 프로세스의 메모리나 레지스터를 읽거나 쓸 때에는 표 2처럼 ptrace()를 호출하면 된다.

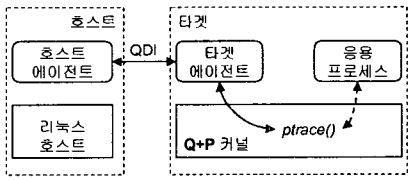

그림 2. 타겟에이전트가 직접 ptrace()를 이용하는 구조

표 2. ptrace()로 메모리와 레지스터에 읽고 쓰는 예

```
/* 프로세스 pid 의 메모리 addr 번지에서 32bit 읽기 */
buf = ptrace(PTRACE_PEEKTEXT, pid, addr, 0);
/* 프로세스 pid 의 메모리 addr 번지에 32bit 쓰기 */
ptrace(PTRACE_POKETEXT, pid, addr, buffer);
/* 프로세스 pid 의 레지스터 읽기 */
ptrace(PTRACE_PEEKUSER, pid, regaddr, 0);
/* 프로세스 pid 의 레지스터에 쓰기 */
ptrace(PTRACE_POKEUSER, pid, regaddr, regs);
```

정지점에 걸린 응용 프로세스를 한 명령어만 실행(step)시키거나 다음 정지점까지 계속 실행(continue)시키는 경우 타겟에이전트는 ptrace() 시스템콜을 호출한

후에 wait() 시스템콜을 호출하여 응용 프로세스가 멈출 때까지 대기해야 한다(표 3).

표 3. 정지점에 걸린 응용 프로세스를 continue시키는 기본 구조

```
/* 응용 프로세스를 continue 시킴 */
ptrace(PTRACE_CONT, pid, 1, signal);
/* 응용 프로세스가 멈출 때까지 대기 */
pid = wait(&status);
```

그림 3은 디버거가 응용 프로세스에게 continue 명령을 내렸을 때 호스트에이전트와 타겟에이전트, Q+P 커널, 그리고 응용 프로세스의 관계를 순차도처럼 표현한 것이다. 타겟에이전트의 QDI_CONTEXT_CONT 서비스가 ptrace()와 wait()를 차례로 호출한 후 응용 프로세스가 정지할 때까지 대기하고 있다. 만약, 응용 프로세스가 다음 정지점을 만날 때까지 오래 걸리거나 응용 프로세스에 버그가 있어서 무한루프에 빠진 경우 타겟에이전트는 호스트에이전트의 요청을 수행할 수 없게 된다. 그런데, 타겟에이전트는 호스트의 디버거뿐만 아니라 원격쉘이나 자원 모니터 등의 요청도 처리해야 하므로 디버거의 continue 요청을 수행하는 동안에도 다른 도구의 요청을 처리할 수 있어야 한다. 따라서, 이 구조는 타겟에이전트에 적합하지 않다.

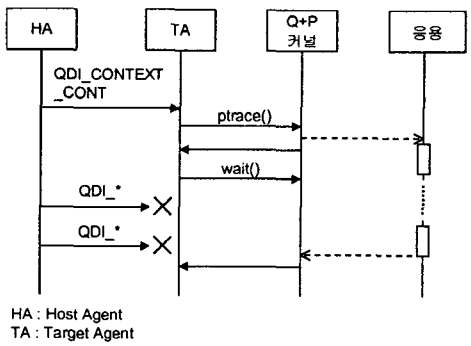

그림 3. 타겟에이전트가 직접 ptrace()로 응용 프로세스를 *continue* 시키는 경우

3.2. gdbserver를 이용하는 구조

3.1절에서 기술한 문제를 해결하기 위해서는 ptrace()와 wait()를 실행하는 별도의 프로세스나 쓰레드가 필요하다. 이에 본 논문은 타겟에이전트가 gdbserver를 이용하여 타겟에서 실행 중인 응용 프로세스를 제어 및 감시하는 구조를 제안한다.

그림 4처럼 호스트에이전트가 타겟에이전트의 QDI 서비스를 호출하면 타겟에이전트는 이 요청을 gdb remote protocol [1]로 변환한 후에 gdbserver에게 보낸다. gdbserver는 ptrace()를 이용하여 응용 프로세스를 적절히 제어하거나 정보를 알아낸 후 gdb remote protocol 형태로 타겟에이전트에게 보낸다. 타겟에이전트는 이 정보를 QDI 메시지 형태로 변환한 후에 호스트에이전트에게 리턴한다.

그림 5는 제안한 구조에서 정지점에 걸린 응용 프로세스를 continue 시킨 경우를 보여준다. 호스트에이전트가 타겟에이전트의 QDI_CONTEXT_CONT 서비스를 호출하면 타겟에이전트는 gdbserver에게 continue 명령("$c#63")을 보낸다. 타겟에이전트가 gdbserver로부터 ACK(+)를 받으면 QDI_CONTEXT_CONT 서비스를 종료하고 리턴한다. 그런 다음, 타겟에이전트는 select() 시스템콜을 사용하여 호스트에이전트의 요청(QDI_*)이나 응용 프로세스가 멈췄을 때 gdbserver가 보내는 결과("$T0508...")를 모두 받을 수 있는 상태로 대기한다. 따라서, 호스트에이전트로부터 QDI 요청이 오면 타겟에이전트는 이를 처리할 수 있다. continue 시킨 응용 프로세스가 정지점에 걸려서 멈추면 gdbserver는 응용 프로세스가 멈춘 상태의 프로그램 카운터 값, 스택포인터의 값 등을 gdb remote protocol 형태로 타겟에이전트에게 보내는데 타겟에이전트 이 값들을 호스트에이전트에게 보낸다. continue 시킨 응용 프로세스가 무한 루프에 빠지더라도 gdbserver만 블록되고 타겟에이전트는 계속해서 원격쉘이나 자원 모니터의 요청을 처리할 수 있다.

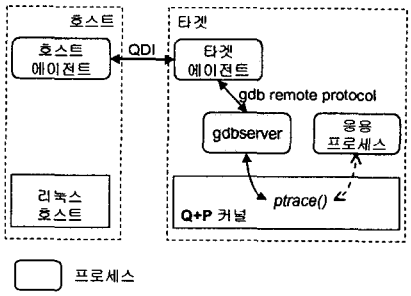

그림 4. 타겟에이전트가 직접 ptrace()로 응용 프로세스를 **continue** 시키는 경우

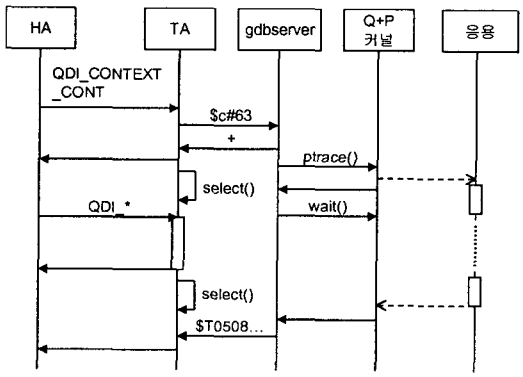

그림 5. gdbserver를 이용하는 구조에서 응용 프로세스를 **continue** 시킨 경우

메모리 읽기와 쓰기(QDI_MEM_{READ, WRITE}), 레지스터 읽기와 쓰기(QDI_REG_{READ, WRITE}), 이벤트 추가 및 삭제하기(QDI_EVENTPOINT_{ADD, DELETE}), 정지점에 걸린 프로세스를 계속 실행시키거나 한 문장씩 실행시키기(QDI_CONTEXT_{CONT, STEP})의 8 가지 서비스만 그림 4처럼 gdbserver를

이용한다. 그외의 QDI 서비스들은 gdbserver를 이용하지 않고 타겟에이전트 프로세스가 모두 처리한다.

4. 타겟에이전트의 구현 및 실험

타겟에이전트는 호스트에이전트와 RPC를 이용하여 통신하며 gdbserver와는 TCP 소켓 통신으로 통신한다. 타겟에이전트의 크기는 55 KB 미만이며, gdbserver는 16 KB 미만이다. gdbserver는 GNU gdb 5.0에 포함된 것을 이용하였으며 멀티쓰레드 프로그램의 디버깅은 지원하지 않는다.

호스트의 Q+P Esto는 레드햇 7.1에서 수행되며 타겟에는 Q+P가 설치되어 있다. 타겟에이전트를 실험하기 위하여 Esto의 디버거인 qpgdb와 원격쉘을 실행시키고 다양한 디버거 명령과 원격쉘 명령을 임의의 순서로 동시에 입력하였다.

타겟에이전트는 Q+P 운영체제가 부팅되면서 백그라운드로 실행된다. 사용자는 가장 먼저 호스트에이전트를 타겟에이전트에 연결해야 하는데, 이 때 QDI_TARGET_CONNECT 서비스가 호출된다. 그런 다음, 호스트의 도구들이 호스트에이전트에 연결됨으로써 도구들은 타겟을 제어하거나 감시할 수 있게 된다. QDI_TARGET_DISCONNECT 서비스가 호출되면 타겟에이전트와 호스트에이전트의 연결이 끊어진다.

개발 중인 응용 프로그램을 타겟에서 실행하거나 디버깅하기 위하여 프로그램을 타겟으로 전송할 경우 QDI_FILE_PUT 서비스가 사용된다.

QDI_CONTEXT_CREATE는 타겟 파일시스템에 전송된 응용 프로그램을 실행시킨다. 디버깅을 해야 하거나 응용 프로그램을 실행하면서 메모리나 레지스터에 접근해야 하는 경우에 타겟에이전트는 gdbserver 프로세스를 fork 하며 그 인자로 응용 프로그램을 전달한다. 그러면, 그림 4처럼 gdbserver는 타겟에이전트의 자식 프로세스가 되고 응용 프로그램은 gdbserver의 자식 프로세스가 된다. 그리고, 타겟에이전트는 gdbserver를 통해서 간접적으로 응용 프로세스를 제어할 수 있게 된다.

응용 프로그램의 메모리나 레지스터에 접근할 필요가 없고 suspend, resume, kill 정도의 제어만 필요한 경우에 타겟에이전트는 gdbserver 없이 바로 응용 프로세스를 fork 하여 자신의 자식 프로세스로 생성한다. 타겟에이전트는 각각 SIGSTOP, SIGCONT, SIGKILL 시그날을 이용하여 응용 프로세스를 suspend, resume, kill 한다.

5. 결론

본 논문은 Q+P Esto의 원격 개발 환경을 지원하기 위한 타겟에이전트의 기능을 설명하고 gdbserver를 이용하는 타겟에이전트의 구조를 제안하였다. gdbserver를 이용함으로써 타겟에이전트는 ptrace()를 사용하는 방법에 대해 고려할 필요가 없이 gdb remote protocol에 맞는 명령만 보내면 되므로 구현이 용이하다.

향후 연구과제로는 타겟에이전트의 성능 개선과 여러 응용 프로세스를 동시에 디버깅하는 기능, 그리고 멀티쓰레드 프로그램의 지원이 있다. 타겟에이전트를 이용하여 디버깅하는 경우가 gdb와 gdbserver를 직접 연결하여 디버깅하는 경우보다 느리므로 성능 개선이 필요하다.

gdb는 한 번에 하나의 프로세스만 디버깅할 수 있다. 따라서, 타겟에서 여러 프로세스를 디버깅하기 위해서는 호스트에 여러 gdb가 실행되어야 하며 타겟에이전트는 서로 다른 gdb의 디버깅 요청과 응답을 구분하여 관리할 수 있는 기능이 필요하다.

타겟에이전트가 사용하고 있는 gdbserver(gdb 5.0 기준)는 아직 멀티쓰레드 디버깅을 제대로 지원하지 못하고 있다. gdbserver에 멀티쓰레드 디버깅 기능을 추가하여 타겟에이전트가 멀티쓰레드 프로그램을 지원하도록 확장할 예정이다.

6. 참고문헌

[1] R. Stallman, R. Pesh, S. Shebs, et al., 'Debugging with GDB: The GNU Source-Level Debugger,' 8th Ed., Mar. 2000.

[2] D. P. Bovet and M. Cesati, 'Understanding the LINUX Kernel,' O'Reilly, 2001.

[3] WindRiver, "VxWorks 5.4", http://www.windriver.com

[4] 김선자, 김흥남, 김채규, "인터넷 정보가전용 RTOS 기술 현황", 한국정보과학회지, 2001. 4.

[5] 공기석 외 3인, "내장형 실시간 소프트웨어의 원격디버깅을 위한 디버그에이전트의 설계 및 구현", 한국정보과학회 추계학술발표회, 1999.10.

[6] 임채덕 외 2인, "내장형 실시간 응용 개발을 위한 도구에서 타겟서버의 구현", 한국정보과학회 추계학술발표회, 1999.10.

[7] 손승우 외 3인, "교차 개발 환경에서 원격 디버깅을 위한 디버깅 프로토콜", 정보처리학회 추계학술발표회, 1999.10.

[8] C.D. Lim, etc., "A Tool Broker in Remote Development Environments for Embedded Applications," IASTED 2000, Feb, 2000.

[9] S.W. Son, etc., "Debugging Protocol for Remote Cross Development Environment," RTCSA 2000, Dec. 2000.

[10] K.Y. Lee, etc., "A Desgin and Implementation of a Remote Debugging Environment for Embedded Internet Software," ACM SIGPLAN 2000 Workshop on LCTES, Jun. 2000.

[11] K.S. Kong, etc., "A Design and Implementation of Debug Agent for Remote Debugging of Embedded Real-time Software," IASTED AI 2000, Feb. 2000.

[12] 공기석 외 2인, "원격 실시간 개발 환경에서 디버그 에이전트 설계 및 구현", 정보처리학회 춘계학술대회, 2000.4.

[13] 임채덕 외 2인, "내장형 소프트웨어의 원격 개발을 위한 Q+용 타겟관리자의 개발", 정보처리학회 춘계학술대회, 2000.4

[14] 이광용 외 3인, "정보가전용 내장형 소프트웨어 개발을 위한 원격 디버거의 설계 및 구현", 정보처리학회 춘계학술발표회, 2000.4.

[15] 김흥남 외 3인, "RTOS 용 원격 대화형쉘 개발에 관한 연구", 정보처리학회 추계학술발표회, 2000. 10.