내장형 소프트웨어의 원격 개발을 위한 Q+용 타겟관리자의 개발

임채덕O, 이우진, 손승우, 김흥남
한국전자통신연구원 컴퓨터.소프트웨어기술연구소
{cdlim, woojin, swson, hnkim}@etri.re.kr

A Target Manager for Remote Developments of Q+ Embedded Applications

Chaedeok LimO, Woojin Lee, Seungwoo Son, Heung-Nam Kim
ETRI-Computer & Software Technology Laboratory

요 약

호스트와 타겟 간의 통신 부담과 타겟 자원이 제한적이라는 문제를 해결하기 위해서, 호스트 중심 원격 개발 환경(Remote Development Environment: RDE)을 구성하였다. 이 원격 개발 환경은 여러 개발 도구로 구성되어 있는데, 이들 도구가 타겟시스템에 접속하여 내장형 응용를 개발하는데 있어서 공통 기능, 공통 데이터가 존재한다. 그런데 이들을 각각 유지시키는 것은 타겟시스템에 부담을 줄 뿐만 아니라, 개발 도구를 추가/구현하는 것도 매우 어렵게 한다. 이 문제를 해결하기 위해 원격 개발 도구들의 공통 요소들을 모아 도구와 타겟시스템 사이에 중개자 역할을 하는 타겟관리자를 둔다. 타겟관리자는 미들웨어로서 호스트와 타겟 간의 통신 채널을 하나로 유지하면서 도구와 타겟 간의 통신을 중재하고, 도구들이 심볼 테이블을 공유할 수 있도록 심볼 테이블을 관리한다. 또한, 타겟에 있는 도구 전용 메모리를 관리하며, 호스트 상에서 개발한 내장형 소프트웨어를 타겟에 로딩하는 일을 처리한다. 이러한 타겟관리자를 사용하는 원격 개발 환경은 도구들에게 공통 인터페이스를 제공하여, 통신 방식 등의 하부 구조에 상관없이 서비스를 받을 수 있고, 새로운 도구를 추가하는 것도 용이하게 할 수 있다는 장점을 갖게 된다. 본 논문에서는 ETRI 에서 개발 중인 실시간 운영 체제인 Q+용 타겟관리자를 설계하고 구현한다. 또한, 타겟관리자가 동작하게 될 내장형 실시간 응용 개발 환경에 대하여 소개하고, 구현 결과를 도구들과 연계하여 보여주며, 타겟 관리자를 둔 원격 개발 환경이 타겟 관리자를 두지 않은 경우에 비해 호스트와 타겟 간의 통신 횟수가 얼마나 감소하는지 시험 결과를 통해 보여준다. 현재 타겟 관리자의 프로토타입을 개발하여 도구들과 통합 시험을 하였는데 기본 기능들이 성공적으로 수행됨을 확인하였다.

1. 서론

최근 새로이 저가형이면서 특정 응용 전용인 32 비트 내장형 마이크로프로세서의 출현과 통신 단말기나 가전 제품에 다양한 기능을 추가하기 위한 내장형 시스템[1] 응용 개발 수요가 증가하고 있다. 더구나 내장형 응용들이 점점 복잡해지고 있으며 이를 위한 개발 환경[2]은 타겟시스템의 자원이 매우 제한적이며 호스트와 타겟 간의 통신 부담이 크다는 점에서 기존의 소프트웨어 개발 환경과 다르다.

호스트와 타겟 간의 통신 부담과 타겟 자원이 제한적이라는 이 문제를 해결하기 위해서, 개발 환경에 있어서 기존의 타겟에 의존적인 개발 도구들을 호스트로 이동시켜 원격 개발 환경을 구성하였다. 이 원격 개발 환경은 대화형 쉘, 원격 디버거, 자원 모니터 등 여러 개발 도구로 구성될 수 있는데, 이들 도구가 타겟시스템에 접속하여 내장형 실시간 소프트웨어를 개발하는데 있어서 공통 기능, 공통 데이터가 존재한다. 이들을 각각 유지시키는 것은 타겟시스템에 부담을 줄 뿐만 아니라, 개발 도구를 구현하는 것도 매우 어렵게 한다.

이 문제를 해결하기 위해 원격 개발 도구들의 공통 요소들을 모아 도구와 타겟시스템 사이에 중개자 역할을 하는 타겟관리자가 필요하게 된다. 타겟관리자는 미들웨어로서 호스트와 타겟 간의 통신 채널을 하나로 유지하면서 도구와 타겟 간의 통신을 중재하고, 도구들이 심볼 테이블을 공유할 수 있도록 심볼 테이블을 관리한다. 또한, 타겟에 있는 도구 전용 메모리를 관리하며, 호스트 상에서 개발한 내장형 소프트웨어를 타겟에 로딩하는 일을 처리한다. 이러한 타겟관리자를 사용하는 원격 개발 환경은 도구들에게 공통 인터페이스를 제공하여, 통신 방식 등의 하부 구조에 상관없이 서비스를 받을 수 있고, 새로운 도구를 추가하는 것도 용이하게 할 수 있다는 장점을 갖게 된다.

본 논문에서는 이러한 미들웨어인 타겟관리자를 설계하고 구현한다. 또한, 타겟관리자가 동작하게 될 원격 개발 환경에 대하여 소개하고, 구현 결과를 도구들과 연계하여 보여주며, 타겟 관리자를 둔 원격 개발 환경이 타겟 관리자를 두지 않은 경우에 비해 호스트와 타겟 간의 통신 횟수가 얼마나 감소하는지 시험 결과를 통해 보여준다. 현재 타겟관리자의 프로토타입을 개발하여 도구들과 통합 시험을 하였는데 기본 기능들이 성공적으로 수행됨을 확인하였다.

본 논문의 구성은 다음과 같다. 본 장에 이어서 제 2 장에서는 타겟관리자가 동작하게 될 내장형 소프트웨어 원격 개발 환경에 대하여 소개한다. 제 3 장에서는 타겟관리자의 구조 및 기능에 대해 기술하고, 제 4 장에서는 구현 결과를 화면으로 보여 주고, 타겟관리자가 있는 원격 개발 환경과 타겟관리자가 없는 원격 개발 환경을 호스트와 타겟 간의 통신 횟수를 분석하여 타겟관리자의 성능을 보여주고, 마지막 장에서 결론을 맺는다.

2. 내장형 소프트웨어의 원격 개발 환경

원격 개발 환경은 통합 개발 환경 혹은 교차 개발 환경과 함께 내장형 소프트웨어를 개발하기 위한 개발 환경이다. 통합 개발 환경은 여러 개발 도구를 타겟관리자의 인터페이스를 통해서만 타겟에 접근이 가능하다는 측면을 강조해서 사용한 용어이다. 또한, 교차 개발 환경은 이기종 프로세서에서 동작하는 목적 코드를 생성하는 데 주로 초점을 맞춰서 사용하는 용어이다. 반면, 원격 개발 환경은 타겟시스템이 이기종이든 아니든 원격 호스트에서 통신을 사용하여 타겟 시스템에서 동작할 응용을 개발할 수 있는 것과 경우에 따라서 타겟관리자를 통하지 않는 도구도 허용한다는 차원에서 본 논문에서는 원격 개발 환경이라는 용어를 사용한다. 그런 의미에서 통합 개발 환경은 원격 개발 환경의 일부라고도 할 수 있다.

원격 개발 환경의 일반적인 구성은 그림 1 과 같이 호스트, 타겟, 그들 사이의 통신 라인으로 구성된다. 호스트는 응용 개발자가 주로 작업하는 환경으로 각종 개발 도구들이 있는 최소한 메인 메모리가 64 Mbyte 이상인 자원이 풍부한 컴퓨터이다. 타겟은 매우 제한된 자원만을 가지고 있는 셋탑 박스와 같은 작은 컴퓨터로 실시간 운영체제가 동작하고 있으며, 그 위에 응용 개발자가 개발하는 소프트웨어가 탑재되어 호스트와 통신을 통하여 최종적인 모습의 소프트웨어를 개발하게 된다. 호스트와 타겟 간의 통신 방식은 시리얼, 이더넷 두 방식을 모두 사용하는 것이 보통이다. 본 논문에서 구현한 타겟관리자도 두 통신 방식을 지원한다. 이는 타겟 보드는 실험실에, 호스트 컴퓨터는 연구실에 두고 개발 행위가 행해질 수 있도록 하는 장점을 제공한다.

내장형 소프트웨어를 호스트 상에서 개발하기 위해서 종래에는 호스트 상에 디버거 클라이언트를 두고 타겟시스템에 디버거 서버를 둔 GDB 와 같은 원격 디버거를 사용하였으나, 내장형 소프트웨어가 멀티미디어 응용과 같이 점점 복잡해짐에 따라 GDB 와 같은 디버거 뿐만 아니라 타겟 시스템 상태를 모니터링해 주는 도구 등 다양한 도구를 호스트 상에 둘 필요성이 제기되었다. 이 필요성을 만족시켜 주기 위해서 원격 개발 환경을 호스트 상에 타겟관리자와 같은 미들웨어를 두어 호스트와 타겟 간의 통신 부담을 줄이며, 여러 도구들은 타겟관리자를 통해 타겟에 연결하여 내장형 소프트웨어를 개발할 수 있는 형태로 구성한다.

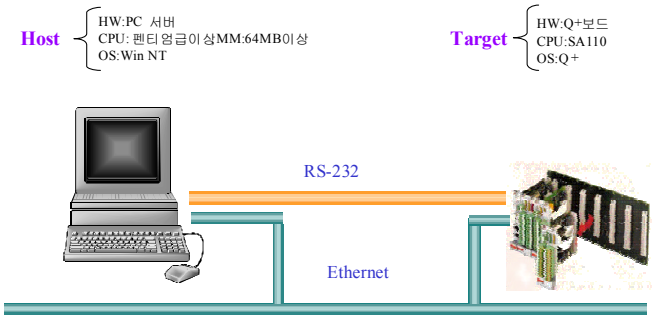

그림 1. 원격 개발 환경의 구성

2.1. 기존의 도구를 활용한 원격 개발 환경

기존의 도구를 이용하여 원격 개발 환경을 구성할 경우에 호스트 상의 도구들이 내장형 소프트웨어를 개발할 수 있도록 지원하기 위한 구조는 그림 2 와 같다.

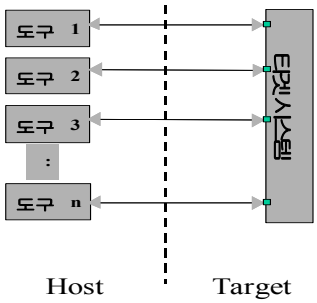

그림 2. 기존의 도구를 활용한 원격 개발 환경

그림 2 는 각각의 도구가 타겟과 연결하여 타겟시스템의 자원에 접근하기 위해서는 타겟시스템에 각 도구에 대응되는 서비스 데몬이 존재해야 한다.[13] 이는 각 도구가 동시에 접근할 경우 타겟시스템의 통신 부담이 매우 커지는 문제점이 있다. 그리고, 도구

를 사용하는 응용 개발자가 타겟시스템에 심각한 오류를 발생시켰을 경우에, 다른 도구를 사용하는 개발자도 타겟시스템을 사용할 수 없게 된다. 서로 다른 도구일 지라도 타겟시스템의 심볼 테이블과 같이 중복된 데이터를 각 도구가 가지고 있어야 하고, 경우에 따라서는 중복 데이터에 대한 일관성도 문제가 될 수 있다. 또한 호스트 상에 새로운 도구를 추가할 경우에도 통신 방식, 타겟 시스템에 대해 상세히 파악해야 하므로 도구 개발자에게 큰 부담을 줄 수 있다.

2.2. 도구 중개자를 활용한 원격 개발 환경

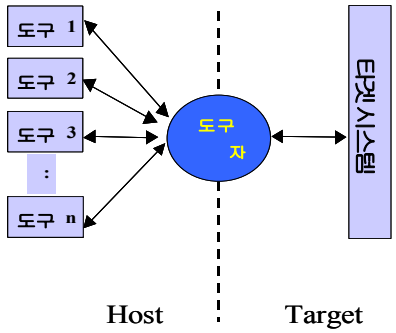

그림 3. 도구중개자를 활용한 원격 개발 환경 구조

제 2 절의 원격 개발 환경과는 달리 그림 3 의 경우는 타겟시스템과 호스트 상의 도구들 사이에 중개자 역할을 하는 도구 중개자를 두는 구조이다. 이 중개자는 호스트와 타겟의 통신을 지원하는데 있어서 호스트와 타겟 간의 통신 채널을 하나로 유지하면서 중재하면서, 도구들이 심볼 테이블을 공유할 수 있도록 심볼 테이블을 관리하고, 타겟에 있는 도구 전용 메모리를 관리하는 등의 일을 처리 한다. 통신 문제와 타겟 자원을 효율적으로 사용할 수 있는 구조는 그림 2 의 구조보다 그림 3 의 구조라고 판단되어 본 논문 연구에서는 도구 중개자(이를 '타겟관리자'라고 명함)를 설계 및 구현하여 이를 활용한 원격 개발 환경을 구성한다.

2.3. 목표 원격 개발 환경

그림 4 는 본 논문의 타겟관리자가 동작되는 내장형 소프트웨어를 개발하기 위한 목표 원격 개발 환경이다.

RDE 메인 프레임은 각 도구들을 제어하기 위한 Control View 를 제공하기 위한 것이며, 크로스 컴파일러, 원격 디버거, 대화형 쉘, 자원 모니터, 타겟관리자 설정 및 관리 창을 선택하기 위한 메뉴 바를 가지고 있다.

크로스 컴파일러는 GNU C 언어 컴파일러를 기본으로 하고 있으며, gcc, make, ld, as 등과 같은 바이너리 유틸리티도 지원한다.

원격 디버거는 GNU 소스 레벨 디버거를 확장한 버전이다. 호스트 상에서 원격 디버거를 이용하여 개발자는 타겟의 실시간 시스템의 태스크를 생성하거나 디버깅할 수 있다. [11, 12, 14, 16]

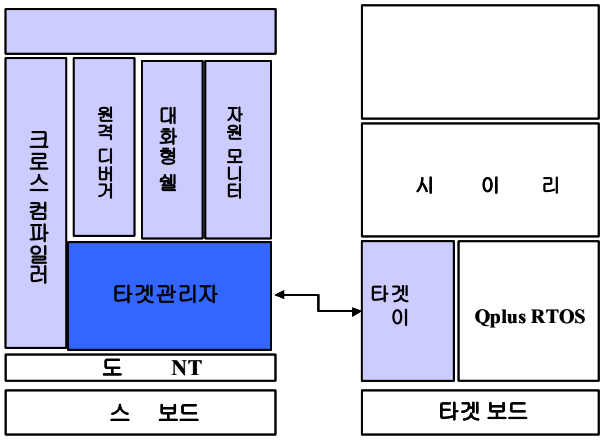

그림 4. 목표 원격 개발 환경

대화형 쉘은 호스트 상에서 타겟 시스템의 모든 실행 시간 유틸리티들을 제공 받을 수 있는 도구이다. 즉, 타겟의 실행 시간 시스템 함수를 호출할 수 있고, 응용 프로그램 실행 모듈을 호출할 수 있으며, 응용 프로그램의 변수 값을 알아보거나 입력할 수 있으며, 새로운 변수를 생성하거나, 메모리에 있는 내용을 확인하고 수정하거나, C 연산자를 이용 계산을 할 수 있다. 또한 필수적인 디버깅 기능을 사용할 수 있다.

자원 모니터는 GUI 를 통해 시스템 정보를 제공해 준다. 메모리 할당 정보, 태스크, 메시지 큐, 세마포어 등 모든 시스템 오브젝트 정보를 쉽게 파악할 수 있다. 이들 정보는 정기적 혹은 개발자 요구에 따라 갱신된다. [18]

타겟 에이전트는 타겟관리자와 함께 개발자가 통신 방식과 타겟 자원에 독립적일 수 있게 하는 RDE 의 주요 구성 요소이다. 호스트 상의 모든 개발 도구들은 호스트 상의 타겟관리자와, 타겟에 있는 타겟 에이전트를 통해서만 정보를 주고 받을 수 있다.[15, 17, 20]

타겟관리자는 타겟과 도구들 사이에서 중개자 역할을 담당한다. 즉, 도구들은 타겟관리자가 제공하는 인터페이스를 통해 타겟시스템에 다양한 요구를 하게 된다. 타겟관리자는 다양한 요구를 해석하여 적절한 처리를 수행한다. 타겟관리자가 타겟과 접속하기 위해서 타겟 에이전트와 연결하여야 하므로 이들 간에도 인터페이스가 존재한다. 이 인터페이스는 통신 방식에 무관하다.[19]

타겟 시스템에는 실시간 운영체제인 Qplus 위에 개발하고자 하는 내장형 응용 프로그램의 개발 버전이 탑재되어 있다.

이미 도구중개자 개념이 들어가 있는 상용 원격 개발 환경 제품[3,4,5,6,7]들이 많지만 이들 모두 하나의 운영체제에 국한하여 사용할 수 있는 도구이므로 운영체제를 독자적으로 개발할 시에는 반드시 원격 개발 환경도 더불어 자체 개발해야 하는 것이 현실이다. 운영체제도 상용 제품을 사용할 수 있으나, 이를 사용할 경우 비싼 로열티를 지불해야 함은 물론이고, 내장형 소프트웨어의 경우 운영체제의 소스를 직접 수정해야만 되는 경우도 빈번하여 우리 나라 자체 기술을 보유하기 위해서도 본 논문의 타겟관리자에 대한 연

구는 필요하다.

3. 타겟 서버의 구조

3.1. 타겟관리자의 설계 방향

타겟관리자는 그림 4 에서 보여 주듯이 호스트 상에 위치하면서 타겟의 타겟 에이젼트와 연결하여 응용을 개발함에 있어서 타겟과 통신 및 타겟 자원의 일부를 관리하는 역할을 한다. 타겟관리자의 설계 원칙은 다음과 같다.

- 타겟 접근의 최소화
- 하나의 타겟에 하나의 통신 채널만 존재
- 도구 확장의 간소화

타겟 접근을 최소화 하기 위해 타겟의 심볼 테이블, 도구가 사용하는 타겟 메모리, 응용 프로그램의 실행 모듈에 대한 관리를 호스트 상에서 할 수 있도록 한다. 하나의 타겟에 하나의 통신 채널 만을 유지하는 것은 여러 개의 서로 다른 도구들이 하나의 타겟을 공유하게 하도록 하기 위함이다. 이미 상용화된 도구나 새로운 도구를 RDE 에 추가하고자 할 경우에 쉽게 확장할 수 있도록 타겟관리자는 Open API 를 제공하고 있다.

3.2. 타겟관리자의 구조 설계

타겟관리자는 호스트 상에서 동작한다. 하나의 타겟에 대하여 하나의 타겟관리자가 동작하고 있어야 한다. 호스트 상의 모든 도구는 타겟관리자를 통해 타겟에 접근하고, 타겟관리자는 이들 도구의 요구를 충족시켜줄 책임을 진다. 타겟관리자는 타겟과 통신하는 것과 관련한 자세한 사항을 모두 관리하며 도구들에게는 투명성을 제공한다. 즉, 도구들은 호스트-타겟 사이의 통신 관련된 사항에 대해 자세히 알 필요가 없도록 하고자 한다. 타겟관리자가 제공 해야 될 기능을 요약하면 다음과 같다.

- GUI 를 통한 타겟관리자 설정 및 구동
- 도구에 Open API 제공
- Object Loading/Unloading
- Symbol Table 관리
- Target 의 Host Pool 관리
- 통신 방식별 타겟과 통신 가능
- 에러, 경고 로깅 기능 호출

타겟관리자는 그림 5 에서 볼 수 있듯이 QTI(Qplus Tool Interface) Protocol, Symbol Table Manager, Object Module Manager, Host-pool Memory Manager, Communication Back-End Manager, QDI(Qplus Debugging Interface) Protocol 로 구성된다. 각 모듈별 설명은 다음과 같다.

- QTI Protocol: 도구들은 반드시 이 프로토콜을 사용하여야만 타겟관리자에 요구할 수 있다. 타겟관리자의 시작과 종료, 실행 시간 타겟 시스템 정보, 이벤트 관리, 디버깅 지원, 실행 모듈 관리, 심볼 관리, 도구가 사용하는 타겟 메모리 관리 등에 관한 기능을 가지고 있다. 표 1 은 인터페이스 프로토콜을 기능에 따라 분류하여 테이블로 나열하면 표 1 과 같다.

기능별 분류	도구 이스	메시지 번호
세션 관리	TOOL_ATTACH	1
	TOOL_DETACH	2
	TS_INFO_GET	3
	TARGET_RESET	4
	INFO_GET	5
	INFO_Q_GET	6
타겟 정보 및 타겟 오퍼레이션	FUNC_CALL	7
	SERVICE_ADD	8
	AGENT_MODE_SET	9
	AGENT_MODE_GET	10
	DIRECT_CALL	11
심볼 관리	SYM_LIST_GET	12
	SYM_TBL_INFO_GET	13
	SYM_FIND	14
	SYM_ADD	15
	SYM_REMOVE	16
실행 파일 로딩 관리	OBJ_MODULE_LOAD	17
	OBJ_MODULE_UNLOAD	18
	OBJ_MODULE_LIST	19
	OBJ_MODULE_INFO_GET	20
	OBJ_MODULE_FIND	21
메모리 관련	MEM_CHECKSUM	28
	MEM_READ	29
	MEM_WRITE	30
	MEM_SET	31
	MEM_SCAN	32
	MEM_MOVE	33
	MEM_ALLOC	34
	MEM_FREE	35
	MEM_INFO_GET	36
	MEM_ALIGN	37
	MEM_REALLOC	38
	MEM_ADD_TO_POOL	39
	REGS_GET	40
	REGS_SET	41
콘텍스트 관리 및 디버깅	CONTEXT_CREATE	42
	CONTEXT_KILL	43
	CONTEXT_SUSPEND	44
	CONTEXT_CONT	45
	CONTEXT_RESUME	46
	CONTEXT_STEP	47
이벤트 관리	EVENTPOINT_ADD	48
	EVENTPOINT_DELETE	49
	EVENTPOINT_LIST	50
	EVENT_GETEGISTER_FOR_EVENT	51
	UN_REGIS_FOR_EVENT	52
	EVENT_ADD	53
		54

표 2. QTI Protocol 의 기능별 분류

- Symbol Table Manager: 타겟의 시스템 심볼 테이블과 타겟에 로딩된 모든 실행 모듈들의 서브루틴과 변수, 모듈 id 에 대한 정보를 관리한다. 원격 디버거와 대화형 쉘과 같은 도구들이 서브루틴을 호출하는 것은 심볼 테이블 관리자를 통해서 가능하다. 특히, 로드된 모듈에 대한 심볼릭 디스어셈블(disassembly) 및 특정 태스크의 서브루틴 호출을 심볼릭하게 추적할 필요가 있는 소스 레벨 원격 디버깅에 유용하다.
- Object Module Manager: 실행 모듈 관리자는 크게 호스트에 있는 실행 파일을 타겟에 로딩/언로딩하는 기능, 로딩된 모듈 리스트를 호스트 상에서 관리하는 기능을 지원 한다. 도구가 요구하는 가장 핵심적인 기능이다. 로더는 로드되고 있는 미정의된 참조를 정의하기 위해 심볼 테이블을 사용한다. 또한 타겟 시스템에 로드된 모듈을 언로드시킬 수 있어야 한다. 하나의 모듈이 언로드될 때 관련된 모든 내용이 심볼 테이블에서 제거되어야 한다. 실행 모듈은 운영 체제에 따라 다양할 수 있다. 로더는 COFF, ELF, a.out 과 같은 여러 형식을 지원하기에 용이한 DLL 구조를 갖고 있다. 타겟관리자 구동 시에 해당 파일 형식을 지원해주는 DLL 이 타겟관리자 데몬에 연결된다.
- Host-pool Memory Manager: 타겟 메모리 관리자

는 도구들의 메모리 할당/해제에 대한 요구를 충족시키기 위해 필요하다. 예를 들어, Object Module Manager 의 로더는 각 도구가 동작 중일 때 실행 모듈 로드를 하기 위해 메모리를 요구하게 된다. 이와 같은 요구를 만족시켜 주기 위해 타겟시스템의 호스트 도구 전용 메모리가 필요하다. 실시간 운영체제 구동 시에 도구들이 사용할 수 있는 타겟 메모리 크기를 운영체제 설정 파일에 세팅할 수 있고, 타겟 에이젼트와 연결이 되면 그 메모리에 대한 정보, 즉 시작 주소 및 크기를 얻어와서, 타겟 메모리 관리자는 처음에는 그 크기만 관리하다가 다 쓰게 되면 타겟의 운영체제에게 추가 요청하여 관리해 준다. 즉, 비어 있는 블록 리스트나 블록 헤더와 같은 정보를 호스트 상에 유지하고 있다가 타겟 메모리 할당에 최소한의 시간을 사용하게 한다. 타겟 메모리의 일정 영역은 캐싱하기에 좋은 후보가 될 수 있다. 당연치로 타겟 메모리 캐쉬는 모든 타겟 상주 모듈의 프로그램 텍스트 부분이 된다. 그 부분은 수정하는 것이 예외 사항 추가 등을 제외하면 매우 이례적인 일이기 때문에 그렇게 하는데, 이는 원격 개발 시 호스트와 타겟 간의 통신 횟수를 줄이는 데 고무적인 역할을 한다.

- Communication Back-End Manager: 이 모듈은 벡엔드 관리자, 이더넷, 시리얼과 같은 통신을 지원해주는 각각의 벡엔드들로 이루어져 있다. 벡엔드 관리자는 도구들이 타겟과 연결에 독립적이도록 해주는데, 타겟과 어떤 통신 방식을 사용하는지, 예를 들어 시리얼인지, 이더넷인지에 상관없이 동작하도록 하기 위함이다. 타겟 관리자를 동작시킬 때 벡엔드를 선택할 때 하나의 벡엔드를 링크시켜 주는 역할을 한다. 시리얼 벡엔드는 시리얼 통신의 경우에, 이더넷 벡엔드는 이더넷 통신의 경우에 동적으로 연결되어 통신 경로를 주선한다. 타겟관리자 벡엔드는 타겟 에이젼트로 해당 인터페이스에 맞게 요구를 하게 된다. 이 부분은 벡엔드에 독립적인 서브루틴 콜로써 인터페이스 한다.
- QDI(Qplus Debugging Interface) Protocol: 타겟 에이젼트와 통신하기 위해 필요하다. 이 프로토콜은 이더넷, 시리얼 통신에 공통적으로 적용된다. 이 프로토콜은 타겟 에이젼트가 서비스 기능을 가지고 있고, 타겟관리자는 호출만을 하므로 구현 시 부담은 작다.

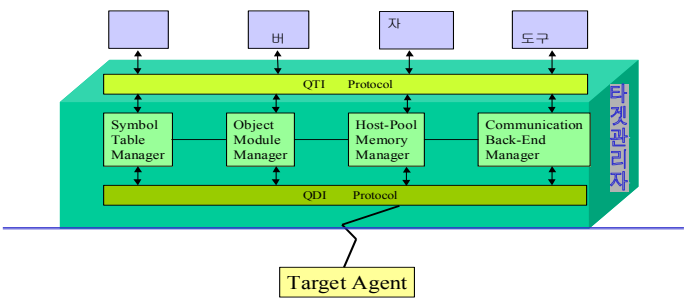

그림 5. 타겟관리자의 구조

4. 타겟관리자의 구현

타겟관리자의 구현 환경은 원격 개발 환경의 호스트와 같다. 즉, Winodws/NT 가 탑재된 펜티엄급 PC 에서 구현하였으며, 타겟과 통신을 위해서 호스트 상에서 SUN/ONCRPC-Compliant RPC[9]를 포팅하여 사용하였다. 프로그래밍 언어는 GUI 를 위해서는 Visual C++를 사용하고, 나머지 기능은 모두 Visual C 언어로 구현하였다. 타겟관리자는 여러 개의 DLL 로 이루어진다. 하나의 도구는 동시에 여러 타겟관리자를 연결할 수 없고, 같은 도구라도 각각 서로 다른 타겟관리자를 연결할 수는 있다. 실행 파일 형식은 ELF, COFF 를 지원한다. 통신 방식은 시리얼과 이더넷을 지원한다.

4.1. 타겟관리자의 구현 결과 화면

본 절에서는 타겟관리자를 구현한 결과를 화면으로 보여줄 수 있는 기능 중심으로 설명한다.

그림 6 은 타겟관리자를 하나의 서비스 데몬으로 구동시키기 위해 각종 플래그들을 설정하는 화면이다. 사용자에게 편의성을 제공하기 위해서 그림 6 의 하단의 코맨드들을 일일이 타입핑하지 않고도 타겟관리자를 설정할 수 있도록 하고 있다. 그림 6 에서 설정한 통신 방식은 이더넷 방식이다. 타겟관리자 데몬이 구동되면 호스트 상의 여러 도구들은 타겟관리자를 통해 타겟에 탑재될 내장형 응용 프로그램을 개발할 수 있다.

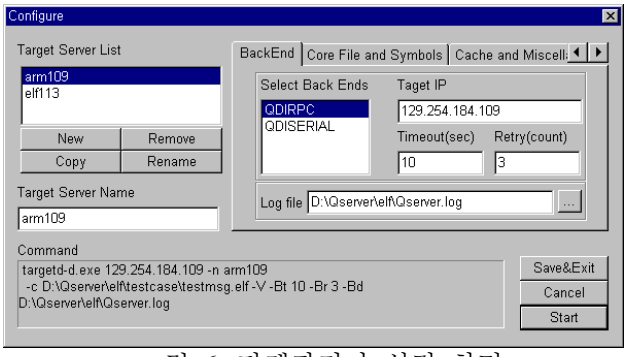

그림 6. 타겟관리자 설정 화면

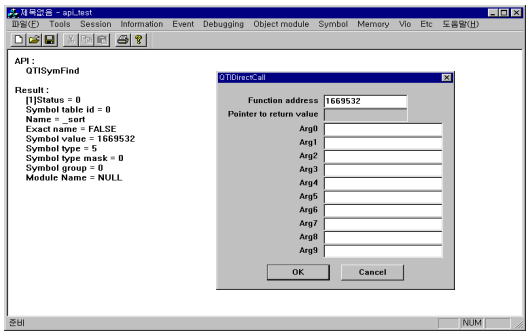

그림 7. 타겟관리자의 시험 화면

그림 7 은 타겟관리자의 시험 화면이다. 이 화면은 도구로부터 불려지는 QTI Protocol 을 구현한 C API 를 각각 시험하기 위해 구현한 결과를 보여주기 위한 예

이다. 이 예는 하나의 OMF 모듈을 로딩하여, 그것의 타겟시스템에서의 모듈 심볼의 주소를 구하여(QTISymFind) 실행(QTIDirectCall)하는 것이다. 이 시험 모듈은 도구와 독립적으로 시험 가능하며 이 시험 모듈이 타겟관리자의 하나의 클라이언트처럼 동작한다는 점에서 도구와의 연결 시험까지 할 수 있다는 장점이 있다.

4.2. 타겟관리자의 성능 분석

본 논문에서는 구현한 타겟관리자가 제대로 기능을 수행하는 지, 제대로 수행한다면 성능은 어느 정도인지 분석하였다. 타겟관리자의 성능 분석은 호스트와 타겟 간의 통신 횟수를 기준으로 수행하였다. 호스트-타겟 간의 통신 횟수를 계산하는 수식을 정리하면 다음과 같다.

$N_{HT} = SUM_{op} [N_{tools} * N_{op}]$ ----------(수식 1)

N_{HT}: 호스트-타겟 간 통신 횟수
N_{tools}: 도구의 수
N_{op}: 타겟 오퍼레이션의 수

(수식 1)의 의미는 다음과 같다. 사용자가 도구의 특정 명령을 내리면, 여러 기능들, 즉 여러 QTI 프로토콜들이 도구로부터 호출되는데, 이들의 횟수를 모두 더하는 것이 호스트-타겟 간의 통신 횟수가 되는 것이다. 같은 기능을 여러 도구에서 사용하면 도구 수만큼 곱하면 된다.

표 3 은 타겟관리자에게 가장 많은 호스트-타겟 오퍼레이션을 요구하는 OMF 모듈 로딩에 대해 실험한 결과를 타겟관리자를 사용한 경우와 그렇지 않은 경우에 대하여 비교한 데이터이다. Testmsg.elf 는 코아 파일의 예이고, 나머지는 재배치 모듈의 예이다. 표 3 에서 볼 수 있듯이, 타겟관리자가 있는 경우 임의의 기능을 수행하기 위해서 도구의 수에 비례하지 않고 호스트 타겟 간의 통신은 거의 일정하다는 것을 알 수 있다.

표 2. 타겟관리자의 성능 분석

		타겟관리자	타겟관리자
Testmsg.elf	4Mbyte		3185
Gdb_test.o	6Kbyte	3(+α)	110
Sort.o	4Kbyte	3(+α)	58

표 2 의 A~C 경우를 각각 분석하면 다음과 같다. 표 1 의 프로토콜 별로 그 횟수를 계산한 것이다.

A. SUM[MemAlign(1), SymAdd(1591), SymFind(1593)] =3185
B. SUM[MemWrite(60), MemSet(2), MemRead(36), MemAlign(3), SymAdd(3), SymFind(6)]=110
C. SUM[MemWrite(3)]=3(한 번에 전송가능한 최대 패킷 크기 따라 몇 개의 패킷으로 나뉘어 전달될 수 있음)

5. 결론

본 논문에서 구현한 타겟관리자가 동작될 원격 개발 환경은 내장형 실시간 응용을 개발하고자 할 때 제한된 호스트와 타겟 간의 통신, 제한된 타겟 시스템의 자원, 도구들 간의 빈약한 공조 체제 등의 문제점을 개선하기 위해 확장한 호스트 중심 원격 개발 환경이다.

본 논문에서는 StrongARM[8]에서 운용되는 Qplus 실시간 운영체제 위에 탑재될 내장형 실시간 소프트웨어를 개발하기 위한 원격 개발 환경의 주요 요소인 타겟관리자[10]의 프로토타입의 설계와 구현에 대해 기술하였으며, 성능 분석을 통하여 구현한 타겟관리자가 호스트-타겟 통신 횟수를 격감시키는 것을 확인하였다. 타겟관리자는 대화형 쉘, 원격 디버거, 자원 모니터와 같은 도구들이 타겟에 접근할 수 있도록 해주는 중개자 역할을 한다. 특히, 도구의 확장성을 고려한 Open API 구현과 다양한 실행 모듈 및 통식 방식 지원을 위해 DLL 형식으로 구현했다는 장점이 있다. 현재의 타겟관리자는 통신 방식은 이더넷과 시리얼 방식을 지원하며, 실행 모듈 포맷은 ELF 와 COFF 형식을 지원하고 있다.

앞으로 도구 간의 통신 기능과 다양한 프로세서를 지원하며, 다양한 OMF 응용 프로그램을 동시에 로딩하거나, 실행 파일의 순서에 상관없이 로딩해도 제대로 실행할 수 있도록 로딩 기능을 강화해 나갈 예정이다.

참고문헌

[1] Phillip A. Laplante, *Real-Time Systems Design and Analysis*, pp. 315-326, IEEE Press, 1997.
[2] C. M. Krishna, Kang G. Shin, *Real-Time Systems,* pp. 176-183, McGraw-Hill, 1997.
[3] Microtec, *Spectra Backplane Concepts*, Microtec, 1997.
[4] Microtec, *Spectra Boot and VRTX Real-Time OS*, 1996.
[5] WindRiver, *VxWorks 5.3.1 Reference Manual*, 1996.
[6] WindRiver, *Tornado API Guide 1.0.1*, 1997.
[7] WindRiver, *Tornado User's Guide*, 1995.
[8] Intel, *StrongARM EBSA-285 Evaluation Board*, Oct. 1998.
[9] W. Richard Stevens, *Unix Network Programming*, Prentice Hall, 1994.
[10] Chaedeok Lim, Heungnam Kim, Young-Kuk Kim, "A Tool Broker in Remote Development Environments for Embedded Applications", Accepted by IASTED AI'2000, Feb. 14, 2000.
[11] David Finkelman, "Combining Visual TEST with ICE for Embedded System Testing," pp. 160-161, Electrical and Electronics Engineers in Israel, 1996.
[12] T. Yasuda, K. Ueki, "A Debugging Technique Using Event History", pp. 137-141, Proc. of the Conference on Real-Time Computing Systems and Applications, 1994.
[13] S. Kapur, C. Sriprasad, "Software Development Tools for Embedded Systems", pp. 331-335, Proc. of Digital Avionics System Conference, 1995.
[14] Jack G. Ganssle, "Debuggers for Modern Embedded Systems", in Embedded Systems Programming, Nov. 1998.
[15] 이은향 외 3 인, "교환 소프트웨어의 교차 디버깅을 위한 디버그 서버 구현", in JCCI'96 Proceedings, 광주, 1996.4.
[16] Jonathan B. Rosenberg, "How Debuggers Work", John Wiley & Sons, 1996.
[17] E. Lee, H. Chang, S. Jun, J. Cho, "A Cross Debugging Architecture for Switching software", Proc. fo ICCT'96, Vol. 1, pp. 214-217, 1996
[18] H. Tokuda, M. Kotera, "A Real-Time Tool Set for the ARTS Kernel", Proc. of RTSS, pp. 289-299, 1998.
[19] N. Iga, N. Ohashi, Y. Nakamoto, H. Monden, "Real-Time Software Development System Rtiplus", Proc. of 12th TRON Project Int'l Symposium, pp. 24-33, 1995.
[20] Y. Byun, Y. Chung, B. Lee, "High-Level CHILL Debugging System in Cross-Development Environments", Proc. of the 6th Euromicro

Workshop on Parallel and Dist. Processing, pp. 211-216, 1998.