내장형 소프트웨어의 성능 분석을 위한 원격 모니터링 시스템

신경호O, 조용윤, 유재우
숭실대학교 일반대학원 컴퓨터학과
delios@orgio.net, yycho@ss.ssu.ac.kr, cwyoo@comp.ssu.ac.kr

Remote Monitoring System to Analyse the Performance of the Embedded System

Kyoung-Ho Shin, Yong-Yoon Cho, Chea-Woo Yoo
Dept of Computing, Soongsil University

요 약

내장형 시스템 개발의 효율성 향상을 위해 개발자들은 내장형 시스템 성능 평가 도구를 사용한다. 성능 평가 도구는 내장형 소프트웨어가 메모리나 프로세서 자원들을 가능한 효율적으로 사용할 수 있도록 개발 단계중 적절한 성능평가를 수행한다. 본 논문은 내장형 소프트웨어의 효율적인 개발을 위한 성능 평가 도구를 기존의 하드웨어적인 도구를 통하지 않고 순수 소프트웨어적인 방법으로 제공하는 내장형 소프트웨어의 성능 분석을 위한 원격 모니터링 시스템을 제안한다. 시스템은 내장형 소프트웨어의 프로그램 성능과 함수 별 측정 및 메모리 관련 성능 평가를 수행하기 위한 모듈과 결과 로그를 분석하여 사용자에게 GUI 형태로 제공하는 모듈로 구성되어 있다. 본 시스템을 이용한 개발자는 추가 비용과 학습 없이 빠르고 정확하게 신뢰성 있는 내장 소프트웨어를 개발할 수 있다.

1. 서론

내장형 시스템이란 다양한 기능을 수행하도록 설계된 범용 시스템과는 달리, 개인 휴대통신 단말기 및 각종 정보형 단말기(PDA)와 같은 소형 정보 기기 또는 각종 전자 제품 안에 내장되어 해당 기기의 기능을 제어하는데 사용되는 컴퓨터 시스템을 말한다. 내장형 시스템은 일반적으로 크기가 작으며 범용 시스템에 비해 성능이 낮은 마이크로프로세서를 내장하고, 독립적인 내장형 실시간 운영체제로 가동되어 여러 개의 실시간 응용 프로그램을 수행할 수 있는데, 이러한 실시간 응용 프로그램을 내장형 소프트웨어라고 한다. 내장형 소프트웨어는 일반적으로 제한된 자원을 사용하는 내장형 시스템을 위해 최적화 되고 안정된 형태로 개발되어야 한다. 현재 내장형 시스템을 채용한 정보 기기들에 대한 사용자의 요구사항이 복잡해지고, 보다 다양한 주변장치들과 연결되어야 하는 필요에 따라, 내장형 소프트웨어 또한 더욱 복잡해져서 효율적인 개발과 시험이 어려워지게 되었다. 또한 내장형 시스템을 이용한 시장이 급속도로 팽창함에 따라, 내장형 소프트웨어는 더욱 짧은 시간 내에 개발이 완료되어야만 시장성을 가질 수 있게 되었다. 내장형 시스템에서 사용되는 소프트웨어의 개발은 일반적으로 범용 시스템(Host System)에서 소프트웨어의 개발이 실시되고, 목적 시스템(Target System)에서 수행되는 교차 개발(Cross Development) 환경이다. 다음의 그림 1은 내장형 소프트웨어 개발을 위한 교차 개발 환경에

대한 구조이다.

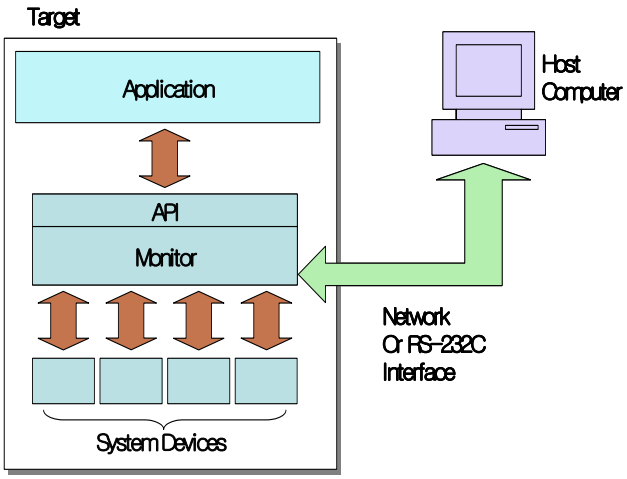

그림 1. 내장형 S/W 교차-개발환경

이와 같은 교차환경에서의 소프트웨어 개발 시간의 단축과 효율성향상을 위해 개발자는 내장형 개발 도구가 필수적이라고 할 수 있다. 이러한 개발 도구는 크게 하드웨어 개발 도구와 소프트웨어 개발 도구로 나눌 수 있다. 하드웨어 개발 도구는 범용 시스템과 목적 시스템 사이에 별도의 하드웨어 장치를 두어 범용 시스템에서 실행 중인 전용 소프트웨어의 요청에 따라, 목적 시스템에서 실행 중인 소프트웨어를 제어하거나 모니터링 하여 그 결과를 범용 시스템으로 전달하는 구조이다. 하드웨어 개발 도구는 성능이 좋고 해당 하드웨어에 최적화 된 테스트를 할 수 있다는 장점이 있지만, 가격이 매우 비싸고 사용하기 어려운 전용 소프트웨어를 다루기 위해 추가적인 학습이 필요하다. 게다가 장치들 간의 연결이 어렵고 복잡하여 많은 기술 지원을 필요로 하고, 하드웨어 장치이기 때문에 시장에 유연성 있게 대처하기에 상대적으로 어렵다는 단점이 있다. 하드웨어를 이용한 개발 도구의 구조는 그림 2에 나타난 것과 같다.

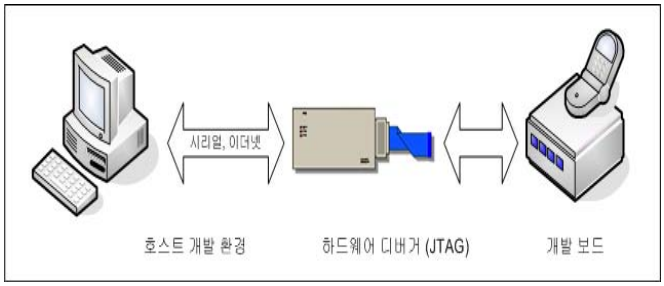

그림 2. 하드웨어를 이용한 개발 도구

소프트웨어 개발 도구는 목적 시스템에 별도의 경량의 소프트웨어를 적재시키고, 이를 통해 개발된 소프트웨어의 실행을 제어하고 결과를 모니터링 하는 방법이다. 이렇게 내장형 시스템에 적재되어 범용 시스템의 범용 개발 도구와 통신하면서, 요청된 작업을 대신 수행하는 소프트웨어를 디버그 에이전트 또는 테스트 에이전트라고 한다. 소프트웨어 개발 도구는 일반적으로 범용 시스템에서 사용되는 개발 도구를 거의 그대로 사용할 수 있어 추가적인 학습이 필요 없고, 하드웨어 장비에 비해 상대적으로 저렴할 수 있다. 그러나 목적 시스템에 테스트 에이전트를 내장하기 위해 메모리가 조금 더 요구되며, 가능한 적은 메모리를 차지하는 테스트 에이전트를 개발하는 것이 중요한 과제이다.

본 논문은 다양하고 복잡한 모바일 및 내장형 타겟 시스템 기반의 S/W에 대한 성능 평가와 프로파일링 결과 분석을 순수 소프트웨어적인 방법으로 수행하기 위한 내장형 소프트웨어 평가 시스템을 제안한다. 본 논문에서 제안하는 평가 시스템은 하드웨어적 개발도구를 이용한 성능 평가 방법이 가지는 추가적인 개발비용과 학습에 대한 부담을 제거하고, 모바일 및 내장형 S/W를 빠르고 정확하게 시험 검증, 분석할 수 있게 되어 개발 소프트웨어의 신뢰성을 보장할 수 있다. 제안하는 시스템은 리눅스 기반의 성능 평가 관련 GNU 툴을 크로스 컴파일을 통해 타겟 보드에 로드(load)하여 생성된 성능 평가 에이전트모듈 과 평가 결과를 처리하기 위한 로그 처리기로 구성된다.

2. 본론

2.1 원격 모니터링을 위한 성능 측정 에이전트

내장형 실시간 소프트웨어를 효율적으로 성능 측정하기 위하여 타겟 시스템에 성능 측정 에이전트라는 태스크를 수행시켜 호스트에서 결과를 받아볼 수 있는 클라이언트-서버 구조를 정의하였다. 성능 측정 에이전트는 호스트의 성능 측정 스터브(stub)에서 시리얼 포트나 이더넷을 통해 요청하는 정지점 관리, 레지스터와 메모리의 조회/수정, 스택 프레임 조회, 이벤트 관리, 예외 핸들러 관리 등에 대한 처리를 하고 호스트로 응답하는 기능을 담당한다. 내장형 소프트웨어를 내장형 시스템이나 개발 보드에서 직접 개발하고 성능 측정하는 방법은 구조적으로는 단순하며 이해하기 쉬울지 모르나, 일반적으로 내장형 시스템은 성능이 떨어지고 충분한 저장 공간도

가지고 있지 못하며, 사용자 인터페이스 지원이 미미하기 때문에 개발자가 개발을 하기에 용이하지 못하다. 개발자가 가장 다루기 쉽고 강력한 기능과 편리한 인터페이스를 지원하는 방법이 내장형 시스템에 타겟 에이전트를 두고 호스트와 연결하여 원격으로 개발하고 성능 측정하는 방법이다. 아래 그림 2. 은 목적 시스템에서 실행되는 에이전트의 구조를 나타낸다.

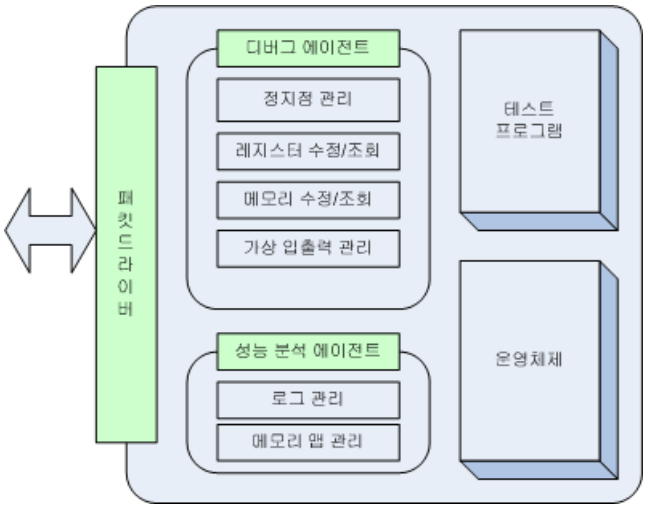

그림 3 목적 시스템에서 실행되는 에이전트의 구조

2.2 교차 도구 체인(cross tool chain)

소프트웨어의 소스 코드를 실행 가능한 코드로 변환하는 과정을 통틀어 컴파일이라 한다. 컴파일 과정에서 필요로 하는 도구에는 컴파일러, 어셈블러, 링커, 라이브러리 집합이 포함되는데, 이들을 통칭하여 도구 체인(tool chain)이라고 한다. 일반적인 경우에는 컴파일러가 실행되는 환경과 동일한 환경에서 동작하는 실행 코드를 출력 결과로 생성하지만, 내장형 소프트웨어를 원격으로 개발하는 방법에 있어서는 컴파일러가 실행되는 환경과는 상이한 내장형 하드웨어 환경에서 동작하는 실행 코드를 출력해야 하는데, 제안하는 시스템은 교차 개발된 내장형 소프트웨어에 대해 타겟에서 실행될 수 있는 실행 파일을 생성하기 위해 교차 도구 체인을 통한 크로스 컴파일러를 내장한다.

2.3. 내장형 소프트웨어의 성능 분석

내장형 소프트웨어의 성능을 분석하기 위해서는 원격 디버깅 에이전트에 실행 중인 프로그램의 상황을 모니터링 하여 호스트로 전송하는 기능을 추가로 작성해 주어야 한다. 이렇게 기능이 추가된 원격 디버깅 에이전트는 교차 도구 체인(cross tool chain)을 이용하여 내장형 시스템에서 실행 가능한 이진 파일로 만들어진다. 다음 그림 4. 는 본 논문이 제안하는 내장형 소프트웨어의 성능 분석을 위한 원격 모니터링 시스템의 구조도 이다.

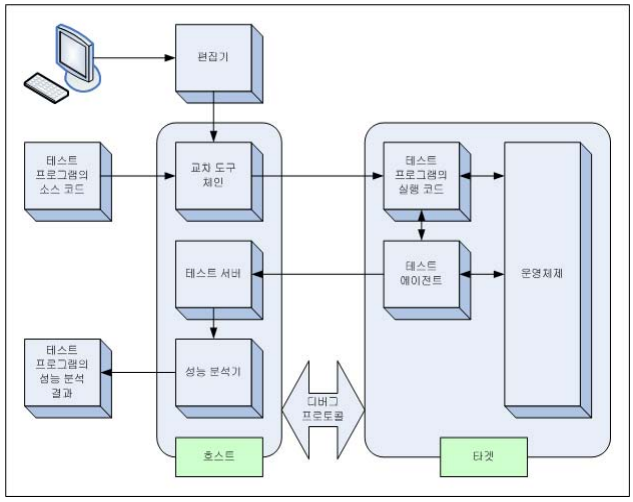

그림 4. 내장형 소프트웨어의 성능 분석을 위한 원격 모니터링 시스템

교차 도구 체인에는 컴파일러와 어셈블러, 링커가 포함된다. 내장형 시스템을 위한 내장형 소프트웨어는 다음과 같은 요구사항을 만족해야 한다.

가. 프로세서 자원을 적게 사용해야 한다. 내장형 시스템에 사용되는 프로세서는 일반적으로 성능이 낮으므로, 빠른 실행을 위하여 불필요한 코드가 없어야 하며, 같은 기능을 한다면 가급적 적은 양의 코드가 생성되도록 작성해야 한다. 실시간성을 요하는 소프트웨어의 경우 프로세스가 해당 소프트웨어에 실시간성을 제공하기 위해 충분한 성능을 지니는지 확인을 할 필요가 있다.

나. 메모리 자원을 적게 사용해야 한다. 내장형 시스템은 범용 시스템에 비하여 매우 작은 크기의 메모리를 가지고 있다. 모든 프로그램은 메모리가 부족하면 실행될 수 없으므로, 실행 코드 자체의 크기도 가능한 작아야 하며, 데이터 저장을 위해 사용하는 메모리도 적어야 한다. 또한, 메모리 누수를 방지하기 위해 한번 할당된 메모리는 프로세스 종료 전까지 반드시 해제되어야 한다.

이를 위해 제안하는 시스템은 다음 그림 5. 와 같은 4가지 부분에 대한 성능 분석 모듈을 포함한다.

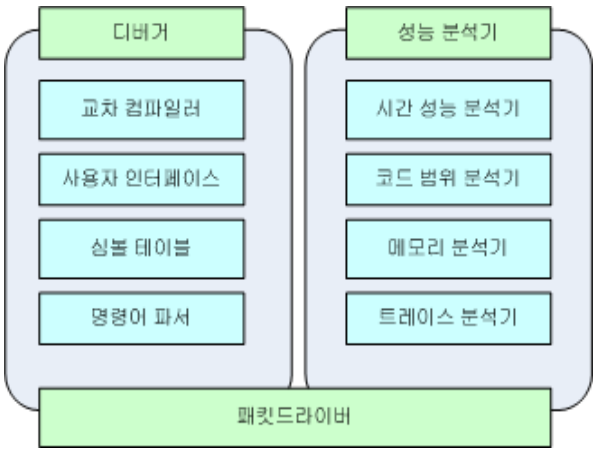

그림 5 소프트웨어 분석 도구 호스트의 구조

2.3.1 성능 (performance)

응용 프로그램 전체 또는 원하는 일부분의 실행에 걸리는 시간을 측정하여 프로세서가 해당 응용 프로그램 실행에 적합한지 또는 해당 응용 프로그램이 얼마나 최적화 되어 있는지 확인할 수 있다. 이를 위해 모든 함수의 진입점(entry point)과 종료점(exit point)에 현재 코드 위치와 현재 시간을 기록하는 코드를 삽입하여 별도의 저장소에 로그를 남기고 나중에 이를 분석하면 해당 함수의 실행에 소요된 시간을 계산할 수 있다.

2.3.2 코드 범위 (code coverage)

응용 프로그램을 실행 시 실제로 사용되는 부분과 사용되지 않는 부분, 자주 사용되는 부분, 거의 사용되지 않는 부분을 확인하여 최적화된 코드를 만들 수 있도록 한다.

2.3.3메모리 (memory)

메모리의 할당과 해제 정보를 출력하고, 전체 메모리 사용량을 확인할 수 있다. 또한, 할당 되었는데 해제되지 않은 메모리를 검사함으로써 메모리 누수를 방지하며, 중복 해제되어 버그를 유발시킬 수 있는 코드를 확인할 수 있다. 이를 위해 메모리 할당과 해제를 담당하는 malloc, free와 같은 시스템 콜을 포장(wrap)하여 각각의 경우에 현재 스택 프레임 상의 호출자(caller)의 주소를 확인하여 어떤 함수의 어떤 위치에서 메모리 할당과 해제 요청이 있었는지 여부를 저장소에 로그로 남기도록 한다. 또한 할당된 메모리들을 맵으로 관리하여 중복 할당되거나 할당 되었는데 해제되지 않은 메모리들을 확인할 수 있다.

2.3.4 트레이스 (trace)

응용 프로그램의 실행과 함께 어떤 순서로 함수 호출이 일어나는지를 출력하여, 응용 프로그램이 정상적으로 진행되고 있는지 여부나, 불필요하게 함수 호출이 많이 일어나지는 않는지를 확인할 수 있다.

3. 결론 및 향후 과제

본 논문은 내장형 소프트웨어의 성능 평가를 위해 원격 모니터링 시스템을 제안하였다. 제안된 시스템은 하드웨어적 성능 분석 방법이 가지는 추가적인 개발비용과 학습에 대한 부담을 제거하기 위해 내장형 소프트웨어를 위한 순수 소프트웨어적인 방법을 통한 성능 분석 기능을 제공하였다.

본 논문에서 제안한 시스템을 통해 개발자는 호스트에서 개발한 내장형 소프트웨어를 타겟에서 안정적으로 실행하여 성능 평가를 할 수 있으며, 성능 평가를 위한 추가적인 하드웨어 설치나 학습을 피할 수 있어 보다 빠르고 안정된 내장형 프로그램의 개발 효율성을 얻을 수 있다.

참고문헌

[1] 공기석, 손승우, 임채덕, 김흥남, “내장형 실시간 소프트웨어의 원격디버깅을 위한 디버그에이전트의 설계 및 구현”, *한국정보과학회 가을 학술발표논문집 Vol. 26. No 2*, pp.125~127, 1999.

[2] In-Band Diagnostic, Debug and Reporting Tunneling Protocol-FatPipe Protocol, http://www.ineoquest.com/pub/docs/Papers/FatPipe_v2.pdf

[3] Dr. Neal Stollon, Rick Leatherman, Bruce Ableidinger, "Multi-Core Embedded Debug for Structured ASIC Systems", *proceedings of DesignCon 2004*, Feb, 2004.