제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

임베디드 소프트웨어를 위한 프로파일링 도구의 설계 및 구현

곽동규°, 조용윤, 유재우
숭실대학교 일반대학원 컴퓨터학과
{coolman°, yycho}@ss.ssu.ac.kr, cwyoo@comp.ssu.ac.kr

A Design and Implementation of A Profiling Tool for the Embedded Software

Dong-Gyu Kwak°, Yong-Yoon Cho, Chea-Woo Yoo
Dept of Computing, Soongsil University

요 약

임베디드 소프트웨어는 제한된 자원을 이용해 빠르고 정확한 기능 수행이 가능하도록 설계 되어야한다. 본 논문은 임베디드 기반 소프트웨어의 효율적인 개발을 위해 개발 소프트웨어의 실행 성능을 분석할 수 있는 프로파일(profile) 도구를 제안한다. 제안하는 시스템은 교차환경(cross-platform environment)에서의 테스트 코드 및 결과 전송을 위한 에이전트 모듈과 프로파일링을 위한 테스팅 엔진 그리고 내장형 소프트웨어의 프로그램 실행 최적화를 위해 개선되어야 할 함수 정보를 GUI 형태로 출력하는 레포팅 모듈로 구성된다. 본 시스템은 효율성과 신뢰성 있는 임베디드 기반 소프트웨어 개발에 기여할 것으로 기대된다.

1. 서론

보통의 응용 소프트웨어는 개발자의 코딩 편의성을 위해 소프트웨어의 성능 최적화의 노력이 희생되기도 한다. 그러나 내장형 소프트웨어는 일반적인 범용 컴퓨터 소프트웨어와는 달리, 개인 휴대통신 단말기 및 각종 정보형 단말기(PDA)와 같은 소형 정보 기기 또는 각종 전자 제품 안에 내장되어 해당 기기의 제안된 기능을 이용하여 최적의 성능을 발휘해야 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 임베디드 소프트웨어 개발자는 교차 개발 환경에서의 소프트웨어 개발 시간 단축과 효율성향상을 위해 임베디드 소프트웨어 프로파일 도구를 사용 할 수 있다. 그림 1은 교차 환경 기반의 프로파일 도구의 전체적인 개념도 이다.

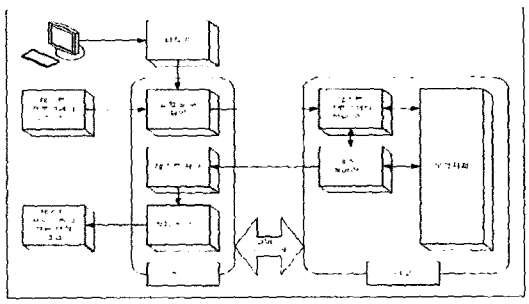

그림 1. 임베디드 프로파일 도구 개념도

소프트웨어 프로파일링(profiling)이란 개발 소프트

웨어의 수행 과정과 그 과정에서 얻어지는 결과를 기록하고 분석하여 실행 환경에 최적화된 소프트웨어를 개발할 수 있는 기회를 주는 방법이다.

소프트웨어 프로파일 도구는 목적 시스템에 별도의 경량의 소프트웨어를 적재시키고, 이를 통해 개발 소프트웨어의 실행을 제어하고 결과를 모니터링하는 도구이다.

본 논문은 다양하고 복잡한 모바일 및 임베디드 타겟 기반의 소프트웨어에 대한 성능 프로파일링과 GUI 기반의 레포팅을 통한 결과 분석을 제공하는 임베디드 소프트웨어 프로파일 도구를 설계한다. 본 논문에서 제안하는 프로파일 도구는 GUI 기반의 순수 소프트웨어 프로그램으로써 별도의 하드웨어 도구를 요구하지 않고 추가적인 분석비용과 학습 부담이 없다. 따라서 모바일 및 임베디드 소프트웨어에 대한 편리하고 쉬운 프로파일 및 분석방법을 제공함으로써 임베디드 관련 소프트웨어 개발 신뢰성과 효율성을 보장할 수 있다.

2. 본론

다음 그림 2는 본 논문에서 제안하는 임베디드 소프트웨어를 위한 프로파일 도구의 전체적인 구성도이다. 제안하는 도구는 원시 테스팅 코드에 대한 타겟 실행 파일 생성을 위한 크로스 컴파일러 및 추가 코드 삽입기와 실행 파일을 타겟 보드에 로드(load)하고 실행 및 결과 전송을 위한 에이전트 모듈 그리고, 결과를 레포팅하기 위한 뷰어(viewer)로 구성된다.

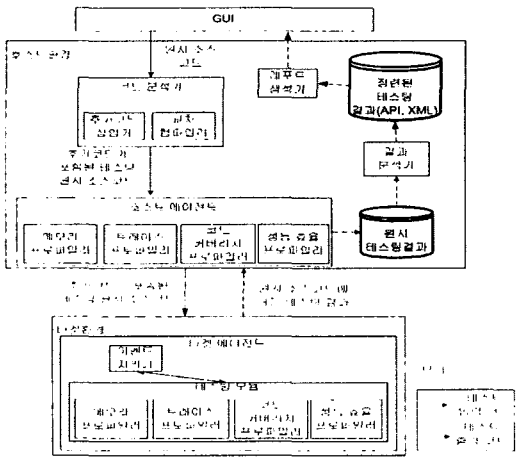

그림 2 제안하는 임베디드 소프트웨어 프로파일 도구의 모듈별 관계도

2.1 코드 분석기

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, 임베디드 소프트웨어를 원격으로 개발하는 방법에 있어서는 컴파일러가 실행되는 환경과는 상이한 내장형 하드웨어 환경에서 동작하는 실행 코드를 출력해야 하는데, 제안하는 시스템은 교차 개발된 임베디드 소프트웨어에 대해 타겟에서 실행될 수 있는 실행 파일을 생성하기 위해 코드 분석기를 포함한다. 코드 분석기는 교차 도구 체인을 통한 크로스 컴파일러와 타겟 머신을 위한 추가 코드를 소스 코드내에 삽입하기 위한 추가 코드 삽입기를 포함한다. 다음 그림 3은 코드 분석기에 대한 개념도 이다.

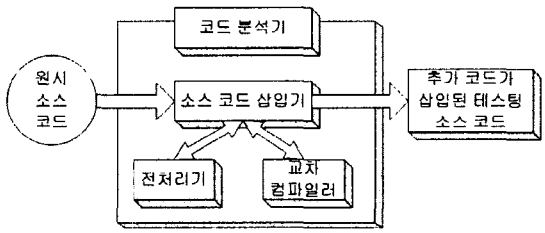

그림 3. 코드 분석기 구성도

소스 코드 삽입기 모듈은 우선 전처리기 모듈을 호출하여 소스 코드 내의 모든 전처리 명령들을 해당되는 문자열로 대치한 중간 코드를 생성한다. 생성된 중간 코드는 어휘, 구문, 의미 분석 과정을 거친 후 소스 코드 삽입기 모듈에 의하여 필요한 위치에 프로파일링 코드가 삽입된 새로운 소스 코드로 변환된다. 이렇게 만들어진 테스트를 위한 최종 소스 코드를 교차 컴파일러 모듈의 입력으로 전달하여 테스트 대상 프로그램이 수행될 환경에 적합한 목적 코드로 컴파일 하게 된다

2.2 에이전트 모듈

임베디드 소프트웨어를 임베디드 시스템이나 개발 보드에서 직접 개발하고 성능 측정하는 방법은 구조

적으로는 단순하며 이해하기 쉬울지 모르나, 일반적으로 임베디드 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 임베디드 시스템에 타겟 에이전트를 두고 호스트와 연결하여 원격으로 개발하고 성능 측정하는 방법이다.

제안하는 프로파일 도구는 교차 환경 기반의 임베디드 소프트웨어의 효율적인 성능 측정을 위해 호스트와 타겟에 클라이언트-서버 형태의 프로파일 에이전트를 가진다. 프로파일 에이전트는 호스트의 성능 측정 스터브(stub)에서 시리얼 포트나 이더넷을 통해 테스팅 대상 소스코드와 테스팅 결과의 전송을 위한 통신 세션을 유지한다. 호스트 에이전트는 입력 테스팅 소스 코드를 타겟 에이전트에 전달하고 결과를 호스트 파일 시스템에 저장한다. 타겟 에이전트는 호스트 에이전트로부터 전달받은 테스팅 소스 코드에 대해 프로파일 테스팅 모듈을 실행키고 발생된 이벤트에 대한 결과를 호스트 에이전트에 전달한다.

2.3. 임베디드 소프트웨어의 프로파일링

일반적으로 제한된 자원을 제공하는 임베디드 시스템 기반의 임베디드 소프트웨어는 다음과 같은 요구사항을 만족해야 한다.[4][5]

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 다음과 같은 4가지 부분에 대한 성능 분석 모듈을 포함한다.

2.3.1 성능 프로파일 (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.

2.3.2 코드 범위 프로파일 (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.

2.3.3 메모리 프로파일(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 프로파일 (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.

2.4 데이터 분석기

타겟에서 실행된 소스 코드에 대한 테스팅 결과를

분석하고 조작하기 쉬운 형태로 결과를 변경하기 위한 모듈이다. 데이터 분석기는 레포트 생성기가 사용자의 요구에 맞는 GUI 형태의 결과를 출력할 수 있도록 API 와 XML 형태로 테스팅 결과를 변경 저장한다. 그림 4는 데이터 분석기 모듈을 구성하는 내부 모듈을 나타낸다.

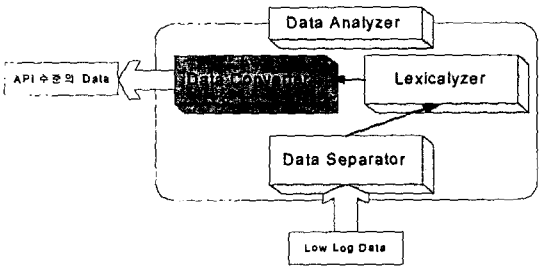

그림 4. 데이터 분석기 모듈 내부 구성도

Data Analyzer는 Data Separator와 Lexicalyzer, Data Converter로 구성되어 있다. Data Separator는 저수준 로그의 종류를 분석하여 분류하는 역할을 하고 Lexicalyzer는 분석된 저수준의 로그를 변환 가능한 형태로 변환하는 모듈이며 Data Converter는 사용 가능한 API형태로 변환하는 모듈이다

2.5 레포트 생성기

레포트 생성기는 결과에 대해 사용자에게 GUI 형태의 레포트를 생성하기 위한 모듈이다. GUI를 통해 사용자가 원하는 테스팅 레포트의 종류를 선택하면, 레포트 생성기는 데이터 분석기가 테스팅 결과에 대해 생성한 API와 XML 파일을 이용해 그래픽 형태의 결과를 출력한다. 그림 5는 레포트 생성기의 초기 화면과 출력 화면이다.

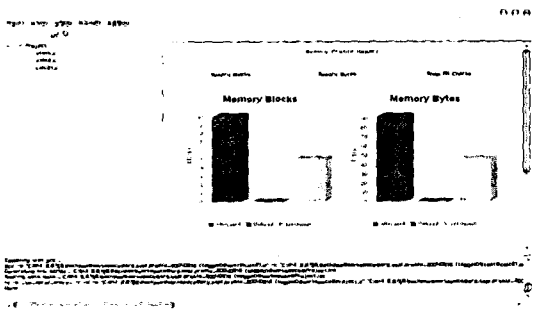

그림 5. 메모리 테스팅 결과

그림 5는 지금까지 설명한 프로파일 도구의 각 모듈을 통해 C로 작성된 임베디드 프로그램에 대한 프로파일 테스팅 결과를 뷰어를 통해 나타낸 것이다. 그리 5에 나타난 뷰어의 출력은 4가지 프로파일 테스팅 중에 메모리 사용에 대한 결과를 그래픽 형식으로 나타내고 있다.

3. 결론 및 향후 과제

본 논문은 임베디드 소프트웨어의 성능 평가를 위한 프로파일 도구를 설계하였다. 제안된 도구는 순수 소프트웨어만을 통한 프로파일 도구로써, 편리한 GUI 테스팅 환경을 통해 개발자가 호스트에서 개발한 임베디드 소프트웨어를 타겟에서 안정적으로 실행하여 성능 평가를 할 수 있도록 지원하며, 그래픽 형태의 다양한 결과 레포트를 얻을 수 있는 뷰어를 제공한다.

따라서, 개발자는 개발 프로그램의 성능 최적화를 위한 시간과 노력을 줄일 수 있어 보다 빠르고 안정된 임베디드 프로그램 개발 효율성을 얻을 수 있을 것으로 기대된다.

향후, 본 논문에서 제안한 프로파일 도구가 생성하는 XML 결과 문서를 이용한 웹 기반 프로파일을 순수 소프트웨어적으로 지원하는 도구에 대한 연구를 진행할 것이다.

참고문헌

[1] 공기석, 손승우, 임채덕, 김홍남, "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.

[4] Bart Broekman, Testing Embedded Software, Addisson-wesley, Dec. 2002.

[5] L.Hatton, Embedded software testing, Software Testing Congress, 2000