임베디드 소프트웨어 성능평가 도구

조용윤*, 김기원**, 김봉기***
*숭실대학교, **초당대학교, ***진주산업대학교

A Performance Evaluation Tool in Embedded Softwares

Yong-yun Cho*, Gi-weon Kim**, Bong-gi Kim***
*Soongsil University, **Chodang University, ***Jinju National University
E-mail : *yycho@ss.ssu.ac.kr, **kwkim@chodang.ac.kr · ***bgkim@jinju.ac.kr

요 약

교차-개발환경을 기반으로 한 임베디드소프트웨어 개발은 일반적인 데스크톱 컴퓨터에서의 개발방법론 및 도구와의 차이점이 발생한다. 이러한 문제점들로 인해 난이도가 높은 임베디드 소프트웨어 애플리케이션을 쉽게 개발 해주는 기술에 대한 수요가 확산됨에 따라 다양한 기종과 규격의 임베디드소프트웨어 개발환경에 최적화된 시험검증시스템이 절실히 필요하다. 본 논문은 내장형 시스템 개발자가 쉽고 편리하게 원하는 GUI 형태의 결과 분석도구를 생성할 수 있도록 하기 위한 프로파일 로그 분석 방법을 제안한다. 제안하는 로그 분석 방법에 의한 API를 통해 개발자나 사용자는 자신의 취향에 맞는 GUI 형태의 결과 분석 도구를 쉽고 빨르게 생성하여 내장형 소프트웨어 개발의 효율성을 높일 수 있으며, 고가의 해외 개발도구의 수입대체 효과를 가져와 관련 산업 발전에 크게 기여할 것으로 기대 된다. 또한 국내 시장의 활성화를 통하여 개발업체간 상호교류를 통하여 보다 나은 국내 산업 시장을 형성하여 기존 임베디드 산업의 경쟁력을 강화하고 고난도의 응용 S/W의 개발과 시험 검증을 용이하게 할 수 있어 넓은 신 시장·창출 효과를 불러올 수 있다.

1. 서론

내장형 S/W는 일반 S/W와는 달리 실시간성, 고신뢰성, 저전력을 요구하는 특성을 가지며 통상적으로 개발 플랫폼과 타겟 플랫폼이 일치하지 않는다. 이러한 개발환경을 "교차-개발환 (Cross-Development Environment)"이라고 부른다. 내장형 시스템이란 특별한 목적으로 설계한 컴퓨터 하드웨어, 소프트웨어, 그리고 대개 부가적인 기계 및 기타 부분의 조합을 말한다. 내장형 시스템을 탑재하고 있는 제품에 대한 요구사항이 증가함에 따라, 내장형 소프트웨어의 복잡도가 증가하고 있다. 내장형 소프트웨어의 복잡도 증가는 효율적인 소프트웨어 개발을 어렵게 만든다. 그러므로 소량의 자원을 보유한 내장형 시스템에서 효율적인 소프트웨어 개발을 위한 내장형 소프트웨어 성능 측정 방법이 요구되고 있다. 일반적인 소프트웨어와는 달리 내장형 소프트웨어는 개발자가 적은 자원을 가지고 있는 타겟에서 CPU나 메모리와 같은 자원을 얼마나 소비하고 어느 정도의 성능을 발휘하는지 파악하는 것이 매우 중요하다. 내장형 소프트웨어에 대한 성능 평가 및 트레이스는 타겟에서 실행되는 프로세스나 쓰레드의 리스트, 각 프로세스의 CPU, 메모리(스택, 힙, 텍스트) 사용량, 타겟 전체의 메모리 사용량 등과 같은 모니터링 정보이다. 성능 측정 정보는 정보의 판독성 향상과 이용 효율성을 위해 다양한 GUI 형태로 제공될 수 있다. 다양한 성능 측정 정보 보고기(reporter) 또는 뷰어(viewer)는 저 수준 텍스트 기반의 성능 측정 정보를 GUI 형태로 제공하기 위해 성능 측정 도구에 포함될 수 있다. 또한, 이러한 GUI 형태의 성능 테스트 결과 보고기 또는 뷰어는 개발자나 사용자의 성향이나 소프트웨어의 특성에 따라 다양하게 존재할 수 있다. 지금까지 소프트웨어의 실행 상태를 로그로 저장하여 저장된 로그를 개발자가 직접 분석하는 방법으로 GUI 형태의 보고기 또는 뷰어를 생성하였다. 하지만, 이 방법은 개발자가 가공되지 않은 저수준의 정보를 직접 분석하여 성능을 측정해야하는 어려움이 있고, 개발비용의 증가를 가져올 수 있다. 그러므로 사용자의 기호에 따른 다양한 형태의 성능 측정 정보 보고기 또는 뷰어의 생성을 위해 원래의 저 수준 테스트 결과 자료는 보다 유연하고 편리하게 가공될 수 있는 형태로 변환되어 제공되어야 한다. 개발자나 사용자의 시각에서 직관적이고 종합적인 성능 측정 정보는 성능 평가에 대한 신뢰성 판단에 빠른 결정을 유도할 수 있어 내장형 소프트웨어 개발비용을 감소시킬 수 있다. 본 논문은 내장형 소프트웨어를 위한 성능 측정 결과를 이용한 다양한 형태의 사용자 GUI로 표현할 수 있도록 가공되지 않은 저수준의 성능 테스트 결과 로그를 분석하고 가공하여 API 수준의 인터페이스로 제공하는 성능 측정 결과 로그 분석기를 제안한다. 제안하는 성능

측정 결과 로그 분석기는 저수준의 성능 측정 결과 로그를 분석하는 프로파일 로그 분석기와 프로파일 로그 분석기로부터 가공된 결과 정보를 분류하고 GUI형태로 쉽게 변경할 수 있도록 API 형태로 제공하는 프로파일 변환기로 구성된다. 내장형 시스템에서 저수준의 로그는 하드웨어적인 성능 측정 방법[3]과 소프트웨어적인 성능 측정 방법[1]을 통해 생성할 수 있다. 하드웨어적인 방법은 빠른 실행과 타겟에 최적화된 테스팅이 가능하고 분석 결과를 저장하기 위한 공간을 별도로 가질 수 있는 장점을 가지고 있다. 하지만 추가적으로 고가의 장비를 필요로 하고 타겟과 호스트 사이의 하드웨어적 연결을 위한 추가적 노력이 요구된다. 소프트웨어적인 방법은 타겟 에이전트를 개발 보드에서 실행시킴으로 자원이 풍부하지 않은 개발 보드의 자원을 필요로 하지만 개발에서는 실제 제품에 사용되는 하드웨어보다 풍부한 자원을 가지고 있고 타겟 에이전트의 경량화를 통해 자원적인 약점을 극복할 수 있다. 그러므로 본 논문은 범용 시스템에서와 유사한 리눅스 기반의 내장형 시스템에서 GNU 툴을 사용한 저수준의 로그 분석 방법을 보인다. 본 논문은 2장에서 관련 연구를 소개하고 3장 본론에서 연구 결과를 서술한 후 4장 결론 및 향후 연구방향으로 맺는다.

2. 관련 연구

내장형 시스템에서 저수준의 로그는 하드웨어적인 성능 측정 방법[3]과 소프트웨어적인 성능 측정 방법[1]을 통해 생성할 수 있다. 하드웨어적인 방법은 빠른 실행과 타겟에 최적화된 테스팅이 가능하고 분석 결과를 저장하기 위한 공간을 별도로 가질 수 있는 장점을 가지고 있다. 하지만 추가적으로 고가의 장비를 필요로 하고 타겟과 호스트 사이의 하드웨어적 연결을 위한 추가적 노력이 요구되고 에뮬레이터에 대한 별도의 학습이 필요하다. 소프트웨어적인 방법은 고가의 에뮬레이터를 필요하지 않는다. 그러므로 에뮬레이터에 대한 추가적인 학습이나 고 비용의 개발비가 없어도 성능 평가 방법을 사용할 수 있는 장점을 가지고 있다. 성능 테스팅을 위한 타겟 테스팅 에이전트를 개발 보드에서 실행시킴으로 호스트에서 서비스를 요청하면 시리얼이나 이더넷을 통해 호스트로 성능 평가 정보를 보내준다. 또한, 성능 평가는 일반적인 디버깅 기능을 포함하여 정지점 관리와 레지스터 수정/조회, 함수 호출 관리, 이벤트 관리, 예외 통보 관리 등의 기능을 제공 할 수 있다. 그러나 발생된 모든 성능 테스트 결과는 저수준의 텍스트 기반 정보이기 때문에 개발자가 직관적으로 성능 테스트 결과 정보를 파악하기는 어렵다. 즉, 실시간 에이전트를 통해 발생한 테스트 결과에 대해 개발자가 직접 대화형 쉘과 자원 모니터를 통해 분석해야 한다. 개발자가 직접 분석하는 방법은 응용 프로그램 개발 이외의 수고를 들게 하여 응용프로그램의 개발을 비효율적으로 만드는 약점이 있다.

3. 본론

일반적으로 내장형 응용프로그램의 개발은 호스트/타겟 방식에 의한 교차 개발(cross development) 방법을 사용한다. 그러므로 성능 측정 로그 분석기 또한, 같은 구조에서 설계되어야 한다. 그림 1은 성능 측정 로그 분석 시스템의 전체 구조를 보여준다.

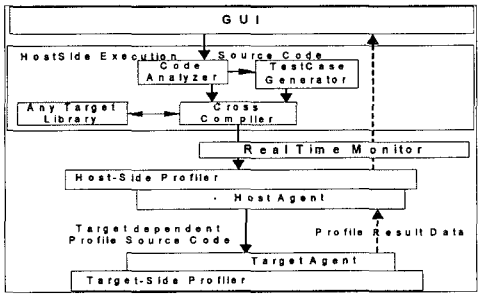

그림 1. 성능 분석 시스템의 전체 구조도

타겟 시스템은 측정 프로그램이 실행되고 타겟 에이전트를 통해 가공되지 않은 저수준의 로그가 기록된다. 타겟 에이전트에서 생성된 로그는 성능 측정 로그 분석기를 통해 정련되어 사용자 GUI 시스템에게 전달된다.

성능 측정 로그 분석기는 프로파일 분석기와 프로파일 결과 분석 API로 이루어져 있고, 타겟/호스트 구조에서 호스트에 위치한다. 타겟 시스템은 내장형 시스템으로써 자원이 풍부하지 않아, 성능 측정 로그 분석기는 측정 프로그램의 성능 측정에 영향을 주지 않도록 자원이 풍부한 범용 컴퓨터인 호스트 시스템에 위치해야 효율적이다.

프로파일 분석기는 로그 분리기와 어휘 분석기, 로그 변환기로 나누어진다. 로그 분리기는 타겟 에이전트에서 생성한 로그를 종류에 따라 분류한다. 분류된 로그는 어휘 분석기를 통해 필요한 부분과 불필요한 부분으로 나누어지고 정보의 토큰으로 추출되어진다. 토큰별로 추출된 로그는 로그 변환기를 통해 정련되어 분석 가능한 정보로 변환된다.

프로파일 분석기는 세 단계를 통해 저수준의 로그를 분석하는데 첫 번째 단계는 통합되어 있는 로그를 정보의 종류에 따라 분할하는 단계이다. 두 번째 단계는 분리된 로그를 적절한 토큰으로 분리하는 단계이고, 세 번째 단계는 분석된 로그를 프로파일 결과 분석 API에서 사용 가능한 형태로 가공하는 단계이다. 앞에서 기술한 세 가지 단계를 처리하는 부분을 각각 로그 분리기와 어휘 분석기, 로그 변환기로 정의한다.

프로파일 결과 분석 API는 사용자에게 메모리에 관한 정보와 함수의 자취에 관한 정보, 블록의 실행 범위에 관한 정보, 함수의 시간적 성능에 관한 정보를 제공한다. 메모리에 관한 정보는 메모리의 할당과 해제에 관한 로그를 분석하여 메모리 누수와 메모리 사용량에 관한 정보를 분석한다. 메모리의 누수는 소프트웨어의 메모리 자원을 불필요하게 소모하는 것으로 메모리 자원이 적은 내장형 시스템에서 치명적이 오류가 된다. 함수의 자취에 관한 정보는 재귀적으로 함수가 호출되는 응용프로그램에서 외부 입력에 따른 함수 호출 경로를 파악할 수 있는 정보가 된다. 그리고 블록의 실행 범위에 관한 정보는 실행되지 않은 블록이 소스 코드에 삽입되어 실행되지 않는 기계어 코드를 제거하는 정보가 된다. 마지막으로 함수의 시간적 성능에 관한 정보는 전체 소프트웨어의 시간적 성능을 모듈별로 측정할 수 있는 정보가 된다.

프로파일 분석기는 세 부분으로 나누어져 있다. 첫째는 로그 분리기이고, 둘째는 어휘 분석기 이며, 셋째는 로그 변환기이다. 본 논문은 리눅스 기반의 내장형 시스템에서 GNU 툴과 메모리 할당/해제를 감시하는 메모리 트레이서를 통해 생성된 저수준의 프로파일 정보를 가공하고 분석하는 방법을 소개한다.

프로파일 분석기의 첫 번째 단계인 로그 분리기는 혼합되어 있는 로그를 종류에 따라 분리한다. 예를 들어 프로파일러로부터 생성된 저수준의 로그에는 함수의 호출 횟수와 발생한 재귀적으로 발생한 사이클, 실행된 시간등과 같은 정보를 포함하고 있다. API 수준의 인터페이스를 제공하기 위해서는 이 정보를 따로 분리하여 분석하여야 한다. 표 1은 프로파일러와 메모리 트레이서로부터 생성된 가공되지 않은 저수준의 로그 중 일부이다.

표 1. 가공되지 않은 저수준의 로그

```
index % time    self  children    called     name
                                    95000           func1 <cycle 1> [3]
[2]     0.0    0.00    0.00       95000           func2 <cycle 1> [2]
                                      900           func1 <cycle 1> [3]
-----------------------------------------------
                                      900           func2 <cycle 1> [2]
                0.00    0.00     1000/1000        main [13]
[3]     0.0    0.00    0.00       1900           func1 <cycle 1> [3]
                                    95000           func2 <cycle 1> [2]
-----------------------------------------------
Index by function name
   [3] func1                   [2] func2                   [1] <cycle 1>

@ <Location>:(Called function + Called Location)[Instruction Location] +/- Address Size
= Start
@ ./ex-n:(mtrace+0x169)[0x80484e9] + 0x804a378 0x12
@ ./ex-n:[0x8048596] - 0x804a378
```

표 1은 프로파일러와 메모리 트레이서가 타겟 시스템에서 생성한 저수준 로그이다. 생성된 로그는 네트워크를 통해 호스트 시스템으로 이동한다. 이동한 로그는 프로파일러가 생성한 로그와 메모리 트레이서가 생성한 로그로 분할한다.

분할된 로그는 어휘 분석기를 통해 가용한 정보를 추출해내고 파싱을 통해 어휘 분석을 한다. 예를 들어 표 1에서 첫 번째 라인의 정보는 불필요한 정보이고, "[1]"과 같은 토큰은 함수의 인덱스정보이다. 그러므로 인덱스 정보를 기준으로 한 함수의 정보를 추출해낸다. 추출된 정보는 로그 변환기를 통해 파로파일 결과 분석 API가 사용할 수 있는 형태의 자료구조로 변환한다. 추출된 정보는 프로파일 결과 분석 API를 통해 분석한다. 그림 2는 호스트/타겟 시스템에서 시스템 성능 측정 로그 분석기를 통해 얻어진 Trace 프로파일의 결과 화면이다.

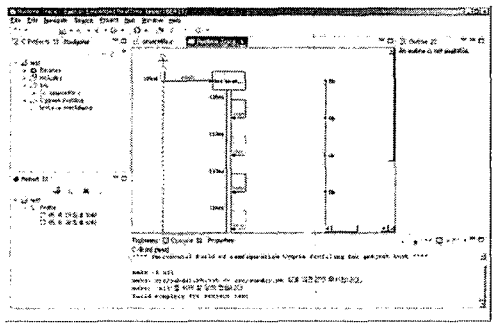

그림 2. Trace 프로파일

프로파일 결과 분석 API는 자료 구조 형태의 가공된 로그 정보를 필요에 따라 분석하여 성능 측정 사용자 GUI에게 정보를 제공한다. 프로파일 결과 분석 API에는 크게 메모리 감시와 함수 자취, 함수의 시간적 성능 측정, 블록의 실행 범위를 측정하는 모듈이 있다. 메모리 감시 모듈은 메모리 트레이스 정보와 함수 프로파일 정보를 분석하여 메모리 누수와 메모리 사용량에 관한 정보를 제공하고, 함수 자취 모듈은 함수 프로파일 정보와 심볼 테이블의 정보를 분석하여 재귀 호출에서의 싸이클 발생과 함수 실행 경로 정보, 함수의 호출과 피호출 정보를 제공한다. 제안하는 성능분석 도구는 내장형 소프트웨어의 성능분석과 함께 테스팅을 실시한다. 그림 3은 제안하는 성능분석 도구가 제공하는 테스팅 결과 화면이다.

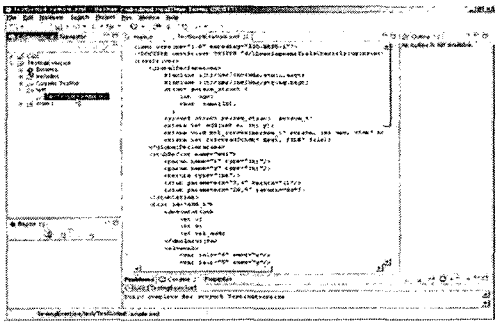

그림 3. 테스팅 결과

4. 결론 및 향후 연구방향

본 논문은 다양한 내장형 소프트웨어 개발에 있어 성능 평가 테스트 결과에 대한 로그를 분석하고 개발자나 사용자가 GUI를 통해 테스트 결과를 볼 수 있는 보고기나 뷰어를 쉽고 빠르게 생성할 수 있도록 결과 분석 API를 제공하였다. 제안한 성능 평가 로그 분석 방법이 제공하는 프로파일 결과 분석 API를 사용함으로써 내장형 소프트웨어 개발자나 사용자는 다양한 GUI 형태의 성능 측정 도구를 개발하기 위해 정련되지 않은 텍스트 기반의 저수준의 로그를 정련하고 분석하는 노력을 줄여 줄 수 있다. 따라서, 전체 내장용 소프트웨어 개발 효율성을 좋게 하고 더욱 효율적인 응용프로그램을 작성 할 수 있다. 본 논문은 싱글프로세서에서의 성능 측정 방법을 제시하였다. 하지만 내장형 시스템은 다수의 프로세서를 사용하는 환경으로 발전하고 있다[3]. 그러므로 다수의 프로세서를 사용하는 환경에서의 성능 측정 방법에 대한 연구가 필요하다.

참고문헌

[1] 공기석, 손승우, 임채덕, 김홍남, "내장형 실시간 소프트웨어의 원격 디버깅을 위한 디버그에이전트의 설계 및 구현", *한국정보과학회 가을 학술발표논문집 Vol. 26. No 2*, pp.125~127, 1999.

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

[3] K. Weiß, T. Steckstor, C. Nitsch, W. Rosenstiel, "Performance Analysis of Real-Time-Operation Systems by Emulation of an Embedded System". 10th IEEE International Workshop on Rapid System Prototyping (RSP) Florida, USA, 1999.