내장형 시스템 소프트웨어를 위한 XML 기반의 프로파일링 도구의 설계 및 구현☆

A Design and Implementation A Software Profiling Tool based on XML for Embedded System

곽 동 규* 유 재 우**
Donggyu Kwak Chae-Woo Yoo

요 약

내장형 시스템의 요구사항이 증가함에 따라 내장형 시스템에서 동작하는 프로그램의 복잡도가 증가하고 있다. 이는 최적의 성능을 발휘하는 소프트웨어의 작성을 어렵게 만드는 요인이 된다. 본 논문은 내장형 시스템에 적합한 호스트/타겟 구조의 프로파일링 도구를 제안한다. 제안하는 도구는 교차 개발환경을 사용하는 내장형 시스템에 적합하도록 호스트에서 작성한 프로그램에 로그를 발생시키는 소스를 삽입하여 타겟 시스템에서 실행한다. 발생된 프로파일링 로그는 통신으로 호스트 시스템에 전송하고 전송한 로그 데이터는 호스트 시스템에서 분석하여 XML 형태로 저장하고 보고서를 생성한다. 보고서는 GUI 기반의 그래픽 뷰어를 통해 개발자에게 제공한다. 제안하는 도구는 자원이 적은 타겟 시스템의 로드를 줄이고 생성하는 로그 XML은 XSLT를 이용하여 다른 형태로 변환하기 용이하다. 또한 제안하는 도구는 이클립스 플러그인 기반으로 이클립스의 다양한 기능을 그대로 사용할 수 있는 장점을 가진다.

ABSTRACT

According to increasing requirments in embedded systems. embedded software has been more complicated then before. a optimum software is difficult in embedded system. software developer make a difficult optimum software. this paper suggests a software profiling tool with which a software developer can easily profiling the embedded system software in cross-development environments. the suggested tool is designed based on host/target architecture. this tool insert program source for make profiling log to target program. a target program executed in target system. a target system communicate profiling log to host system. this tool in host system analyze profiling log data. and make a XML of profiling log and a profiling report. a profiling report is graphic viewer based GUI. a target system in this tool needs a few computing power. and XSLT can conversion of profile log XML to other format data. and suggested tool based on eclipse plug-in, therefore developer can use operates in eclipse.

☞ KeyWords : 소프트웨어 프로파일링(Software Profiling), 내장형 시스템(Embedded System), 교차개발환경(Cross-Platform Environment), XML

1. 서 론

내장형 컴퓨팅 시스템(Embedded Computing System)은 일반적인 컴퓨팅 시스템과 달리 적은 컴퓨팅 자원을 갖는다. 그러므로 내장형 시스템 소프트웨어 개발자는 한정된 자원에서 요구사항을 충족하는 소프트웨어를 작성해야만 한다. 내장형 시스템의 요구사항이 증가함에 따라 내장형 소프트웨어의 복잡도가 증가하고 최적의 성능을 발휘하는 소프트웨어의 작성이 어려워지고 있다. 이에 따라 내장형 소프트웨어의 성능을 측정하고 분석하는 소프트웨어 프로파일링(Profiling) 도구의 필요성이 대두된다. 소프트웨어 프로파일링 도

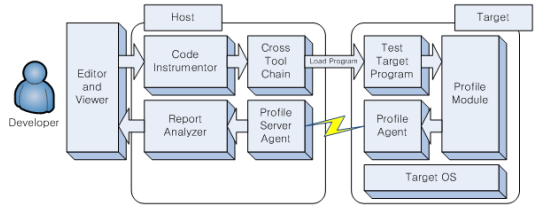
* 정 회 원 : 숭실대학교 대학원 컴퓨터학과 박사과정
coolman@ss.ssu.ac.kr
** 정 회 원 : 숭실대학교 컴퓨터학과 교수
cwyoo@ssu.ac.kr
[2009/06/22 투고 - 2009/07/01 심사 - 2009/08/26 심사완료]
☆ 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.

구란 개발 소프트웨어의 수행 과정과 그 과정에서 얻어지는 결과를 기록하고 분석하여 실행 환경에 최적화된 소프트웨어를 개발할 수 있는 기회를 주는 도구이다[1].

소프트웨어는 작성하는 알고리즘에 따라 같은 입출력을 갖더라도 많은 성능의 차이를 보일 수 있다. 그러므로 소프트웨어 프로파일링에 관한 요구는 일반적인 컴퓨팅 시스템에서도 있다. 일반적으로 프로파일링 도구는 두 가지 방법을 많이 사용한다. 그 중 한 방법은 운영체제의 시스템 콜(system call)이 발생할 때 로그를 발생시키는 방법이다[2][3]. 그리고 다른 한 방법은 프로그램 소스에 로그를 발생시키는 소스를 삽입하여 프로그램 실행 시 로그를 발생시키는 방법이다[4]. 이 중 첫 번째 소개한 시스템 콜이 로그를 발생시키는 방법은 프로그램의 실행 시간에 영향을 주지 않는 장점이 있으나 운영체제가 프로파일 기능을 제공해야 한다. 즉 다양한 운영체제에 같은 방법으로 적용하기 어렵다는 약점을 가진다. 로그를 발생시키는 소스를 삽입하는 방법은 프로그램 실행 시간에 로그를 발생 시키는 시간이 포함되는 약점을 가지나 다양한 환경에 적용하기 용이하여 교차 개발 환경에 적용하기 쉽다. 시스템 콜이 발생할 때 로그를 발생시키는 방법의 대표적인 예로는 GNU(GNU's Not UNIX) 프로젝트인 gprof[2]나 선 마이크로시스템(Sun Microsystems)에서 개발한 DTrace[3]가 있다. 하지만 이 도구들은 교차 개발 환경에 적용하기 어려운 약점을 가진다. 소스를 삽입하는 방법으로 교차 개발 환경에 적용하기 용이한 프로파일링 도구로는 IBM사의 RTRT(Rational Test RealTime)[4]가 있다. RTRT는 교차 개발 환경에서 실시간으로 내장형 시스템 소프트웨어의 테스팅 및 프로파일링을 실시할 수 있다. 하지만 직관적인 뷰어를 제공하고 있지 않아 프로파일링 결과를 분석하기 어렵다.

국내에서는 RTOS(Real Time Operating System)에 적용할 수 있는 원격 멀티 태스크 디버거인 Qplus가 연구되었다[5]. Qplus는 운영체제 커널에 디버깅을 위한 디버그 에이전트를 삽입하여 디버깅이 가능하도록 설계하였다. Qplus는 시스템 콜에 로그를 발생시키는 방법으로 다양한 운영체제에 적용하기 어렵다.

내장형 시스템의 개발 환경은 일반적인 컴퓨팅 시스템과 달리 호스트(Host)/타겟(Target) 구조의 교차 개발 환경(Cross-Platform Environment)을 이용한다. 교차 개발 환경은 내장형 시스템과 같이 컴퓨팅 자원이 적고 입출력 디바이스에 제약이 있는 경우 많이 사용하는 개발 환경이다. 교차 개발 환경에서 개발자는 컴퓨팅 자원이 풍부하고 입출력 디바이스의 제약이 적은 호스트 컴퓨터에서 프로그램을 개발하고 교차 컴파일러(Cross Compiler)를 이용하여 타겟 컴퓨터에서 실행될 목적 코드(Object Code)를 생성한다. 그림 1은 제안하는 프로파일링 도구의 호스트/타겟 구조를 보인다.



(그림 1) 프로파일링 도구의 구조도

제안하는 도구는 그림 1과 같이 실시간으로 프로그램을 프로파일하기 위해 개발자가 작성한 원시 소스 코드(Original Source Code)에 로그를 발생시키는 코드를 삽입하여 테스트 타겟 소스 코드(Test Target Source Code)를 생성한다. 타겟 소스 코드는 타겟 시스템에 로드되어 실행될 프로그램 소스 코드를 의미한다. 생성한 테스트 타겟 소스 코드는 실행 시 저 수준의 테스트 로그를 생성한다.

본 논문은 내장형 소프트웨어의 성능 측정을 위해 교차 개발 환경에 적합한 XML 기반의 프로파일링 도구를 제안한다. 제안하는 도구는 호스트 환경에서 컴파일링하여 타겟 시스템에서 실행할

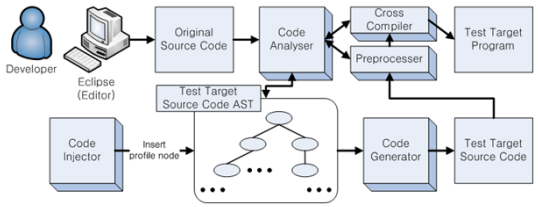
수 있는 교차 개발 환경과 실행 결과를 실시간으로 호스트 환경에서 확인할 수 있는 GUI 기반의 호스트 뷰어를 제공한다. 또한, 호스트 개발 환경은 이클립스(eclipse)[6]의 플러그인(plug-in)형태로 제공하여 이클립스가 제공하는 디버거(debugger)나 SVN(Subversion)[7]과 같은 다른 개발 도구도 함께 사용할 수 있는 장점을 가진다. 그리고 프로파일링에 관한 로그를 XML로 생성하여 저장하고 이를 분석하여 직관적인 GUI 뷰어를 통해 사용자에게 제공한다. XML은 현재 범용적으로 사용하고 있는 데이터 저장 방식으로 다른 많은 응용에서 사용하고 있다. XML로 생성한 저 수준의 로그는 사용자가 요구하는 다양한 형태로 뷰잉할 수 있는 장점을 가진다.

본 논문은 2장에서 제안하는 도구의 구조를 보이고 3장에서 제공하는 소프트웨어 프로파일의 종류와 뷰잉에 관해 논한 후 4장 실험에서 본 도구의 실험 결과를 기술하고 5장에서 결론을 맺는다.

2. 제안하는 도구의 구조

제안하는 도구는 호스트와 타겟으로 나누어 구성된다. 개발자가 편집기를 통해 작성한 프로그램은 코드 삽입기(Code Instrumentor)를 통해 로그를 생성하는 프로그램이 삽입된다. 로그를 생성하는 코드는 교차 도구 체인(Cross Tool Chain)을 통해 타겟 시스템에 로드되고 이 테스트 타겟 프로그램(Test Target Program)이 실행되며 로그를 생성한다. 이 로그는 실시간으로 호스트 컴퓨터에 전송되고 결과 분석기(Report Analyzer)를 통해 XML로 변환된다. 생성된 XML은 프로파일의 결과로 뷰어를 통해 그래프와 표로 사용자에게 제공된다.

2.1 호스트 시스템 구조



(그림 2) 테스트 타겟 프로그램 생성기

본 도구의 호스트 시스템은 두 가지 부분으로 나누어 설계되어 있다. 한 부분은 원시 코드를 입력으로 받아 분석하여 테스트 타겟 프로그램을 생성하는 코드 삽입기이다. 그리고 다른 부분은 테스트 타겟 프로그램과 통신하며 로그를 생성하고 이를 분석하여 보고서를 생성하는 리포트 뷰어이다. 그림 2는 호스트 시스템에서의 타겟 프로그램 생성기의 구조를 보인다.

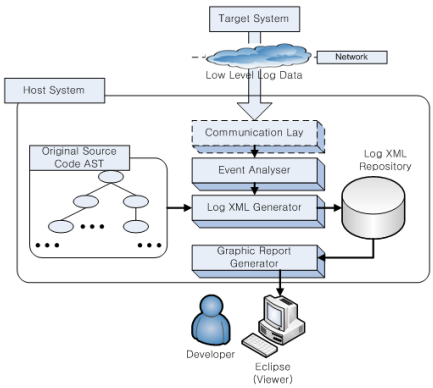
개발자는 그림 2와 같이 편집기를 이용하여 원시 소스 코드를 생성한다. 개발자가 작성한 원시 프로그램은 코드 분석기(Code Analyzer)를 통해 분석하여 AST(Abstract Syntax Tree)를 생성하고 로그를 생성하는 소스에 해당하는 노드(Node)를 트리에 삽입한다. 코드 생성기(Code Generator)는 AST를 입력으로 받아 컴파일 가능한 테스트 타겟 소스 코드를 생성한다. 이 소스를 교차 컴파일(Cross Compile)하여 타겟 서비스에서 실행 가능한 테스트 타겟 프로그램을 생성한다. 표 1은 테스트 타겟 프로그램을 생성하기 위한 알고리즘이다.

(표 1) 코드 삽입 알고리즘

```
1   class TargetCodeGeneratorVisitor extends Visitor{
2   TargetCodeAST tn = new TargetCodeAST();
3   public void visit(FunctionStartNode on){ //함수의 시작 위치
4   tn.add(on);
5   tn.add(sessionIdStartCallNode); //함수 시작의 프로파일 정보 생성
6   if((parameters = on.getParameters()).length() != 0){
7     TargetParameterNode paraNode = bound(parameters);
8     tn.add(paraNode);}
9   for(int i = 0; i < on.childNodes.size(); i++){
10  on.childNodes.elementAt(i).accept(this);}}
11  public void visit(DeclLocalVariableNode on){
12  //함수 지역변수 선언부 시작
13  tn.add(on); //함수 지역변수에 관한 프로파일 정보 생성
14  if(localVariables = on.getLocalVariables()).length() != 0){
15  TargetLocalVariableNode variableNode = bound(localVariables);}
16  for(int i = 0; i < on.childNodes.size(); i++){
17  on.childNodes.elementAt(i).accept(this);}}
18  public visit(StatementNode node){
19  tn.add(node);}
20  public void visit(LoopNode on){
21  //반복문 시작, 반복문에 관한 프로파일 정보 생성
22  tn.add(loopStartCallNode);
23  tn.add(on);}
24  public void visit(FunctionReturnNode on){
25  //함수 반환에 관한 프로파일 정보 생성
26  tn.add(sessionReturnNode);
27  tn.add(on);}
28  public TargetVairiableNode[] bound(VairiableNode[] node){
29  //변수에 관한 의미(semantic) 정보를 분석하여 반환}}
```

표 1의 코드 삽입 알고리즘은 프로파일 대상 프로그램의 AST를 순회하면서 프로파일 로그가 필요한 위치에 로그 생성 코드를 삽입한다. 알고리즘에서 3줄과 20줄, 24줄에 해당하는 visit 함수는 대상 프로그램의 블록에 해당하는 AST 노드에서 블록의 시작/끝 로그 생성 코드를 삽입한다. 그리고, 11줄과 28줄에 해당하는 visit 함수는 대상 프로그램의 지역변수에 관한 로그 생성 코드를 생성한다.

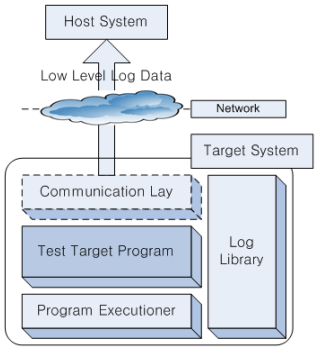
호스트 시스템은 타겟 시스템에서 생성한 저수준의 로그를 분석하여 로그 XML을 생성한다. XML은 범용적으로 사용하고 있는 데이터 저장 방식으로 XSLT(Extensible Stylesheet Language Transformations)을 이용하면 다양한 형태로 쉽게 변환이 가능한 장점을 가지고 있다. 로그 XML은 그래픽 보고서 생성기(Graphic Report Generator)로 GUI 기반의 보고서를 생성한다. 그림 3은 호스트 시스템의 보고서 생성기의 구조를 보인다.



(그림 3) 호스트 시스템의 보고서 생성기

2.2 타겟 시스템 구조

일반적으로 내장형 시스템은 컴퓨팅 자원이 부족하다. 그러므로 타겟 시스템을 경량화할 필요가 있다. 또한 경우에 따라서는 로그를 저장하는 저장장치를 보유하고 있지 않을 수 있다. 그러므로 생성하는 저 수준의 로그는 통신을 통해 호스트 시스템으로 전송된다. 그림 4는 타겟 시스템의 구조를 보인다.



(그림 4) 타겟 시스템의 구조

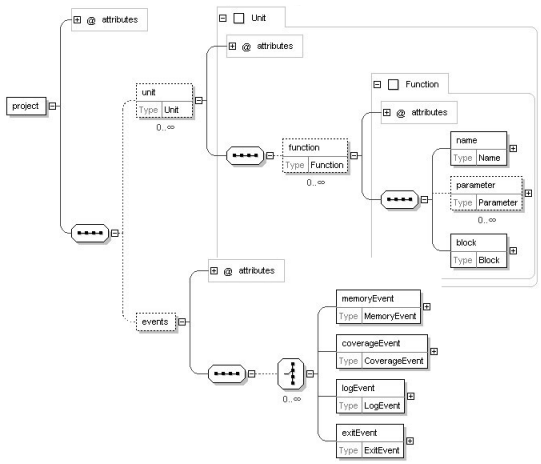
그림 4와 같이 타겟 시스템은 프로그램 실행기(Program Executioner)와 통신 계층(Communication Lay), 로그 라이블러리(Log Library)로 구성된다. 본 도구의 타겟 시스템은 자원이 적은 내장형 시스

템에 합당하도록 경량화하여 설계한다. 테스트 타겟 프로그램은 삽입된 코드에 의해 저 수준의 로그를 발생시키고 이를 호스트에 전송한다.

가공되지 않은 저 수준의 로그는 분석하여 가독성이 높은 XML 로그로 변환한다. XML은 현재 다양한 응용에서 사용되고 있으며 XSLT를 이용하여 다수의 응용에서 사용할 수 있는 장점을 가진다.

2.3 타겟 프로그램의 XML 로그

제안하는 도구는 내장형 시스템의 소프트웨어 개발에 사용하기 용이하도록 호스트/타겟으로 구성되어 있다. 타겟 시스템에서 생성한 저 수준의 데이터는 호스트 시스템에서 로그 XML로 변환한다. 그림 5는 로그 XML의 스키마이다.



(그림 5) 프로그램 정보와 로그 XML의 스키마

로그 XML은 테스트 프로그램의 구조정보와 이벤트로 구성되어 있다. 테스트 프로그램의 구조정보는 뷰잉 도구에서 프로그램 소스의 분석 없이 보고서를 생성할 수 있는 정보를 제공한다. 그리고 이벤트는 메모리 이벤트와 커버리지 이벤트, 로그 이벤트, 종료 이벤트로 구성되어 있다. 아래 항목은 네 가지 이벤트를 보인다.

- 메모리 이벤트(Memory Event) : 메모리 할당과 해제에 관한 이벤트
- 범위 이벤트(Coverage Event) : 소스 프로그램 중 실행된 블록에 관한 이벤트
- 로그 이벤트(Log Event) : 실행 시간에 관한 이벤트
- 종료 이벤트(Exit Event) : 프로그램 종료 이벤트

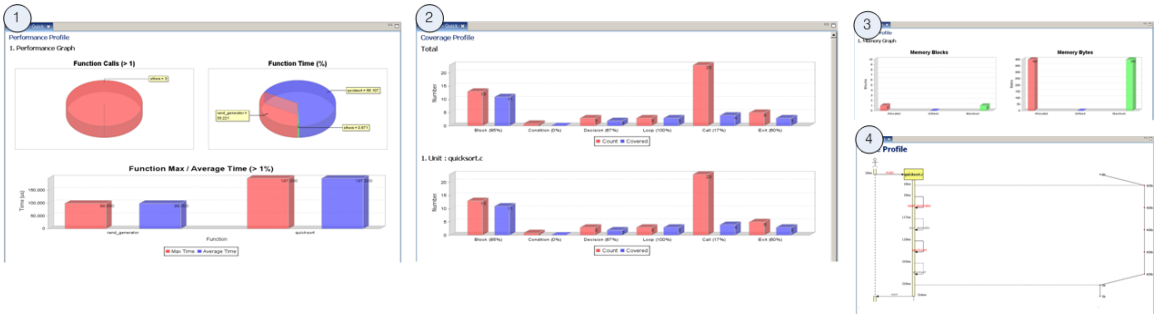
위와 같은 네 가지 이벤트는 조합하여 네 가지 프로파일 정보를 생성한다.

3. 소프트웨어의 프로파일

일반적으로 제한된 자원을 제공하는 내장형 시스템의 소프트웨어는 다음과 같은 요구사항을 만족해야 한다[8][9].

- 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.
- 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 작아야 한다. 또한, 메모리 누수를 방지하기 위해 한번 할당된 메모리는 프로세스 종료 전까지 반드시 해제되어야 한다.

이를 위해 제안하는 도구는 다음과 같은 4가지 부분에 대한 성능 분석 모듈을 포함한다. 본 장에



(그림 6) 그래픽 결과 뷰어

서는 제안하는 도구의 프로파일링 방법과 뷰어의 구조에 관해 논한다.

성능 프로파일은 응용 프로그램의 전체 또는 단위 프로그램의 실행에 걸리는 시간을 측정한 결과이다. 본 도구의 프로파일 결과는 함수단위로 아래와 같은 항목을 갖는다.

- 성능 프로파일의 항목
  - Call : 함수가 호출된 횟수
  - F time : 함수가 실행된 총 시간(다른 함수가 호출되어 실행된 시간 제외)
  - F+D time : 함수가 실행된 총 시간(다른 함수가 호출되어 실행된 시간 포함)
  - Min F time : 함수의 호출 중 가장 짧게 실행된 시간
  - Max F time : 함수의 호출 중 가장 길게 실행된 시간
  - Avg F time : 함수의 평균 호출 시간

코드 범위 프로파일은 응용 프로그램 실행 시 실제로 사용된 부분과 사용되지 않은 부분을 측정한 결과이다. 본 도구의 코드 범위 프로파일 결과는 함수단위로 아래와 같은 항목을 갖는다.

- 코드 범위 프로파일의 항목
  - Function and Exits : 함수의 총 개수와 실행된 함수의 개수
  - Blocks : 블록의 총 개수와 실행된 블록의 개수
  - Conditions : 조건문의 총 개수와 실행된 조건문의 개수
  - Decisions : 분기 결정문의 총 개수와 실행된 분기 결정문의 개수
  - Loops : 반복문의 총 개수와 실행된 반복문의 개수
  - Calls : 함수 호출의 개수와 실행된 함수 호출의 개수

메모리 프로파일은 메모리 할당과 해제 정보 그리고 전체 메모리 사용량, 메모리 누수 등을 검사한 결과이다.

- 메모리 프로파일의 항목
  - Allocated : 할당된 메모리의 개수와 크기
  - Unfreed : 할당되고 해제되지 않은 메모리의 개수와 크기
  - Maximum : 프로그램 수행 중 가장 많은 메모리를 할당하였을 때의 값

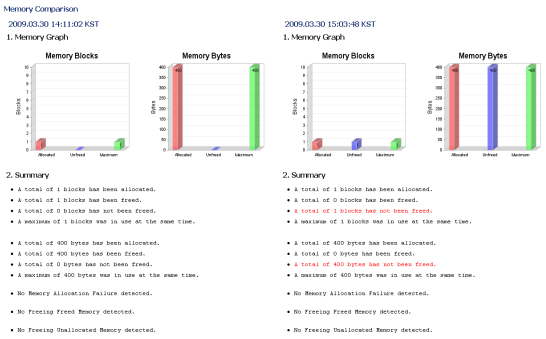
트레이스 프로파일은 응용 프로그램의 실행과 함수의 호출 순서를 검사한 결과이다. 프로그램의 트레이스를 표현하는 방법으로는 UML(Unified Modeling Language)[10]의 시퀀스 다이어그램(Sequence Diagram)이 있다[11]. 표 2는 제안하는 시스템과 gprof, DTrace의 프로파일 항목을 비교하여 보여준다.

(표 2) 프로파일 항목 비교

시스템 / 항목	제안하는 시스템	gprof	DTrace
성능	Call	Call	Call
	F time	self Ts/call	기본적으로 제공하지 않고 스크립트로 작성 가능
	F+D time	total Ts/call	
	Min F time	제공하지 않음	
코드범위	Fun & Exit	제공하지 않음	기본적으로 제공하지 않고 스크립트로 작성 가능
	Blocks		
	Conditions		
	Loops		
	Calls		
메모리	Allocated	제공하지 않음	기본적으로 제공하지 않고 스크립트로 작성 가능
	Unfreed		
	Maximum		
실시간 트레이스	UML	제공하지 않음	제공하지 않음

본 도구는 프로그램의 트레이스를 UML의 시퀀스 다이어그램을 통해 표현한다. 그림 6은 앞에서 기술한 네 가지 프로파일 정보를 보이는 뷰어 화면이다. 그림 6의 ①는 성능 프로파일의 뷰잉 결과 화면이다. 화면의 파이 그래프(Pie Graph)는 전체 실행 시간에서 각 함수가 실행된 비율을 표현하고 막대 그래프를 이용하여 각 함수의 절대적인 실행 시간을 표현한다. ②는 코드 범위 프로파일의 결과 화면이다. 코드 범위 프로파일은 막대 그래프로 실행에 사용된 코드와 사용되지 않은 코드를 문장단위로 표현한다. ③은 메모리 프로파일을 막대 그래프로 표현한 결과 화면이다. ④는 트레이스 프로파일을 UML 시퀀스 다이어그램을 표현한 결과 화면이다.

이와 같은 GUI 기반의 프로파일 결과는 로그 XML을 분석하여 생성하고 저장된 로그 XML을 이용하여 이전에 실행한 프로파일의 결과를 확인할 수 있다. 또한, 생성된 두 로그 XML을 이용하여 그림 7과 같은 두 프로파일 결과를 비교할 수 있다. 프로파일 결과는 프로그램 소스에 따라 또는 테스트 데이터에 따라 달라질 수 있다.



(그림 7) 두 프로파일의 비교 화면

그림 7의 오른쪽 그래프는 메모리를 사용하고 해제하지 않아 해제하지 않은 메모리(Unfreed) 항목이 증가해 있다. 분석한 프로파일 결과를 바탕으로 메모리 해제에 관한 소스를 첨가한 후 다시 프로파일링하여 왼쪽 그래프와 같은 그래프로 모든 메모리가 해제되었음을 확인할 수 있다.

4. 실 험

본 논문은 제안하는 도구를 실험하기 위해 행렬의 곱을 이용하여 도구의 성능을 평가한다. 행렬 곱의 시간 복잡도는 O(n^4)이다. 행렬의 곱 연산은 시간 복잡도가 높아 시스템의 성능측정에 많이 이용되고 있다.

제안하는 도구를 이용하여 행렬 곱의 성능을 측정한다. 실험 대상이 되는 행렬 곱 프로그램은 init과 matrix, 두 개의 함수로 구성되어 있다. init은 행렬의 값을 초기화하는 함수이고 matrix는 행렬의 한 행과 열을 연산하는 함수이다. 그러므로 init 함수는 프로그램에서 1회 호출되고 matrix는 연산에 첫 번째 행렬의 행의 개수와 동일하게 호출된다.

제안하는 프로파일링 도구는 소스 프로그램에 로그 생성 코드를 삽입하여 테스트 타겟 프로그램을 생성한다. 생성한 프로그램은 타겟 시스템에 로드하여 실행한다. 본 장에서는 각 알고리즘의 테스트 결과와 코드 삽입으로 인한 실행 시간의 변화를 분석한다. 표 3은 테스트 대상이 되는 알

고리즘의 실행 시간의 비율을 보인다. 표 3에서 matrix 전체시간은 다수 호출된 함수의 전체시간의 비율을 의미하고 matrix 평균시간은 한 번 호출된 실행 시간의 평균을 의미한다.

(표 3) 성능 측정(단위 %)

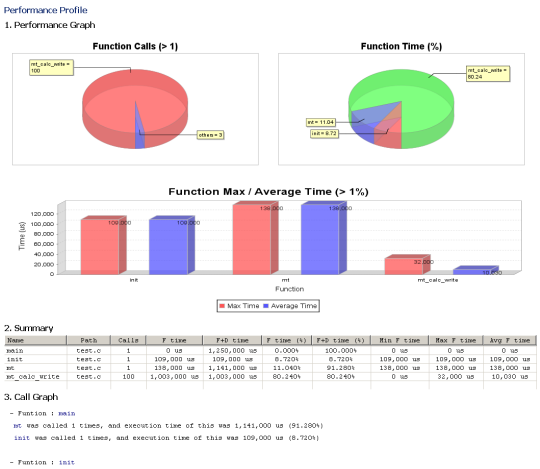
항목 \ 함수	init	matrix 전체시간	matrix 평균시간
제안하는 도구를 이용한 측정	8.72	91.28	0.8
도구를 사용하지 않고 측정	4.81	95.18	0.9

표 3은 각 함수의 실행 시간을 제안하는 도구를 사용한 측정과 도구를 사용하지 않고 측정한 형태로 보인다. 도구를 사용할 경우 실행 시간은 제안하는 도구를 이용한 경우 실행 시간이 증가한다. 이는 로그를 발생 시키는 코드를 추가하여 발생한다. 추가되는 코드는 프로그램 전반에 추가되어 실행시간은 전체적으로 증가한다. 성능의 분석은 비율에는 영향을 주지 않아 전체 프로그램 중 시간을 많이 소모하는 함수와 적게 소모하는 함수를 파악할 수 있다. 표 3과 같이 제안하는 도구를 사용할 경우와 사용하지 않을 경우 비율에는 큰 영향을 주지 않는다.

그림 8은 측정한 결과에 해당하는 그래프 기반 보고서이다. 이 그래프 기반의 보고서는 개발자에게 함수의 실행 비율의 정보를 직관적으로 제공하여 빠른 정보 수취에 도움을 준다.

5. 결 론

내장형 시스템의 요구사항이 증가함에 따라 프로그램의 복잡도도 증가하고 있다. 내장형 시스템은 낮은 컴퓨팅 자원을 갖는 특성이 있다. 프로그램의 복잡도 증가는 최적의 프로그램을 작성하기 어렵게 한다. 이에 따라 내장형 시스템의 프로그램에 대한 성능 측정이 요구된다. 내장형 시스템은 호스트/타겟 구조의 교차 개발 환경을 이용한다.



(그림 8) 그래픽 기반의 성능측정 보고서

본 논문은 교차 개발 환경에서 사용 가능한 호스트/타겟 구조의 프로파일링 도구를 제안한다. 제안하는 도구는 호스트 시스템에서 사용자가 작성한 프로그램에 자동으로 로그를 생성하는 루틴을 추가한다. 로그 생성 루틴이 추가된 프로그램은 교차 컴파일러로 컴파일링하여 타겟 시스템에 업로드하고 실행하여 프로그램 실행 시 저 수준의 로그를 생성한다. 저 수준의 로그는 호스트 시스템에서 분석하여 가독성이 높은 로그 XML 문서로 변환한다. XML은 범용적으로 사용하고 있는 데이터 저장 방식으로 많은 응용에서 사용하고 있고 XSLT를 이용하여 쉽게 다른 형태의 XML로 변환할 수 있어 사용자가 요구하는 다양한 형태의 뷰잉을 지원할 수 있는 장점을 가진다. 그리고 사용자에게 직관적인 뷰잉을 제공하기 위해 로그 XML을 분석하여 GUI 뷰어를 통해 프로파일 결과를 제공한다. 또한, 본 도구는 이클립스 플러그인 기반으로 개발되어 이클립스가 제공하는 다른 기능을 그대로 사용할 수 있다.

제안하는 도구에서 생성하는 보고서는 성능 프로파일과 코드 범위 프로파일, 메모리 프로파일, 트레이스 프로파일로 구분하여 생성한다. 프로파일의 결과는 프로그램의 입력이 되는 테스트 데이터에 따라 달라질 수 있는데 이를 위해 두 프로

파일링 결과를 비교 분석할 수 있는 화면을 제공한다. 이는 개발자에게 내장형 시스템 프로그램의 성능 향상을 위한 기회를 제공한다.

참 고 문 헌

[1] 곽동규, 조용윤, 유재우, “임베디드 소프트웨어를 위한 프로파일링 도구의 설계 및 구현”, 2004 한국정보처리학회 추계발표대회 논문집, 제11권, 제2호, 2004년 11월.

[2] GUN gporf, http://www.gnu.org.

[3] DTrace, http://www.solarisinternals.com/wiki/index.php/DTrace_Topics.

[4] RTRT, http://www.ibm.com/developerworks/download/s/r/rtrt.

[5] 이광용, 김흥남, “Qplus-TRTOS를 위한 원격 멀티 태스크 디버거의 개발”, 컴퓨팅의 실제 제9권 제4호, 2003년 8월.

[6] Eclipse, http://www.eclipse.org.

[7] Subversion, http://subversion.tirgris.org.

[8] Bart Broekman, *Testing Embedded System Addisson-wsley*, Dec 2002.

[9] L. Hatton, Embedded *Software Testing, Software Testing Congress*, 2000.

[10] UML, http://www.omg.org/spec/UML/2.0/

[11] Petri Kukkala, Jouni Riihimaki, Marko Hannikainen, Timo D. Hamalainen, Klaus Kronlof, “UML 2.0 Profile for Embedded System Design”, Proceedings of the 25th IEEE International Real-Time System Symposium (RTSS’04), pp459-468, 2004.

[12] Dr. Neal Stollon, Rick Leatherman, Bruce Ableidinger, “Multi-Core Embedded Debug for Structured ASIC System”, proceedings of Design Con 2004, Feb 2004.

[13] Donggyu Kwak, Yonggyun Cho, Jaeyoung Choi, Chae-Woo Yoo, “A XML-Based Testing Tool for Embedded Software”, 2007 International Conference on Multimedia and Ubiquitous Engineering, 2007.

◐ 저 자 소 개 ◑

곽 동 규

2002년 서경대학교 응용수학과 졸업(학사)
2004년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2004년~현재 숭실대학교 대학원 컴퓨터학과 박사과정
관심분야 : 프로그래밍 언어, 컴파일러, XML, 임베디드, 유비쿼터스, etc.
E-mail : coolman@ss.ssu.ac.kr

유 재 우

1976년 숭실대학교 전자계산학과 졸업(학사)
1985년 한국과학기술원 대학원 전산학과 졸업(박사)
1983~현재 숭실대학교 컴퓨터학과 교수
1986~87년, 1996~97년 코넬대학교 객원교수
1999~97년 한국정보과학회 프로그래밍언어 연구회 위원장
관심분야 : 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호작용, etc.
E-mail : cwyoo@ssu.ac.kr