비트마스킹 기법을 이용한 임베디드 모니터링 시스템

신 원[†] · 김 태 완[††] · 장 천 현[†††]

요 약

임베디드 소프트웨어의 적용범위가 넓어짐에 따라 소프트웨어의 개발시간을 줄이기 위한 많은 개발도구가 개발되었다. 하지만, 기존 도구들은 특정 플랫폼 적용을 목적으로 개발되었기 때문에 적용 가능한 범위가 제한된다. 이러한 문제 해결을 위하여 다양한 통신 환경을 지원함으로써 플랫폼 제약을 제거한 임베디드 모니터링 시스템을 개발하였다. 임베디드 모니터링 시스템은 코드 인라인 기법을 통하여 모니터링 과정을 진행한다. 하지만, 코드 인라인 기법은 모니터링 센서의 수행에 대한 부하문제를 가지고 있기 때문에 모니터링 센서의 최적화에 대한 고려가 필요하다. 이에 본 논문에서는 모니터링 과정 초기화 방안과 모니터링 센서 최적화를 위한 비트마스킹 기법을 제안한다. 개발된 임베디드 모니터링 시스템은 임베디드 시스템을 사용하는 모든 분야에 적용가능하다.

키워드 : 모니터링 시스템, 임베디드 시스템, 임베디드 소프트웨어

Embedded Monitoring System using Bit-masking Technique

Won Shin[†] · Tae Wan Kim[††] · Chun Hyon Chang[†††]

ABSTRACT

As the embedded software spreads in various areas, many development tools have been made to minimize the developing time. But these tools cannot be applicable to all the environment because they have been created for the specific platform. As such, this paper proposes an Embedded Monitoring System, which supports the various communication environment and removes the limitation of adaptability to the various platforms. Using the Code Inline technique, this system can perform the monitoring process. However, we should consider the optimization for the monitoring process and monitoring sensors because the technique has the monitoring sensor overhead. As such, this paper proposes an approach for initializing the monitoring process and a bit-masking technique for optimizing the monitoring sensor. The Embedded Monitoring System will be applicable to all the areas using embedded systems.

Key Words : Monitoring System, Embedded System, Embedded Software

1. 서 론

임베디드 장치는 가정, 자동차, 사무실에서부터 원격 어셈블리 설비 및 해상 드릴링 플랫폼에 이르는 거의 모든 분야에 이용되고 있다. 이렇듯 많은 분야에서의 임베디드 장치가 사용됨에 따라 임베디드 시스템 제조업자들은 빠른 시간 내에 각각의 분야에 맞는 시스템을 개발해야 하는 어려움이 따르게 되었다. 임베디드 장치는 크게 임베디드 하드웨어와 소프트웨어로 나뉘게 되는데 하드웨어의 경우 이러한 빠른 변화에 적용할 수 있는 많은 기술들이 존재하여 개발 가능하다. 하지만 소프트웨어의 경우 아직까지도 개발환경이 미흡하여 소프트웨어를 빠르게 개발하고 유지보수 할 수 있는 개발도구들의 중요성이 대두되고 있다.

소프트웨어의 동작과정에서의 오류를 찾아내기 위해 사용되는 모니터링 도구는 여러 종류의 임베디드 소프트웨어 개발도구 중에서도 소프트웨어의 개발기간 단축에 가장 큰 역할을 한다. 하지만, 기존의 모니터링 도구는 특정 플랫폼 적용을 목적으로 개발되었기 때문에 적용 가능한 환경이 제한된다. 또한, 모든 변수에 대한 데이터 추출을 위하여 시스템의 자원사용량과 처리량을 증가시킴으로써 많은 부하를 발생시키기 때문에 임베디드 환경처럼 제한된 자원을 갖는 환경에는 적용 불가능하다. 이러한 문제를 해결하기 위하여 특정 대상에 대한 값만을 도출함으로써 모니터링으로 인한 부하를 줄이는 모니터링 센서기법을 적용하고 다양한 통신 환경을 지원함으로써 특정 플랫폼에 제한되지 않는 임베디드 모니터링 시스템을 개발하였다[2]. 하지만, 이러한 임베디드 모니터링 시스템은 모니터링 센서의 동작과정이 직접적으로 응용 프로그램의 동작과정에 영향을 미침으로써 모니

※ 이 논문은 2005년도 건국대학교 학술진흥연구비 지원에 의한 논문임.
† 준 회 원 : 건국대학교 컴퓨터공학과 박사과정
†† 정 회 원 : 건국대학교 컴퓨터공학과 박사과정
††† 종신회원 : 건국대학교 컴퓨터공학과 정교수
논문접수 : 2006년 2월 9일, 심사완료 : 2006년 6월 7일

터링 데이터의 신뢰성을 저하시킨다는 점과 데드라인 위반 시의 처리방안에 대한 고려가 없다는 문제점이 발생된다.

이러한 문제를 해결하기 위하여 본 논문에서는 임베디드 모니터링 시스템에서 모니터링 과정의 소프트웨어에 대한 영향력을 최소화하기 위하여 기존의 모니터링 센서의 처리 구조에 비트마스킹 기법을 적용하고 데드라인 위반 시의 메시지출력, 로그기록 등의 처리 방법론을 개발한다. 여기서 비트마스킹 기법이란 데이터 도출이 필요한 데이터와 필요하지 않은 데이터를 구분하기 위하여 사용하는 것으로서, 비트연산을 통하여 데이터 추출이 필요하지 않은 데이터의 값을 변형시키는 기법을 말한다.

본 논문은 2장에서 기존의 모니터링 시스템의 구조와 문제점을 제시하고, 3장에서는 모니터링 센서의 최적화방안에 대하여 기술하고 4장에서는 데드라인 검출 방안에 대하여 기술한다. 5장에서는 실행시간 분석을 위한 실험에 대하여 설명하고 마지막으로 6장에서 본 논문의 결론 및 향후 방향에 대하여 기술한다.

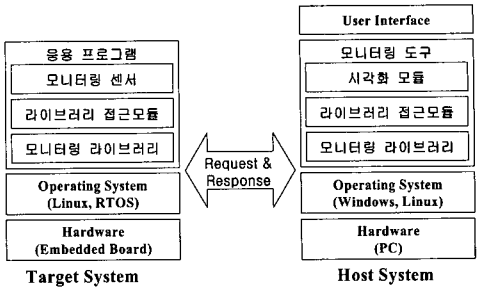
2. 관련연구

2.1 동적 프로그램 모니터링 기술

동적 프로그램 모니터링 기술이란 응용프로그램 수행 중에 프로그램의 이벤트, 변수 등을 감시하여 프로그램이 동작 상태를 점검하고 문제발생 시에 문제의 원인을 분석하기 위한 기술이다. 이러한 동적 프로그램 모니터링 기술은 크게 커널의 지원을 통한 방법, 인터프리터에 의해 수행되는 방법, 프로그램을 변환하는 방법으로 나뉜다. 커널의 지원을 통한 방법은 운영체제의 지원 없이는 구현하기 힘든 방법이며, 문맥-전환으로 인한 오버헤드가 크다는 문제점이 있다. 인터프리터에 의한 방법은 실행 속도를 크게 저하시킨다는 문제점이 있다[3-6]. 프로그램을 변환하는 방법은 코드 인라인 기법이라고 부르며, 데이터 도출을 위한 센서를 소스레벨에서 삽입하고 실행시간에 삽입된 센서를 통하여 데이터를 도출하는 기법을 말한다[1]. 코드 인라인 기법은 각 응용에 따라서 필요한 부분만을 효과적으로 모니터링 할 수 있는 방법으로 그 효율성으로 인해 최근 들어 주목받고 있다. 하지만, 삽입되는 센서에 의한 부하문제가 발생하고 이러한 부하는 응용프로그램의 실행에 직접적으로 영향을 미칠 수 있기 때문에 모니터링에 대한 신뢰성을 저하시킬 수 있다는 문제점이 있다. 현재까지는 이러한 문제를 해결할 수 있는 방안이 없는 실정이다. 이에 이러한 문제를 해결하기 위한 노력이 필요하다[6, 7].

2.2 기존 임베디드 모니터링 시스템 구조

기존에 모니터링 과정을 진행하기 위해서는 디버깅모듈을 삽입하고 디버깅모듈에 의하여 모든 값들을 추출한다. 추출된 데이터를 기반으로 개발자는 프로그램의 문제점을 분석하고 해결한다. 이러한 구조에서는 디버깅모듈의 삽입으로 생기는 자원사용량의 증가와 모든 값을 추출하기 위하여 생기는 처리량의 증가로 인하여 많은 부하가 발생한다. 이렇게 발생되는 부하는 응용 소프트웨어의 동작과정에 영향을 미쳐 추출하는 모니터링 데이터의 신뢰성을 저하시킨다.



(그림 1) 모니터링 시스템의 전체 구조도

이러한 문제를 해결하기 위하여 기존의 디버깅 모듈을 대체할 수 있는 코드 인라인 기법을 적용한 임베디드 모니터링 시스템을 제안하였다. 임베디드 모니터링 시스템에서는 코드 인라인 기법을 통하여 모니터링 센서와 원시 소스가 결합된 형태인 모니터링 소스를 생성하고 생성된 모니터링 소스는 목적시스템에서 실행된다. 실행시간에 센서는 데이터를 도출하여 호스트 시스템으로 전송한다. 호스트 시스템은 전송받은 모니터링 데이터를 분석하여 저장하거나 시각화하여 개발자에게 보여준다. (그림 1)은 모니터링 시스템의 전체 구조도이다[2].

하지만, 임베디드 모니터링 시스템에서는 기존의 코드 인라인 기법의 근본적인 문제인 센서의 부하에 대한 문제를 해결하지 못하고 있기 때문에 이에 대한 연구가 필요하다.

2.3 문제 해결방안

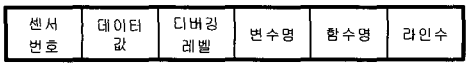
원시 소스에 삽입되는 모니터링 센서는 변수에 대한 값을 도출하는 역할을 수행한다. 센서의 최적화과정은 센서의 역할인 데이터 도출기능 외에 불필요한 부분을 제거하거나 보완하는 것이다. 기존의 센서에서는 디버깅 레벨을 비교하는 구문과 센서정보 테이블의 작성 여부를 비교하는 구문을 포함하고 있다. 일반적으로 임베디드 시스템에서 응용소프트웨어의 실행과정은 일련의 루틴이 무한 반복되는 형태이기 때문에 모니터링 센서의 두 가지 비교구문은 실행시간에 적지 않은 영향을 미칠 것이다. 이러한 문제를 해결하기 위하여 제안하는 구조에서는 실제 프로그램이 동작하기 이전에 비교구문을 대체하기 위한 방안으로 센서에 대한 정보를 미리 구성하고 디버깅 레벨을 비교하지 않도록 비트마스킹 기법을 사용한다.

데드라인 위반 시의 해결방안으로는 메시지 창을 출력함으로써 프로그램의 위험을 알리거나, 데드라인을 위반한 변수의 세부정보를 기록하는 로깅기법을 제공한다.

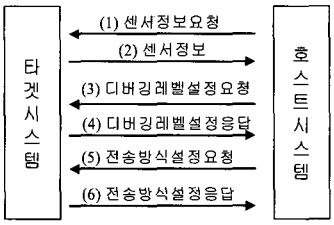
3. 모니터링 센서의 최적화 과정

3.1 모니터링 센서의 구조

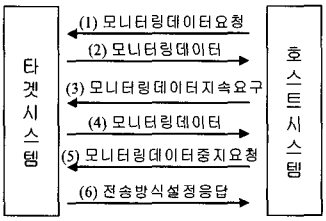
모니터링 과정에서 가장 핵심적인 역할을 수행하는 모니



(그림 2) 모니터링 센서의 구조



(그림 3) 초기화 단계



(그림 4) 전송 단계

터링 센서는 센서의 고유번호, 소속정보 그리고 디버깅 레벨을 갖게 된다.

디버깅 레벨은 디버깅레벨 기법에서 쓰이는 변수의 중요도를 나타내는 수치이다. 이러한 디버깅레벨 기법은 소스인라인 기법의 단점을 보완하기 위한 구조로서, 모니터링 대상에 0부터 9사이의 디버깅 레벨을 지정해 놓고 응용 프로그램의 실행시간 중에 모니터링 하고 싶은 레벨의 값을 변경함으로써 모니터링 대상을 변경할 수 있게 한다. 즉, 디버깅레벨 기법을 통하여 모니터링 대상 설정이 컴파일 시간 전에서 끝나는 것이 아니라 실행시간에 동적으로 변경 가능하게 한다.

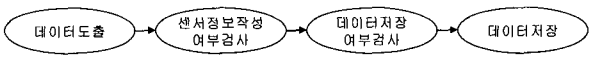
모니터링 센서의 동작과정은 크게 데이터 수집과 데이터 전송과정으로 나뉠 수 있다. 데이터 수집과정은 센서가 위치한 곳의 모니터링 데이터를 수집하여 데이터저장소에 저장하는 과정이다.

데이터 전송과정은 데이터전송을 위해 목적시스템에 디버깅레벨, 전송방식 등을 설정하고 목적시스템으로부터 센서정보를 받는 초기화단계와 모니터링 데이터를 전송받는 전송단계로 나뉜다.

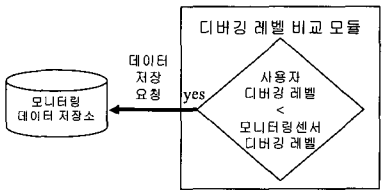
모니터링 데이터의 전송은 전송되는 패킷의 양을 줄이기 위하여 초기화단계에서 전송받은 모니터링 센서 정보를 이용하여 전송패킷을 구성한다. 전송패킷은 센서고유번호와 모니터링 데이터 값으로 구성된다.

3.2 모니터링 센서의 동작구조

기존의 모니터링 센서는 데이터를 도출하여 데이터를 저장하기 전에 센서정보를 생성하는 과정과 디버깅레벨을 비교하는 과정을 포함하고 있다. (그림 5)는 모니터링 센서의 모니터링 데이터 도출 과정을 나타낸다.



(그림 5) 기존 모니터링 센서의 데이터 도출 과정



(그림 6) 디버깅레벨 비교 동작 구조

모니터링 데이터가 전송되기 전에 목적시스템에서는 호스트시스템에 센서정보를 전송해야 하는데 이를 위해서 센서가 동작할 때 센서정보작성 여부에 따라 센서정보를 구성한다.

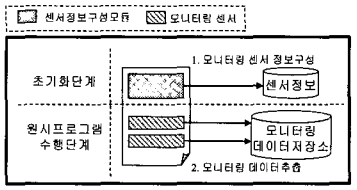
모니터링 데이터는 개발자가 요구하는 디버깅레벨보다 높은 디버깅레벨을 가지고 있는 모니터링 센서의 값을 도출한다. 이러한 이유로 모니터링 센서의 값은 센서가 가지고 있는 디버깅레벨에 의해 전송여부가 결정된다.

디버깅레벨 비교를 거쳐 저장여부가 결정되면 모니터링 데이터는 지정된 모니터링 데이터 저장소에 순차적으로 저장된다.

이러한 구조에서는 센서정보작성 여부를 결정하기 위한 비교구문과 데이터 값의 저장 여부를 결정하기 위한 비교구문으로 인해 많은 부하가 발생된다.

3.2.1 센서정보작성 여부 비교구문의 제거 방안

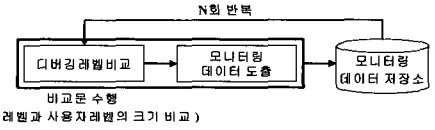
기존의 센서동작구조에서 문제가 되었던 센서정보작성 여부를 결정하기 위한 비교구문을 제거하기 위하여 센서정보 구성을 응용 프로그램이 동작하기 전인 초기화단계에서 구성한다. 이로써 응용프로그램의 실행시간에는 영향을 미치지 않는다.



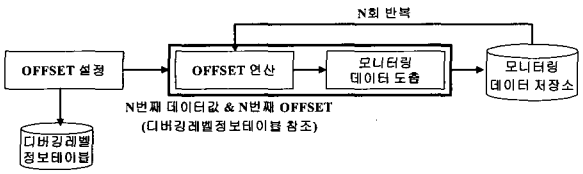
(그림 7) 센서정보작성 비교구문의 제거 방안

3.2.2 데이터 값의 저장 여부 비교구문의 제거 방안

기존 구조에서는 모니터링 센서가 동작할 때마다 각각의 센서마다 결정되어 있는 디버깅레벨에 따라서 모니터링 데이터 도출 여부를 검사한다. (그림 8)은 기존 모니터링 센서의 데이터 도출 단계이다.



(그림 8) 기존 모니터링 센서의 데이터 도출 단계



(그림 9) 비트마스킹 기법을 이용한 데이터 도출 단계

모니터링 센서의 최적화를 위하여 비교문의 수행을 줄일 수 있는 비트마스킹 기법을 사용한다.

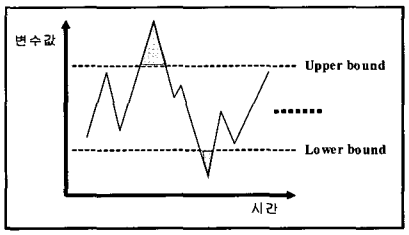
비트마스킹 기법을 사용하기 위해서는 센서 초기화단계에서 센서의 디버깅레벨과 사용자가 지정한 디버깅레벨을 비교하여 디버깅 레벨 정보 테이블을 구성한다. 모니터링 센서는 미리 구성된 디버깅 레벨 정보테이블의 값과 모니터링 데이터 값의 '&' 연산을 수행하여 모니터링 데이터 저장소에 저장한다. '&'연산의 수행은 추출이 불필요한 데이터의 경우에 데이터의 값을 0으로 바꾸게 된다.

기존 시스템의 구조에서 M개의 센서를 삽입하고 N번의 반복을 진행한다고 한다면, M*N회의 비교구문을 수행하여 모니터링 데이터를 추출하게 된다. 개선된 구조를 사용한다면 M회의 비교구문 수행으로 데이터 도출이 가능하다. 결국, 기존 구조에 비해 N분의 1회만큼의 비교구문 수행으로 데이터 도출을 가능하게 할 뿐만 아니라 M회의 비교구문 수행은 응용소프트웨어의 실행시간에 영향을 거의 미치지 않게 된다.

4. 데드라인 위반 검출 기능

응용프로그램 내에서 중요한 역할을 수행하는 변수가 데드라인을 위반할 경우에 시스템에 치명적인 손상을 일으킬 수 있다. 때문에 실행시간 모니터링과정에서 데드라인 위반 검출 및 처리는 가장 중요한 부분이다. 일반적으로 데드라인의 위반이라고 하는 것은 모니터링 데이터의 값이 지정된 데드라인의 범위를 넘는 경우를 말한다.

데드라인 위반 처리는 데드라인 위반 시에 곧바로 처리를 하는 경우와 위반 사항에 대한 로그파일을 작성하고 로그파일 분석을 통하여 소프트웨어의 오류부분을 찾아서 수정하는 경우로 나뉜다. 전자의 경우는 메시지를 출력하여 위반사항을 개발자에게 알려주는 것이고, 후자의 경우는 위반내용의 상세 내역을 파일로 기록하고, 기록된 내용을 주기적으로 분석하여 그 정보를 개발자가 원하는 시간에 보여주는 것이다.



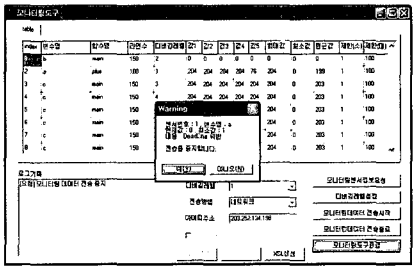
(그림 10) 데드라인 위반

데드라인 위반 검출을 하기 위해서는 기존의 모니터링 센서의 구조를 변경해야 한다. 모니터링 센서의 구조는 기본적인 센서의 소속정보만이 저장되어 있기 때문에 센서에 대한 데드라인의 범위를 처리할 수 있는 데드라인 최대값, 최소값 필드를 추가한다. 추가된 필드는 호스트시스템으로부터 모니터링 요청이 왔을 때, 기존 센서정보와 같이 전송하게 되고 실행시간 모니터링 중에 데드라인 위반 검출을 위하여 사용된다.

데드라인의 검출은 목적시스템으로부터 전송받은 센서의 데드라인 최대값, 최소값을 기반으로 이루어진다. 모니터링 데이터를 화면에 출력하기 전에 최대값과 최소값의 비교과정을 거쳐 데드라인 위반 여부를 결정한다. 데드라인을 위반하게 되면 에러 메시지를 출력하거나 로그파일에 위반에 대한 상세정보를 기록한다.

4.1 메시지 창 출력

데드라인 위반 검출이 되면 에러정보를 담은 메시지 창을 출력한다. 에러 정보에는 현재 에러를 발생시킨 센서의 정보와 위반 정보가 포함된다. 개발자는 메시지 창의 정보를 참조하여 프로그램의 중지 또는 지속 결정을 내리게 된다. (그림 11)은 데드라인 위반 검출 방식인 메시지 창 실행화면이다.



(그림 11) 모니터링 도구와 에러 메시지 창 출력화면

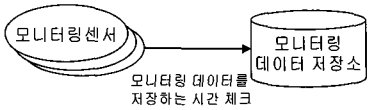
4.2 로그파일 기록

로그파일에 기록되는 정보는 모니터링 데이터의 세부정보와 데드라인을 위반한 시간이다. 모니터링 데이터의 정보에는 변수명, 소속함수, 센서번호, 데드라인 범위, 현재 값이 포함된다.

5. 실 험

5.1 실험구조와 목적

모니터링 센서에 대한 실행시간 측정을 위하여 동일한 응용프로그램에 일정한 수의 모니터링 센서를 삽입한다. 모니터링 센서가 동작하여 모니터링 데이터 저장소에 데이터를 저장하는 시간을 측정한다. (그림 12)는 실행시간 측정 구조이다.



(그림 12) 모니터링 센서 실행시간 측정 구조

〈표 1〉 모니터링 센서 실행시간 측정 방식

실행시간 = 센서실행시간 + 원시소스실행시간 센서 한 개당 실행시간 = (모니터링소스실행시간 - 원시소스실행시간) / 원시소스에 삽입된 센서개수

〈표 2〉 벤치마킹 프로그램 정보

프로그램명	실행시간(μs)	프로그램 설명
Bubble Sort	24.023	50개의 데이터 정렬
Compress	12.248	데이터 압축
Jdfctint	8.021	8*8 블록 이미지 DCT 변환

삽입되는 모니터링 센서의 종류는 최적화되지 않은 센서, 센서정보작성 비교구문을 제거한 센서, 데이터 값 저장여부 비교구문을 제거한 센서이다. 각각의 모니터링 센서에 대한 실행시간을 측정하여 모니터링 센서가 응용소프트웨어에 미치는 영향을 분석하고 최적화되지 않은 센서와 최적화된 센서의 실행시간 차이를 확인함으로써 최적화과정의 중요성을 인지할 수 있다.

실험환경은 응용프로그램이 동작하는 목적시스템과 모니터링 데이터를 전송받는 호스트시스템으로 구성된다. 실행시간 측정은 프로그램의 시작시간과 종료시간의 차이를 계산하여 구한다. 모든 프로그램의 동작에는 하드웨어 또는 운영체제에 의한 영향으로 인하여 측정할 때마다 많은 차이가 발생할 수 있다. 이러한 점을 고려하여 10,000번의 실험을 실시하고 상위와 하위 10%의 값을 뺀 나머지의 평균을 실험치로 사용한다. 또한 센서 한 개는 프로그램에서 한 번의 수행만을 하도록 한다. 그 이유는 반복문 안에 센서가 삽입될 경우 센서의 실행횟수는 반복문의 반복횟수와 같아지기 때문에 정확한 실험치의 측정이 불가능해지기 때문이다. 실험상에서 쓰이는 실행시간은 μs 단위이고 <표 1>의 식을 이용하여 측정한다.

실험에 사용되는 벤치마킹 프로그램은 <표 2>와 같으며 <표 1>의 식을 적용하기 위하여 필요한 데이터인 원시소스의 실행시간을 측정하여 기술하였다.

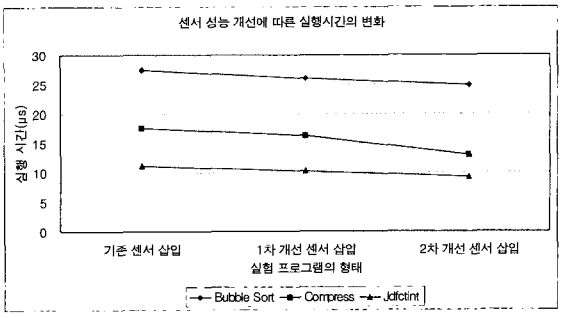
(1) 실험1 : 모니터링 센서의 구조 변화에 따른 실행시간을 측정하는 실험

모니터링 센서의 동작으로 생기는 부하가 클 경우, 원시소스의 실행에 영향을 미쳐 추출되는 모니터링 데이터의 신뢰성을 저하시킨다. 이에, 센서를 삽입한 경우와 센서를 삽입하지 않은 경우에 대한 실행시간을 측정하여 센서가 원시소스에 미치는 영향을 측정한다.

측정대상은 기존 모니터링 센서, 1차 개선된 센서, 2차 개선된 센서를 각각 10개씩 삽입한 소스이다.

<표 3>은 측정된 실행시간과 원시소스의 실행시간의 차를 나타낸다. 각각의 소스에 대한 실행시간을 <표 1>의 식을 이용하여 측정하여 센서 한 개의 실행시간과 센서의 성능향상 수치를 알아본다.

Bubble Sort의 경우, 기존 구조에서 개선된 센서구조는 3.355μs에서 0.840μs로 2.515μs의 성능이 향상되었고, 2차 개선된 센서 한 개의 실행시간은 0.08μs이다.



(그림 13)모니터링 센서의 개수 증가에 따른 실행시간

〈표 3〉 센서 성능 개선에 따른 실행시간의 변화치

(단위 : μs)

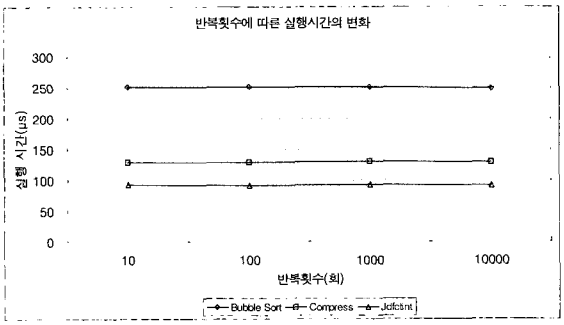
	Bubble Sort	Compress	Jdfctint
기존 센서	3.355	5.350	3.153
1차개선센서	1.983	4.001	2.295
2차개선센서	0.840	0.730	1.194

(2) 실험2 : 응용프로그램의 반복횟수를 증가하였을 때의 실행시간을 측정하는 실험

임베디드 시스템은 일반적으로 프로그램이 한번 동작하고 종료하는 형태가 아닌 무한적으로 반복하는 형태이다. 만약, 모니터링 센서가 반복적으로 실행될 때 부하의 누적현상이 발생한다면 이는 원시프로그램에 많은 문제를 발생시킬 수 있다. 이러한 임베디드 시스템의 특성을 고려하여 프로그램을 반복 수행하였을 때 센서의 실행시간 누적현상이 발생하는지 알아본다.

모니터링 센서 100개를 삽입하고 반복횟수를 증가하면서 실행시간을 측정한다.

100회 반복을 진행하였을 때, 실행시간은 약 251, 129, 92μs이다. <표 1>의 식을 이용하여 센서 한 개의 실행시간을 측정하면 실험 1에서와 같은 값이 측정된다. 즉, 이 실험을 통하여 반복횟수의 증가로 인하여 센서의 부하가 축적되어 실행시간이 증가되는 현상이 발생하지 않는다는 것을 알 수 있다.



(그림 14) 반복회수의 증가에 따른 실행시간

〈표 4〉 반복횟수 증가에 따른 실행시간의 변화

(단위 : μs)

	Bubble Sort	Compress	Jdfctint
100회	25.178	12.911	9.205
10000회	24.863	12.978	9.215

5.2 실험의 결론

전체실험의 목적은 모니터링 센서를 삽입하였을 때, 센서가 응용프로그램에 미치는 영향을 측정하는 것이다.

실험 1은 기존의 센서 구조와 개선된 구조의 성능 차이를 측정하는 실험으로서, Bubble Sort의 경우 2차 최적화 단계를 통하여 센서의 실행시간이 2.515μs 개선되었고 센서 한 개의 실행시간은 0.08μs이다.

실험 2는 일정한 센서를 삽입하고 반복회수를 증가시킴에 따라 센서의 부하가 축적되는 현상이 발생되는 것인지를 측정하기 위한 실험으로 반복횟수는 센서의 실행시간에 영향을 미치지 않는다는 것을 확인할 수 있다.

실험 1에서는 센서 10개, 실험 2에서는 센서 100개를 삽입하고 측정한 결과 센서 한 개의 실행시간은 같게 측정되었다. 이는 센서의 개수가 증가하더라도 실행시간 축적현상이 발생하지 않고 개수에 비례적으로 증가한다는 것을 뜻한다.

Bubble Sort의 경우 센서 한 개의 동작이 전체 프로그램의 0.3% 정도의 실행시간을 차지하므로 프로그램 실행에 거의 영향을 주지 않으며 프로그램의 크기가 커진다면 센서의 영향력은 더 없어진다고 할 수 있다.

6. 결론 및 향후계획

임베디드 소프트웨어의 활용범위가 넓어짐에 따라 개발기간을 줄이기 위한 많은 개발도구들이 개발되었다. 그러나 기존의 시스템은 개발기간에만 초점이 맞추어져 있어 실행시간 또는 유지보수 단계에서는 적용이 불가능하다는 문제점이 발생되었다. 이러한 문제를 해결하기 위해 임베디드 모니터링 시스템을 개발하였다. 그러나 임베디드 모니터링 시스템은 감시과정을 진행하기 위하여 사용하는 센서의 부하문제를 그대로 가지고 있기 때문에 추출되는 모니터링 데이터에 대한 신뢰성을 보장할 수 없다. 또한 데드라인 위반 시의 해결방안에 대한 고려가 미흡하다. 이에 모니터링 센서의 최적화를 위하여 응용프로그램의 실행 전에 센서정보와 디버깅정보테이블을 구성하는 초기화과정을 거치고 비트 마스킹 기법을 이용하여 모니터링 데이터를 추출하는 방식을 사용한다. 또한, 메시지 창 출력과 로그기록 기능을 지원함으로써 데드라인 검출 및 처리에 대한 기능을 추가한다. 이렇게 개발된 임베디드 모니터링 시스템은 특정 플랫폼에 제한되지 않기 때문에 임베디드 환경을 사용하는 모든 산업 시스템에 적용함으로써 임베디드 시스템의 개발시간을 단축시키고 소프트웨어의 신뢰성을 향상시킬 수 있다.

향후에는 모니터링 데이터 전송과정 최적화에 대한 연구와 임베디드 모니터링 시스템을 분산 환경에서 적용하기 위한 연구를 진행할 예정이다.

참 고 문 헌

[1] 문세원, 창병모, "자바 병행 프로그램의 모니터링 시스템", 한국컴퓨터종합학술대회 논문집 Vol.32, No.1(B), pp.904-906, 2005.

[2] 신원, 김태완, 장천현, "개선된 모니터링 센서를 이용한 임베디드 모니터링 시스템의 설계 및 구현", 정보과학회 춘계논문집 A권, pp.778-780, 2005.

[3] Aloysius K. Mok and Guangtian Liu, "Efficient Run-Time Monitoring of Timing Constraints", 3rd IEEE RTAS, pp.252-262, 1997.

[4] Roman Obermaisser, "Monitoring and Configuration in a Smart Transducer Network", IEEE Real-Time Embedded System Workshop, pp.1-7, 2001.

[5] Rob Law, "An Overview of Debugging Tools", ACM software Engineering Notes, Vol.22 No.2, pp.43-47, 1997.

[6] Ulfar Erlingsson, F. B. Schneider, 2000. "IRM Enforcement of Java Stack", Symposium on Security and Privacy, Oakland, California, May, 2000.

[7] Ulfar Erlingsson, "The Inlined Reference Monitor Approach to Security Policy Enforcement," PhD thesis, Department of Computer Science, Cornell University, Available as Technical Report 2003-1916, 2003.

신 원

e-mail : wonjjang@konkuk.ac.kr

2003년~2005년 건국대학교 컴퓨터공학과 (공학석사)

2005년~현재 건국대학교 컴퓨터공학과 박사과정

관심분야 : 임베디드 시스템, 실시간 시스템, 컴파일러, 모니터링 시스템

김 태 완

e-mail : twkim@konkuk.ac.kr

1994년 건국대학교 전자계산학과(공학사)

1996년 건국대학교 전자계산학과(공학석사)

1996년~2001년 현대중공업 기전연구소 연구원

2003년~2004년 경민대학 인터넷비즈니스과 겸임교수

1996년~현재 건국대학교 컴퓨터공학과 박사과정

2004년~현재 건국대학교 컴퓨터공학부 강의교수

관심분야 : 프로그래밍 언어, 실시간 프로그래밍, 자동화 소프트웨어, 산업기기 가시 진단 제어 시스템

장 천 현

e-mail : chchang@konkuk.ac.kr

1977년 서울대학교 계산통계(학사)

1979년 KAIST 전산학(석사)

1985년 KAIST 전산학(박사)

현 재 건국대학교 컴퓨터공학과 정교수

관심분야 : 프로그래밍 언어, 컴파일러, 실시간 시스템