원인 프로세스 추적 기능을 가진 실행시간 프로세스 모니터의 설계

정윤석, 김태완, 장천현
건국대학교 컴퓨터 정보통신 공학과
e-mail : ysjeong@cse.konkuk.ac.kr

Design of a Run-time Process Monitor with a Function of Detecting Root Cause Process

Yoon Seok Jeong, Tae Wan Kim, Chun Hyon Chang
Dept. of Computer Science and Engineering , Konkuk University

요 약

산업 전반에 보급된 실시간 시스템의 수행 상태를 파악하기 위해 실시간 감시 기능을 사용한다. 감시의 기본 목적은 감시 대상이 정상적으로 수행하는지를 파악하는 것이다. 특히 실시간 감시 기법으로는 실시간 시스템 상에서 동작하는 실시간 프로세스 상태를 감시하는 프로세스 감시 기법이 있다. 이러한 감시 기법을 지원하는 실행시간 프로세스 모니터의 구조 및 데이터 저장소의 구조에 대해서는 이미 설계 및 구현을 하였다. 하지만, 기존의 프로세스 모니터는 프로세스 상태 데이터를 수집하는 기본 기능만을 제공하였다. 개발자에게 있어서 의미 있는 정보는 프로세스 상태 데이터 만이 아니라 프로세스의 동작 상의 문제점을 발견하고 원인 규명을 할 수 있도록 하는 고급 정보이다. 이러한 정보를 도출하기 위해 본 논문에서는 기존의 실행시간 프로세스 모니터의 구조 기반 위에서 프로세스 동작 여부를 확인하는 기능과 흐름 추적 기능을 새롭게 설계 추가하였다. 이들 기능을 통해 개발자는 어떤 프로세스에서 동작 상의 문제가 발생했는지, 또한 문제를 발생시킨 원인 프로세스가 무엇인지를 추적해 낼 수 있으며, 개발단계에서 문제 해결 능력을 높일 수 있다. 본 논문에서 설계한 원인 프로세스 추적 기능을 가진 실행시간 프로세스 모니터는 기본적으로 실시간 감시 및 제어를 필요로 하는 분야에서 이용될 수 있다.

1. 서론

실시간 시스템이 보편화되면서, 실시간 시스템의 운영 상태를 파악하기 위해 실시간 감시 및 제어 기능이 이용된다. 특히 실시간 감시 기법으로는 실시간 시스템 상에서 동작하는 실시간 프로세스 상태를 감시하는 프로세스 감시 기법이 있다[4][6]. 이러한 감시 기법을 지원하는 실행시간 프로세스 모니터의 구조 및 관련 데이터 저장소의 구조에 대해서는 기존 논문을 통해 설계하고 구현한 바가 있다.[2][3] 그러나 기존의 실행시간 프로세스 모니터는 프로세스의 상태 정보를 수집하고, 출력하는 기본 기능 중심으로 설계되었다. 그러나 개발자의 입장에서는 보다 의미 있는 정보는 정상적으로 동작하는 프로세스의 수행 시간만은 아니다. 프로세스가 수행을 멈춘 경우 이를 발견해 낼 수 있는가하는 점과 또한 비정상적인 동작 상황이 일어나게 된 것이 어떤 프로세스로 말미암은 것인가를 추적해 내는 것이다. 이러한 문제를 해결하기 위해서는 프로세스의 생존 여부를 확인할 수 있어야 하며, 흐름 추적을 추적할 수 있어야 한다. 본 논문에서는 이러한 기능을 지원하기 위해 프로세스의 동작 여부를 확인하는 기능, 흐름 추적 기능을 새롭게 설계하여 추가함으로써, 기존의 프로세스 모니터의 기능을 강화하였다.

본 논문의 2 장은 관련 연구로써 기존에 실행시간 프로세스 모니터의 기능을 분석하여, 한계점을 도출하고 이러한 한계점을 극복하기 위한 기존 감시 기능을 분석하여 문제점을 도출하여, 해결 방안을 제시하였다. 3 장에서는 프로세스 동작 확인 기능과 흐름 추적을 추적하는 기능을 기술하였다. 마지막 4 장은 결론으로 본 논문의 성과 및 향후 연구 방향에 대해서 기술한다.

2. 관련 연구

2.1 기존 실행시간 프로세스 모니터의 기능 분석

실시간 시스템의 보급과 함께, 실시간 시스템의 수행을 감시하는 실시간 감시 기능에 대한 다수의 연구가 수행되었다[2][3][5]. 이들 연구를 종합하면 감시란,

본 연구는 2003 년도 대학 IT 연구센터 육성·지원사업의 연구결과로 수행되었음

'데이터를 수집하고, 데이터를 통하여 각각의 시간 별로 어플리케이션 및 운영체제가 어떤 작업을 수행했는지를 파악하는 것'이다[1] [2] [7][9].

이러한 감시 기능 중 시스템 상에서 동작하는 실행시간 프로세스를 감시하기 위한 기능을 프로세스 감시라 한다. 본 논문에 앞서 프로세스 감시 기능을 지원하기 위해 시스템 구조 및 데이터 저장소 구조를 설계하고, LTMOS 기반에서 TMO 모델을 이용하여 구현 한 바 있다. 기존에 설계하고 구현한 실행시간 프로세스 모니터의 기본 기능을 요약하면 다음과 같다.

- 실행 중인 실시간 프로세스의 상태 데이터 획득
- 프로세스의 수행 시간을 측정하는 성능측정 및 정확성 검사
- 프로세스의 수행시간 위반에 따른 보고

그러나 개발자들에게 보다 의미 있는 정보는 정상적으로 동작하는 프로세스의 수행 시간 만은 아니다. 오히려 프로세스가 비정상적으로 동작하는 경우, 예를 들어 프로세스가 동작을 멈추었을 때, 이를 발견해 낼 수 있는가하는 점이다. 또한 비정상적인 동작 상황이 일어나게 된 것은 어떤 프로세스에서 문제가 발생했는지를 추적해 내는 데 관심이 있다. 이를 위해 다음과 같은 기능이 실행시간 프로세스 모니터에 추가되어야 한다.

- 프로세스의 동작 여부를 확인하는 기능
- 흐름 추적 기능

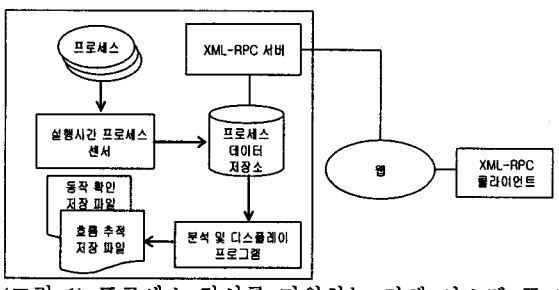
본 논문에서는 기존의 실행시간 프로세스 모니터의 상기 두 가지 기능을 추가하기 위한 개선 구조 및 메커니즘을 제시한다.

3. 설계

3.1 전체 시스템 구조

그림 1 은 통합 감시를 지원하는 시스템의 전체 구조를 보여준다. 시스템의 구성 요소는 다음과 같다.

- 실행시간 프로세스 센서 : 센서는 시스템 상에서 동작하는 실시간 프로세스의 상태 데이터를 수집하는 역할을 한다.
- 프로세스 데이터 저장소 : 프로세스 데이터 저장소는 실행시간 프로세스 센서가 추출한 상태 데이터를 저장한다.
- 분석 및 디스플레이 프로그램 : 실시간 프로세스의 감시 결과나 분석 결과를 출력한다.

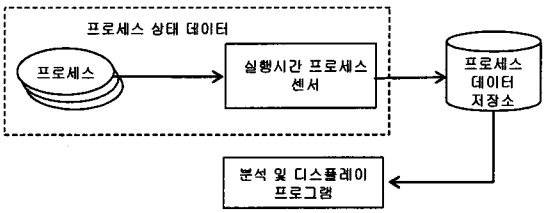


(그림 1) 프로세스 감시를 지원하는 전체 시스템 구조

3.2 프로세스 동작 확인 기능

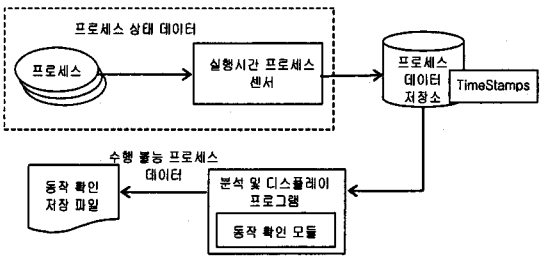
프로세스는 여러 원인에 의해서 비정상적인 수행을 하게 된다. 실시간 프로세스에서 최악의 경우는 서비스를 제공하는 프로세스가 동작을 멈추는 경우이다. 개발자 입장에서 생각한다면, 어떠한 프로세스가 동작을 하는지 하지 않는지 파악하는 것은 실행 단계에서 발생할 수 있는 문제를 개발 단계에서 막을 수 있다는 장점을 갖는다. 따라서 프로세스의 동작 여부를 확인할 수 있는 구조로 변경되어야 한다.

그림 2 는 기존의 데이터 추출 및 출력 구조를 보여준다. 실행시간 프로세스 센서는 실시간 프로세스 내에 코드 형태로 삽입된다. 따라서 특정 프로세스가 동작을 멈추는 경우, 그 프로세스의 데이터를 수집할 수 있는 센서까지도 동작을 멈추는 상황에 이른다. 그럼에도 불구하고, 사용자 혹은 개발자는 프로세스가 어떠한 상황에 빠져있는지를 확인할 수 없다. 이는 프로세스의 동작 여부에 상관없이 프로세스 데이터 저장소에 저장되어 있는 기존의 데이터를 주기적으로 읽어오도록 설계되었기 때문이다.



(그림 2) 기존의 프로세스 상태 데이터 추출 및 출력 구조

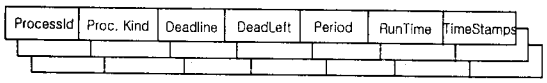
이러한 문제점을 해결하기 위해서 본 논문에서는 프로세스 동작 확인 기능을 추가적으로 제시한다. 그림 3 은 프로세스 동작 확인 기능을 추가한 변경된 프로세스 상태 데이터의 흐름을 나타낸다. 프로세스 동작 확인 기능은 프로세스의 동작 상황을 고려하여, 프로세스 자체가 동작하지 않을 때는 분석 및 디스플레이 프로그램이 프로세스 데이터 저장소에 있는 프로세스 상태 데이터를 추출하거나 출력하지 않도록 한다.



(그림 3) 변경된 프로세스 상태 데이터 추출 및 출력 구조

먼저 구체적으로 기존에 제시한 데이터 저장소의 구조를 개선하였다. 그림 4 는 기존에 제시한 데이터 저장소의 데이터 구조에 TimeStamps 라는 필드를 추가

한 것을 보여준다. TimeStamps 필드의 역할은 현재 데이터 저장소가 저장하고 있는 프로세스 데이터가 어느 정도 현재 상황을 반영하고 있는가를 확인하기 위해서이다. TimeStatmps 필드에는 프로세스 데이터를 추출한 시점의 시간을 저장한다.



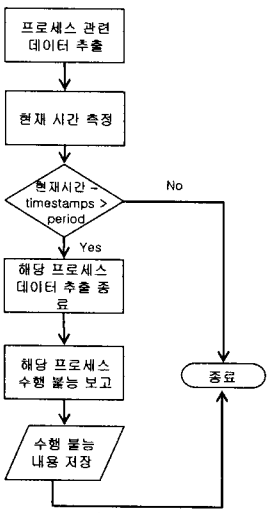
(그림 4) 개선된 데이터 저장소 구조

데이터 저장소에 데이터는 기본적으로 프로세스의 주기 마다 새롭게 갱신된다고 생각할 수 있다. 특정 프로세스가 동작하면, 센서는 그 프로세서의 상태정보를 수집하여 데이터 저장소에 저장한다. 따라서 데이터 저장소에 저장된 데이터의 경우, TimeStamps 가 갖어야 하는 값은

$$time - timestamps_i > period_i$$

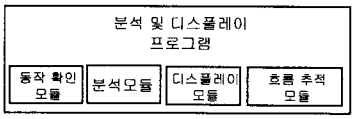
이다. 따라서 분석 및 디스플레이 프로그램에서는 프로세스 상태 데이터를 출력하고자 할 때, 상기의 공식을 이용해서 프로세스의 동작 여부를 확인할 수 있다.

그림 5 는 프로세스 동작 확인을 위한 전체 흐름도이다. 먼저 프로세스 관련 데이터를 프로세스 데이터 저장소에서 추출한다. 그 후 현재 시간과 timeStamps 를 비교하여 해당 프로세스가 종료되었는지를 확인한다. 프로세스가 동작하지 않는 경우, 해당 프로세스의 데이터 추출을 종료하고, 수행 불능 내용을 보고하고, 동작확인 저장 파일에 로그를 저장한다. 저장된 내용은 향후 원인 프로세스 발견하기 위한 기초 데이터로 이용한다.



(그림 5) 프로세스 동작 확인 순서도

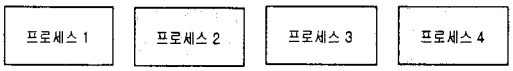
분석 및 디스플레이 프로그램은 그림 6 과 같은 구조를 갖는다. 기존의 모듈들 외에 동작 확인 모듈을 추가하여, 상기한 프로세스 동작 확인 기능을 수행한다.



(그림 6) 개선된 분석 및 디스플레이 프로그램의 구조

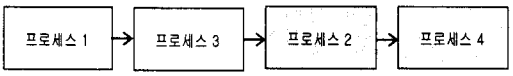
3.3. 흐름 추적 기능

흐름 추적은 프로세스에 의해 어떤 프로세스에 의해서 호출되지는 나타내는 일련의 순서이다. 흐름 추적을 통해서 현재 시스템 상에서 동작하고 있는 여러 프로세스 간의 연관성을 파악할 수 있다. 또한 비정상적인 상황이 발생했을 때 흐름 추적을 이용하여 각 프로세스의 역학관계를 조사하고 원인 프로세스를 찾아 낼 수 있다. 예를 들어 그림 7 에서 문제를 야기한 프로세스가 음영처리 된 프로세스 2 와 프로세스 4 라고 하자. 하지만, 두 프로세스가 모두 문제가 있는 것인지, 혹은 프로세스 2 의 문제가 프로세스 4 의 문제를 야기시킨 것인지 그 반대인지 알 수 없다.



(그림 7) 흐름 추적 기능이 적용되지 않는 프로세스의 정렬

그림 8 의 경우는 흐름 추적 기능에 의해서 프로세스를 나열하게 되면, 프로세스 2 와 프로세스 4 간에 연관관계가 있는 것을 알 수 있다. 또한 프로세스 4 보다 프로세스 2 가 문제를 야기시켰을 가능성이 크다는 것을 알 수 있다. 이와 같이 흐름 추적을 통해 원인 프로세스를 추적하고 문제 해결 능력을 프로세스 2 에 집중할 수 있다.



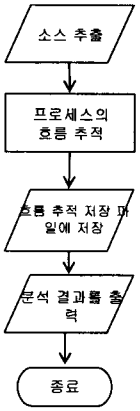
(그림 8) 흐름 추적 기능이 적용된 프로세스의 정렬

본 논문에서 흐름 추적 기능을 보강하는 목적 역시, 프로세스간의 관계를 통해 문제의 원인을 찾아내기 위해서이다. 흐름 추적에는 실행시간에 이루어지는 동적인 흐름 추적과 비 실행시간에 이루어지는 정적인 흐름 추적이 있다. 본 논문에서는 우선 정적인 흐름 추적을 중심으로 설계하였다.

흐름 추적은 호출 순서(calling sequence)와 달리 함수 호출이 이루어지는 순서를 나타내는 것이 아니라, 실시간 프로세스의 주기적인 활성화의 순서를 나타내는 것이다. 이는 실시간 운영체제나 실시간 미들웨어에서 동작하는 실시간 프로세스 가운데는 주기적으로 동작하는 프로세스들이 존재하며, 이들 프로세스들은 일정 시간마다 활성화 단계로 전환하기 때문이다.

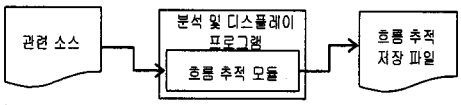
그림 9 는 흐름 추적을 수행하기 위한 전체적인 절

차를 나타낸 흐름 추적의 순서도이다. 흐름 추적의 수행을 위해서는 먼저 해당 실시간 프로그램 소스를 입력 받고, 흐름 추적을 수행한다. 흐름 추적은 소스 차원의 분석을 하는 것으로써, 소스 내에 각 프로세스들을 추출하고 이들간의 수행 흐름을 찾아내는 것이다. 흐름 추적의 결과는 흐름 추적 저장 구조를 가진 파일에 분석 결과를 저장하고, 출력한다. 분석 결과를 통해 개발자는 실시간 프로세스의 동작 흐름을 파악할 수 있으며, 이에 따라 프로세스간의 관계를 파악할 수 있다.



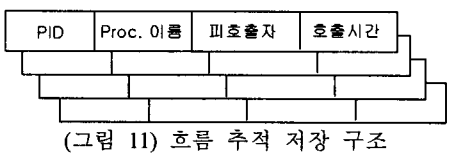
(그림 9) 흐름 추적의 순서도

그림 10 은 흐름 추적 모듈이 기능을 수행하기 위해 필요한 데이터 흐름을 보여준다. 흐름 추적은 분석 및 디스플레이 프로그램 내에 흐름 추적 모듈을 통해 이루어진다. 물론 정적 흐름 추적을 지원하기 때문에 흐름 추적 모듈이 동작하는 시점은 실시간 프로세스가 동작하지 않는 오프라인 상황이다. 흐름 추적 모듈은 실시간 프로그램의 소스를 입력 받아 흐름 추적 분석을 수행한 후, 분석 결과를 흐름 추적 저장 파일에 저장한다. 저장된 데이터는 필요에 따라 다른 분석 과정 없이 출력에 사용할 수 있다.



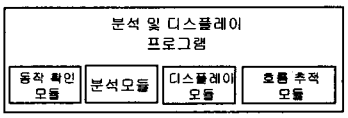
(그림 10) 흐름 추적 데이터의 흐름 구조

프로세스의 흐름 추적 정보를 저장해야 한다. 저장된 데이터를 이용해 호출 관계 등을 파악할 수 있다. 그림 11 과 같이 프로세스 ID, 프로세스 이름, 피호출자, 호출 시간 데이터를 저장한다. 저장되는 데이터는 각 프로세스가 수행될 때 마다 관련 데이터를 흐름 추적 저장 구조에 저장한다.



(그림 11) 흐름 추적 저장 구조

흐름 추적 기능을 추가하기 위해 분석 및 디스플레이 프로그램은 그림 12 와 같은 구조를 갖는다. 음영처리된 흐름 추적 모듈은 프로세스가 비정상적으로 동작할 때 실행하며, 흐름 추적을 파악함으로 개발자에게 용이한 원인 분석을 할 수 있도록 돕는다.



(그림 12) 개선된 분석 및 디스플레이 프로그램의 구조

4. 결론 및 향후 방향

본 논문에서는 기존의 실행시간 프로세스 모니터가 제공하는 프로세스 감시 기법을 정교화하기 위해 추가적으로 프로세스의 생존 여부를 확인하는 기능, 흐름 추적을 추적하는 기능을 추가하였다. 이와 관련하여 구체적으로 기존의 데이터 저장소를 개선하고 흐름 추적 저장 구조를 추가하였다. 또한 분석 및 디스플레이 구조를 개선하여 동작확인 모듈 및 흐름 추적 추적 모듈을 추가하여, 본 논문에서 제시한 두 가지 기능을 실현할 수 있도록 설계하였다. 본 논문에서 제시한 기능을 통해 개발자는 프로세스의 동작 상태를 보다 정확하게 파악하고, 문제 발생시 원인 규명을 할 수 있는 추가적인 정보를 획득할 수 있다.

향후 과제로는 특정 실시간 프로그래밍 모델(TMO 모델 등)을 이용해 본 논문에서 제시한 원인 프로세스 추적 기능을 구현할 예정이며, 동적 흐름 추적 및 동적 분석 기능을 수행하는 실행시간 프로세스 모니터의 설계 및 구현할 예정이다.

참고문헌

[1] "How To Enable Process Accounting on Linux", http://kldp.org

[2] 정윤석, 김태완, 장천현, "실행시간 프로세스 모니터를 위한 구조 설계", 정보처리학회 춘계 학술발표논문집 제 10 권 1 호 2003.

[3] Yoon Seok Jeong, Tae Wan Kim, Chun Hyon Chang, "Design and Implementation of a Run-time TMO Monitor on LTMOS", Proc. Embedded Systems and Applications, Jun. 2003.

[4] B.J. Min, et al., Implementation of a Run-time Monitor for TMO Programs on Windows NT, IEEE Computer Jun. 2000.

[5] A.K. Mok and G. Liu, "Efficient Run-Time Monitoring of Timing Constraints", Proc. Real-Time Technology and Application, Jun. 1997.

[6] Hyung-Taek Lim, et al., "Monitor based Fault Management in a Distributed Environment", 1995.

[7] B.A. Schroeder, "On-line Monitoring: A Tutorial", IEEE Computer, June. 1995.

[8] S. Sankar and M. Mandal, "Concurrent Runtime Monitoring of Formally Specified Programs", IEEE Computer Mar. 1993.

[9] Mike Loukides, "System Performance Tuning", O'Reilly, 1990.