2001 년 한국정보처리학회 춘계 학술발표논문집 제 8 권 제 1 호

유닉스 환경에서의 프로세스 보호 시스템

정준목
LG 전자 네트워크연구소

e-mail : chunmok@lgic.co.kr

Process Protection System on UNIX

Chun-Mok Chung

Network Lab., LG Electronics Inc.

요 약

유닉스에서 구동 되는 중요한 프로세스가 임의의 사용자나 프로세스에 의해 강제로 종료되는 것을 방지하며, 또한 프로세스의 구동 상태를 감시하여 프로세스의 비정상적인 종료가 발견될 때 이를 대체할 새로운 프로세스를 구동 시켜주는 기능을 제공함으로써, 유닉스 시스템의 안정성 및 항상성을 높여주는 기능을 제공하는 유닉스 프로세스 보호 시스템에 관한 것이다. 이는 데몬 방식의 프로세스 모니터링 및 보호 기능뿐 아니라, 커널 수준에서 시그널을 발생하는 시스템 호출을 감시하고 제어함으로써, 견고하게 중요한 프로세스를 보호할 수 있다.

1. 서론

네트워크 기술과 인터넷의 발전에 힘입어 네트워크 상에서 WWW, email 등의 서비스들을 제공하기 위한 서버들의 사용이 증대되고 있다. 이러한 서버들은 서비스를 제공하기 위해 httpd, sendmail 등의 특정 프로세스를 항상 구동하고 있어야 한다. 만일 임의의 사용자나 프로세스에 의해서 이러한 특정 프로세스들의 구동이 중단된다면, 서비스를 제공하는 업체는 서비스 중단으로 인한 금전적 손실 뿐만이 아니라 신뢰성과 같은 기업의 이미지에서도 커다란 손실을 입게 된다. 그러므로, 서버의 안정적인 서비스를 위해서는 이러한 특정 프로세스들이 임의로 중지되는 것을 방지해야 하며, 또한 비정상적으로 프로세스의 구동이 중단되었을 때는 이를 대체할 프로세스를 조속히 구동 시켜주는 기능이 필요하게 된다.

유닉스는 프로세스, 파일 등의 시스템 자원에 대해서 사용자 ID(UID)를 기준으로 접근을 제어한다[1]. 이는 자신의 소유가 아닌 프로세스를 임으로 종료시킬수 없도록 프로세스를 보호해준다. 그러나, 이러한 사용자 UID 기반의 프로세스 제어에서 하나의 예외가 있다. 바로 슈퍼 유저(root)라는 사용자이다. 슈퍼 유저는 자신의 소유가 아닌 모든 프로세스에 접근할 수 있으며, 임의로 프로세스를 제어할 수도 있다.

물론, 슈퍼 유저는 시스템을 관리하는 기능을 담당하므로, 필요에 따라서 자신의 소유가 아닌 프로세스들을 종료하거나 제어할 필요성을 가진다. 그러나, 현재의 시스템들은 그 규모가 커짐에 따라서 시스템을 여러 명이 기능을 나누어 관리하는 경향을 가진다. 즉, 시스템 하드웨어의 변경을 담당하는 하드웨어 관리자, 패스워드를 분실한 사용자들의 패스워드를 초기화해주는 패스워드 관리자, 로그 파일 수집이나 분석과 같은 시스템의 보안을 담당하는 보안 관리자 등으로 구분되어지고 있다. 이처럼 관리의 기능이 분화되고 있으나, 유닉스의 구조적 특성상 이런 모든 기능은 슈퍼 유저만이 수행을 할 수 있으므로, 이들은 모두 슈퍼 유저 ID(root)를 공유하게 된다. 결국, 프로세스에 대한 접근은 이들 모두에게 허가되므로, 이들의 의도적인 조작이나 실수로부터 중요한 프로세스를 보호해야 할 필요성이 증대된다. 즉, 슈퍼 유저라 하더라도 중요한 프로세스를 임의로 종료 시키지 못하게 함으로써, 중요한 프로세스가 비정상적으로 종료되는 것을 방지할 방법이 필요하게 된다.

2 장에서는 유닉스에서 프로세스 보호를 위한 방법을 설명하고, 3 장과 4 장에서는 이를 위해 설계된 유닉스 프로세스 보호 시스템의 구조와 동작 과정을 상세히 설명한 후, 5 장에서 결론을 맺는다.

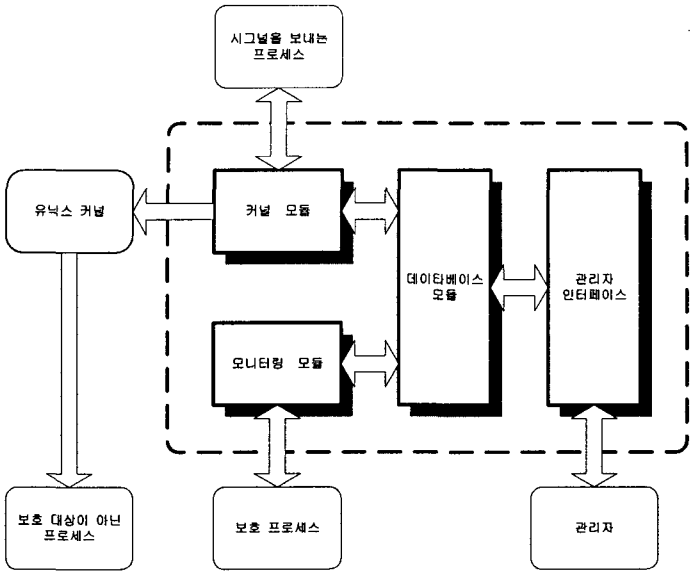

그림 1. 프로세스 보호 시스템의 구조 : 그림에서 굵은 점선 부분이 프로세스 보호 시스템을 나타낸다.

2. 유닉스 환경에서의 프로세스 보호 방법

유닉스에서 프로세스를 강제로 종료 시키는 방법으로는 종료를 원하는 프로세스에 프로세스를 종료 시키는 시그널(signal)을 발생시키는 방법이 있다[2]. 대부분의 프로세스는 시그널을 받았을 때, 무시하거나 특별한 처리를 하도록 설정되어있지 않을 경우에는, 정상적으로 종료된다. *SIGKILL*, *SIGTERM* 등의 시그널에 대해서는 어느 프로세스도 무시하거나 특별한 처리를 할 수 없기 때문에 이러한 시그널을 받은 프로세스는 반드시 종료하게 된다. 프로세스들은 kill 시스템 호출을 통해 시그널을 서로 보낼 수도 있고 또는, 커널이 내부적으로 시그널을 보낼 수도 있다[3].

유닉스 프로세스 보호 시스템(이하 보호기)은 강제적인 종료로부터 보호해야 할 중요한 프로세스(이하 보호 프로세스)가 임의의 사용자나 프로세스에 의해 강제로 종료되는 것을 방지한다. 보호기는 커널 수준에서 시그널을 발생시키는 kill 시스템 호출을 감시하고 제어하여, 보호 프로세스로 시그널이 발생하는 것을 방지함으로써, 프로세스를 보호한다. 보호기는 또한 보호 프로세스의 구동 상태를 감시하여 보호 프로세스 자체의 오류로 인해 비정상적인으로 종료되었을 때 이를 대체할 새로운 보호 프로세스를 구동 시켜준다. 이를 위해 보호기는 데몬(daemon) 형식의 모니터를 통해 보호 프로세스를 감시한다.

3. 보호기의 구조

그림 1 은 보호기의 모듈 구조와 각 모듈간의 연관 관계를 보여준다. 굵은 점선부분이 보호기이다. 보호기는 시스템 호출을 감시, 제어하는 커널 모듈, 보호 프로세스의 상태를 감시하는 모니터링 모듈, 보호 프로세스에 대한 정보를 관리하는 데이터베이스 모듈, 데이터베이스에 보호 프로세스의 등록 및 제거를 위한 관리자 인터페이스로 구성된다. 보호기의 구조는 선행 개발 단계에서 설계된 UPS[4]의 기본 구조를 따르므로, 상세한 내부구조는 UPS 를 참조하기 바란다.

커널 모듈은 외부 프로세스로 인하여 보호 프로세스가 종료되는 것을 방지하는 기능을 담당한다. 커널 모듈은 다른 프로세스가 시그널을 발생시켜 보호 프로세스를 강제로 종료하려 할 때, 이를 감지하여 시그널을 차단하는 기능을 수행한다. 커널 모듈은 유닉스의 커널 영역에 적재되어, 임의의 프로세스에서 시그널 발생을 위해 kill 시스템 호출을 커널에 요청하면, 유닉스 커널이 kill 시스템 호출을 수행하기 전에 kill 시스템 호출의 인자 값을 분석하여, 시그널을 받을 프로세스를 검색하고, 보호 프로세스인 경우에는 kill 시스템 호출의 수행을 거부하여 보호 프로세스가 시그널에 의해 종료되는 것을 방지한다. 그러나, 보호 프로세스 이외의 프로세스에 발생하는 시그널은 수행이 되도록 한다.

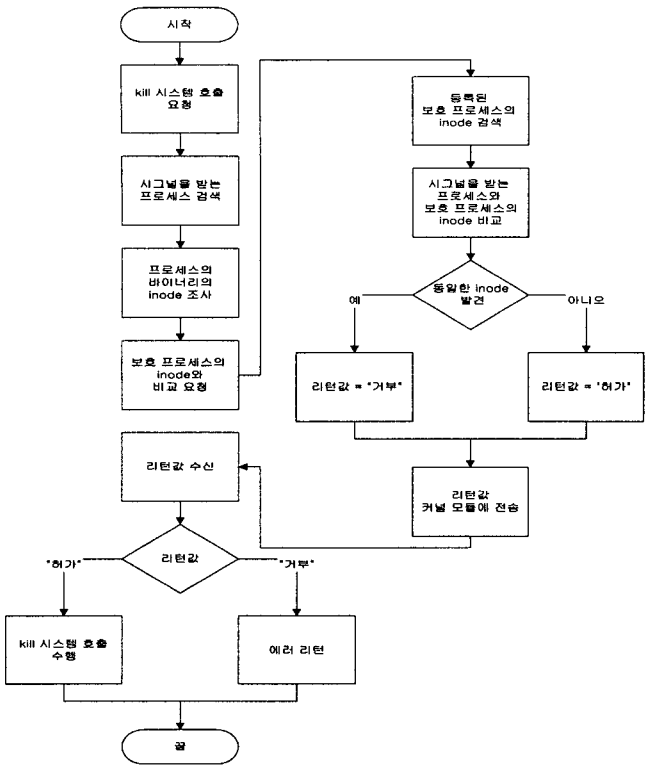

그림 2. 시그널 처리 과정 : 보호 프로세스에게는 시그널을 보내지 못하도록 제어한다.

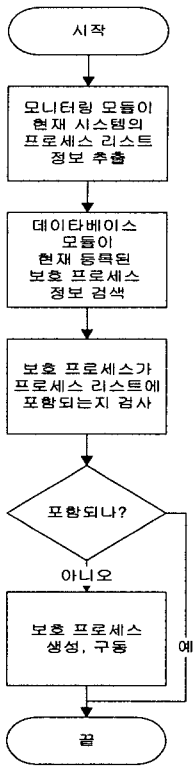

그림 3. 자체 오류에 의한 종료된 보호 프로세스를 새로운 프로세스로 대체하는 과정.

모니터링 모듈은 보호 프로세스의 구동 상태를 감시하고, 자체적인 오류로 인하여 보호 프로세스가 비정상적으로 종료된 경우에 이를 감지하여 대체적인 보호 프로세스들을 구동 시키는 기능을 수행한다. 모니터링 모듈은 데몬(daemon) 형태로 시스템에 상주하며, 일정 시간을 주기로 현재 시스템에서 보호 프로세스가 정상적으로 구동 중인지를 검사한다. 만일 보호 프로세스가 자체적인 오류로 인해 비정상적으로 종료되어 구동 되고 있지 않을 때는, 모니터링 모듈이 새로운 프로세스로 보호 프로세스를 구동 시킨다. 이런 방식으로, 보호 프로세스가 시스템에서 언제나 구동 상태로 존재하도록 유지시켜준다.

데이터베이스 모듈은 보호 프로세스에 대한 정보를 저장하고 관리하며, 커널 모듈이나 모니터링 모듈에서 보호 프로세스에 대한 정보를 요청할 때 제공한다. 커널 모듈이나 모니터링 모듈은 데이터베이스 모듈과의 통신을 통해 등록된 보호 프로세스에 대한 정보를 얻게 되고, 관리자 인터페이스도 데이터베이스 모듈과의 통신을 통해 보호 프로세스에 대한 정보의 검색이나 수정을 요청한다.

관리자 인터페이스는 관리자가 보호 프로세스를 데이터베이스 모듈에 추가하거나, 데이터베이스 모듈에서 삭제할 수 있도록 데이터베이스 모듈에 대한 인터페이스를 제공한다. 관리자 인터페이스는 관리자와 데이터베이스 모듈과의 유일한 상호 통신 채널을 제공한다. 관리자는 관리자 인터페이스를 통해서만 데이터베이스 모듈에 접근할 수 있으며, 이를 통해서만 보호 프로세스에 대한 정보 검색이나 수정 등의 작업을 처리할 수 있다.

4. 보호기의 동작 과정

그림 2 는 시그널에 의한 보호 프로세스 종료 방지 기능의 동작 과정을 보여준다. 다른 프로세스에서 보호 프로세스를 종료 시키는 시그널을 발생하기위해, 보호 프로세스의 프로세스 ID 를 인자 값으로 가지는 kill 시스템 호출을 커널에 요청한다. 커널 모듈은 유닉스의 커널에 앞서, 요청한 kill 시스템 호출의 인자 값을 분석하여, 시그널을 받을 프로세스를 검색한다. 커널 모듈은 검색된 프로세스에 대한 파일 시스템상에서의 바이너리(binary) 파일의 inode 번호를 조사한다. 커널 모듈은 조사한 바이너리 파일의 inode 번호를 사용하여, 데이터베이스 모듈에게 데이터베이스에 등록된 보호 프로세스의 바이너리 파일의 inode 번호와 일치하는지에 대한 확인을 요청한다. 데이터베이스

모듈은 커널 모듈의 요청에 따라 데이터베이스를 검색하여 보호 프로세스로 등록된 프로세스들의 바이너리의 inode 번호들을 검색한다. 검색한 inode 번호들을 커널 모듈이 보낸 inode 번호와 비교하여, inode 번호가 일치하는 보호 프로세스가 존재하는지 조사한다. 만일 동일한 inode 번호가 발견되면, 보호 프로세스가 시그널을 받은 것으로 인식하고, 커널 모듈에 되돌려줄 리턴값을 "거부"로 설정한다. 이는 시그널 발생을 거부한다는 의미이다. 동일한 inode 번호가 발견되지 않으면, 보호 프로세스가 아닌 프로세스에 대한 시그널 발생으로 인식하고, 커널 모듈에 되돌려줄 리턴값을 "허가"로 설정한다. 이는 시그널 발생을 허가한다는 의미이다. 설정된 리턴값은 커널 모듈에 전송된다. 커널 모듈은 데이터베이스 모듈로부터의 전송 결과에 따라 다른 프로세스가 요청한 kill 시스템 호출에 대한 수행 여부를 판단한다. 만약 "거부"가 전송되면, 요청한 kill 시스템 호출을 수행하지 않고, 요청한 프로세스에 에러를 리턴하며, "허가"가 전송되면, 커널의 kill 시스템 호출의 수행을 계속한다. 커널은 kill 시스템 호출을 수행한 후, 결과값을 요청한 프로세스에 리턴한다.

프로세스는 구동 중에 메모리 부족, 메모리 참조 실패, 잘못된 연산 수행 등의 자체적인 오류로 인해 종료될 수가 있다. 그림 3 은 자체적인 오류로 인한 보호 프로세스의 비정상적인 종료 방지 기능의 동작 과정을 보여준다. 모니터링 모듈은 시스템에서 현재 구동중인 프로세스 리스트를 추출한다. 그리고, 데이터베이스 모듈에게 현재 등록된 보호 프로세스들의 정보를 요청한다. 모니터링 모듈은 데이터베이스 모듈로부터 받은 보호 프로세스가 시스템으로부터 추출한 프로세스 리스트에 포함되어 있는지 검사한다. 만약, 보호 프로세스가 시스템의 프로세스 리스트에 포함되어 있지 않으면, 보호 프로세스가 비정상적으로 종료된 것으로 판단하여 보호 프로세스를 새로운 프로세스로 구동 시킨다. 이러한 과정은 일정 시간에 한번씩 주기적으로 수행되며, 데이터베이스에 등록된 모든 보호 프로세스에 대해 적용된다. 모든 보호 프로세스에 대한 검사와 구동 과정이 끝나면, 모니터링 모듈은 다음 주기가 될 때까지 대기 상태가 된다.

5. 결론

보호기는 유닉스에서 구동 되는 보호 프로세스가 임의의 사용자나 프로세스에 의해 강제로 종료되는 것을 방지한다. 또한 프로세스의 구동 상태를 감시하여 비정상적인 보호 프로세스의 종료가 발견되면, 이를 대체할 프로세스를 구동 시켜주는 기능을 제공한다. 그럼으로써, 서버의 안정성 및 항상성을 높여주는 기능을 제공한다. 이는 중요한 서버를 운영하는 기업에게 서버의 안정적 운영을 가능하게 해준다.

효과적인 보호를 위해 보호기는 시스템 호출을 커널 수준에서 감시하고 제어하여 프로세스를 보호함과 더불어 데몬 방식의 프로세스 감시, 보호 기능을 포함한다. 이는 프로세스가 비정상적으로 종료되는 내적, 외적 문제들에 대해 모두 처리를 하여주므로, 보다 견고하게 프로세스를 보호할 수 있을 뿐만이 아니라 커널 수준의 작업으로 서버의 부담도 적은 장점을 가진다.

참고문헌

[1] 조유근, "UNIX 의 내부구조", 홍릉과학출판사, 1994.
[2] Steve D Pate, "UNIX Internals : A Practical Approach", Addison-Wesley. 1996.
[3] W.Richard Stevens, "Advanced Programming in the UNIX Environment", Addison-Wesley. 1992.
[4] 정준목, 오석남, "시스템 자원 접근 제어 방법", 한국정보처리학회 제 13 회 춘계학술대회, 2000 년, http://chunmok.hihome.com/paper/kips2000spring.zip