실행환경의 변화를 통한 디바이스 드라이버 고장 복구

박현숙° 이인환
한양대학교 전자통신컴퓨터공학과
{meetesang°, ihlee}@hanyang.ac.kr

Recovering from Device Driver Failures by Environment Diversity

Hyunsook Park° Inhwan Lee
Dept. of Electronics and Computer Engineering, Hanyang University

요 약

디바이스 드라이버는 커널 고장을 유발하는 가장 큰 원인이며, 커널 영역에서 동작하므로 커널에 큰 영향을 미칠 수 있다. 본 논문은 운영체제의 신뢰성을 향상시키기 위하여, 디바이스 드라이버 실패가 발생했을 때, 고장난 드라이버와 이 드라이버를 사용하는 응용 프로그램을 복구하는 방법을 제시한다. 우리는 논문에서 제시한 복구 시스템을 FRS (Failure Recovery Subsystem) 이라고 부르겠다. 디바이스 드라이버가 고장 났을 때, FRS는 디바이스 드라이버의 실행 환경을 다양하게 변경하고 드라이버를 재실행함으로써 같은 고장이 반복되지 않도록 한다. FRS은 고장을 복구하고 디바이스 드라이버를 정상적으로 실행함으로써 전체 시스템의 안정성을 향상시킨다.

1. 서 론

디바이스 드라이버 고장이 커널 코드의 다른 곳의 고장보다 훨씬 빈번하게 발생된다는 연구가 보고된 적이 있다[1]. 디바이스 드라이버는 커널 영역에서 동작하지만 확장 프로그램이기 때문에, 드라이버는 커널과 완벽하게 조화하기 어려워서 고장을 발생시킬 확률이 크다. 따라서 드라이버의 고장이 시스템에 미칠 수 있는 피해는 심각하다. 디바이스 드라이버에서 발생한 고장은 운영체제를 불안정하게 하고, 결국 시스템의 신뢰성을 떨어뜨린다.

본 논문은 디바이스 드라이버 고장을 복구하여 시스템의 신뢰성을 향상시키는 시스템을 제안한다. 이 시스템은 고장이 발생한 실행 환경을 변화시켜 고장을 복구하고 디바이스 드라이버를 다시 실행하여 같은 고장이 재발하지 않도록 한다.

2. 관련 연구

디바이스 드라이버 고장이 일으키는 가장 큰 문제는 발생한 고장이 커널에 직접적인 영향을 줌으로써 시스템 크래시를 유발한다는 것이다. 그동안 시스템 안정성을 위한 연구가 진행되었고, 대부분의 연구는 고장이 커널에 미치는 영향을 차단함으로써 시스템의 안정성을 향상시키는 방법을 제시하였다. 시스템 안정성을 향상시키기 위해 수행되었던 연구를 크게 네 가지로 분류하면 다음과 같다.

- 기존 운영체제 보수: 운영체제는 보호 영역을 두어 이곳에서 디바이스 드라이버를 실행함으로써 발생한 에러가 다른 곳으로 전파되는 것을 막는다[2][3].
- 버추얼 머신 : 버추얼 머신은 실제 시스템을 흉내낸 인스턴스를 여러 개 만들고, 그 위에서 디바이스 드라이버를 실행함으로써 한 곳에서 발생한 에러가 다른 인스턴스에 영향을 주지 못하게 한다[4].
-멀티서버 커널 : 운영 체제의 핵심 커널만 커널 모드에서 동작하고, 운영 체제의 나머지 부분은 사용자 모드에서 동작한다. 디바이스 드라이버도 사용자 모드에서 동작하므로 디바이스 드라이버의 에러는 커널에 영향을 주지 못한다[5].
- 언어 기반의 보호 : 이 시스템은 프로그램 에러를 발생할 수 있는 언어 대신에 세이프-타입(safe-type) 언어를 쓴다. 예를 들면, 포인터 에러를 유발할 수 있는 C언어 보다 C#언어를 사용한다.

기존의 연구가 디바이스 드라이버의 고장을 차단하여 시스템 크래시를 예방하는 방법을 제시한 반면에, 본 논문은 드라이버의 고장을 복구함으로써 시스템 크래시를 예방하는 방법을 제시한다. 본 논문은 고장난 디바이스 드라이버를 복구하기 위하여 Nooks[3] 환경에 Rx[6]의 아이디어를 적용한다. Rx는 사용자 공간에서 응용 프로그램의 고장을 복구한다. 구체적으로 고장이 발생한 응용 프로그램의 실행환경을 바꿈으로써 같은 고장이 다시 발생하지 않도록 한다. Nooks[3]는 디바이스 드라이버를 커널공간의 보호 영역 안에서 실행하면서, 드라이버의 실행을 모니터링 한다.

3. FRS 설계

본 논문에서는 FRS(Failure Recovery Subsystem)을 제안한다. FRS는 디바이스 드라이버의 고장을 복구하고 디바이스 드라이버를 재실행함으로써 드라이버와 응용 프로그램이 정상적으로 동작하여 시스템의 신뢰성을 향상시킨다. 구체적으로, FRS는 고장을 유발한 드라이버의 실행 환경을 변경하고 드라이버를 재실행하여 같은

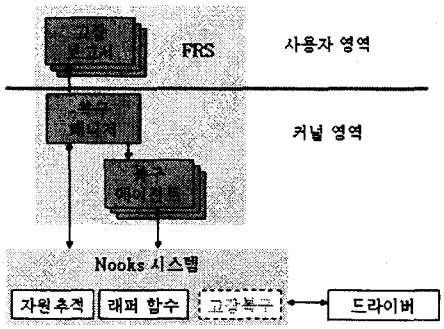

그림 1 FRS의 구조

고장이 발생하지 않도록 한다.

그림 1은 FRS의 구조를 나타내고 있다. FRS는 Nooks가 내장된 기존의 운영체제 위에서 동작한다. Nooks는 디바이스 드라이버 실행을 관리하기 위해서 고장차단, 자원추적, 래퍼함수, 고장복구와 같은 기능을 제공한다. FRS는 Nooks가 제공하는 자원추적, 래퍼함수를 이용하여서 디바이스 드라이버를 모니터링하고 고장을 복구한다. 즉, FRS는 Nooks의 복구 매니저를 대신에 해서 고장 복구 과정을 관리한다. FRS에서 복구 매니저는 복구 과정을 통제하고 복구 결과를 고장 보고서에 쓰며, 복구 에이전트는 복구를 수행한다.

3.1 고장 보고서

고장 보고서는 발생한 고장에 대한 정보를 저장하고 있는 파일이다. 고장 보고서는 고장이 발생하여 고장이 복구될 때마다 갱신된다. 이 때 갱신되는 정보는 발생한 고장의 종류에 따른 고장 발생 수, 고장 발생 빈도, 복구 성공 비율, 오버헤드 등이다. 고장 보고서는 두 가지 용도로 사용된다. FRS은 고장을 복구하기 위해서 고장 보고서를 복구 규칙(3.4참고)에 적용하기 위한 데이터베이스로 사용하며, 개발자는 디바이스 드라이버에서 발생한 버그를 디버깅하기 위한 자료로 사용한다. FRS는 새로운 디바이스 드라이버를 관리하기 시작하면, 이 드라이버가 발생시킨 고장에 대한 정보를 담고 있는 새로운 고장 보고서를 만든다.

3.2 복구 매니저

복구 매니저는 고장 복구 과정 전체를 통제한다. 복구 매니저가 복구를 관리하지만 실제로 복구과정을 실행하지 않는다. 복구 매니저는 복구 에이전트를 생성하여 FRS가 관리하는 드라이버에 할당하고, 복구 에이전트가 실제 복구 과정을 담당하게 한다. 복구 매니저는 고장이 발생했을 때, 고장을 복구 에이전트에게 알리고 복구 에이전트를 복구 모드에서 동작하게 한다.

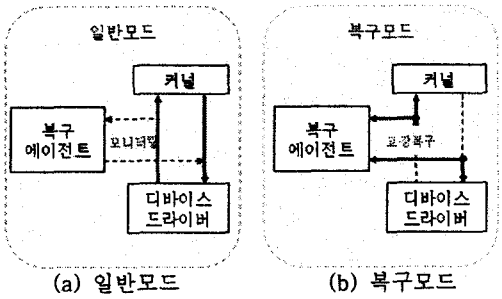

그림 2 복구 에이전트 동작

3.3 복구 에이전트

복구 에이전트는 디바이스 드라이버를 모니터링하고, 실제 복구 작업을 수행한다. 복구 에이전트는 Nooks가 제공하는 정보를 이용해서 드라이버 실행을 모니터링하고, 그 결과를 이용하여서 복구 작업을 수행할 수 있다.

그림2(a)는 일반모드에서 복구 에이전트의 동작을 보여준다. 디바이스 드라이버 고장이 발생하기 전에는 복구 에이전트는 디바이스 드라이버 모니터링 작업만 한다. 복구 에이전트는 드라이버와 커널간의 통신을 추적하지만, 실제로는 드라이버와 커널간의 통신에 영향을 미치지 않는다. 그림2(b)는 복구모드에서 복구 에이전트의 동작을 보여준다. 디바이스 드라이버 고장이 발생하면, 복구 에이전트는 커널과 디바이스 드라이버의 통신을 가로채어 복구 작업을 시작한다. 기존의 커널과 디바이스 드라이버 사이의 연결은 끊어지고, 커널과 드라이버간의 통신은 복구 에이전트를 통해서 이루어진다.

3.4 복구 규칙

FRS는 일시적이지 않은 고장과 일시적인 고장을 복구 할 수 있다[7]. 일시적이지 않은 고장은 메모리 관리와 관련된 고장으로 디바이스 드라이버가 실행될 때마다 매번 발생한다. 일시적인 고장은 프로세스 간의 동기화와 관련된 고장으로 대부분 디바이스 드라이버를 재실행함으로써 해결될 수 있다. 일시적이지 않은 고장을 발생시킨 원인은 다양하기 때문에 이에 적용될 환경변화도 고장에 따라 다르다. 표1은 고장의 종류와 고장의 종류에 따른 다양한 환경변화를 보여준다[6].

복구 에이전트는 적은 오버헤드로 신속하게 복구 과정을 수행하기 위해서 복구 규칙을 사용한다. 복구 규칙은 두 단계로 진행된다. 먼저, 복구 에이전트는 고장 보고서의 고장 빈도를 참고하여 가장 빈번하게 발생한 고장을 여러 개 선택한다. 다음으로 복구 에이전트는 선택된 고장 중에서 오버헤드가 적은 순서로 환경변화를 선택하여 복구에 적용한다.

표 1 고장 종류에 따른 환경 변화

종류	반복적인가?	환경 변화
버퍼 오버플로	예	버퍼 양끝에 패딩을 삽입한다.
초기화 되지 않은 변수	예	변수를 '0'으로 채운다.
메모리 오염	예	새로운 공간에 메모리를 할당한다.
메모리 중복해제	예	해제된 메모리 사용을 지연한다.
데이터 경쟁	아니오	디바이스 드라이버를 재실행한다.
잘못된 요청	예 또는 아니오	요청을 무시한다.

4. FRS 동작

4.1 디바이스 드라이버 실행

FRS은 일반모드와 복구모드에서 디바이스 드라이버를 실행한다. 일반모드는 고장이 발생하기 전의 정상적인 디바이스 드라이버의 실행모드이고, 복구모드는 고장이 발생하고 난 후의 드라이버를 복구하는 실행모드이다. 일반모드에서 디바이스 드라이버가 실행하고 있을 때 고장이 발생하지 않는다면, 복구 에이전트는 드라이버의 실행을 추적하며 저장한다. 일단 디바이스 드라이버 고장이 발생하면, 복구 매니저는 복구 에이전트의 실행모드를 복구모드로 바꾼다. 디바이스 드라이버의 실행이 복구 모드로 들어가게 되면, 복구 에이전트는 드라이버의 실행을 마지막으로 저장된 상태로 되돌리고 복구 과정을 진행한다. 드라이버가 복구가 성공하면 실행모드는 일반모드로 바뀐다. 만약 복구가 실패하면 디바이스 드라이버의 실행은 마지막으로 저장된 상태로 다시 되돌아가고, 복구 에이전트는 다른 복구 방법을 드라이버에 적용한다. 복구 에이전트가 복구를 시도하는 중에 타임아웃이 발생하면, 복구 에이전트는 고장난 디바이스 드라이버를 리로딩 하거나 전체 시스템을 리부팅한다. 그림 3은 일반모드와 복구 모드에서 디바이스 드라이버의 실행 흐름을 보여준다.

4.2 일반 모드

복구 에이전트는 디바이스 드라이버 실행을 모니터링 하기 위해서 몇 가지 작업을 한다[8]. 첫째, 복구 에이전트는 응용 프로그램의 요청을 저장한다. 복구 에이전트가 고장을 복구하기 위해서는 드라이버가 이전의 상태로 돌아가야 한다. 복구 에이전트는 드라이버의 실행 흐름을 저장하여 드라이버 실행을 추적할 수 있다. 둘째, 복구 에이전트는 환경 설정을 저장한다. 복구 에이전트는 디바이스와 드라이버의 환경 설정을 저장함으로써 이전과 동일한 환경에서 복구된 드라이버를 재실행할 수 있다. 셋째, 복구 에이전트는 디바이스 드라이버가 사용하는 자원을 추적한다. 고장난 디바이스 드라이버의 복구와 재실행을 위하여, 복구 에이전트는 드라이버가 요청한 커널 자원이나 사용하고 있는 자원을 관리해야 한다. 드바이스 드라이버를 모니터링 하기 위해서 저장된 정보들을 복구 매니저가 주기적으로 삭제하여서

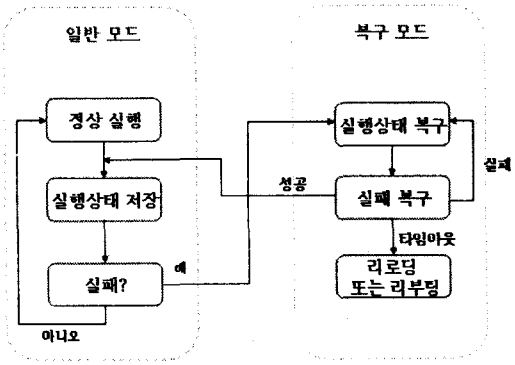

그림 3 일반모드와 복구 모드에서 디바이스 드라이버의 실행 흐름

시스템의 메모리를 최소한으로 사용한다.

4.3 복구 모드

복구모드에서 복구 에이전트는 실질적인 복구 과정을 처리한다. 복구는 세 단계의 과정을 통해 이루어진다. 첫째, 복구 에이전트는 디바이스 드라이버가 사용하고 있는 커널 자원을 해제하고 디바이스 드라이버로 들어오는 인터럽트를 무시한다. 만약 다른 프로세스가 드라이버가 사용하고 있는 자원을 사용하고 있다면, 복구 에이전트는 프로세스가 자원을 사용하지 않을 때까지 대기한다. 복구 과정 중에 드라이버가 커널과 디바이스로부터 오는 인터럽트에 의해 방해 받는다면 복구 과정은 지연될 것이다. 둘째, 복구 에이전트는 복구 규칙을 적용하여 고장을 복구한다. 구체적으로 복구 에이전트는 고장난 디바이스 드라이버의 실행을 되돌리고, 복구 규칙에 의하여 선택된 환경 변화를 드라이버에 적용한다. 만약, 복구가 성공하지 않았다면, 복구 에이전트는 다른 복구 해결책을 적용하여 다시 복구를 시도한다. 셋째, 복구 에이전트는 복구 결과를 복구 매니저에게 알린다. 복구 매니저는 복구 에이전트로부터 복구 결과에 대한 보고를 받으면, 발생한 고장의 종류, 적용한 환경 변화 등에 대한 정보를 고장 보고서에 업데이트한다.

복구가 진행되는 동안에는 디바이스 드라이버는 응용 프로그램의 요청을 즉시 처리할 수 없기 때문에 복구가 끝난 다음에 처리해야 한다. 복구 에이전트는 복구 중에 들어온 요청을 임시 버퍼에 저장하고, 복구가 끝난 다음에 디바이스 드라이버에게 전달한다.

5. 구현

우리는 Vmware위에서 Nooks 시스템이 포함된 리눅스 커널 2.4.18을 기반으로 FRS를 구현하고 있다. 본 논문이 FRS를 구현하기 위해서 Nooks을 선택한 이유는 Nooks가 기존 운영체제에서 동작할 수 있으며, 커널 소스 코드를 거의 변경하지 않기 때문이다. FRS는 디바이스 드라이버 실행에 대한 정보를 Nooks를 통해

얻으며, FRS는 이 정보를 이용해서 디바이스 드라이버 고장을 복구하는 과정만 처리한다. 현재 복구 매니저를 구현하고 있으며, 사운드 블러스터 16 사운드 카드의 디바이스 드라이버를 위한 복구 에이전트를 구현하고 있다. 우리는 향후 다른 디바이스 드라이버를 위한 복구 에이전트를 구현할 계획을 가지고 있다.

6. 결론

본 논문은 FRS의 개념을 제안했다. FRS는 운영체제의 신뢰성을 향상하기 위해서 기존의 방법들과 다른 방법으로 디바이스 드라이버의 고장을 다룬다. FRS는 디바이스 드라이버의 고장을 발생시키는 주요한 소프트웨어 에러의 종류를 정의하고, 이들을 유발하는 환경을 변경함으로써 고장의 원인이 되는 환경적인 요인을 제거한다. FRS는 효율적으로 고장을 복구하기 위해 복구 규칙에 의해 적용할 환경 변화를 선택한다. FRS가 디바이스 드라이버의 모든 고장 유형을 복구할 수는 없지만 많이 발생하는 고장유형에 대처함으로써 시스템의 안정성을 높일 수 있다.

7. 참고 문헌

[1] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An Empirical Study of Operating Systems Errors," ACM SIGOPS Operating Systems Review, Proceedings of the eighteenth ACM symposium on Operating systems principles SOSP '01, Volume 35 , Issue 5, Pages: 73-88, 2001

[2] Robert Wahbe, Steven Lucco, Thomas E. Anderson and Susan L. Graham, "Efficient Software-Based Fault Isolation," ACM SIGOPS Operating Systems Review, Proceedings of the fourteenth ACM symposium on Operating systems principles SOSP '93, Volume 27, Issue 5, Pages: 203-216, 1993

[3] Michael M. Swift, Brian N. Bershad, and Henry M. Levy, "Improving the Reliability of Commodity Operating Systems," Transactions on Computer Systems (TOCS), Volume 23, Issue 1, Pages: 77-110, 2005

[4] Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz, "Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines", Proceedings of the 6th Symposium on Operating System Design and Implementation, 2004

[5] Jochen Liedtke, "On micro-kernel construction,", ACM SIGOPS Operating Systems Review, Proceedings of the fifteenth ACM symposium on Operating systems principles SOSP '95, Volume 29, Issue 5, Pages: 237-250, 1995

[6] Feng Qin, Joseph Tucek, JagadeesanSundaresan and Yuanyuan Zhou, "Rx: Treating Bugs As Allergies— A Safe Method to Survive Software Failures," ACM SIGOPS Operating Systems Review, Proceedings of the twentieth ACM symposium on Operating systems principles SOSP '05,Volume 39, Issue 5, Pages: 235-248, 2005

[7] Subhachandra Chandra and Peter M. Chen, "Whither Generic Recovery from Application Faults? A Fault Study using Open-Source Software," Proceedings of the 2000 International Conference on Dependable Systems and Networks /Symposium on Fault-Tolerant Computing (DSN/FTCS), 2000

[8] Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy, "Recovering Device Drivers," Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation, 2004