2000 년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 1 호

시스템 자원 접근 제어 방법

정준목, 오석남
LG 정보통신
e-mail : {chunmok, snoh}@lgic.co.kr

The Method of Controlling System Resource Accesses

Chun-Mok Chung and Seok-Nam Oh
LG Information and Communications, Ltd.

요 약

본 논문에서는 사용자 수준에서 시스템 호출을 제어하는 방법 및 구성을 제안하고, 이를 위해 유닉스 시스템 호출을 가로채는 방법을 기술한다. 제안된 방법은 유닉스의 소스 코드의 수정을 필요로 하지 않으므로, 동적으로 적용 및 변경이 가능하다. 또한, 시스템 호출 제어방법의 응용 모델로서, 사용자의 시스템 자원 접근을 제어하는 보안 모델로 설계된 UPS(Unix Protection System)에 대하여 기술한다. UPS 의 성능 평가는 벤치마크 프로그램들의 수행을 통해 추가 부하(overhead)를 살펴본다. 실험 결과에 의하면 UPS 를 사용한 경우는 사용하지 않은 경우보다 1~19%의 추가적인 시간이 소요되었다.

1. 서론

운영체제는 하드웨어와 직접적으로 상호 작용하여 프로그램에 공통 서비스를 제공하며, 하드웨어의 특이성으로부터 프로그램을 차폐 시켜 준다. 유닉스(Unix) 시스템은 1969 년에 처음 소개된 운영체제로 이후 마이크로프로세서에서 메인프레임에(mainframe) 이르는 다양한 처리 능력을 가진 수많은 컴퓨터에 사용되고 있으며, 1984 년 초에는 세계적으로 100,000 대에 가까운 컴퓨터에서 운용되고 있다.[1]

커널은 시스템의 자원을 관리하여 모든 시스템이 원활히 돌아갈 수 있도록 제어하는 유닉스의 핵심 부분으로, 유닉스안에 있는 모든 응용 프로그램들은 커널이 제공하는 서비스에 의존한다. 커널의 기능으로는 프로세스의 실행에 대한 제어, 프로세스들에게 CPU 의 공평하게 스케줄하는 기능, 주기억 장치의 할당, 보조기억 장치의 할당, 주변장치에 대한 프로세스의 접근 제어 등이 있다. 사용자는 시스템 호출을 통해 커널에 이러한 기능을 요청할 수 있다.

유닉스에서의 시스템 호출은 사용자가 시스템의 자원을 사용하기 위해 커널에 요청하는 인터페이스이다. 시스템 호출은 커널로 하여금 그것을 요청하는 프로그램을 위해 여러 다양한 연산을 하도록 지시하며, 커널과 프로그램 사이에 데이터를 교환하게 한다. 사용자가 파일이나 프로세스등의 시스템의 자원을 사용하기 위해서는 유닉스에서 정의된 시스템 호출을 통해서만 가능하다.

본 논문에서는 사용자 수준에서 시스템 호출을 제어하는 방법 및 구성을 제안하고, 이를 위해 유닉스 시스템 호출을 가로채는 방법을 기술한다. 제안된 방법은 유닉스의 소스 코드의 수정을 필요로 하지 않으므로, 사용자에 의해 동적으로 적용 및 변경이 가능하다. 또한, 시스템 호출 제어방법의 응용 모델로서, 사용자의 시스템 자원 접근을 제어하는 보안 모델로 설계된 UPS(Unix Protection System)에 대하여 기술한다. UPS 의 성능 평가는 벤치마크 프로그램들의 수행을 통해 추가 부하(overhead)를 살펴본다.

이후의 구성은 다음과 같다. 2 장에서는 시스템 호

출을 가로채는 방법을 제시하며, 3 장에서는 시스템 호출 제어를 통한 시스템 자원 접근 제어 방법과 이를 구현한 UPS(Unix Protection System)에 대해 기술한다. 4 장에서는 벤치마크 수행을 통해 UPS 의 성능을 평가하며, 5 장에서는 결론을 맺는다.

2. 시스템 호출 가로채기

유닉스의 커널은 여러 개의 모듈로 구성되어 있다. 이러한 모듈 구성은 커널의 변경을 용이하게 함으로써 확장성을 증대 시킨다. 즉, 커널에 기능을 추가하거나 수정이 필요할 때는 원시 코드를 고치지 않고, 새로운 기능의 코드를 작성하여 커널 영역에 적재하면 된다. 이러한 모듈 변경은 시스템이 구동 되는 도중에도 가능하므로, 확정성에서의 이점뿐만 아니라 시스템의 구동성도 높이게 된다. 새로운 하드웨어 장치에 대한 디바이스 드라이버들은 하나의 새로운 모듈로 작성되어 커널 영역에 적재함으로써 사용할 수 있다.[2]

유닉스 운영체제에서 시스템 호출은 커널 내 시스템 호출 벡터 테이블에 시스템 호출 처리기가 등록되어있고, 사용자 프로세스에서 시스템 호출을 요청할 때 등록된 시스템 호출 처리기를 수행함으로써 처리된다.[3] 그러므로, 시스템 호출 벡터 테이블에 등록된 시스템 호출 처리기를 대체 시스템 호출 처리기로 변경 등록함으로써, 해당 시스템 호출을 가로챌 수 있다. 본 장에서는 유닉스에서 커널 모듈을 사용하여 원시 코드의 수정 없이 사용자 수준에서 동적으로 시스템 호출을 가로채는 방법에 대하여 기술한다.

2.1. 구성 및 기능

시스템 호출을 가로채는 인터셉트 모듈은 엔진과 초기화 프로세스 부분으로 구성된다. 엔진은 커널 모듈의 형태로 커널 영역에 적재되어 사용자가 요청한 시스템 호출을 가로채는 역할을 수행한다. 이는 기존의 시스템 호출 벡터 테이블(System call vector table)을 대신하는 벡터 테이블과 기존의 시스템 호출 처리 루틴(System call process routine)을 대신하는 대체 처리기를 포함한다. 벡터 테이블은 인터셉트 모듈에서 가로챌 시스템 호출에 대한 정보와 이들을 처리하는 대체 처리기의 위치 정보를 포함한다. 벡터 테이블은 시스템 호출 벡터 테이블과 동일한 수의 엔트리를 가진다. 대체 처리기는 벡터 테이블에 명시된 시스템 호출들을 사용자 프로세스가 요청할 때, 벡터 테이블에 의해 기존의 시스템 호출 처리 루틴을 대신하여 호출되고 수행되는 부분이다.

초기화 프로세스는 인터셉트 모듈의 초기화 기능을 담당한다. 이는 커널에 적재된 벡터 테이블에 대체 처리기들의 위치 정보를 기록한다. 그리고, 벡터 테이블의 내용에 따라 대체 처리기를 시스템 호출 벡터 테이블에 등록하여 엔진이 수행되도록 한다.

2.2. 수행 과정

시스템 호출 가로채기 과정은 크게 초기화 과정과 엔진 구동 과정으로 나누어진다. 초기화 과정에서는 시스템 호출 벡터 테이블에서 특정 시스템 호출에 해당하는 항을 찾아 처리기의 위치 정보를 기존의 시스템 호출 처리 루틴 대신 대체 처리기의 위치 정보로 내용을 수정한다. 그럼으로 사용자 프로세스가 특정 시스템 호출을 요청하였을 때, 대체 처리기가 수행되도록 한다. 엔진 구동은 사용자 프로세스가 시스템 호출을 요청하면, 시스템 호출 벡터 테이블에 의해 대체 처리기가 구동 된다. 수행 과정을 정리하면 다음과 같다.

① 벡터 테이블과 대체 처리기를 커널 영역에 적재한다.
② 벡터 테이블에 가로채기할 시스템 호출에 해당하는 항의 처리기 정보에 대체 처리기의 위치 정보를 기록한다. 해당되지 않는 항의 처리기 정보는 NULL 로 설정한다.
③ 벡터 테이블의 각 항을 읽어서, 시스템 호출 벡터 테이블에 등록된 시스템 호출 처리 루틴의 위치 정보를 대체 처리기가 등록된 항의 값으로 변경한다.
④ 시스템 호출 처리 루틴의 위치 정보는 시스템 호출 가로채기 기능을 제거하기 위하여 저장함으로써 초기화를 마감한다.
⑤ 사용자가 시스템 호출을 요청하면, 시스템 호출 벡터 테이블에 의해서 가로챌 시스템 호출에 대해서는 대체 처리기가 자동으로 구동 된다.

가로채기 기능을 제거하기 위해서는 저장해둔 시스템 호출 처리 루틴의 위치 정보로 시스템 호출 벡터 테이블의 처리기 정보를 복원시킨다.

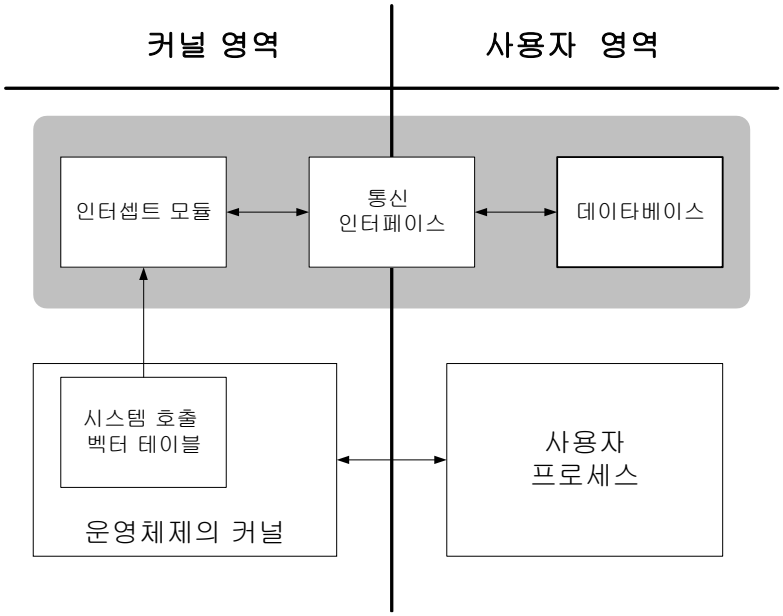

그림 1. UPS 의 블록 구성도

3. 시스템 자원 접근 제어

유닉스 사용자는 파일과 디렉토리를 사용하거나 프로세스를 생성하는 과정을 수행한다. 이들은 모두 유닉스에서 자원으로 관리되며, 이러한 처리들은 사용자가 유닉스의 시스템 호출이나 이를 이용하는 라이브러리들을 이용하여 수행한다. 즉, 사용자의 유닉스 자원에 대한 접근은 시스템 호출을 통하여 유닉스에 요청하여 사용 권한을 얻게 된다. 그러므로, 유닉스의 시스템 호출을 제어함으로써, 사용자의 자원 접근을 제어할 수 있다. 본 장에서는 2 장에서 설명된 시스템 호출 가로채기 방법을 사용하여, 동적으로 사용자의 자원 접근을 제어하는 방법을 제안하며, 이러한 기능을 수행하는 UPS(Unix Protection System)에 대하여 기술한다.

3.1. 구성 및 기능

그림 1 은 UPS 가 적재된 시스템의 유닉스 시스템의 간략적 구조를 보여준다. 음영으로 표시된 영역이 UPS 를 의미하며, UPS 는 크게 인터셉트 모듈, 통신 인터페이스, 데이터베이스의 세 부분으로 구성된다.

인터셉트 모듈은 사용자가 요청한 시스템 호출을 가로채는 부분으로 2 장에서 기술한 구조를 가진다. 데이터베이스는 사용자 프로세스가 요청한 시스템 호출을 수행할 것인지를 판단하는 부분으로, 사용자의 허가 정보를 관리하며, 인터셉트 모듈로부터의 요청에 따라 해당 정보를 검색하여 제공한다. 데이터베이스는 사용자 영역에서 구동 되며, 사용자의 허가 정보의 변경을 통해 사용자의 접근 제어 모델의 동적 적용이 가능하다. 통신 인터페이스는 인터셉트 모듈과 데이터베이스 간의 통신을 담당한다. 접근 허가 여부에 관한 정보는 데이터베이스에서 관리하며, 데이터베이스는 그림 1 에서 보는 바와 같이 사용자 영역에서 수행되므로, 이들간의 통신을 위해서는 커널 영역과 사용자 영역간의 통신 방법이 필요하다. UPS 에서는 커널 영역에 적재된 인터셉트 모듈과 사용자 영역에서 구동되는 데이터베이스 간의 통신 방법으로 디바이스 드라이버(device driver)를 사용한다. 디바이스 드라이버는 커널 영역에 존재하면서 동시에 사용자 영역에서 사용 가능한 파일 시스템에도 등록된다. 그러므로, 커널 영역에 존재하는 인터셉트 모듈과의 통신도 가능하며, 사용자 영역의 데이터베이스와의 직접적인 통신도 가능하다.

인터셉트 모듈과 데이터베이스 간의 통신은 비동기적으로 발생한다. 일반적으로 비동기적인 통신을 위해 폴링(polling)이나 busy-waiting 방법을 사용할 수 있으나, 이는 추가적인 대기 시간을 필요로 하며, 커널 내에서의 대기 시간은 시스템 자체의 성능 저하를 발생시킨다. UPS 에서는 비동기적인 인터셉트 모듈과 데이터베이스 간의 효율적인 통신을 위하여 버퍼를 가지는 스트림 디바이스 드라이버(stream device driver)[4]를 사용한다. 이는 통신이 준비되지 않았을 때는 자신을 수면(sleep)상태로 만들고 CPU 를 다른 프로세스에 양보하므로, 시스템의 성능 저하 없이 인터셉트 모듈과

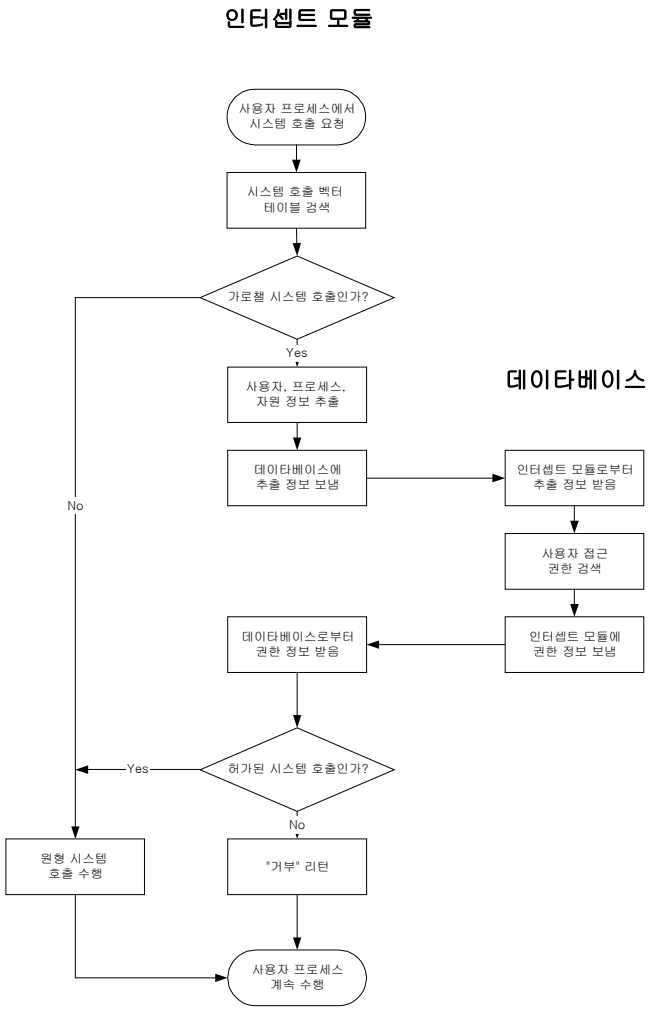

그림 2. UPS 의 접근 제어 수행 흐름도

데이터베이스 간의 비동기적인 통신이 가능하다. 인터셉트 모듈과 데이터베이스는 이러한 통신 인터페이스를 통해서 간접적으로 통신을 수행한다.

3.2. 수행 모델

그림 2 는 시스템 호출 가로채기를 통한 접근 제어 알고리즘을 보여준다. 접근 제어는 인터셉트 모듈과 데이터베이스 간의 상호 협력을 통해 이루어진다. 사용자 프로세스가 시스템 호출을 요청하면, 운영체제의 커널에 의해 시스템 호출 벡터 테이블이 검색된다. 가로챌 시스템 호출은 대체 처리기로 분기하며, 가로챌 대상이 아닌 시스템 호출에 대해서는 원형 시스템 호출 처리 루틴으로 분기한다. 대체 처리기는 시스템 호출을 요청한 사용자, 프로세스, 자원, 요청 종류 등에 대한 정보를 추출한다. 인터셉트 모듈은 추출한 정보를 통신 인터페이스를 통해 데이터베이스에 전송하고, 자원에 대한 사용자의 접근 권한을 데이터베이스에 요청한다. 데이터베이스는 인터셉트 모듈로부터 전송받은 정보를 데이터베이스에 저장된 정보와 비교하여 접근 허가 여부를 결정한다. 데이터베이스는 접근 허가 정보를 통신 인터페이스를 통해 인터셉트 모듈에게 전송한다. 데이터베이스로부터 접근 허가 정보를 받은 인터셉트 모듈은 접근 허가 정보에 따라 요청된 시스템 호출에 대한 수행 여부를 판단한다. 사용자에게 자원에 대한 접근 권한이 있을 경우, 시스템 호출 처리 루틴으로 분기하여 시스템 호출을 수행한 후, 사용자 프로세스에 결과를 리턴한다. 접근이 허가되지 않는 사용자인 경우에는 수행 거부를 리턴한후, 시스템 호출을 종료한다.

표 1. 벤치마크의 소요 시간(초 단위). 이를 통해 UPS 를 사용하였을 때, 기존 시스템에 추가되는 부하(overhead)를 파악할 수 있다.

		cp	du	gcc	gzip	gunzip	ls
UPSoff	real	30.17	119.61	57.26	175.14	13.46	133.28
	user	5.90	4.60	40.90	0.40	9.90	5.90
	sys	12.60	14.70	20.00	2.00	5.40	16.90
UPSon	real	30.41	142.58	63.69	193.78	13.53	137.10
	user	0.00	3.20	45.20	0.00	10.60	3.90
	sys	12.20	34.30	10.70	2.70	2.80	27.40
Overhead	real	1%	19%	11%	11%	1%	3%
	user		-30%	11%		7%	-34%
	sys	-3%	233%	-46%	35%	-48%	62%

4. 성능 평가

본 장에서는 실험을 통해 UPS 의 성능을 평가한다. 실험은 기존 유닉스 시스템과 UPS 를 구동 시킨 유닉스 시스템에서 응용 프로그램들을 수행시키고, 그 수행 시간을 측정하여 UPS 로 인해 발생하는 추가적인 부하(overhead)를 살펴본다.

4.1. 실험 내용

실험의 벤치마크로는 유닉스에서 사용자들이 자주 사용하는 명령어와 응용프로그램중에서 6 개를 선택하였다. cp 는 cp(1) 명령어로 전체 크기가 100MB 정도인 여러 파일들을 복사하였고, du 는 du(1M) 명령어에 -sk 옵션을 사용하여 8GB 의 크기를 가지는 유닉스 파일 시스템(UFS)의 사용 현황을 살펴보았으며, gcc 는 gcc 컴파일러를 사용하여 xxgdb 소스를 컴파일하였다. gzip 은 gzip 을 사용하여 tar 로 묶인 33MB 크기의 gcc 소스 파일을 압축하였고, gunzip 은 gunzip 을 사용하여 gzip 으로 압축된 gcc 소스 파일의 압축을 풀어주었으며, ls 는 ls(1) 명령어에 -aR 옵션을 사용하여 8GB 크기의 파일 시스템의 모든 리스트를 출력하였다.

실험은 솔라리스 2.5.1 을 사용하는 SPARCstation20 에서 수행되었다. 벤치마크들을 10 번씩 반복적으로 수행시켰고, 측정된 수행 시간의 평균을 구하였다. 수행 시간의 측정에는 솔라리스에서 제공하는 time(1) 명령어를 사용하여 각 프로그램들의 수행 시간을 측정하였다.

4.2. 실험 결과

표 1 은 벤치마크들의 평균 수행 시간을 초단위로 보여준다. UPSoff 는 기존 시스템에서의 수행 시간을 나타낸다. 이는 사용자가 수행한 시스템 호출들을 그대로 모두 수행한 결과이다. UPSon 은 기존 시스템에 UPS 를 구동시켰을때의 수행 시간을 나타낸다. 이는 사용자가 요청한 시스템 호출을 수행할 뿐만 아니라 시스템 호출을 가로채고, 데이터베이스와의 통신을 수행하여 허가를 인증 받는 과정을 추가적으로 수행한 시간이다. 현재의 UPS 의 데이터베이스는 권한 허가에 대한 검색을 수행하지 않고, 사용자의 ID(uid)만을 확인하여 허가여부를 결정하므로, 데이터베이스 내에서의 검색에 소요되는 시간은 고려되지 않았다고 본다. 또한, 실험에서는 데이터베이스 내에서의 거부 결정에 의한 벤치마크 수행이 중지되는 것을 방지하기 위하여, 모든 사용자에 대하여 모든 자원에 대한 접근을 허가하도록 설정한 후 수행하였다.

UPS 를 사용한 경우(UPSon)에서 사용하지 않은 경우(UPSoff)보다 1~19%의 추가적인 시간이 소요되었다. 여기서 각 벤치마크마다 다양한 비율의 추가 소요시간을 가지는 것은 각 벤치마크에서 사용되는 시스템 호출의 분포에 영향을 받기 때문이다. UPS 는 모든 시스템 호출을 가로채는 것이 아니라, 파일이나 프로세스와 같이 자원을 접근하는 시스템 호출에 대해서만 가로채기를 수행하기 때문이다. 그러므로, 파일 접근이 많은 du 에서 가장 많은 추가 부하를 가지게 되며, 파일 접근보다 연산이 많은 gunzip 에서 추가 부하가 적다. 또한, 동일한 자원에 대한 동일한 접근을 수행하는 경우-예를 들면 파일을 읽거나 쓰기 위해 반복

적으로 read(2)나 write(2)를 수행하는 경우-에는 반복적으로 시스템 호출을 가로채지 않기 때문에 동일한 파일을 읽거나 쓰는 작업이 많은 cp 나 ls 의 경우는 유사한 작업을 수행하는 du 에 비하여 부하가 적게 걸린다.

5. 결론

유닉스의 커널은 시스템의 자원을 관리하여 모든 시스템이 원활히 돌아갈 수 있도록 제어하는 유닉스의 핵심 부분으로, 유닉스 안에 있는 모든 응용 프로그램들은 커널이 제공하는 서비스에 의존한다. 시스템 호출은 사용자가 시스템의 자원을 사용하기 위해 커널에 요청하는 인터페이스이다.

본 논문에서는 사용자 수준에서 시스템 호출을 제어하는 방법 및 구성을 제안하고, 이를 위해 유닉스 시스템 호출을 가로채는 방법을 기술한다. 제안된 방법은 유닉스의 소스 코드의 수정을 필요로 하지 않으므로, 동적으로 적용 및 변경이 가능하다. 또한, 시스템 호출 제어방법의 응용 모델로서, 사용자의 시스템 자원 접근을 제어하는 보안 모델로 설계된 UPS(Unix Protection System)에 대하여 기술한다. UPS 의 성능 평가는 벤치마크 프로그램들의 수행을 통해 추가 부하(overhead)를 살펴본다. 실험 결과에 의하면 UPS 를 사용한 경우는 사용하지 않은 경우보다 1~19%의 추가적인 시간이 소요되었다.

참고문헌

[1] 조유근, "UNIX 의 내부구조", 홍릉과학출판사, 1994.

[2] SunSoft, "SunOS 5.2 Writing Device Drivers", SunSoft, 1993.

[3] Steve D Pate, "UNIX Internals : A Practical Approach", Addison-Wesley, 1996.

[4] SunSoft, "SunOS 5.2 STREAMS Programmer's Guide", SunSoft, 1993.