Linux Filesystem을 위한 Force Unmount 설계 및 구현

김동욱*°, 임은지*, 차규일*, 정성인*
*한국 전자통신연구원 시스템소프트웨어팀

Design and Implementation of Force Unmount for Linux Filesystem

Kim, Dong-Wook°, Lim Eun-Ji, Cha Gyuli, Jung Sung-In
Electronics and Telecommunications Research Institute
E-mail : {kdw9234, ejlim, gicha, sijung}@etri.re.kr

요 약

이 논문은 리눅스 파일시스템상에 FU(Force Unmount) 기능 구현에 관한 내용을 기술하고 있다. 현재 리눅스는 엔터프라이즈 서버로 진화하기 위해서 많은 기능이 요구 되고 있는 상황이다. OSDL의 CGL 문서는 FU 기능 제공에 대해서 Property 1로 정의하고 있다. FU 기능 제공은 다른 기술 제공을 위한 기반 기술로 인식되고 있기 때문이다. 이 논문은 FU 구현 시 문제점과 고려 사항을 기술 하며, 구현 후 안정성과 기능성 검증을 위해서 테스트 결과를 담고 있다.

1. 서론

이전의 리눅스는 Web server, FTP와 같은 Network-edge service 목적으로 사용되었지만, 현재 점차적으로 엔터프라이즈 서버로 발전하고 있다. 이를 지원하기 위해서 OSDL과 같은 그룹에서 리눅스 발전 방향을 제시하고 있다.

OSDL의 CGL(Carrier Grade Linux)[1]은 많은 기능들을 정의하고 있는데, 그 중에 FU(Forced Unmount) 기능이 Property 1로 포함되어 있다.

FU는 그 이름에서 알 수 있듯이 파일시스템상에 오픈 파일 또는 수행 테스크가 존재하는 상황하에서도 그 파일시스템을 강제로 해제할 수 있는 기능을 제공한다.

엔터프라이즈 서버는 수많은 클라이언트 요청들을 처리하기 위해서 무수한 파일 작업을 수행해야만 한다. 이러한 상태가 지속적으로 발생하면, 아무리 견고한 시스템이라 할지라도 사용된 모든 리소스들을 정상적으로 해제하는 것을 보장하지 못한다. 이 상황에서 블럭 디바이스나 시스템을 업그레이드를 위해서 잠시 파일시스템을 해제해야 한다면, 파일시스템 해제를 위해서 일반적으로 사용하는 umount는 정상 처리되지 못할 것이며, 사용자는 시스템 재기동 이외에는 다른 대안을 찾을 수 없을 것이다.

이 문제를 해결하기 위해서 리눅스에 FU를 개발하였다. 개발된 FU는 nested filesystem 지원과 종류에 상관없이 모든 파일시스템을 지원하여 다른 FU 기능에 비해서 높은 기능성을 갖고 있으며, LTP와 I/O bound 도구를 이용한 테스트를 통해 높은 신뢰성을 제공하고 있다.

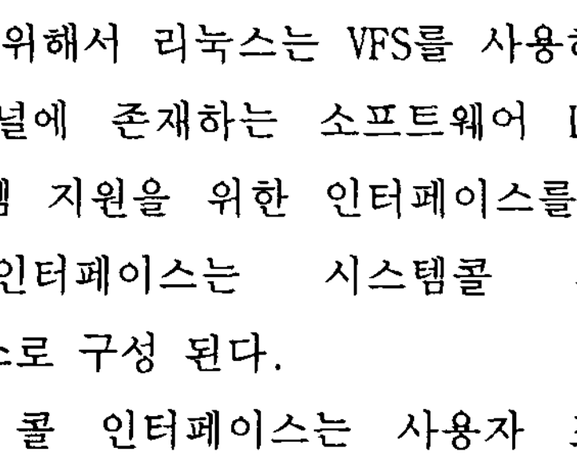
2. Linux Filesystem

파일시스템 등록과 해제는 리눅스 가상 파일시스템(VFS, Virtual FileSystem)과 밀접한 관련 있다. 파일시스템은 등록된 후 그 파일시스템의 관리 정보들이 VFS로 올라오게 되고, 그 파일시스템의 자원 접근시 이용된다. FU는 이때 변경된 정보를 저장하고, 할당된 자원들을 해제한 후, 파일시스템을 해제하는 작업을 수행한다.

2.1 VFS(Virtual Filesystem)

Ext3와 Resierfs와 같은 이기종 파일시스템들을 지원하기 위해서 리눅스는 VFS를 사용하고 있다. VFS는 커널에 존재하는 소프트웨어 Layer이며, 파일시스템 지원을 위한 인터페이스를 제공하고 있다. 인터페이스는 시스템콜 과 내부 인터페이스로 구성 된다.

시스템 콜 인터페이스는 사용자 프로그램에 의해서 발생 되는 파일 작업을 처리하기 위해서 사용되며, 내부 인터페이스는 파일시스템 관련 데이터와 함수들을 커널의 다른 기능들에 의해서 사용될 수 있도록 지원한다.

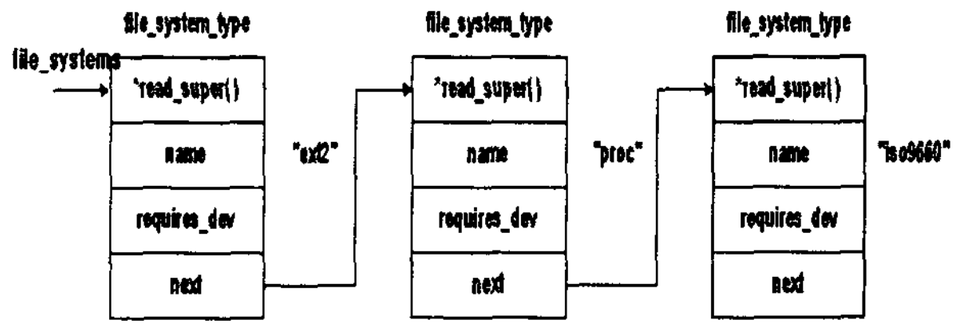


<그림 1> 가상 파일시스템

2.2 파일시스템 등록

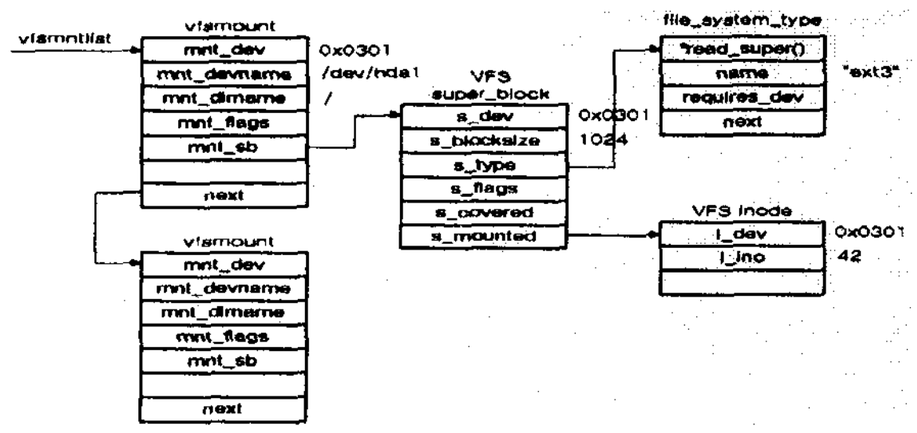
사용자가 파일시스템 등록을 시도하면, 시스템콜 sys_mount가 호출되어진다. 이때 파일시스템 종류, 디바이스 이름, mount point가 커널로 전달되어진다.

등록의 최초 작업은 커널이 갖고 있는 지원 가능한 파일시스템 정보들을 검색하여 원하는 파일시스템 정보를 획득하는 것이다. 이 작업은 구조체 file_system_type의 리스트인 file_systems 탐색으로 수행된다.



<Figure 2> List of file_system_type

요청된 파일시스템 종류에 매칭되는 file_system_type이 발견되면, 여기에 담긴 정보를 이용하여 물리 저장매체에 존재하는 파일시스템의 superblock을 VFS로 로딩하여, 등록 작업을 완료하게 된다. 아래 그림은 ext3 파일시스템을 VFS에 등록한 후 커널에 유지되는 정보들이다. 파일시스템 해제는 이 정보들을 삭제하는 작업을 포함하게 된다.

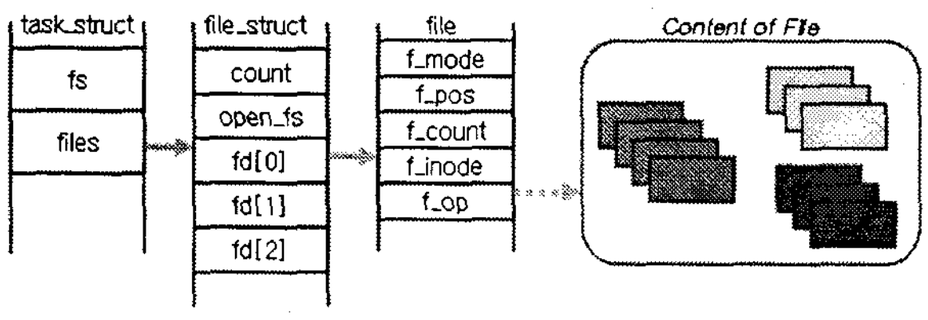


<그림 3> 마운트된 파일시스템

2.3 테스크의 파일 오픈

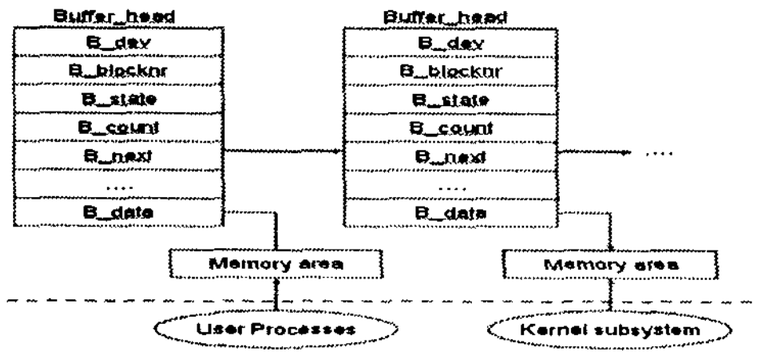
사용자 테스크에 의해서 파일 오픈 요청이 발생하면, 파일 경로를 이용하여 해당 Inode를 찾는 작업을 수행한다. 만일 그 Inode가 VFS에

있는 경우는 테스크에 할당하여 그대로 사용할 수 있지만, 존재하지 않는 경우는 파일시스템에서 해당 Inode를 VFS로 올리게 된다. 이때 사용되는 메쏘드는 read_inode() 함수이다. 이 함수는 마운트된 파일시스템의 superblock내에 유지되고 있다.



<그림 4> 테스크에 의해서 오픈된 파일

파일 오픈을 요청한 테스크에 Inode가 할당되면, 테스크는 이 Inode를 이용하여 파일의 내용을 메모리로 로드하고, Read/Write 작업을 수행한다. 이때 파일의 내용은 VFS의 buffer에 담기게 되면, 이 buffer는 리눅스 메모리 관리의 page들로 이루어졌다.



<그림 5> Buffer's Memory area

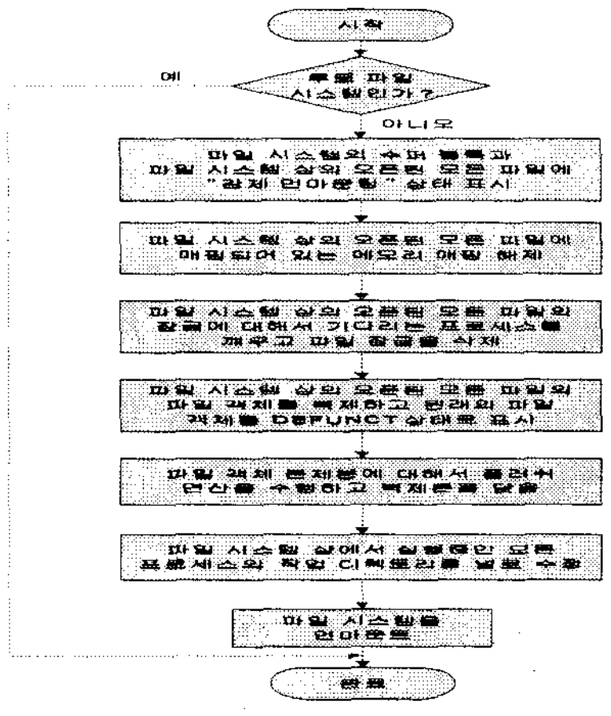
파일시스템 해제를 위해서는 테스크에 할당된 파일과 그 파일에 할당된 버퍼들을 해제하는 작업이 선행되어야 한다.

3. FU 설계 및 고려

VxFS와 같은 기존 상업용 파일시스템들은 자사 제품에 대해서만 FU 기능을 지원하고 있다. 그러나 리눅스는 VFS을 통해서 이기종 파일시스템들을 지원하고 있기 때문에 VFS를 통해 파일시스템 종류에 관계없이 FU 기능을 지원할 수 있다[2]. 우리는 FU 설계시 이 점을 고려하여 모든 파일시스템에 FU 기능을 지원하도록 하였다.

3.1 FU 수행 절차

사용자가 FU를 수행하기 위해서 시스템 콜(sys_umount)을 발생하면, 아래 <그림 6>과 같은 흐름에 따라 FU 수행이 이루어진다. 호출된 sys_umount는 먼저 해당 파일시스템의 상태 정보에 MNT_FFORCE 플래그를 추가하여 파일시스템 접근을 봉쇄하고, superblock의 오픈 파일 리스트를 이용하여 모든 오픈 파일의 메모리 매핑과 Lock을 삭제하도록 한다. 그리고 전체 테스크들을 탐색하여 CWD(current working directory)와 mount point의 값을 NULL로 변경한다. 마지막으로 파일시스템의 reference count가 2되는 경우 umount 시킨다.



<그림 6> FU control flow

3.2 unmount 수행

파일시스템 해제를 위한 기존의 umount는 해당 파일시스템이 busy 상태인 경우 수행이 실패된다. 파일시스템을 busy 상태로 만드는 경우는 다음 4가지 경우로 분류할 수 있다.

- 테스크의 CWD가 파일시스템하의 위치로 설정되어 있는 경우
- 오픈된 파일이 존재하는 경우
- 수행중인 테스크의 바이너리 파일이 존재하는 경우
- Nested Filesystem이 존재하는 경우

일반 Umount 기능과 FU의 큰 차이점은 설계된 FU가 파일시스템을 "busy"에서 "idle" 상태로 전환 하는 기능을 갖고 있는 점이다. FU는 해제 대상이 되는 파일시스템에 관련된 테스크와 오픈 파일의 리소스 해제 작업을 통해 그 파일시스템을 "idle" 상태로 전환시킨다.

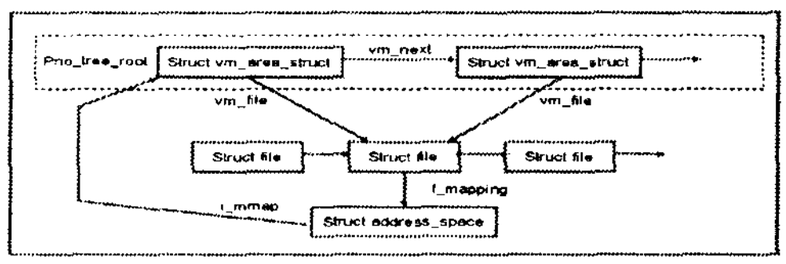
3.3 오픈 파일

파일시스템 해제를 위해서 FU는 강제적으로 오픈 파일들에 닫기 작업을 수행해야 한다. 닫기 작업은 관련된 리소스들을 해제한 후 수행되어야 한다. 먼저 오픈 파일의 리스트 획득을 위해서 등록된 파일시스템의 superblock의 정보를 탐색한다. 그리고, 각 오픈 파일의 변경 내용을 디스크에 작성하고, 오픈 파일에 할당된 메모리 매핑 영역을 삭제한다. 더 이상의 변경을 하지 못하도록 파일 오퍼레이션들 NULL로 초기화 시킨다.

A. Remove Page-Mapping

커널 2.6은 물리적 어드레스를 가상 어드레스로 변환하기 위해서 ramps 대신에 file 별로 address_space를 사용하고 있다[4]. 이것은 파일의 내용 로드하기 위해서 사용된 메모리 가상 어드레스 정보들을 갖고 있기 때문에 이것을 이용하여 파일에 매핑된 메모리를 삭제할 수 있다.

파일 구조체에는 메모리 매핑을 나타내기 위해서 두 종류의 address_space가 존재한다[6]. 첫번째 Linear Page-Mapping는 파일 내용이 선형적 어드레스 형태로 유지하고 있으며, NonLinear Page-Mapping는 Linear Page-Mapping으로 재 매핑이 발생하기 전에 로드된 파일 내용을 임시적으로 담고 있다. 이 두 개의 address_space 정보를 이용하여 파일에 매핑된 메모리를 해제하게 된다. <그림 7>의 struct file은 오픈 파일을 나타내며, 여기에 연결된 vm_area_struct 영역이 FU에 의해서 강제적으로 해제 된다.



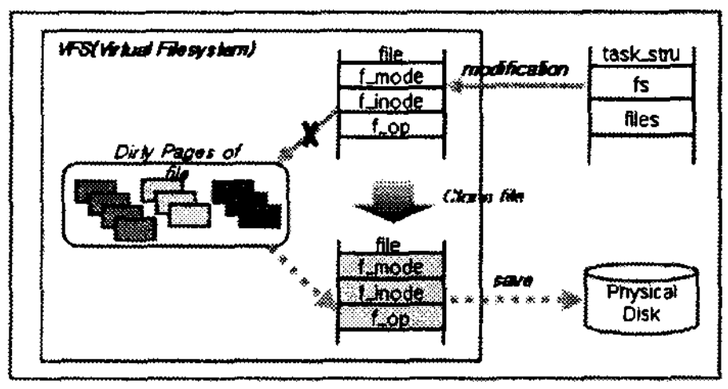
<그림 7> 파일과 메모리 관계

B. Clone File Object

Memory Mapping을 해제 이후, 테스크가 더 이상 그 파일의 내용을 변경하지 못하도록 파일 접근 메소드들을 해제해야 한다. 이 작업은 파일에 등록된 접근 메소드들의 어드레스를 NULL 설정하는 것으로 간단히 해결할 수 있다.

그러나 문제는 테스크가 이전에 변경한 파일 내용은 Dirty buffer들에 존재하며, 이 buffer들은 디바이스로 저장해야 한다. 이때 저장을 위해서 사용하는 도구가 파일에 등록된 접근 메소드들이다. 따라서 접근 메소드는 테스크로 부터 사용은 막으면서, Dirty buffer를 저장하기 위해서 사용되어야 한다.

이 문제를 해결하기 위해서 Clone File Object 개념을 도입하였다. 먼저 테스크에 속한 파일 구조체를 새로운 파일 구조체로 복제시키고, 이 복제본을 super_block에 등록시킨다. 그리고 테스크의 속한 파일 구조체의 접근 메소드 정보를 NULL로 변경하면, 이후 테스크는 파일 내용을 변경할 수 없고, VFS는 Dirty buffer들을 싱크할 수 있다.



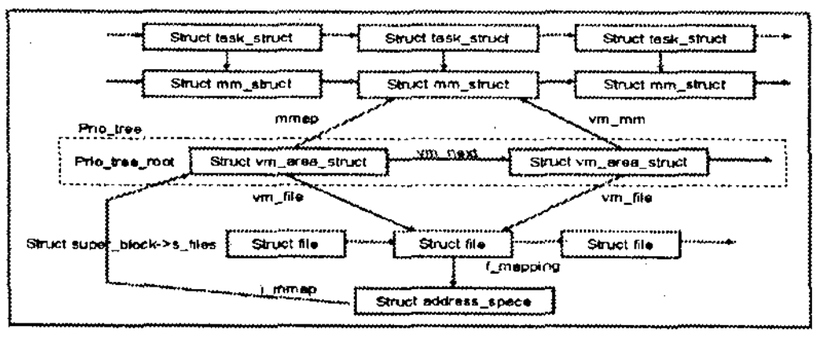
<그림 8> clone file object

3.4 바이너리 파일

바이너리 파일은 오픈 파일과 다르게

task_struct가 추가 생성된다.(<그림 9> 참조)[6]. 이 경우 리소스 해제 작업은 오픈 파일 해제 작업과 비슷한 방법으로 이루어진다.

오픈 파일 처리 방식과 다른 점은 테스크를 나타내는 task_struct를 강제적으로 해제하지 않고, 테스크 수행시 세그먼트 폴트(Segment Fault) 유발하여 커널의 에러 처리 기능에 의해 task_struct를 해제 하도록 한다. 테스크는 수행할 명령어인 opcode를 CPU로 로딩하기 위해서 mm_struct를 통해 vm_area_struct 영역에 접근하게 된다. 그러나 메모리가 해제된 이후에는 더 이상 유효한 메모리 주소를 나타내지 못하기 때문에 커널에 의해서 세그먼트 폴트가 발생되고 task_struct는 자동적으로 해제 된다.



<그림 9> task_struct와 file 연결

3.5 작업 디렉토리

테스크의 작업 디렉토리로 지정되면, 해당 파일시스템의 reference count가 증가된다. 이것은 파일시스템을 “busy” 상태로 만들어 파일시스템을 해제할 수 없게 한다. 이 문제를 처리하기 위해서 해당 파일시스템을 사용하는 테스크의 CWD 와 현재 위치하는 디렉토리에 대한 처리 함수들을 NULL 설정하여 “idle” 상태로 만들고, reference count를 감소시키다.

3.6 Utile fumount과 Nested filesystem

기존의 FU 기능들은 nested filesystem이 존재하는 경우 FU를 수행시키지 못하였다. 이 문제를 해결하기 위해서 fumout 도구를 설계하였다. 이 도구는 사용자가 파일시스템을 FU하려고 할 때, 커널에 MNT_FFOURCE 플래그와 함께 sys_umount를 발생시켜 커널의 FU 기능을 기동시키는 역할을 수행한다.

fumount는 다른 파일시스템이 sub-mounted 되어 있는 경우, 하단의 파일시스템부터 강제 해제하여 원하는 파일시스템을 해제 시킨다.

이 경우 고려할 점은 nested filesystem의 존재 유무를 파악하여 처리하는 주체를 결정하는 문제이다. Nested filesystem을 지원하는 주체가 커널 또는 사용자 유틸이 될 수 있다. 커널의 시스템 콜은 atomic operation 형태를 지향하는 것이 좋은 방법이며, 사용자 유틸에 의해서 관리되는 마운트 파일시스템 정보를 유지하기 위해서는 nested filesystem을 처리하는 주체가 사용자 유틸이 되는 것이 낫다

4. FU 구현 및 검증

리눅스 커널 2.6.8 기반에서 위의 사항들을 고려하여 구현된 FU는 다음과 같은 기능을 제공한다.

- 해제 되는 파일시스템을 네임 스페이스에 나타나게 하지 않는다.
- 그 파일시스템에 접근하는 테스크가 I/O operation 발생시 I/O error(EIO)를 발생시킨다.
- 파일 오픈과 같은 추가 자원 접근을 방지한다.
- 또한 그 파일시스템상의 오픈 파일들은 Close를 제외한 다른 Operation의 접근을 차단한다.
- 테스크 또는 커널 내부 연산에 의해서 발생된 Lock들을 Release 시킨다.
- 파일 I/O operation이 memory-mapped file에 접근할 때 어플리케이션은 Segmentation violation을 유도한다.
- 마지막으로 파일시스템 종류에 관계없이 모든 파일시스템을 강제로 해제 할 수 있다.

FU의 기능과 안정성을 검증하기 위해서

POSTMARK[3] 도구를 사용하여 실험환경을 구성하였다. POSTMARK는 파일시스템 성능 측정을 위해서 파일시스템에 많은 파일들을 생성하고 Read/Write 작업을 반복적으로 수행하여 파일시스템 성능을 측정하는 도구이다. 우리는 POSTMARK 수행시 10000개의 파일 생성과 10000개의 transaction을 수행 하도록 설정하고, POSTMARK 수행 중간에 FU 수행시켜 정상적으로 해제되는 것을 확인하였다.

아래 표는 /home을 POSTMARK 수행 중간에 FU 실행 시킨 결과이다. POSTMARK의 작업 디렉토리를 /home/postmark로 설정하여 /home에 커다란 I/O 연산을 발생시켰다.

```
#df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1              39G   15G   22G  40% /
none                  506M     0  506M   0% /dev/shm
/dev/hda3             146G   58G   88G  40% /home


# /root/PostMark/postmark < /root/PostMark/benchinp
ut > log&
[1] 6723


# /root/FU/fumout /home
.........
Error: Cannot delete '/home/postmark/15042'
Error: Cannot delete '/home/postmark/15043'
/home umounted
[1]+  Done    /root/PostMark/postmark </root/PostMar
k/benchinput > log


# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1              39G   15G   22G  40% /
none                  506M     0  506M   0% /dev/shm
```

<표1> POSTMARK 수행시 FU 수행 예제

또한 FU 지원을 위해서 리눅스 커널 수정은 불가피한 작업이기 때문에 수정 이전과 동일한 안정성이 유지되고 있다는 보장이 필요하다. 커널의 안정성 테스트를 위해 LTP(Linux Test Project) 도구를 사용하였다. 아래 결과는 수정 이전 커널 2.6.8과 FU를 지원하도록 수정된 커널 2.6.8-fu의 LTP 수행 결과이다.

Kernel 2.6.8.1	Total Tests: 807 Total Failures: 4 Kernel Version: 2.6.8.1 Machine Architecture: i686 Hostname: ia32e-1
Kernel 2.6.8.1-FU	Total Tests: 807 Total Failures: 4 Kernel Version: 2.6.8-fumount Machine Architecture: i686 Hostname: ia32e-1

<표2> LTP 수행 결과

5. FU(Force Unmount) 관련 연구

Unix 계열의 FU는 2000년 VxFS3.4 부터 Sun의 Solaris 7.0과 8.0 기반으로 개발되어 지원되고 있다[5]. 이 FU는 파일시스템 VxFS만을 지원하고 시스템의 다른 파일시스템은 지원하지 못하는 단점을 갖고 있다.

리눅스에서 FU 기능 지원은 Montavista사의 커널 2.4대 임베디드 리눅스 개발 플랫폼에서부터 지원되기 시작하였다. 현재 Montavista에 의해서 버전 2.6에 대한 지원은 존재하지 않는다.

6. 결론

현재 ODSL의 CGL 문서는 Force Unmount 기능을 Property 1으로 정의하고 있다. 이 기술은 리눅스를 엔터프라이즈 서버로 진입시키기 위해서 해결되어야 할 문제 중 하나이다.

우리가 개발한 FU는 nested filesystem 지원과

모든 파일시스템을 지원하여 다른 그룹에서 지원하고 있는 FU에 비해서 높은 기능성을 갖고 있으며, 또한 LTP와 I/O bound 어플리케이션을 이용한 테스트를 통해 높은 신뢰성을 제공하고 있다. 향후 FU 수행시 발생 가능한 Data Lose 문제를 해결하기 위해서 연구할 계획이며, 개발한 FU 코드를 공동체 그룹에 공개하여 성능 및 기능 개선을 할 예정이다.

[참고문헌]

[1] OSDL Group : Carrier Grade Linux Requirements Definition V2.0.2. http://www.osdl.org

[2] Daniel P. Bovet, Marco Cesati : Understanding the linux kernel, O'ReillyY&Associates, Inc.

[3] Jeffery Katcher : PostMark – A New File System Benchmark, Network Appliance Inc.

[4] Martin J. Bligh, David Hansen : Linux Memory Management on larger machines, proceedings of the linux symposium 2003

[5] VERITAS Software Inc. : VERITAS Techtalk - #232976

[6] Mel Gorman :understanding the linux virtual memory manager : Prentice Hall Inc.