Linux Devfs (Device File System) FAQ

Richard Gooch

rgooch (at) atnf dot csiro dot au

도운이

최종필

viatoris (at) nownuri dot net

김영휘

lachesis (at) kldp dot org

이 문서의 원본은 http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html (http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html) 에서 찾을수 있고, 커널 소스와 함께 제공되는 텍스트 문서보다 보기에 더 편하다. 미러사이트는 http://www.ras.ucalgary.ca/~rgooch/linux/docs/devfs.html (http://www.ras.ucalgary.ca/~rgooch/linux/docs/devfs.html) 에 있다.

또한 devfs와 같이 사용되는 데몬이 있는데, http://www.atnf.csiro.au/~rgooch/linux/ (http://www.atnf.csiro.au/~rgooch/linux/) 에서 더 많은 정보를 얻을수 있다.

당신이 신청할 경우 메일링 리스트의 사용이 가능하다. mailto:majordomo (at) oss.sgi.com (mailto:majordomo@oss.sgi.com) 로 메일 본문에 **subscribe devfs** 과 같은 내용을 포함하여 메일을 보내라. 신청취소는 **unsubscribe devfs** 을 포함한 메일을 보내면 된다.

이 리스트는 http://oss.sgi.com/projects/devfs/archive/ (http://oss.sgi.com/projects/devfs/archive/) 에 보관된다.

참고: http://your.destiny.pe.kr/l (http://your.destiny.pe.kr/) 에서 최신 수정본을 보실 수 있습니다.

고친 과정

고침 2-FEB-2002 2002. 2. 2 고친이 R.Gooch
원본 문서 업데이트
고침 4-FEB-2002 2002. 2. 4 고친이 viatoris
번역 완료
고침 18-FEB-2002 2002. 3. 2 고친이 라키시스
DocBook 으로 변환
고침 158-MAR-20022002. 3. 15고친이 최종필
XFree86 부분의 오역 수정

1. 이것은 무엇인가?

Devfs는 루트 파일시스템에서 "실제" 캐릭터 및 특수 블럭디바이스를 취급하는 것에 대한 다른 방법이다. 커널 디바이스 드라이버는 메이저/마이너 번호에 의해 등록되는 것보다 더 나은 이름으로 디바이스를 등록할수 있다. 이 디바이스들은 그 드라이버에 지정된 소유권과 보호 권한을 가진 상태로 자동적으로 devfs에 나타날 것이다. 데몬(devfsd)은 이러한 기본값을 덮어써서 사용할 수 있다. Devfs는 커널 2.3.46 이후 버전부터 사용할수 있다.

> 참고**:** devfs는 옵션이라는 것을 명심하자. 만약 예전방식의 disc-based 디바이스 노드를 선호한다면, 간단하게 CONFIG_DEVFS_FS=n (기본값)라고 두면 된다. 이 경우에, 아무것도 바뀌지 않는다. 또한 devfs를 활성화시켰다면 기본값은 예전의 디바이스 이름과 완벽하게 호환되도록 관리된다는 것을 기억하라.

devfs를 바라보는데에 두가지 관점이 있다 : 하나는 디바이스 네임스페이스가 단지 마운트된 파일시스템처럼 네임스페이스라는 것에 기초한 것이다. 다른 하나는 그 디바이스 네임스페이스의 관점을 제공하는 파일시스템의 코드에 대한 관점이다. 내(원저자)가 이런 구분을 만든 이유는 devfs는 각 마운트마다 같은 디바이스 네임스페이스를 보여주면서 여러번 마운트 될 수 있기 때문이다.

devfs에 대한 댓가는 커널코드의 크기와 메모리 사용량이 소량 증가하는 것이다. 약 7페이지의 코드(__init 섹션에서)와 그 네임스페이스에서 각 개체에 대하여 72바이트의 메모리가 필요하다. 보통의 시스템은 수백개의 디바이스 개체를 가지며, 이것은 좀더 많은 페이지를 필요로 한다. ramdisc 에 /dev 를 넣는 방법과 이것을 비교해 보라.

일반적인 머신에서, 그 댓가는 0.2 퍼센트 이하이다. 64메가의 램을 가지는 보통의 시스템에서는 0.1 퍼센트 이하이다. devfs 에 대해 "bloatware"라는 비난은 정당성이 없다.

2. devfs 를 사용할 이유

devfs를 필요로 하는 여러 문제점이 있다. 이들 문제들 중에 어떤 것들은 (당신의 관점에 따라서) 매우 심각하고, 어떤 것들은 devfs 없이 해결할 수 있다. 그러나, 전체적인 면에서 보면 이 문제들은 devfs를 필요로 한다.

선택할 수 있는 것은 복잡하고 헛점투성이인 필요없는 userspace 에서의 해결 패치를 내놓는 것과, 간단하고 견고하며 효과적인 devfs를 사용하는 것의 둘 중 하나이다.

devfs에 대한 많은 반론이 있고, 모두다 devfs를 구현하지 않으면 얻어지는 이익들을 포함하고 있다. 하지만, 지금까지는 코드가 없거나, devfs가 제공하는 모든 기능들을 제공할수 있는 어떤 대안도 없다. 게다가, 제안된 대안들은 사용자 공간에서 더 복잡하다. (그리고 devfs보다 더 적은 기능을 가진 상태로 제공된다) 어떤 사람들은 "커널 크기"의 축소를 목표로 삼지만, 그들은 userspace 에 미치는 영향은 고려하지 않는다.

커널공간과 사용자 공간의 전체적인 복잡함의 정도를 제한한 (복잡성이 덜한) 방법이 좋은 솔루션이다.

2.1. 메이저 번호와 마이너 번호의 할당

현존하는 스키마는 각각의 디바이스 마다 메이저/마이너 디바이스 번호의 할당을 요구한다. 이것은 각 디바이스마다 고유한 번호를 가질 수 있게 하기 위하여 (당신이 "개인적"인 디바이스 드라이버를 개발하지 않는한) 이들 디바이스 번호들을 처리하기 위하여 중앙집중적인 권한이 요구된다는 것을 의미한다. Devfs는 네임스페이스에 대한 부하를 덜어준다. 이것은 크게 부하를 덜어주는 것처럼 보이지 않지만, 실제로는 엄청난 효과를 낸다. 드라이버 관리자들은 자연적으로 그 디바이스의 기능을 반영하는 디바이스 이름을 선택하므로, 네임스페이스의 충돌에 대한 잠재적 위험은 적다. 이것을 해결하기 위해서는 커널의 변경이 필요하다.

2.2. /dev 관리

현재의 디바이스 접근 방식은 디바이스 노드를 통해서 행해지는 방식이기 때문에, 시스템 관리자에 의해서만 디바이스 노드가 만들어져야만 한다. 표준 디바이스에 대하여 당신은 모든 (수백개의!)노드를 만들어내는 MAKEDEV라는 프로그램을 찾을 수 있다. 이것은 커널에서의 변화는 MAKEDEV 프로그램에서 반영되어야 하거나, 시스템 관리자가 수동으로 디바이스 노드를 만들어야 한다는 것을 의미한다.

기본적인 문제는 메이저 번호와 마이너 번호를 구분하는 두개의 데이터베이스가 존재한다는 것이다. 하나는 커널에 있고, 나머지는 /dev에 있다 (또는 만약 당신이 그런 방법으로 보기 원한다면 MAKEDEV 프로그램에 있다.). 이것은 실용적이지 못한 정보의 중복이다. 이 문제를 해결하기 위해서는 커널의 변경이 필요하다.

2.3. /dev 디렉토리의 증가

전형적인 /dev 디렉토리에는 1200개 이상의노드가 있다! 이 디바이스들의 대부분은 하드웨어가 없기 때문에 존재하지 않는다. 이 거대한 /dev 는 디바이스 접근 시간을 증가시킨다 (나는 단지 dentry 탐색 시간과 존재하지 않는 디스크의 inode를 읽는데 걸리는 시간만을 계산한 것이다 : 다음 항목에서는 좀더 끔찍한 예를 보여준다)

다음의 예는 SCSI 디바이스를 고려한다면 /dev 디렉토리가 얼마나 커질수 있는 가를 보여준다.

```
host            6  bits  (say up to 64 hosts on a really big machine)
channel         4  bits  (say up to 16 SCSI buses per host)
id              4  bits
lun             3  bits
partition       6  bits
TOTAL          23 bits
```

가능한 모든 디바이스 노드를 저정하기 원한다면 8메가(1024*1024)의 inode를 필요로 한다. 만약 id, partition 을 제외하고 SCSI 타겟(id)당 하나의 SCSI 버스와 하나의 논리적 유닛을 가지는 호스트 어뎁터로 가정한다 할지라도, 여전히 10bits 또는 1024개의 노드가 남는다. 각 VFS inode는 대략 256 bytes를 차지하므로 (kernel 2.1.78), (실제 inode 역시 VFS inode와 유사한 양의 공간을 차지한다고 가정하면) 디스크상에 256kBytes의 inode 저장공간이 필요하다. 오늘날 디스크는 저렴하므로, 나쁘지는 않다. 임베디드 시스템은 /dev inode의 256kBytes에 대해 주의해야 하지만 임베디드 시스템은 /dev 디렉토리를 수동으로 조정할 수 있다. 나 자신의 임베디드 시스템에서는 그렇게 해야겠지만, devfs를 사용하는 방법을 택할 것이다.

다른 문제는 최초로 참조될때 inode 를 탐색하는데 걸리는 시간이다. 메모리상의 리스트를 통하여 탐색하는데 걸리는 시간 뿐만 아니라, 디스크에서 inode를 읽는데 걸리는 시간도 있다. 이 문제는 커널로그를 분석하여 필요하지 않으면 /dev 개체를 삭제하고 필요할때 생성하는 프로그램을 사용하여 사용자 공간상에서 해결할 수 있다. 이 프로그램은 새로운 모듈이 로드될때마다 실행될 것이고, 많은 부분 느려지게 될 것이다.

SCSI 디바이스에 대하여 디바이스 노드를 자동으로 만드는 scsidev 라 불리는 프로그램이 있다. 이 프로그램은 /proc/scsi에서 파일을 탐색하는 방법으로 작동한다. 불행히도, 다른 디바이스 노드에 대해 이 개념을 확장하기 위해서는 현존하는 드라이버들을 많은 부분 수정하는 것이 필요하다 (/proc 파일 시스템에 정보를 제공해야 하기 때문에). 이것은 간단한 변경이 아니다 (나는 devfs가 유사한 일을 한다는 것을 알고 있다). 당신이 이러한 모든 노력을 하더라도, (이 정보를 제공하고 있는) devfs를 사용하는 것이 나을 것이다. 게다가, 여러개의 서로 다른 디바이스 드라이버들이 서로 다른 방법으로 각각의 정보를 제공하는 것처럼, 그러한 시스템은 특수한 목적을 위해 만들어지는 시스템들과 마찬가지로, 각각의 특성에 맞게 구성되어야 한다.

Devfs는 디바이스 노드 스스로 자연스럽게 정보를 제공하게끔 하는 일정한 메커니즘을 가지고 있으므로 시스템의 구성이 보다 더 명확해진다!!

2.4. 디바이스 드라이버의 **struct file_operations** 구조체에 대한 노드

디바이스 넘버를 가지고 (disc-based) /dev 디렉토리 아래의 문자 / 블럭 장치 노드들과 실제 디바이스 드라이버를 연결하는 방법과, devfs 에서 등록된 개체들이 실제 디바이스 드라이버에 연결되는 방법 사이에는 매우 중요한 차이점이 있다.

8비트의 메이저/마이너 번호를 가지고 disc-based c&b 노드들과 해당 메이저 넘베의 디바이스 드라이버를 연결하는 것은 고정된 128 개의 엔트리 테이블을 통하여 이루어진다. 파일시스템들은 문자/블록 장치파일에 대한 inode operation 을 {chr,blk}dev_inode_operations 로 설정해 둔다. 따라서, 디바이스가 open 되었을 때, 약간의 indirect call 만으로 빠르게 file_operation 구조체의 함수 포인터들을 참조할 수 있게 된다.

다른 기타의(miscellaneous) 문자 디바이스들 -- (역자주 : 마이너 넘버들에 따라 전혀 다른 장치가 될 수도 있는 장치들, 예:tty (major number 3,4)) -- 때문에 두번째 단계가 필요하다 : 그 파일이 열렸을때 같은 마이너 번호를 가진 드라이버 엔트리를 탐색하고, 적당한 마이너 오픈 메쏘드를 호출한다. 이 탬색은 디바이스 노드를 열때 항상 수행된다. 어쩌면, 오픈 메쏘드를 찾기 전에 수십개의 miscellaneous 개체를 탐색할 것이다. 많은 오버헤드를 일으키지는 않는다 하더라도, 필요없는 작업으로 보인다.

리눅스는 때때로 8비트의 메이저/마이너 경계를 넘어서야만 하는 경우도 있다. 만약 16비트로 각각의 번호의 크기를 증가시킨다면, 메이저 테이블은(캐릭터 및 블럭 디바이스에 대해 하나의) 64k 엔트리가 필요하기 때문에 (x86 에서는 512kBytes, 64 비트 시스템에서는1 메가바이트), 그 메이저 드라이버를 사용하는 인덱싱 스키마는 사용할 수 없게 된다. 따라서 miscellaneous 캐릭터 디바이스를 사용하는 것과 비슷한 스키마를 사용해야만 한다. 이것은 탐색시간이 당신의 시스템에서 메이저 디바이스 드라이버의 평균 수에 따라 선형적으로 증감한다는 것을 나타낸다. 모든 "디바이스"가 하드웨어는 아니다. 어떤 것은 KGI처럼 좀더 고수준의 드라이버들이다. 따라서 당신은 하드웨어의 증가 없이 더 많은 "디바이스"들을 얻을 수 있다. 당신은 질서정연한(균형잡힌:-) 바이너리 구조를 생성함으로써 이것을 향상 시킬수 있고, 이 경우에 탐색시간은 log(N)으로 된다. 다른 방법으로는, 탐색시간을 향상시키기 위해 해싱(hashing)을 사용할 수 있다. 그러나 꼭 해야만 하는 것이 아니라면 왜 모두를 탐색하려 하는가? 다시 한번, 이것은 필요없는 것처럼 보인다.

devfs는 메이저&마이너 시스템을 사용하지 않는 다는 것을 기억하라. devfs 엔트리에 대한 연결은 /dev 개체를 탐색했을때 일어난다. devfs_register() 함수가 호출되면, 내부의 테이블은 그 엔트리 이름과 file_operations을 추가한다. 그 dentry 캐시가 미리 /dev 엔트리를 가지고 있지 않다면, 이 내부의 테이블은 file_operations를 얻기 위하여 탐색되고, inode는 생성된다. 만약 dentry 캐쉬가 이미 그 엔트리를 가지고 있다면, 탐색시간은 필요없다 (other than the dentry scan itself, but we can't avoid that anyway, and besides Linux dentries cream other OS's which don't have them:-). 또한, devfs에 노드 엔트리의 수는 사용가능한 디바이스 엔트리의 수와 같다. 즉, "상상할 수 있는" 수가 아니다. disc-based /dev에서 필요없는 엔트리를 지운다 하더라도, 공간을 절약하기 위하여 스스로 제한하더라도 상상할 수 있는 엔트리의 수는 여전히 같은 채로 남아있다.

Devfs는 **VFS** 노드와 디바이스 드라이버 사이의 빠른 연결을 제공한다**.**

2.5. 시스템 관리 도구로서의 **/dev**

/dev 디렉토리는 내가 가지고 있지 않는 대부분의 디바이스를 포함하여 가능한 모든 디바이스의 노드를 포함한다. Devfs는 단지 시스템에서 사용가능한 디바이스만 보여준다. 이것은 /dev의 목록을 만든다는 사용가능한 디바이스를 편리한 방법으로 체크한다는 것을 의미한다.

2.6. 메이저**&**마이너 크기

현재 메이저/마이너 번호들은 각각 8비트로 제한되어 있다. 이것은 하나의 메이저 넘버를 소비하는 SCSI 디스크 드라이버와 같은 드라이버에서는 한계요인이 된다. 16개의 디스크만 지원되고 각 디스크는 15개의 파티션을 가질 수 있다. 아마 이것은 당신에게는 문제가 아닐지

모르나, 디스크 어레이를 사용하여 큰 리눅스 시스템을 만드는 이에게는 큰 문제가 된다. Devfs에서 임의의 포인터는 32비트 디바이스 구분자를 (i.e. 32비트 마이너 넘버를 가지는 것처럼 보이는) 사용가능한 각각의 디바이스 엔트리로 연결시킬수 있다. 이것은 커널에만 관련된 것이므로, 메이저/마이너 번호를 크기를 증가시키는 것에 대하여 C 라이브러리와의 호환에 대한 논란은 필요없다. 사용자 공간에서 호환성을 유지하는 것에 대한 세부사항은 "디바이스 번호의 할당"에 대한 섹션을 보라.

이 문제를 해결하는데는 커널의 변경이 필요하다**.**

이것을 작성함으로써, 커널은 SCSI 디스크 드라이버가 더 많은 메이저 번호를 할당받고, 128 디스크 이상을 지원하도록 수정되었다. 이들 메이저 번호들이 비연속적 (무계획적인 확장의 결과로 인해) 이므로, 그것의 구현은 처음보다 더 성가신 일이 되었다.

IP 주소의 부족을 일시적으로 극복하기 위해 IPv4 를 변경하는 것과 마찬가지로, 사람들은 제한사항이 생기면 그것을 둘러갈 트릭을 찾는다. 그러나 IPv6 나 devfs 와 같은 근본적인 해결책의 도입은 더이상 늦춰져서는 안된다.

2.7. 읽기전용 루트 파일시스템

루트 파일시스템에 디바이스 노드를 가진다는 것은, 만약 루트 파일 시스템이 읽기 전용으로 마운트되었을 경우엔 여러분의 시스템이 제대로 작동하지 못한다는 것을 의미한다. 그것은 tty 디바이스의 소유권과 보호권을 변경해야 하기 때문이다. 실제로 루트 파일시스템 디바이스로서 읽기 전용 장치인 CD-ROM 을 사용하는 것은 허용되지 않는다. 확실히, 당신은 CD-ROM을 이용하여 부팅을 할수는 있으나, tty의 소유권을 변경하지는 못한다. 따라서 그 방법은 인스톨할때나 좋은 방법이다.

또한, 디스크가 없는 (discless) 리눅스 머신 에서 (보통 /dev 에서 변경된 tty 소유권을 가지는 것은 좋지 못하다) 공유된 NFS 루트 파일시스템을 가질 수도 없다. 또한 ROM-FS 에 당신의 루트 파일시스템을 포함시킬수도 없다.

당신은 부팅시에 RAMDISC 를 만들고, 그것에 ext2 파일시스템을 만든후, 그것을 어떤 장소에 마운트하고 /dev의 목록을 그것에 복사한 후, 마운팅을 해제하고 /dev에 그것을 다시 마운팅 하는 방법을 사용하게 될 것이다.

Devfs는 이 문제를 해결하는데 있어 보다 명확한 방법을 제공한다.

2.8. Non-Unix 루트 파일시스템

Non-Unix 파일시스템(NTFS와 같은)은 그것들이 다양한 캐릭터/블럭 스페셜 파일 또는 심볼릭 링크들을 지원하지 않기 때문에 루트 파일시스템으로 사용할 수 없다. 당신은 마운트 하기 전에 디바이스 노드를 필요로 하기 때문에 /dev 에 마운트된 파일시스템이 disc-based 인지 RAMDISC-based 인지 구분할 수 없다. Devfs는 디바이스 노드 없이 마운트 할 수 있다. Devlink는 심볼릭 링크(symlink)의 지원이 안되기 때문에 작동하지 않을 것이다. 다른 해결책은 RAMDISC 초기 루트 파일시스템 (소수의 선택된 디바이스 노드를 포함하는) 을 마운트 하기 위하여 initrd를 사용하고, 다른 RAMDISC 에 새로운 /dev를 만든 후, 마지막으로 non-Unix

루트 파일시스템으로 바꾸는 것이다 이것은 영리한 부트스크립트와 헛점이 많고 복잡한 부트 과정이 필요하다.

Devfs는 견고하고 개념적으로 간단한 방법으로 이 것을 해결한다.

2.9. PTY 보안

현재의 pseudo-tty(pty)는 루트가 소유권자이고, 모든 사용자가 읽고 쓰기가 가능하다. pty-pair의 사용자는 suid-root 없이는 소유권/보호권을 바꿀수 없다.

이 문제는 루트로 수행되고 실제 pty-pairs 를 만들어 내는 userspace 에서 작동하는 보안 데몬을 사용함으로써 해결될 수 있다. 이 같은 데몬은 이런 새로운 매커니즘을 사용하고자 하는 "모든" 프로그램에 수정을 요한다. 또한 pty-pairs를 생성하는 데에도 속도저하가 일어난다.

다른 해결책은 사용자공간의 데몬과 비슷한 일을 하는 새로운 open_pty() 시스템콜을 만드는 것이다. 다시 한번 말하지만, 이것은 pty 핸들링 프로그램들의 수정을 요구한다.

devfs 의 해결책은 열려있지 않은 디바이스가 열렸을때 디바이스 드라이버가 어떤 디바이스 파일을 "tag"하도록 허용하고, 소유권은 열린 프로세스의 현재 euid와 egid로 변화되고, 보호권은 그 드라이버에 의해 등록된 기본값으로 변경된다. 그 디바이스가 닫히면, 소유권은 루트로 돌아가고 보호권은 모든 이에게 읽고쓰기 가능으로 바뀐다. 어떤 프로그램도 수정될 필요가 없다. devpts 파일시스템은 Unix98 ptys 에 대하여 이러한 자동-소유권의 기능을 제공한다. 이것은 옛방식의 pty 디바이스는 지원하지 않고, devfs가 제공하는 다른 기능들을 모두 제공하지 않는다.

2.10. Intelligent device 유지

Devfs는 사용자 공간에서 작동하는 디바이스 관리 데몬(devfsd)과의 통신을 위하여 간단하지만 강력한 프로토콜을 구현한다. 이것은 디바이스 엔트리의 등록/등록해제, 디바이스의 open/close, inode 의 탐색 디렉토리와 다른 것들의 탐색 등과 같은 이벤트가 일어나면 (동기 또는 비동기적으로) devfsd 에 메세지를 보내는 것이 가능하다. 이것은 많은 잠재성을 가지게 된다. 그 잠재성 중의 어떤 것은 이미 구현되어 있다. http://www.atnf.csiro.au/~rgooch/linux/ (http://www.atnf.csiro.au/~rgooch/linux/)을 살펴보라

디바이스 엔트리의 등록 이벤트는 새롭게 생성되는 디바이스 노드의 퍼미션을 변경할때 devfsd에 의해 사용될 수 있다. 이것은 디바이스의 퍼미션을 제어하기 위한 하나의 매커니즘이다.

디바이스 엔트리의 등록/등록해제 이벤트는 프로그램이나 스크립트를 구동시키는데 사용된다. 이것은 새로운 블럭 디바이스 미디어에 드라이버가 추가되었을때 파일시스템의 자동적인 마운트를 제공하는데에도 사용될 수 있다.

비동기 디바이스의 open/close 이벤트는 퍼미션 관리를 구현하는데 사용될 수 있다. 예를 들어, /dev/dsp에 대한 기본 퍼미션은 모든 사용자가 그 디바이스를 읽는 것을 허용하지 않는다. 이것은 당신이 콘솔에 얘기하는 것을 원격 사용자가 녹음하는 것을 원하지 않을 것이기 때문에 합당한 일이다. 그러나, 콘솔 사용자 역시 녹음하는 것이 막혀있다. 이것은 바람직한 것이 아니다. 비동기 디바이스에 open/close 이벤트를 이용하여 당신은 콘솔디바이스가 다른 디바

이스 노드 (예를 들어 /dev/dsp)에 대한 소유권을 변경하기 위해 열렸을때, devfsd가 어떤 프로그램이나 스크립트를 수행하도록 할 수 있다. 마지막으로, 당신은 다른 스크립트를 이용하여 퍼미션을 복구 할 수 있다. C 라이브러리의 tty 핸들링의 수정이 필요없는 이런 설계의 이득은 당신의 프로그램이 충돌이 일어나더라도 (당신은 존재하지 않은 로긴을 위하여 지체되는 엔트리를 가지는 utmp 데이터 베이스를 수없이 보아오지 않았는가?) 작동한다는 것이다.

동기 디바이스의 열기 이벤트는 디바이스가 접근권한을 설정하는데 사용될 수 있다. 디바이스 드라이브의 open() 메소드가 호출되기 전에, 데몬은 외부 프로그램이나 스크립트를 실행시킴으로써 처음으로 열기시도를 확인할 것이다. 이것은 그에 대한 접근 자체가 UID와 GID 대신에 다른 시스템의 상태에 기초하여 결정되기 때문에 접근 제어 리스트를 사용하는 것보다 훨씬 더 유연하다.

Inode 탐색 이벤트는 모듈의 자동로드 요청을 인증하는데 사용될 수 있다. kmod를 직접적으로 사용하는 대신에, 이 이벤트는 모듈 스스로가 적재되기 전에 임의적으로 인증을 구현할수 있는 devfsd 에 보내진다.

Inode 탐색 이벤트는 존재하지 않는 디바이스에 대하여 심볼릭 링크로 devfs 를 위치시키기 위해 복구하지 않고도 임시로 네임스페이스를 구성하는데 역시 쓰일수 있다.

2.11. 이론적인 디바이스 탐색

모듈이 올라와 있던 아니던 시스템에 있는 모든(SCSI, IDE 및 다른 타입들) CD-ROM 디바이스를 찾기 위한(cdparanoia와 같은) 어플리케이션을 고려해보자. 이론적으로 어플리케이션은 해당 모듈이 적재되어 있는지 확인하기 위해서는 디바이스 노드를 (SCSI CD-ROM의 경우 /dev/sr0) 열어야만 한다. 표준 디바이스 네이밍 스키마 (최근에 레드햇은 이것을 다르게 구현한 것을 확인했다) 를 따르는 모든 리눅스 배포판은 이러한 작업을 필요로 한다. Devfs는 그러한 네이밍 문제를 해결한다.

그와 같은 어플리케이션은 또 시스템에 실제 사용가능한 디바이스를 찾기를 원할 수도 있다. 현존하는 시스템으로 이같은 일은 /dev 디렉토리를 읽고, 그 디바이스가 존재하는지 안하는지를 결정하기 위해서는 각 /dev/sr* 디바이스들을 open 해 봐야 한다. 큰 /dev 를 가지는 시스템에서, 그 시스템이 특히 많은 수의 /dev/sr* 노드를 가지고 있다면, 이것은 비효율적인 작업이다. scsidev와 같은 해결책은 /dev/sr* 엔트리의 수를 줄여줄수 있다 (그러나 역시 불필요한 디렉토리 탐색을 하게 된다).

Devfs를 사용하면, 그 어플리케이션은 `/dev/sr` 디렉토리 (필요하다면 모듈의 자동적재를 유도해내는)를 읽을수 있고, `/dev/sr`를 읽기 위하여 계속 진행된다. 오직 사용가능한 디바이스들만이 엔트리를 가지므로, 불필요한 디렉토리 탐색이나, 디바이스의 오프닝은 필요없다.

3. 누가 이 일을 하는가?

FreeBSD는 devfs를 구현한다. 솔라리스와 AIX 역시 pseudo-devfs(scsi와는 비슷하지만 비명시적인 커널 지원과 함께 모든 디바이스를 지원하는)를 가지고 있다. BeOS, Plan9와 QNX 역시 이것을 가지고 있다. SGI의 IRIX 6.4 이상의 것 역시 디바이스 파일시스템을 가지고 있다.

우리는 다른 사람들이 하기 때문에 자동적으로(의무적으로) 이것을 할 필요는 없지만, 다른 사람의 작업을 무시해서도 안된다. FreeBSD 는 이 작업을 하는데 있어 수많은 개발자를 소유하고 있고, 따라서 그들의 의견은 간단하게 무시되어서는 안된다.

4. 어떻게 작동하나?

4.1. 디바이스 엔트리를 등록하기

devfs에 기초한 /dev 안에서 각 엔트리(디바이스노드)에 대해 드라이버는 devfs_register()를 호출해야만 한다. 이것은 내부 테이블에 디바이스 엠트리의 이름과 file_operation 구조체 포인터와 약간의 다른 것들을 추가한다. 디바이스 엔트리는 어떤 때라도 추가되거나 제거될 수 있을 것이다. 디바이스 엔트리가 등록되면, devfs가 마운트된 곳에 자동으로 나타난다.

4.2. Inode 탐색

엔트리에 대한 탐색이 수행되고 만약 그 엔트리에 대한 어떠한 드라이버 정보도 없을때 devfs는 devfsd의 호출을 시도한다. 만약 어떠한 정보도 찾아지지 않는다면, negative dentry가 얻어지고, 다음 단계의 수행은 VFS(create() 또는 mknod() 함수처럼 inode 를 조작하는 메쏘드와 같은)에 의해 호출될 것이다. 드라이버의 정보가 찾아지면 inode는 생성될 것이고(미리 존재하는 것이 없다면) 그것으로 잘 될 것이다.

4.3. 직접 device노드를 만들기

mknod() 메쏘드는 사용자에게 devfs 안에 고유하게 이름붙여진 파이프를 만들도록 허용하거나, 이미 존재하지 않는다면 블럭 스페셜 inode를 만들수 있도록 허용한다. 사용자는 사용자 스스로 퍼미션과 소유권을 설정할 수 있는 캐릭터나 블럭 스페셜 inode를 생성하기를 원할 수도 있다. 나중에, 만약 디바이스 드라이버가 같은 이름으로 엔트리를 등록한다면, 그 퍼미션과 소유권과 시간은 계속 유지된다. 이것은 그 드라이버가 적재되기 전이라도 디바이스에 대한 보호권을 설정할 수 있도록 한다. 한번이라도 inode를 만들었다면 디렉토리에 나타난다.

4.4. 다비이스 엔트리의 등록해제

디바이스 드라이버는 엔트리를 등록해제하기 위해서 devfs_unregister()를호출한다.

4.5. Chroot()의 제한들

- **2.2.x**대의 커널들

  inode 생성의 의미는 devfs가 "특별한" 옵션과 함께 마운트 되었을때는 다르다. 지금, 디바이스 엔트리가 등록되었을때, 그 디바시으를 생성하기 위해 mknod() 를 사용하기 전까지는 나타나지 않을 것이다. mknod()가 devfs_register() 를 사용하여 그 디바이스가 되기 전이던 중요하지 않다. 이런 작업의 목적은 당신이 그 제한안에 최소한의 devfs를 마운트 하기 원하는 곳에서 chroot(2)의 제한을 지원하는 것이다. 단지 당신이 특별히 사용가능하도록 설정하는(당신만의 mknod() 설정을 통하여) 디바이스만이 접근 가능하게 될 것이다.

- **2.4.x** 대의 커널들

  현재의 2.3.99 커널은 VFS가 전체 파일시스템의 네임스페이스의 한 부분을 그 네임스페이스의 다른 부분으로 리바인드 시킬수 있다. 이것은 심지어 leaf-node 레벨에서도 작동하며, 각각의 파일들과 디바이스 노드가 그 네임스페이스의 다른 부분으로 바인드 될수 있다는 것을 의미한다. 이것은 링크를 만드는 것과 유사하지만, 파일시스템(하드 링크와는 다른)과 chroot() 의 제한(심볼릭 링크와 다른)을 통하여 작동하기 때문에 더 낫다.

  VFS에서의 이런 향상 덕분에, devfs에서 다중 마운트에 대한 기능은 더이상 필요하지 않다. 관리자는 VFS 바인딩을 사용하여 chroot(2) 제한 안에 최소한의 디바이스 트리를 만들게 된다. 이것은 devfs의 다중마운트 능력이 가진 대부분의 기능을 제공하기 때문에, 나는(저자는) 다중 마운트 코드를 (RFC 에서의 논의 후에) 빼버렸다. 이것은 코드크기의 감소와 간소함을 가져다 주었다.

  만약 최소한의 chroot() 제한을 만들기 원한다면, 다음의 명령어를 치면 된다 :

  ```
  mount --bind /dev/null /gaol/dev/null
  ```

  보여주기 원하는 다른 디바이스 노드에도 반복하라. 간단하다!

5. 운영상의 문제점

5.1. 참을성 없는 이들을 위한 소개

문서읽기를 좋아하는 사람은 없다. 사람들은 바로 얻고 수행하기를 원한다. 따라서 이번 부분에서는 빠르게 /dev 상에 devfs를 마운트 하여 수행하는데 있어 필요한 단계 를 설명할 것이다. 이번 단계를 건너뛰어라. 그러면 당신은 거의 부팅할수 없는 시스템을 가진채 끝날 것이다. 아래의 부분들은 좀더 자세한 부분에 대해 설명하고 있고, 필수적이지 않은 설정옵션에 대해 논의할 것이다.

5.1.1. Devfsd

OK, 당신이 이부분을 읽고 있다면, 나는 당신이 devfs를 사용하기를 원한다고 생각할 것이다. 첫번째로 당신은 `/usr/src/linux`에 최신의 커널소스를 포함하고 있는지 확실하게 해야 한다. 그리고 디바이스 관리 데몬인 devfsd 를 컴파일 할 필요가 있다. 이것은 http://www.atnf.csiro.au/~rgooch/linux/ (http://www.atnf.csiro.au/~rgooch/linux/) 에서 얻을수 있다. 그 커널은 옛 버전의 네이밍 스키마와는 상당히 다른 네이밍 스키마를 가지기 때문에 예전의 네이밍 스키마가 깨지지 않고 계속 사용할 수 있도록하는 소프트웨어와 설정파일인 devfsd를 설치할 필요가 있다.

devfsd를 컴파일 하고 설치하라. 당신은 예전 네이밍 스키마에 대하여 호환성을 가지는 심링크를 제공하는 기본 설정파일인 `/etc/devfsd.conf`를 제공받을 것이다. 충분한 지식을 얻기 전까지 이 파일을 수정하지 말라. 충분한 지식을 가지고 있다고 생각한다 할지라도, 아래에서 부팅된 devfs가 가능한 시스템과 이것이 제대로 작동하는지에 대해 검증이 끝날때까지 수정하지 말라.

> 이제 devfsd가 가장 처음(파일시스템의 체크가 일어나기 전에)에 시작되도록 메인 시스템의 부트 스크립트를 수정해야 한다. 가금 `/etc/rc.d/rc.sysinit`이 SysV 형식의 부트 스크립트와 함께 시스템에서 메인 부트 스크립트로 사용된다. BSD 형식의 부트 스크립트를 가지는 시스템에서는 `/etc/rc` 도 사용된다. `/sbin/rc`도 체크해보라.
>
> 참고**:** 부트 스크립트에 넣어야 될 줄은 정확하게 다음과 같아야 함을 명심하라 :
>
> ```
> /sbin/devfsd /dev
> ```
>
> 절대로 어떠한 데몬 실행 프로그램도 사용하지 말라. 그렇지 않으면 부트스크립트는 devfsd가 초기화하는 것을 기다려주지 않는다.

5.1.2. 시스템 라이브러리들

잘못된 소프트웨어가 디바이스 네임에 대해 가정하기 때문에 일어나는 문제점들이 있다. 자세하게 말하면, 어떤 소프트웨어는 심볼릭 링크로 된 디바이스를 사용하지 못한다. 만약 libc5를 사용하는 시스템을 사용하고 있다면, libc5.4.44를 설치해라(만약 libc5.4.46을 사용하고 있다면 올바르게 작동하는 libc5.4.44로 다운그레이드하라). 만약 glibc에 기초한 시스템을 사용한다면 glibc 2.1.3 이상인지 확인해야 한다.

5.1.3. /etc/securetty

PAM(Pluggable Authentication Modules)은 더 나은 사용자 인증과 서비스를 제공하기 때문에 유연한 메커니즘으로 평가받고 있다. 불행하게도, 그것은 깨지기 쉽고, 복잡하고 문서화 되어 있지 않다(레드햇 6.1과 다른 배포판을 확인해보라). PAM은 심볼릭링크와는 문제를 일으킨다. 다음의 줄을 `/etc/securetty` 파일에 추가시켜라 :

```
vc/1
```

```
vc/2
vc/3
vc/4
vc/5
vc/6
vc/7
vc/8
```

이것은 보안을 약화시키지 않을 것이다. 만약 2.10.h 이전의 util-linux 버전을 사용하고 있다면 그 이후의 버전으로 업그레이를 하기 바란다. 만약 업그레이드 할 수 없다면, 다음의 줄을 `/etc/securetty` 파일에 추가하라 :

```
1
2
3
4
5
6
7
8
```

이것은 네트워크를 통한 루트의 로그인을 허용하기 때문에(패스워드는 요구된다) 보안을 약화시킨다. 그러나, 심링크를 취급하는데 있어서 문제가 있기 때문에, 나는 어떤 경우에라도 제공되는 보안레벨에 의심을 가지고 있다. (역자 주 : PAM이 심링크에 문제점을 보이기 때문에, 어차피 보안에는 문제점이 있다는 말)

5.1.4. Xfree86

필수적이지는 않지만, 좀더 devfs에 밀접하도록 하는 패치의 일환으로 XFree86 4.0으로 업그레이드 하는 것은 좋은 생각이다. 만약 그렇게 하지 않는다면, 일반 사용자가 startx를 할수 있도록 `/etc/security/console.perms`에 다음의 패치를 적용해야 할 것이다. 모든 배포판이 이 파일을 가지고 있는 것은 아니므로(e.g. 데비안) 이것이 존재하지 않는다고 해도 걱정할 필요는 없다.

```
--- /etc/security/console.perms.orig    Sat Apr 17 16:26:47 1999
+++ /etc/security/console.perms Fri Feb 25 23:53:55 2000
@@ -14,7 +14,7 @@
# man 5 console.perms

# file classes -- 이것은 정규표현식이다.
-<console>=tty[0-9][0-9]* :[0-9]\.[0-9] :[0-9]
+<console>=tty[0-9][0-9]* vc/[0-9][0-9]* :[0-9]\.[0-9] :[0-9]

# device classes -- these are shell-style globs
<floppy>=/dev/fd[0-1]*
```

만약 패치가 적용되지 않는다면, 다음과 같이 변경하라.:

```
<console>=tty[0-9][0-9]* :[0-9]\.[0-9] :[0-9]
```

을

```
<console>=tty[0-9][0-9]* vc/[0-9][0-9]* :[0-9]\.[0-9] :[0-9]
```

으로 수정하면 된다.

5.1.5. devpts를 중지시키기

나는 /dev/pts에 마운트된 devpts가 제대로 작동하지 않는다는 보고를 받았다. devfs는 /dev/pts를 관리하므로, 더 이상 devpts를 마운트할 필요는 없다. /etc/fstab파일을 수정해서 마운트되지 않도록 하거나, 커널 설정에서 devpts를 사용하지 않도록 하라.

5.1.6. 지원되지 않는 드라이버

모든 드라이버가 devfs를 지원하지는 않는다. 만약 이러한 드라이버 중에 하나를 가지고 있다면, 적당한 때에 디바이스 노드를 생성하도록 부팅시에 사용할 수 있는 스크립트나 tarfile를 작성하는 것이 필요할 것이다. 5.3절 은 이부분을 설명하고 있다. 5.3절 은 devfs를 지원하는 드라이버들의 목록이다.

5.1.7. /dev/mouse

많은 배포판들이 XFree86과 GPM의 마우스 디바이스를 /dev/mouse로 설정한다. 나는 이것을 원래와 다르게 우회해서 추가하기 때문에 좋지 못한 생각이라고 생각한다. 만약 당신이 설정파일에서 살펴볼 때, /dev/mouse를 보게 된다면 마우스가 참조된 곳이 어디인지 궁금하게 여길 것이다. 따라서 나는 /etcX11/XF86Cnfig파일(그리고 GPM 설정 파일)에 실제 마우스 디바이스(예를 들어,/dev/psaux 와 같은)를 기입하는 것을 추천한다.

다른 방법으로는 위에 설명된 지원되지 않는 드라이버를 위한 테크닉을 사용해도 된다.

5.1.8. 커널

마지막으로, devfs가 커널에에서 컴파일 되도록 해야 한다. 가장 좋아하는 커널 설정툴(예를 들면, **make config** 또는 **make xconfig**)을 사용하여 CONFIG_EXPERIMENTAL=y, CONFIG_DEVFS_FS=y 그리고 CONFIG_DEVFS_MOUNT=y 로 설정하고 **make dep;make clean** 하고 커널과 모듈을 재컴파일 하라. 부팅시에 devfs가 /dev 에 마운트 될 것이다.

만약 부팅시에 문제(예를 들어, 설정 단계를 잊었다던지 하는)가 발생한다면, 커널 부트 커맨드 라인에 **devfs=nomount**를 넣고 통과할 수 있다. 이것은 커널이 부팅시에 devfs를 /dev에 마운팅 하는 것을 막아준다. /proc/partitions, irrespective of whether devfs is mounted.

일반적으로 CONFIG_DEVFS_FS=y로했지만 /dev에 마운트되지 않도록 만들어진 커널은 안전하다, 또한 설정의 변경도 필요없다. 주의를 기울여야 할 한가지 예외는 **LABEL=** 지시자가 /etc/fstab에 사용되었을 때이다. 이런 경우에는 정상적으로 부팅할 수 없을 것이다. 이것은 **mount(8)** 프로그램이 그 볼륨 레이블을 찾는 과정의 일부분으로 /proc/partitions를

사용하고, 찾은 디바이스 이름이 CONFIG_DEVFS_FS=y로 설정하는 것이 `/proc/partitions` 에서 그 이름을 변경시키고, devfs가 마운트 되었는지 상관하지 않기 때문에 사용불가능하게 되기 때문이다.

이제 필요한 모든 단계를 끝냈다. 이제 당신의 빛나는:-) 새로운 커널을 부팅할 준비가 되었다. 즐겨보라.

5.1.9. 설정파일 변경하기

OK, 이제 devfs가 사용가능한 시스템을 부팅시켰고, 모든 것이 제대로 작동한다. 이제 설정파일을 변경시킬 필요를 느낄 것이다(일반적으로 설정대상은 `/etc/fstab`와 `/etc/devfsd.conf`이다). 제대로 작동하는 시스템을 갖고 있기 때문에, 설정을 변경하고나서 제대로 작동하지 않더라도, 디폴트로 설정파일을 복구하면 잘 작동한다는 것을 알 수 있다.

5.2. 리부팅 후의 퍼미션 유지

만약 디바이스 파일을 생성하기 위하여 **mknod(2)**를 사용하지 않고 소유권/퍼미션을 변경하기 위해 **chmod(2)** 또는 **chown(2)**을 사용하지 않는다면 inode의 ctime는 0(12 am, 1-JAN-1970, GMT)으로 남겨질 것이다. 이후의 ctime을 가지는 것들은 소유권/퍼미션의 변화가 있을 것이다. 따라서, 간단한 스크립트 또는 프로그램을 사용하여 셧다운하기 전에 모든 변경된 inode들을 묶는데 사용한다. 효과적이긴 하지만 이것을 적용하기 위해서는 많은 고려사항이 있다.

더 나은 접근방법은 퍼미션을 저장하고 북구하는데 devfsd를 사용하는 것이다. 이것은 퍼미션에서의 변경을 기록하기 위해서 설정되고, 데이타베이스(실제로는 디렉토리 트리)에 저장하고, 부팅시에 복구할 것이다. 이것은 현재의 퍼미션을 저장하는데 있어서 효율적인 방법이며 효과적인 결과를 보여준다(알수 없는 미래의 시간에 퍼미션을 저장하는 tar 의 접근과는 다르게)

devsd에 제공되는 기본설정파일은 지속적인 관리가 가능하도록 하기 위해 주석이 달린 설정부분을 가지고 있다.

만약 어떤 방법으로던 tar 접근을 사용하기로 결정했다면, tar은 새로운 디바이스 노드를 생성하기 전에 먼저 inode를 **unlink(2)** 할 것이라는 것을 주의하라. **unlink(2)**는 devfs 엔트리와 디바이스 드라이버 사이의 연결을 끊어버리는 효과를 가진다. 만약 부팅 옵션에서 **"devfs=only"**를 사용하고 있다면, 모듈을 적재하기 위해 필요한 디바이스 드라이버에 대한 통로를 잃게 된다. 나는 이것을 tar의 버그라고 생각하고 있다(실제로 tar가 inode를 먼저 **unlink(2)**할 필요는 없다)

다른 방법으로는, 디바이스의 좀더 철학적인 퍼미션 관리를 제공하기 위하여 devfsd를 사용할 수도 있다. 당신은 디바이스당 하나의 엔트리를 할당하는 진부한 방법보다, 하나의 설정을 통하여 모든 디바이스 그룹에 대하여 퍼미션을 저장할 수 있다.

5.2.1. `/dev`에 마운트된 퍼미션 데이타베이스의 저장

만약 devfs가 /dev에 마운트 된채로 disc-baed /dev에 디바이스의 퍼미션을 저장하거나 복구하기를 원한다면 그렇게 할 수도 있다. 이것은 2.4.x 커널 (실제로는, 2.3.99 이후)의 VFS 바인딩 기능을 필요로 한다. 이 기능을 설정하기 위해서는 아래와 같이 하라:

- 부팅시에 devfs가 마운트 되지 않도록하라
- /dev-state디렉토리를 만들어라
- 부트 스크립트의 앞부분에 다음의 줄을 추가하라

```
mount --bind /dev /dev-state
mount -t devfs none /dev
devfsd /dev
```

- /etc/devfsd.conf에 다음을 추가하라:

```
REGISTER ^pt[sy]/.* IGNORE
CHANGE ^pt[sy]/.* IGNORE
REGISTER .* COPY /dev-state/$devname $devpath
CHANGE .* COPY $devpath /dev-state/$devname
CREATE .* COPY $devpath /dev-state/$devname
```

- 리붓

5.2.2. 일반 디렉토리에 퍼미션 데이터베이스 복구하기

만약 VFS 바인딩을 지원하지 않는 오래된 커널을 사용중이라면, /dev에 마운트되어 있는 퍼미션 데이타베이스를 가져올 수 없을 것이다. 그러나 그 데이타베이스를 복구하기 위하여 일반 디렉토리를 만들 수 있다. 위의 /etc/devfsd.conf파일은 여전히 사용가능 할 것이다. 당신은 devfsd를 인스톨 하기에 앞서 /dev-state디렉토리를 만들어야 한다. 만약 /dev에 예전의 퍼미션을 가지고 있다면, 새로운 디렉토리에 디바이스 노드들을 카피(또는 move)하면 된다.

5.2.3. 더 나은 방법

가장 좋은 방법은 /dev에 마운트되어 있는 퍼미션의 데이터베이스를 가져오는 것이다. 그 이유는 디바이스 노드들을 /dev-state에 복사할 필요가 없고, /dev-state(devfs를 위한) 과 /dev/(devfs를 사용하지 않을 경우에) 사이에 퍼미션을 복사할 필요가 없이 devfs와 devfs를 사용하지 않는 커널을 바꿀 수 있기 때문이다.

5.3. devfs의 지원없이 드라이버를 취급하기

현재, 커널에서 모든 드라이버들이 devfs를 사용하도록 수정되지는 않았다. 여전히 devfs가 지원되지 않는 디바이스 드라이버들은 자동으로 devfs에 나타나지 않을 것이다. 이러한 드라

이버들에 대한 디바이스를 생성하는 가장 간단한 방법은 필요한 디바이스 노드들을 포함하고 있는 tarfile을 푸는 것이다. 이것을 부트 스크립트에 넣어도 된다. 이제 당신의 모든 드라이버들은 전처럼 작동할 것이다.

고무적이게도 대부분의 사람들에게 devfs는 대부분의 기능을 잃지 않고 (예를 들어 다양한 디바이스에 대한 접근 상실) /dev 위에 devfs디렉토리를 마운트 할 수 있는 충분한 지원을 한다. 1998 년 1월 22일 이후 (devfs 패치 버전 10)부터 나는(저자는) 이 방법(devfs)를 운영하고 있다. 내가 가지고 있는 모든 디바이스들은 devfs에 나타나므로 나는 아무것도 잃은 것이 없다.

> **경고**
>
> 만약 당신의 설정이 예전방식의 디바이스 이름(예를 들어 `/dev/hda1` , `/dev/sda1`)을 필요로 한다면, 호환엔트리를 유지하도록 devfsd를 설치하고 설정해야 한다. 당신이 이것을 필요로 할 것이라는 것은 거의 확실하다. 커널이 루트 디바이스에 대해 호환항목을 생성하기 때문에 initrd를 필요로 하지 않는다는 것을 기억하라.

만약 Unix98 PTYs를 사용하기 있다면 devfs 가 `/dev/pts`를 관리할 수 있기 때문에 더 이상 devpts 를 마운트할 필요가 없다는 것을 기억하라. 이것은 devpts를 컴파일 하고 설치할 필요가 없기 때문에, 약간의 램을 절약한다. glibc의 어떤 버전은 devfs 시스템에서 Unix98 pty를 취급하는데 버그를 가지고 있다는 것을 명심하라. 수정을 위해서는 glibc 메인테이너와 연락하라. Glibc 2.1.3 은 수정되었다.

만약 devfsd를 설치하지 않았다면 `/etc/fstab`를 수정하는 것에서부터 다른 것들도 수정해야 한다는 것을 명심하라. (X 서버와 같은) 어떤 소프트웨어는 그 소스에 디바이스 이름을 고정시켜 놓는다. 실제로 호환 엔트리를 생성하는데 devfsd를 사용하는 것이 훨씬 쉽다. 당신은 당신의 시스템을 devfsd가 호환항목을 생성하도록 제한하면서 서서히 새로운 디바이스 이름(예를 들어, /etc/fstab부터 시작하여)을 사용하도록 할수 있다.

DEVFS가 가능한 커널을 부팅하기 전에 **DEVFSD**를 설치하였는지 확인하라**!!**

devfs가 2.3.46 커널에 들어갔을 당시, 나는 수많은 보고를 받았다. 대부분의 보고들은 사람들이 devfsd 없이 운영했기 때문이었고, 그 중에 어떤 것들이 깨졌다(사용할 수 없는 시스템이 되었다). 나는 순간적인 설정잘못으로 인한 버그보다 실제 버그에 전념하기를 원한다. 만약 다른 코드(예를 들어 glibc, X server)에서 버그/오류를 수정해야 한다고 느낀다면, 그 메인테이너들에게 확인하라. 그들은 (나보다) 훌륭하게 해결할 것이다.

5.4. Devfs 의 방법

devfs의 커널패치는 합리적인 디바이스 트리를 생성한다. 위에 기술된대로, 만약 예전 방식의 `/dev` 네이밍 스키마를 유지하기 원한다면, 적당하게(맨 페이지에서 보는대로) devfsd를 설정하면 된다. 예전 방식을 선호하는 사람들은 이번 섹션을 건너뛰어라. 체계화된 이름과 명확한 `/dev`를 좋아하는 사람들은 읽어보길 바란다.

만약 devfsd를 실행하지 않았다면, 호환 엔트리의 관리를 할수 없고, 새로운 이름을 사용하기 위해서는 당신의 시스템을 설정해야 할 것이다. 예를 들어, 새로운 디스크의 네이밍 스키마를 사용하기 위해서 `/etc/fstab`를 수정해야 할 것이다. 만약 devfs를 사용하지 않는 커널을 부

팅하시기 원한다면, devfs없이 커널을 부팅했을때를 위하여 예전 방식의 이름으로 돌아갈 disc-based /dev에 기초한 호환 심링크가 필요할 것이다.

당신은 호환엔트리를 유지할 디바이스를 선택할 수 있다. 예를 들어, BSD pseudo-terminal 디바이스에 대한 호환 엔트리를 원할 수도 있다(그렇지 않으면 C 라이브러리를 패치하거나, 대신에 Unix98 ptys를 사용해야 할 것이다). 이것은 /dev/devfsd.conf 에 정확한 정규표현식을 넣으면 되는 일이다.

당신이 선호하는 다른 네이밍 스키마를 선택할 수 있다. 예를 들어, 나는 너무 수직적이기 때문에 커널 제공 이름를 사용하지 않는다. 일반적인 잘못된 생각은 커널이 제공하는 이름들은 설정파일에 직접적으로 사용되어야 한다는 것이다. 이것은 그런게 아니다. 그것들은 장착된 디바이스들의 레이아웃을 반영하고 쉽게 분류하기 위하여 디자인된 것이다.

만약 커널이 제공하는 이름을 좋아한다면, 그것으로 좋다. 만약 그렇지 않다면, 당신이 좀더 선호하는 네임스페이스를 생성하기 위하여 **devfsd**를 사용해야 할 것이다. **devfsd**는 논리적이고 관리하기 쉽도록 하는 네임스페이스를 생성하기 위한 코드를 포함하고 있다. 근본적으로는, 커널이 제공하는 네임스페이스의 생략형을 생성한다.

당신은 자신만의 네임스페이스를 만들기 위한 과정에 있다. devfsd는 당신이 이 일을 쉽게 하도록 하는데 필요한 모든 기반을 가지고 있다. 필요한 일은 스크립트를 만드는 것이다. 심지어 c 코드를 작성할 수도 있으며 devfsd는 호출가능한 확장으로 그 공유 오브젝트를 적재할 수 있다.

5.5. 다른 논점

5.5.1. init 프로그램

주의를 기울여야 할 다른 점은 **init**프로그램이 /dev/telinit 유닉스 소켓을 생성하는지에 대한 것이다. init의 어떤 버전은 /dev/telinit를 생성하고 **telinit**프로그램은 init 프로세스와 통신한다. 만약 이런 시스템을 가지고 있다면, init 가 시작되기 전에 devfs가 /dev에 마운트되도록 해야 한다. 다른 말로 하면, **init**후에 수행되는 /etc/rc에는 devfs를 마운트 하도록 넣어서는 안된다. **init**의 다른 버전들은 **init**가 시작하기 전에 존재해야만 하는 named pipe /dev/initctl 가 필요하다. 다시말하지만 **init**가 시작되기 전에 그 named pipe를 생성하고 devfs를 마운트해야 한다.

기본적인 행동은 2.3.x 이후의 커널이 부팅할때에는 /dev 에 devfs를 마운트 하지 않는 것이다. 이것은 부팅 옵션에 **"devfs=mount"**를 넣음으로써 고칠 수 있다. 이 문제는 **init**의 문제를 해결하고, 다음의 끔찍한 상황(Cannot open initial console 메세지)을 막아준다. devfs 패치가 적용되어야 하는 2.2.x의 커널에서는 기본적으로 마운트 한다.

만약 /dev에 devfs를 자동으로 마운트한다면, 부트 스크립트에서 /dev/initctl를 생성하도록 해야 한다. 다음의 줄을 추가하라:

```
mknod /dev/initctl p
kill -SIGUSR1 1  # /dev/initctl이 현재 존재하는지를 init에 알려줌
```

다른 방법으로는, 커널이 /dev에 devfs를 마운트 하기를 원치 않는다면, /dev/initctl문제를 해결하기 위하여 다음의 프로시져를 사용할 수 수 있다:

```
# cd /sbin
# mv init init.real
# cat > init
#! /bin/sh
mount -n -t devfs none /dev
mknod /dev/initctl p
exec /sbin/init.real $*
[control-D]
# chmod a+x init
```

init의 최신 버전은 /dev/initctl를 자동으로 생성하므로, 이것에 대해서 걱정할 필요가 없다는 것을 기억하라.

5.5.2. 모듈 자동로딩

모듈의 자동로딩을 하기 위해서는 **devfsd**를 설정할 필요가 있다. 다음의 줄을 /etc/devfsd.conf파일에 추가시켜라:

```
LOOKUP .* MODLOAD
```

devfsd-v1.3.10 이후로, MODLOAD 에 의해 사용되는 포괄적인 /etc/modules.devfs 설정 파일이 설치된다. 이것은 대부분의 설정들에 대해 충분하다. 만약 설정이 더 필요하다면, /etc/modules.conf파일을 수정하라. devfs가 모듈의 자동로딩을 하는 방법은 :

- 프로세스가 디바이스 노드의 탐색시도를 한다(e.g. /dev/fred)
- 만약 디바이스 노드가 존재하지 않으면, 전체경로가 문자열로 devfsd에 넘겨진다.
- devfsd는 modprobe 프로그램(위에서 보여진 설정 라인이 존재하는) 에 그 문자열을 전해주고, 그 설정파일인 /etc/modules.devfs에 명시한다.
- /etc/modules.devfs는 로컬 설정에 접근하기 위해 /etc/modules.conf을 포함한다.
- modprobe는 그 경로를 모듈이름으로 변환하는 앨리어스를 탐색함으로써 그것의 설정파일을 찾을 것이다.
- 변환된 경로는 그 모듈을 로딩하는데 사용된다.

만약 mymod모듈을 로딩하기 위하여 /dev/fred를 탐색하기 원한다면, /etc/modules.conf에 다음의 줄을 추가하라:

```
alias /dev/fred mymod
```

그 /etc/modules.devfs 서정 파일은 표준 디바이스 이름에 대해서 각각의 앨리어스를 제공한다. 만약 이 파일을 자세하게 찾는다면, 어떤 모듈은 여러개의 앨리어스 설정 줄이 필요하다는 것을 알것이다. 그것은 엣 방시과 새로운 방식의 디바이스 이름에 대해 모듈 자동로딩을 지원하기 위해 필요하다.

5.5.3. devfs 디바이스에 root를 마운트하기

만약 "devfs=only" 부트 옵션을 설정했을때 devfs 디바이스가 root를 마운트 하기를 원한다면, 부팅시 커널옵션에 "root=<device>" 옵션을 넣어야 한다. LILO를 사용한다면, lilo.conf에 이것을 가지고 있어야만 한다:

```
append = "root=>device>"
```

놀라운가? 옙, 나 역시 그렇다. 다음의 줄을 당신이 가지고 있다면 (대부분의 사람들이 그런 것처럼) 해결한다:

```
root = <device>
```

LILO는 <device>의 디바이스 번호를 결정할 것이고, 그 커널을 시작하게 전에 커널 이미지의 특정 부분에 그 번호를 기록할 것이며, 그 커널은 루트 파일시스템을 마운트 하기 위하여 그 디바이스 번호를 사용할 것이다. 그래서, "append" 변수를 사용하는 것은 LILO가 그 루트 파일시스템을 devfs가 사용할 수 있는 문자열로 넘겨 주도록 확실히 하는 것이다.

만약 "devfs=only"를 사용하지 않는다면 신경쓸 것 없다.

5.5.4. TTY 문제

C 라이브러리의 어떤 버전에서는 **ttyname(3)**가 심볼릭 링크인 디바이스 엔트리에 대해서는 오류를 만들어낸다. **tty(1)** 프로그램은 이 함수에 의존한다. 나는 이것을 수정하기 위하여 libc 5.4.43에 패치를 만들었다. 이것은 libc 5.4.44에 포함되어 있고 비슷핸 패치가 glibc 2.1.4에 있다.

5.6. 커널 네이밍 스키마

커널은 기본 네이밍 스키마를 제공한다. 이 스키마는 특정한 디바이스나 디바이스 형태에 따라 검색하고, 사용가능한 디바이스를 찾아보는데 쉽도록 디자인 된다. 어떤 디바이스 형태(하드 디스크 같은)는 그런 부류의 디바이스가 사용가능한가를 알아보기 쉽게 하기 위해 그 항목의 디렉토리를 가지고 있다. 종종, 그 항목들은 사용가능한 디바이스의 구조를 반영하기 위한 디렉토리 트리에 심볼릭 링크로 되어 있다. 기하학적인 트리는 당신의 디바이스가 어떻게 배열되어 있는지 보는데 유용하다.

아래는 대부분의 일반적인 드라이버들에 대한 네이밍 스키마의 목록이다. 예약된 디바이스 이름들 (http://your.destiny.pe.kr/devfs/device-names.txt)의 목록은 참고가 가능하다. 그 목록에 추가되기 원한다면 rgooch (at) atnf.csiro.au (mailto:rgooch@atnf.csiro.au) 에 메일을 보내라. 다른 이름들도 그 메인테이너의 판단에 따라 요청된 이름대신 할당될 수 있다.

5.6.1. 디스크 디바이스들

SCSI, IDE 또는 무엇이라도, 모든 디스크는 /dev/discs 아래에 위치한다.

```
/dev/discs/disc0 first disc
/dev/discs/disc1 second disc
```

이들 각 항목들마다 그 디바이스에 대하여 디렉토리에 심볼릭 링크가 걸려있다. 디바이스 디렉토리는 다음을 포함한다.

```
disc 전체 디스크에 대하여
part* 각각의 파티션에 대하여
```

5.6.2. CD-ROM Devices

SCSI, IDE 또는 어떤 CD-ROM이던지간에 모든 CD-ROM들은 /dev/cdroms 계열아래 존재한다.:

```
/dev/cdroms/cdrom0 first CD-ROM
/dev/cdroms/cdrom1 second CD-ROM
```

이들 항목 각각은 그 디바이스에 대한 실제 디바이스 항목에 대한 심볼릭 링크이다.

5.6.3. Tape Devices

SCSI, IDE 를 막론하고 모든 테잎 장치들은 /dev/tapes계열아래에 위치한다.

```
/dev/tapes/tape0 first tape
/dev/tapes/tape1 second tape
```

이들 항목 각각은 그 디바이스에 대한 디렉토리의 심볼릭 링크이다.

```
mt for mode 0
mtl for mode 1
mtm for mode 2
mta for mode 3
mtn for mode 0, no rewind
mtln for mode 1, no rewind
mtmn for mode 2, no rewind
mtan for mode 3, no rewind
```

5.6.4. SCSI Devices

어떤 SCSI 디바이스를 유일하게 판별하기 위해서는 다음의 정보가 필요하다:

```
controller (host adapter)
```

```
bus (SCSI channel)
target (SCSI ID)
unit (Logical Unit Number)
```

모든 SCSI 디바이스는 /dev/scsi (devfs가 /dev에 마운트 되었다는 전제하에) 아래 위치한다. 그러므로, 다음 파라미터:c=1, b=2, t=3, u=4 를 가지는 SCSI 디바이스는 다음과 같이 나타날 것이다.

```
/dev/scsi/host1/bus2/target3/lun4 device directory
```

이 디렉토리 안에서 SCSI 디바이스 타잎 드라이버가 인스톨된 것에 의존하여 디바이스 항목 번호가 만들어질 것이다.

SCSI 디스크 드라이버가 만드는 항목을 알기위해서는 디스크 네이밍 체계에 대한 섹션을 보라.

SCSI 테잎 드라이버 항목이 만들어지는 것에 대해 알고 싶으면 테잎 네이밍 체계에 대한 섹션을 보라.

SCSI CD-ROm 드라이버는 다음을 만든다:

```
cd
```

SCSI generic 드라이버는 다음을 만든다:

```
generic
```

5.6.5. IDE Devices

IDE 디바이스를 유일하게 지정하기 위해서는 다음의 정보가 필요하다:

```
controller
bus (aka. primary/secondary)
target (aka. master/slave)
unit
```

모든 IDE 디바이스들은 /dev/ide아레 위치하고, SCSI 서브시스템과 비슷한 네이밍 체계를 사용한다.

5.6.6. XT Hard Discs

모든 XT 디스크들은 /dev/xd아래 위치한다. 첫번째 XT 디스크는 /dev/xd/disc0디렉토리를 가진다.

5.6.7. TTY devices

tty 디바이스들은 다음과 같이 나타난다.

New name	Old-name	Device Type
/dev/tts/{0,1,...}	/dev/ttyS{0,1,...}	Serial ports
/dev/cua/{0,1,...}	/dev/cua{0,1,...}	Call out devices
/dev/vc/0	/dev/tty	Current vir-
tual console		
/dev/vc/{1,2,...}	/dev/tty{1...63}	Virtual consoles
/dev/vcc/{0,1,...}	/dev/vcs{1...63}	Virtual consoles
/dev/pty/m{0,1,...}	/dev/ptyp??	PTY masters
/dev/pty/s{0,1,...}	/dev/ttyp??	PTY slaves

5.6.8. RAMDISCS

RAMDISC들은 그들 자신의 디렉토리에 위치하며, 다음과 같은 이름을 가진다:

```
/dev/rd/{0,1,2,...}
```

5.6.9. Meta Devices

meta 디바이스들은 그들 자신의 디렉토리에 위치하며, 다음과 같다:

```
/dev/md/{0,1,2,...}
```

5.6.10. Floppy discs

플로피 디스크들은 `/dev/floppy`디렉토리에 위치한다.

5.6.11. Loop devices

Loop 디바이스들은 `/dev/loop`디렉토리에 위치한다.

5.6.12. Sound devices

Sound 디바이스들은 `/dev/sound`디렉토리에 위치한다. (`audio, sequencer,...`).

5.7. Devfsd 네이밍 스키마

Devfsd는 커널제공 네임스페이스 의 편리한 축약형인 네이밍 스키마를 제공한다. 어떤 경우에는, 커널이 제공하는 네이밍 스키마는 매우 편리하고, 따라서 devfsd는 다른 네이밍 스키마를 제공하지 않는다. devfsd가 만드는 편리한 이름은 실제로 오리지날 devfs 커널 패치가 만드는 이름과 동일하다(리누즈가 Big Name Change를 지시하기 전까지). 이것들은 "새로운 호환 엔트리"로서 참고된다.

이들 편리한 이름을 생성하기 위해 devfsd를 설정하기 위하여, 다음의 줄이 `/etc/devfsd.conf`에 추가되어야 한다:

```
REGISTER        .*          MKNEWCOMPAT
UNREGISTER      .*          RMNEWCOMPAT
```

이것은 커널이 제공하는 이름을 지정하기 위하여 devfsd가 심볼릭 링크를 만들(없앨)도록 한다.

5.7.1. SCSI 하드 디스크들

모든 SCSI 디스크들은 `/dev/sd`아래에 위치한다 (devfs는 `/dev`에 마운트되었다고 가정하면). 그러므로, 다음의 파라미터:c=1,b=2,t=3,u=4를 가지는 SCSI 디스크들은

```
/dev/sd/c1b2t3u4 for the whole disc
/dev/sd/c1b2t3u4p5 for the 5th partition
/dev/sd/c1b2t3u4p5s6 for the 6th slice in the 5th partition
```

에 나타난다.

5.7.2. SCSI Tapes

모든 SCSI 테입들은 `/dev/st`아래 위치한다. SCSI 디스크들과 유사한 네이밍 체계가 사용된다. 다음과 같은 파라미터 : c=1, b=2, t=3, u=4 를 가지는 SCSI 테잎들은 다음과 같이 나타난다:

```
/dev/st/c1b2t3u4m0 for mode 0
/dev/st/c1b2t3u4m1 for mode 1
/dev/st/c1b2t3u4m2 for mode 2
/dev/st/c1b2t3u4m3 for mode 3
/dev/st/c1b2t3u4m0n for mode 0, no rewind
/dev/st/c1b2t3u4m1n for mode 1, no rewind
/dev/st/c1b2t3u4m2n for mode 2, no rewind
/dev/st/c1b2t3u4m3n for mode 3, no rewind
```

5.7.3. SCSI CD-ROMs

모든 SCSI CD-ROM들은 `/dev/sr`아래 위치한다. SCSI 디스크들과 비슷한 네이밍 체계를 사용한다. 다음과 같은 파라미터: c=1, b=2, t=3, u=4 를 가지는 SCSI CD-ROM은 다음과 같이 나타난다:

```
/dev/sr/c1b2t3u4
```

5.7.4. SCSI Generic Devices

모든 SCSI CD-ROM들은 `/dev/sg`아래 위치한다. SCSI 디스크와 비슷한 네이밍 체계를 사용한다. 다음과 같은 파라미터: c=1, b=2, t=3, u=4 를 가지는 SCSI generic 디바이스는 다음과 같이 나타난다:

```
/dev/sg/c1b2t3u4
```

5.7.5. IDE Hard Discs

모든 IDE 디스크들은 SCSI 디스크들과 비슷한 방법을 사용하여, `/dev/ide/hd`아래 위치한다. 다음의 맵핑이 새로운 이름과 예전 이름 사이에 존재한다.

```
/dev/hda /dev/ide/hd/c0b0t0u0
/dev/hdb /dev/ide/hd/c0b0t1u0
/dev/hdc /dev/ide/hd/c0b1t0u0
/dev/hdd /dev/ide/hd/c0b1t1u0
```

5.7.6. IDE Tapes

IDE 디스크들과 유사한 네이밍 체계가 사용된다. 그 항목은 `/dev/ide/mt`디렉토리 안에서 나타날 것이다.

5.7.7. IDE CD-ROM

IDE 디스크와 비슷한 네이밍 체계가 사용된다. 각 항목은 `/dev/ide/cd`디렉토리 안에서 나타날 것이다.

5.7.8. IDE Floppies

IDE 디스크와 비슷한 네이밍 체계를 사용한다. 각 항목은 `/dev/ide/fd`디렉토리 안에서 나타날 것이다.

5.7.9. XT Hard Discs

모든 XT 디스크들은 /dev/xd아래에 위치한다. 첫번째 XT 디스크는 /dev/xd/c0t0처럼 나타날 것이다.

5.8. 옛 방식의 호환 이름

에전의 호환이름들은 /dev/hda, /dev/sda, /dev/rtc 등등의 예전부터 유지되어온 디바이스 이름을 사용한다. Devfsd는 오래된 이름을 계속 사용할 수 있도록 호환 심링크를 생성하도록 설정파일에 설정될 수 있고, 따라서 오래된 응용프로그램들은 계속하여 잘 작동하게 될 것이다.

devfsd가 이런 예전의 이름을 생성하도록 설정하기 위해서 다음의 줄이 /etc/devfsd.conf에 추가되어야 한다

```
REGISTER .* MKOLDCOMPAT
UNREGISTER .* RMOLDCOMPAT
```

이것은 devfsd가 커널이 제공하는 이름을 지정하도록 하는 심볼릭 링크를 만들(없애)도록 한다.

5.9. SCSI 호스트 탐색과 관련된 문제

Devfs는 SCSI 호스트 번호에 대해 기초한 SCSI 디스크를 식별하도록 허용한다. 만약 당신의 컴퓨터에 오직 하나의 SCSI 호스트(card)가 있다면, 호스트 번호는 0을 받게 될 것이다. 당신이 여러개의 SCSI 호스트를 가지고 있다면 그다지 쉽지 않다. 불행하게도, 때때로 SCSI 호스트의 번호를 얻기 원하는 것을 추측하는데 어려움이 있을 수 있다. 이것을 쉽게 하기 위해서, "scsihosts"라 불리는 커널 부트 파라미터가 있다. 이것은 SCSI 호스트들의 다른 타입에 대해 순서를 결정하도록 해준다. 이 파라미터의 문법은:

```
scsihosts=<name_1>:<name_2>:<name_3>:...:<name_n>
```

이다. 여기서 <name_1>,<name_2>,...,<name_n> 들은 /proc 파일시스템에서 사용된 드라이버들의 이름들이다. 예를 들어:

```
scsihosts=aha1542:ppa:aha1542::ncr53c7xx
```

는

```
- 첫번째 aha1542 컨트롤러   - /dev/scsi/host0/bus#/target#/lun# 일 것이다.
- 첫번째 병렬포트 ZIP       - /dev/scsi/host1/bus#/target#/lun# 일 것이다.
- 두번째 aha1542 컨트롤러   - /dev/scsi/host2/bus#/target#/lun# 일 것이다.
- 첫번째 NCR53C7xx 컨트롤러 - /dev/scsi/host4/bus#/target#/lun# 일 것이다.
- 나머지 컨트롤러           - /dev/scsi/host5/bus#/target#/lun#,
```

/dev/scsi/host6/bus#/target#/lun# 등등 일 것이다.
- 만약 위의 어떤 컨트롤러도 발견되지 않았다면 - 그 예약된 이름들이 다른 디바이스에 의해 사용되지 않는다는 것이다.
- /dev/scsi/host3/bus#/target#/lun# 는 절대로 사용되지 않는다.

에 연결된 디바이스를 의미한다. 원한다면 분리자로 ':' 대신에 ','를 사용할 수 있다. 나는 여기서 devfsd 네이밍 스키마 를 사용한다.

만약 같은 타입으로 여러개의 카드(NCR53c8xx와 같은)를 가지고 있다면, 이 스키마가 SCSI 호스트를 지정하지는 못한다. 이런 경우에 당신은 이것을 제어하기 위해서 드라이버를 지정하는 부트 파라미터를 사용할 필요가 있다.

6. 현재 포팅된 디바이스 드라이버들

- 모든 miscellaneous 캐릭터 디바이스들은 devfs를 지원한다 (이것은 misc_register()를 통하여 투명하게 이루어 진다)
- SCSI 디스크와 일반 하드 디스크들
- 캐릭터 메모리 디바이스들(null, zero, full, 기타등등) -- C. Scott Ananian <cananian (at) alumni.princeton.edu>에게 감사하다.
- 루프 디바이스들(/dev/loop?)
- TTY 디바이스들 (콘솔, 시리얼 포트, 터미널과 pseudo 터미널들) -- C. Scott Ananian <cananian (at) alumni.princeton.edu> 에 감사한다.
- SCSI 테입장치(/dev/scsi 와 /dev/tapes)
- SCSI CD-ROM(/dev/scsi와 /dev/cdroms)
- SCSI 일반 디바이스들(/dev/scsi)
- RAMDISCS (/dev/ram?)
- 메타 디바이스들 (/dev/md*)
- 병렬포트 프린터 (/dev/printers)
- 사운드 디바이스 (/dev/sound) -- C. Scott Ananian <cananian (at) alumni.princeton.edu> 와 Eric Dumas <dumas (at) linux.eu.org> 에게 감사한다.
- 조이스틱(/dev/joysticks)
- Sparc 키보드 (/dev/kbd)
- DSP56001 디비털 시그널 프로세서 (/dev/dsp56k)
- 애플 데스크탑 버스 (/dev/adb)
- Coda 네트웍 파일시스템 (/dev/cfs*)

- 버츄얼 콘솔 캡쳐 디바이스 (/dev/vcc) -- Dennis Hou <smilax (at) mindmeld.yi.org> 에 감사한다.
- 프레임 버퍼 디바이스 (/dev/fb)
- 비디오 캡쳐 디바이스 (/dev/v41)

7. 디바이스 번호의 할당

Devfs는 커널 내부의 오퍼레이션에 대해 디바이스 번호(메이저&마이너 번호) 를 할당할 필요가 없는 드라이버들을 작성할 수 있도록 허용한다. 그러나, 디바이스에 대해 유일하게 조정가능한 것으로써 그 디바이스를 사용하는 사용자 공간 프로그램들의 번호가 있다. 그 중에 한 예가 inode가 다른 inode와 다른 파일시스템에 있는지 구분하기 위하여 다비이스 번호를 사용하는 **find** 프로그램이다. 그 사용된 디바이스 번호는 파일시스템이 사용하고 있는 블럭 디바이스를 위한 것이다. 사용자 공간 프로그램들과 호환성을유지하기 위하여 devfs를 사용하는 블럭 디바이스들은 유일한 디바이스 번호를 할당받을 필요가 있다. 또한, POSIX는 디바이스 번호들을 명시하고 디바이스 번호중 일부분은 사용자 공간에 보여줄 필요가 있다.

가장 간단한 옵션(특히 devfs에 포팅된 드라이버일때)은 예전의 메이저/마이너 번호들을 계속 사용하는 것이다. Devfs는 메이저&마이너 번호에 주어진 값이 무엇이던간에 사용자 공간에 보낼 것이다.

다른 방법으로는, devfs가 유일한 디바이스 번호를 선택하도록 할 수 있다. **devfs_register**를 사용하여 캐릭터나 블럭 디바이스를 등록하였을때, 자동적으로 고유한 디바이스 번호를 하당하는(그 할당은 캐릭터와 블럭디바이스를 구분한다) DEVFS_FL_AUTO_DEVNUM 플래그를 사용할 수 있다.

이 디바이스 번호는 16 bit 번호이며, 따라서 많은 수의 디스크와 파티션에 대해 충분한 공간을 마련해준다. 이 스키마는 또한 현재 256 pseudo-ttys 로 제한된(이 제한은 xterms 과 원격 로긴의 총 동시 접속자 수이다) tty 디바이스들과 같은 캐릭터 디바이스에도 적용된다. 디바이스 번호는 현존하는 공식적인 할당과의 가능한 충돌을 피하기 위하여 36864-61439(메이저 144-239) 의 범위로 제한되어 있다는 것을 명심하라.

동적으로 할당된 블럭 디바이스 번호들은 원격 마운트의 활성 시간동안 계속해서 주어지는 디바이스에 대해 dev_t 를 기다리는 NFS 데몬과는 작동하지 않는다 (사용자와 커널 모드 다) 는 것을 명심하라.

이 스키마에서 마지막으로 명심할 것은 : 이것이 디바이스 번호의 크기를 증가시키지 않기 때문에 사용자 공간에서의 호환과는 문제가 없다 는 것이다.

8. 질문과 답변

8.1. Making things work

여기에는 통상적인 질문과 답변이 있다.

- **Devfsd**가 모든 퍼미션을 관리하지 않는다

  당신이 적당한 이벤트를 캡쳐하고 있는지 확인하라. 예를 들어,커널이 만든 디바이스 엔트리는 **REGISTER** 이벤트를 발생시키지만, **devfsd**가 생성한 것들은 **CREATE**이벤트를 발생한다.

- **Devfsd** 는 모든 **REGISTER**이벤트를 캡쳐하지 않는다**.**

  앞의 항목을 보라 : 당신은 **CREATE**이벤트를 캡쳐할 필요가 있다.

- **X**가 시작하지 않는다**.**

  위에 약술된 모든 단계를 따라했는지 확인해보라.

- 왜 네트워크 디바이스가 **devfs**에 나타나지 않는가**?**

  이것은 버그가 아니다. 네트웍 디바이스는 네임스페이스와는 완전히 분리되어 있다. 그것들은 **socket(2)**와 **setsockopt(2)**에 의해 접근되고, 따라서 디바이스 노드가 필요없다. 나는 네트웍 디바이스를 디바이스 네임스페이스에 옮길 것을 제안하였지만, 아무런 응답을 받지 못했다.

- 커널에 **devfs**가 컴파일되어 있는지 확인하는 방법은 무엇인가**?**

  커널에 내장되어 있거나 현재 로딩되어 있는 모든 파일시스템들은 `/proc/filesystems`에 나타난다. 만약 `devfs` 엔트리가 보인다면, devfs가 컴파일되어 있는 커널이라는 것을 알수 있다. 만약 정확하게 설정하고 커널을 재컴파일 한다면, devfs는 내장되어 있을 것이다. 생각하기에 제대로 설정했음에도 `/proc/filesystems`에 나타나지 않는다면, 당신이 실수를 한 것이다. 일반적인 실수는:

  - devfs 패치를 적용하지 않은 2.2.x 커널을 사용중이다(만약 커널을 패치하는 방법을 모른다면, 대신에 2.4.x를 사용하고, 나에게 패치하는 법을 질문해서 성가시게 하지 말라)
  - **CONFIG_EXPERIMENTAL=y** 로 설정하는 것을 잊었다.
  - **CONFIG_DEVFS_FS=y**로 설정하는 것을 잊었다.
  - **CONFIG_DEVFS_MOUNT=y**로 설정하는 것을 잊었다. (만약 부팅시 devfs가 자동으로 마운트 되기를 원한다면)
  - make config 나 make xconfig 대신에 **.config** 를 직접 수정하라.
  - 설정을 변경하고 컴파일하기 전에 **make dep; make clean**하는 것을 잊었다.
  - 커널과 모듈을 컴파일 하는 것을 잊었다.
  - 커널을 설치하는 것을 잊었다.
  - 모듈을 설치하는 것을 잊었다

  제발 버그 리포트를 보내기 전에 위의 모든 단계를 수행하였는지 두번씩 체크하도록 하라.

- **devfs**가 **`/dev`**에 마운트되었는지 어떻게 확인하죠**?**

디바이스 파일시스템은 데몬과 통신하기 위해 사용되는 "`.devfsd`" 라 불리는 항목을 항상 생성한다. 데몬이 실행중이 아니라 할지라도, 그 항목은 존재한다. 이 항목의 존재에 대해 확인하는 것은 devfs가 마운트 되었는지 아닌지 결정하는 좋은 방법이다. 항목의 형태(i.e. 일반적인 파일, 캐릭터 디바이스, named pipe, 등등)은 사전주의 없이 바뀔수도 있다. 오직 그 항목(.devfsd:역자 주)의 존재만이 믿을만한 것이다

8.2. devfs의 대안

나는 모든 devfs를 반대한 제안들과 대조해 보았고, 그것들의 한계를 설명했다.

참고**:** 준비중입니다.

8.2.1. 왜 데몬에서 **create/remove** 이벤트를 보내지 않는가**?**

여기에서 그러한 제안은 디바이스들이 create/remove 이벤트들을 등록할 수 있고, 데몬은 그 이벤트들을 들을 수 있는 커널의 API를 개발하자는 것이다. 그 데몬은 (디스크상의 한부분인)`/dev`를 생성/해제 할 것이다.

이것은 여러가지 제한을 가지고 있다:

- 그것은 오직 커널이 부팅을 끝낸후에 모듈을 올리거나 제거하는 (또는 디바이스를 추가하거나 제거하는) 것에 대한 작업이다. 이벤트의 데이터베이스 없이는 데몬이 완전한 `/dev`를 만들 수 있는 방법은 없다.
- 만약 이 스키마에 대한 데이터베이스를 추가한다면, 문제점은 사용자 공간에서 그 데이터베이스를 보여주는 방법이다. 만약 데몬에서 파이프를 통해 통과되는 이벤트 코드들의 문자열 리스트를 만든다면, 오직 데몬만이 사용할 수 있다. 나는 이 데이터를 보여주기 위한 장연적은 방법으로 devfs와 같은 파일시스템 비슷한 것을 설명할 것이다. 파일시스템으로써 그 데이터를 보여주는 것은 사용자가 사용가능한 것을 알아보기 쉽게 하고 또한 그 "데이터베이스"를 탐색하기 쉽도록 하는 스크립트를 작성하기 쉽게 한다.
- 디바이스 노드와 드라이버 사이의 빡빡한 바인딩이 더 이상 가능하지 않다. (requiring the otherwise perfectly avoidable 테이블 검색)
- `/dev`에서 모듈의 자동로딩은 생성된 디바이스 노드를 필요로 하기에 inode 탐색 이벤트를 잡을 수 없다. 이것은 커다란 셋으로부터 생성된 매우 적은수의 inode들의 드라이버에 대한 문제이다.
- 이 기법은 root 파일시스템이 읽기 전용으로 마운트 되었을때는 사용할 수 없다.

8.2.2. **scsidev**에 대한 더 나은 구현

이 제안은 **scsidev** 프로그램을 포함시켜서 그것을 SCSI 디바이스 뿐만 아니라, 모든 디바이스를 탐색하는데에도 확장하자는 것이다. **scsidev** 프로그램은 `/proc/scsi` 를 탐색하는 방법으로 작동한다.

문제점:

- 커널은 현재 사용가능한 모든 디바이스들의 목록을 제공하지 않는다. 모든 드라이브들이 커널 메세지를 생성하거나 `/proc`에 엔트리를 등록하지는 않는다.
- devfs API와는 다르게 디바이스를 등록시키는 일정한 메커니즘이 없다.
- 그와 같은 API를 구현하는 것은 위의 제안과 같다.

8.2.3. 램 디스크에 /dev를 넣는 것

이 제안은 램디스크를 생성하고 디바이스 노드들을 거기에 둔 후 `/dev`에 그것을 마운트 하자는 것이다.

문제점:

- 이것은 루트 파일시스템을 마운트 하기 위해서는 여전히 디바이스 노드가 필요하므로 루트 파일시스템을 마운트 할때는 도움이 되지 못한다.
- 만약 루트 디바이스 노드에도 이 기법을 사용하기 원한다면, initrd를 사용하여야 한다. 이것은 부팅 과정을 복잡하게 하고, 관리자와 설정하는 것을 더 어렵게 한다. initrd 는 쉬운 시스템 관리를 앗아감으로써 본질적으로 불투명하다.
- 정확하게 램디스크를 위치시키기 위한 정보가 불충분하다. 그래서 우리는 그 문제를 해결하기 위해서 위의 제안으로 돌아간다.
- ramdisc-based 해결책은 각 항목마다 일반 VFS inode와 dentry 에 대해서 284 바이트와 112 바이트의 기본적인 저장공간이 필요하므로, 더 많은 커널 메모리가 필요하다. devfs가 72 바이트를 요구하는 것과 비교해보라.

8.2.4. 아무것도 하지 않는다: 아무런 문제가 없다

때때로 사람들은 현재의 스키마가 좋다라는 주장을 듣곤 한다. 이것은 그들이 무시하는 것에 대한 것이다.

- 디바이스 번호의 크기(메이저/마이너 번호 당 8비트)은 실제 제한사항이고, 어떤 방식으로던 수정되어야만 한다. 예를 들어 큰 SCSI 디바이스 번호를 가지는 시스템에서는 할당되지 않은 메이저 번호들을 계속 사용할 것이다. USB는 8비트의 마이너 번호 제한을 넘을 필요가 있다.

- 단순히 디바이스 번호를 증가시키는 것만으로는 불충분하다. 많은 노력을 필요로 하는 것과는 달리, 수천개 이상의 디바이스 노드를 가지는 /dev 를 관리하는 문제를 해결하지는 못한다.
- 거대한 /dev의 문제를 무시하는 것은 동적인 /dev를 원하는 많은 수의 합리적인 주장을 철회하거나 오래가지 못할 것이다.
- 표준 응답: "디바이스 관리 데몬을 작성하라" 이것은 우리가 위의 제안으로 돌아가게 한다.

8.3. devfs에 대해 좋아하지 않는것

여기 devfs에 대한 불평불만이 있고, 어떤 제안과 해결책은 좀더 당신의 마음에 들것이다. 나는 모든 사람을 기쁘게 할 수 없지만, 노력하겠다 :-)

8.3.1. 나는 그 네이밍 스키마가 싫다

첫번째로, 모든 사람을 기쁘게 해 줄수 있는 네이밍 스키마는 없다는 것을 기억하라. 당신이 그 스키마가 싫어도, 다른 사람은 좋아한다. 누가 옳고 그름을 말할수 있는가? 궁극적으로, 코드를 작성하는 사람은 선택을 하고, 현재 남아있는 것은 devfs 저자 (http://www.atnf.csiro.au/~rgooch/)와 커널 관리자(Linus)에 의해 만들어진 것을 조합하는 것이다.

그러나, 모든 것을 잃지는 않는다. 만약 자신만의 네이밍 스키마를 만들고 싶다면, 독립적인 스크립트를 작성하거나, devfsd 를 핵 하거나, devfsd 에 의해 호출되는 스크립트를 작성하는 등의 간단한 일을 하면된다. 여러분이 좋아하는 네이밍 스키마가 무엇이던지 간에 만들 수 있다.

또한, /dev로부터 devfs 네이밍 스키마의 모든 것을 제거하기를 원한다면, devfs를 아무곳이나(/devfs등의)마운트하고, /devfs에 /dev를 링크하여 만들 수 있다. 그것은 원한다면 devfsd를 사용함으로써 자동적으로 해결될 수도 있다.

심지어 심볼릭 링크를 사용하기보다 VFS 바인딩 키능이 링크를 만들도록 사용할 수도 있다. 이 방법은, 이 심볼릭 링크들의 "목적지"가 어디인지 알 필요가 없다.

8.3.2. 커널에서 Devfs 정책

이미 커널안에서의 정책이 있다. 디바이스 번호는 실제로 정책에 의한 것이다 (왜 내가 사용하는 디바이스 번호가 무엇인지를 커널이 지시해야만 하는가?). 이것에 대해, 커널에는 어떤 정책이 있다. 정책으로써의 디바이스 이름과 정책으로서의 디바이스 번호의 실제 차이점은 디바이스 번호는 사람에게는 의미없는 숫자이고 너무 조잡스럽기 때문에 누구도 직접적으로 사용하지 않는다는 것이다. 최소한 devfs 디바이스 이름에서는, (자신만의 네이밍 스키마를 더 할 수 있다 하더라도) 어떤 사람들은 devfs가 제공하는 이름을 직접적으로 사용할 것이다. 이것은 어떤 사람들에게는 기분 나쁜 일이다 :-)

8.3.3. Devfs는 블로트웨어(bloatware)이다

앞에서 보여진 것처럼, 이것은 전혀 진실이 아니다. 코드와 데이터 크기 모두 매우 적은 양이다.

8.4. 버그 리포트 방법

만약 devfs에 대한 버그를 알고 있다면(또는 알고 있다고 생각한다면), 다음의 단계대로 행하라:

1. 최신의 devfs 패치를 적용했는지 확인해보라. 최신의 커널버전에 최신의 devfs 패치가 적용되지 않았을 수도 있다(Linus는 매우 바쁘다)
2. 당신의 버그 리포트에 완전한 커널 로그 (**dmesg**를 사용하여)를 저장하여 포함하라. 모든 부트 메세지를 잡아내기 위한 내부버퍼의 크기를 증가시키기 위해 `-s`를 사용할 필요가 있을 것이다.
3. 커널 부트 커맨드 라인에 `devfs=dall`를 넣고 부팅해보고 (부트로더에서 이것을 사용하는 방법은 문서를 읽도록 하라), 그 결과를 파일로 저장하라. 이것은 매우 장황스러운 일이고, 메제시 버퍼를 넘어설 수도 있지만, 할수만 있다면 더 많은 것을 잡아내도록 노력해 달라.
4. 당신의 devfsd 설정파일을 복사해 보내달라.
5. 나 (http://www.atnf.csiro.au/~rgooch/index.html) 에게 먼저 버그 리포트를 보내라. 당신이 리눅스 커널 메일링 리스트에 버그를 리포트 한다고 해서 내가 알 것이라고 기대하지는 말라. 위에 나열된 모든 정보를 포함하고, 그외 관련이 있다고 생각하는 것을 추가시켜도 좋다. 제목에 **devfs**를 넣어서, 나의 메일 필터가 긴급한 상황임을 알수 있게 하라.

여기, 답장받을 기회를 향상시켜주는 질문하는 방법이 있다:
http://www.tuxedo.org/~esr/faqs/smart-questions.html
(http://www.tuxedo.org/~esr/faqs/smart-questions.html) 만약 버그보고를 하고 싶다면,
http://www.chiark.greenend.org.uk/~sgtatham/bugs.html
(http://www.chiark.greenend.org.uk/~sgtatham/bugs.html) 를 읽을 필요가 있을 것이다.

8.5. 이상한 커널 메세지

당신의 커널로그에서 devfs와 관련된 메세지를 볼 수 있을 것이다. 아래는 그 메세지들의 의미를 나열했다 (그리고 가능하다면 그 메세지와 관련해서 해야 할 것들의 목록이다)

- devfs_register(fred):could not append to parent, err: -17

  당신은 그 에러코드가 의미하는 바를 확인할 필요가 있지만, 보통 17은 EEXIST를 의미한다. 이것은 드라이버가 디렉토리에 `fred` 항목을 생성하려고 했지만, 같은 이름의 항목이 이

미 존재한다는 것을 나타낸다. 이것은 종종 퍼미션을 복구하기 위한 수단으로 `/dev`에 inode들을 풀어놓으려는 스크립트가 잘못되었을때 일어난다. 이 메세지는 그 디바이스 노드가 여전히 그 드라이버에 접근하도록 허용하기 때문에 (온 정성을 다하여:-) `devfs=only`부트 옵션을 사용하더라도) 위험하다. 이러한 불필요한 메세지를 제거하기 원한다면, devfsd-v1.3.20으로 업그레이드 하고, 퍼미션을 저장하기 위하여 추천된 `RESTORE` 지시자를 사용하라.

- devfs_mk_dir(bill): using old entry in dir: c1808724 ""

  이것은 드라이버가 부모디렉토리가 같은 이름을 가지는 `bill`이라 불리는 디렉토리를 생성하려고 한다는 것을 제외하고위의 메세지와 유사하다. 이 경우에, 드라이버가 작동을 올바르게 계속하기 위해서, 옛 항목이 그 드라이버에 주어지고 재사용된다. 2.5 커널에서는, 그 드라이버는 NULL 항목이 주어지고, 따라서, 어떤 환경에서는 요청된 디바이스 노드를 생성하지 못할 것이다. 이것의 해결책은 위와 같다

8.6. devfsd의 컴파일 문제

보통, **devfsd**컴파일은 단지 소스 디렉토리에서 **make**를 친 후, **make install** (루트로)을 치는 것으로 된다. 가끔은, 잘못된 설정으로 인해 문제가 일어날 수도 있다.

- `DEVFSD_NOTIFY_DELETE`에 관련된 에러메세지

  이것은 `/usr/include/linux`나 `/usr/src/linux`에 예전의 커널 헤더를 가지고 있기 때문에 일어난다. `KERNEL_DIR`변수를 (만약 `/usr/src/linux`에 새 커널소스를 설치하지 않았다면) **make**에 전달하거나, `/usr/include/linux`에 커널 소스에 있는 `devfs_fs.h`를 복사하면 된다.

9. 이 문서의 번역본

이 문서는 다른 언어로 번역되어 있다.

- rgooch (at) atnf.csiro.au 에 의해 주 문서(영어로 된)는 http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html (http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html) 에 있다.
- viatoris (at) nownuri.net 가 번역한 한국어 문서는 http://your.destiny.pe.kr/devfs/devfs.html (http://your.destiny.pe.kr/devfs/devfs.html)에 있다.

Most flags courtesy of ITA (http://www.theodora.com/ita.html)'s Flags of All Countries (http://www.theodora.com/flags) used with permission.

다른 참고문서

http://www.torque.net/sg/devfs_scsi.html (http://www.torque.net/sg/devfs_scsi.html) : SCSI 서브 시스템을 탐색하고 devfs 와 같이 사용할 수 있게 하는것을 설명한 유용한 문서 , Gilbert Douglas .

http://www.torque.net/scsi/scsihosts.html (http://www.torque.net/scsi/scsihosts.html) : `scsihosts=` 부트 옵션에 대해 논하는 좋은 문서 , Gilbert Douglas .

http://www.torque.net/scsi/SCSI-2.4-HOWTO/ (http://www.torque.net/scsi/SCSI-2.4-HOWTO/) : 2.4 에서 리눅스 SCSI 서브시스템에 대해 논한 문서 , Gilbert Douglas .

http://johannes.erdfelt.com/hotswap.txt (http://johannes.erdfelt.com/hotswap.txt) : 요구사항이 납득할만한 해결책인지와 어떻게 왜 devfs + devfsd 를 사용할지 등에 대해 기술한 페이지. 초안임. plain-text 형식임. johannes 는 html 버젼을 만들겠다고 약속했다. , Erdfelt Johannes .

2000 년 10 월 미국 플로리다 마이애미에서 개최된 2nd Annual Storage Management Workshop (http://www.sistina.com/gfs/Pages/Miami2000.html) 에서 발표한 나(저자) 의 논문 (http://www.atnf.csiro.au/~rgooch/linux/papers/Miami2000/index.html) , Gooch Richard .