12장. 커널 메커니즘 모듈 (Modules)

이 장에서는 리눅스 커널이 파일 시스템같은 함수들을 자신이 필요로 할 때 동적으로 로드 하는 방법을 설명한다.

리눅스는 단일(monolithic) 커널이다. 즉 커널의 모든 기능적인 요소들이 자신의 내부 자료구 조와 함수들에 모두 접근할 수 있는 하나의 거대한 프로그램이다. 운영체제 설계의 다른 방 법으로는 커널의 각 기능적인 부분들이 별도의 단위로 쪼개지고, 그 사이에 엄격한 통신 매 커니즘으로 연결되는 마이크로커널(micro-kernel) 구조가 있다. 이는 시간이 소모되는 프로세 스[1]가 아닌 환경 설정 프로세스를 통하여 새로운 컴포넌트를 커널에 추가할 수 있게 한다. 가령 사용자가 NCR 810 SCSI용 드라이버를 사용하려고 하는데 이것이 커널에 포함되어 있 지 않다고 하자. 그러면 커널의 설정을 바꾸고 다시 컴파일해야 NCR 810 SCSI를 사용할 수 있게 될 것이다. 그러나 여기에 다른 대안이 있다. 리눅스는 운영체제를 구성하는 컴포넌트 들을 필요로 할 때 동적으로 로드 또는 언로드할 수 있게 한다. 리눅스 모듈은 시스템이 부 팅된 후 언제라도 커널에 동적으로 링크될 수 있는 코드 덩어리이다. 또한 모듈이 더이상 필요하지 않을 때는 커널과의 연결을 해제하고 제거할 수 있다. 리눅스 커널의 상당수는 디 바이스 드라이버와, 네트웍 드라이버나 파일시스템 같은 유사 디바이스 드라이버(pseudo device driver)이다.

사용자는 insmod나 rmmod같은 명령으로 리눅스 커널 모듈을 명확하게 로드 또는 언로 드를 할 수 있으며, 또는 커널 자신이 자신이 필요로 할 때 커널 데몬(kerneld)에게 모듈 을 로드/언로드 할 것을 요구할 수 있다. 필요로 할 때 코드를 동적으로 로드하는 것은 커 널 크기를 최소화할 수 있고, 커널을 매우 유연하게 할 수 있어 매력적이다. 필자가 사용하 는 인텔 커널은 모듈을 광범위하게 사용하여 크기가 겨우 406 Kbyte 밖에 되지 않는다. 나는 VFAT 파일 시스템을 가끔씩 사용할 뿐이므로, 내가 VFAT 파티션을 마운트 할 때만 리눅스 커널이 VFAT 파일 시스템 모듈을 자동으로 올리도록 했다. 그리고 그 VFAT 파티션의 마운 트를 해제하면 시스템이 더이상 VFAT 파일 시스템 모듈이 필요하지 않다는 것을 알아차리 고 시스템에서 제거하도록 했다. 모듈은 또한 새로운 커널 코드를 다시 컴파일하고 커널을 재부팅하지 않고 테스트를 해보고자 할 때 유용하다. 물론 아무런 댓가도 없는 것은 아니지만, 커널 모듈과 관련하여 성능과 메모리에서 약간의 손해가 있을 뿐이다. 이것은 로드할 수 있도록 모듈이 제공해야 하는 약간의 코드가 있고, 별도의 자료구조가 메모리를 조금 차지 하기 때문이다. 또한 커널 자원에 접근할 때 한 단계를 거쳐야 하므로 모듈의 효율성이 아 주 조금 떨어지게 된다.

로드된 리눅스 모듈은 다른 보통 커널 코드처럼 커널의 한 부분이 된다. 모듈은 커널 코드 와 똑같은 권한과 책임을 진다. 다르게 말하면, 리눅스 커널 모듈은 모든 커널 코드나 디바 이스 드라이버처럼 커널을 망가뜨릴 수도 있다는 것이다.

모듈이 자신이 필요로 할 때 커널의 자원을 사용할 수 있으려면, 그것이 어디 있는지 찾을 수 있어야 한다. 가령 모듈이 커널 메모리를 할당하는 함수인 kmalloc()을 호출해야 한다 고 하자. 모듈을 컴파일할 때에는 메모리의 어느 위치에 kmalloc()이 있는지 모르므로, 모 듈이 로드될 때 커널은 모듈이 제대

로 동작할 수 있도록 kmalloc()에 대한 참조를 맞춰주 어야 한다. 커널은 커널의 모든 자원의 목록을 커널의 심볼 테이블(symbol table)로 관리하며, 이를 이용해 모듈이 로드될 때 이들 자원에 대한 참조를 해결할 수 있다. 리눅스는 한 모듈 이 다른 모듈의 서비스를 필요로 하는 경우, 모듈이 층층이 쌓아질 수 있도록 한다[2]. 예를 들어, VFAT 파일 시스템 모듈은 FAT 파일 시스템 모듈의 서비스를 필요로 한다. 이는 VFAT 파일 시스템이 FAT 파일 시스템을 다소 확장한 것이기 때문이다. 이렇게 한 모듈이 다른 모 듈이 제공하는 서비스나 자원을 필요로 하는 것은, 모듈이 커널 자체의 서비스와 자원을 필요로 하는 경우와 매우 비슷하다. 단지 여기서 필요로 하는 서비스가 다른, 이전에 로드된 모듈에 있는 것일 뿐이다. 각 모듈이 로드될 때, 커널은 새로 로드되는 모듈에서 외부로 보 여주는 자원과 심볼을 모두 커널 심볼 테이블에 추가한다. 이는 다음에 로드되는 모듈이 이 미 로드된 모듈의 서비스를 이용할 수 있도록 하기 위한 것이다.

모듈을 언로드하려 할 때 커널은 모듈이 현재 사용되고 않고 있는지 알아야 하며, 모듈에게 자신이 언로드되려고 한다는 것을 알려줄 수 있어야 한다. 이렇게 해서 모듈은 커널에서 제 거될 때, 자신이 할당받은 커널 메모리나 인터럽트 같은 시스템 자원을 해제할 수 있다. 모 듈이 언로드될 때 커널은 모듈이 커널 심볼 테이블에 추가한 심볼들을 모두 제거한다.

로드된 모듈이 잘못 만들어진 것이어서 운영체제를 망가트릴 가능성과는 별도로, 다른 위험 가능성이 있다. 만약 지금 실행하고 있는 커널보다 이전 버전이나 이후 버전 용으로 컴파일 된 모듈을 로드하려고 한다면 어떻게 될까? 모듈이 커널 루틴을 호출할 때 잘못된 인자를 넘겨준다면 문제가 생길 수 있을 것이다. 커널은 모듈을 로드할 때 엄격한 버전 검사를 하 여 이런 문제를 선택적으로 막을 수 있다[3].

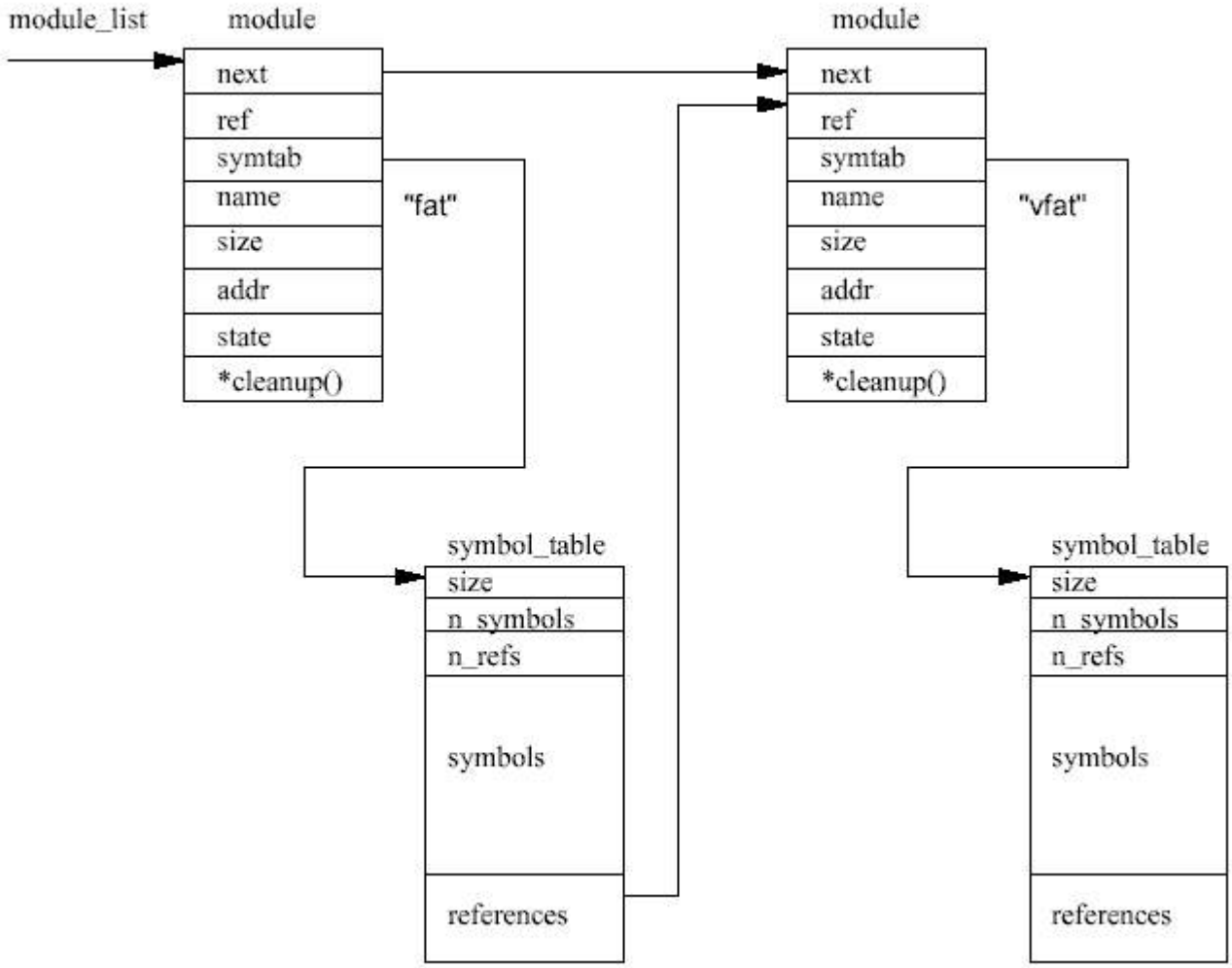

그림 12.1 : 커널 모듈의 리스트

12.1 모듈을 로드하기

커널 모듈을 로드하는 방법은 두가지가 있다. 하나는 insmod 명령을 사용하여 수동으로 모 듈을 커널에 추가하는 것이다. 두번째는 이보다 더 똑똑한 방법으로 모듈을 필요로 할 때 로드하는 것으로, 이를 요구시 로딩(demand loading)이라고 한다. 커널이 어떤 모듈을 필요로 한다는 것을 발견하면 (예를 들어 사용자가 커널에 없는 파일시스템을 마운트 한 경우), 커 널은 커널 데몬(kerneld)에게 맞는 모듈을 로드하라고 요구한다.

커널 데몬은 비록 슈퍼유저 권한을 가지고 있기는 하지만 보통의 사용자 프로세스이다. 이 프로세스는 보통 시스템이 부팅할 때 시작하여, 커널과 프로세스간 통신(IPC) 채널을 하나 연다. 이 연결은 커널이 kerneld에게 여러가지 작업을 요청하기 위해 메시지를 보내는데 사용한다. kerneld의 주된 역할을 커널 모듈을 로드하고 언로드하는 것이지만, 필요할 때 직렬라인 상에 PPP 연결을 시작하거나, 필요하지 않을 때 이를 닫는 것 같은 다른 작업을 할 수 있는 능력도 있다. kerneld는 직접 이런 일들을 하는 것이 아니라, 이런 일을 하기 위해 필요한 프로그램(insmod 같은 것)을 실행한다. kerneld는 단지 커널의 대리인이며, 커널의 다른 한편에서 일을 스케쥴링한다.

insmod 프로그램은 자신이 로드해야 하는 요청한 커널 모듈을 찾을 수 있어야 한다. 요구 시 로드하는 커널 모듈은 보통 /lib/modules/kernel-version에 들어 있다. 커널 모듈 은 시스템에 있는 다른 프로그램과 비교하면 링크된 오브젝트 파일이라는 점에 같지만, 재 배치가능한 이미지로 링크되어 있다는 점이 다르다. 즉, 특정 주소에서 시작하도록 링크되어 있지 않다는 것이다. 이 이미지는 a.out 포맷이나 ELF 포맷의 오브젝트 파일일 수 있다. insmod는 커널이 익스포트(export)하는 심볼을 찾기 위해 특권층의 시스템 콜을 사용한다. 커널은 익스포트 심볼을 심볼의 이름과 그것의 값(심볼의 주소같은)의 쌍으로 가지고 있다. 커널의 익스포트 심볼 테이블은, 커널이 관리하는 모듈의 목록인 module_list 포인터가 가리키고 있는, 첫번째 module 자료구조에 들어 있다. 커널에 있는 모든 심볼들이 모듈에게 익스포트 되는 것은 아니다. 단지 커널을 컴파일하고 링크할 때 특별히 지정한 심볼만이 이 테이블에 들어간다[4]. 드라이버가 시스템의 특정 인터럽트의 제어권을 갖고 싶을 때 호출 해야 하는 커널루틴인 "request_irq" 심볼을 예로 들어보자. 필자가 갖고 있는 현재 커널 에서 이것의 값은 0x0010CD30 이다. 커널의 익스포트 심볼과 값은 /proc/ksyms를 살펴보 거나 ksyms 프로그램을 사용하여 볼 수 있다. ksyms 프로그램을 이용하여 커널에 있는 모 든 익스포트 심볼을 볼 수도 있고, 로드된 모듈이 익스포트하는 심볼들의 목록만 볼 수도 있다. insmod는 모듈을 자신의 가상 메모리 공간으로 읽어들이고, 아직 해결되지 않은 커 널 루틴과 자원에 대한 참조를 커널에 있는 익스포트 심볼을 통하여 맞추어준다. 이렇게 위 치를 고정하는 것은 메모리상에 있는 모듈 이미지를 수정하는 형태로 이루어진다. insmod 는 모듈에 있는 해당하는 위치에 물리적으로 심볼의 주소를 써넣는다.

insmod가 모듈의 익스포트된 커널 심볼에 대한 참조를 모두 해결하였다면, 특권 시스템 콜을 이용하여 커널에게 새로운 커널을 포함할 수 있는 충분한 공간이 있는지 묻는다. 커널 은 새 module 자료구조와, 새 모듈을 충분히 포함할 수 있는 크기의 커널 메모리를 할당하 고, 이 구조체를 커널 모듈 리스트의 끝에 넣는다. 새 모듈은 초기화되지 않았다고 (UNINITIALIZED) 표시된다. 그림 12.1은 FAT와 VFAT 두 모듈이 커널에 로드된 후의 커 널 모듈의 리스트를 보여준다. 이 그림에는 나타나지 않았지만, 리스트에 있는 첫번째 모듈 은 유사 모듈(pseudo module)로서 단지 커널의 익스포트 심볼 테이블을 갖기 위해 존재한다. 로드된 커널의 목록과 그들의 상관관계를 보고 싶으면 lsmod 명령어를 쓰면 된다. lsmod 명령은 단지 커널 module 자료구조의 리스트로 부터 만들어지는 /proc/modules의 포맷을 바꾸어서 보여주는 것 뿐이다. 커널이 모듈을 위해 할당한 메모리는 insmod가 이에 접 근할 수 있도록 insmod 프로세스의 주소공간에 매핑이 된다. insmod는 모듈을 할당받은 공간으로 복사를 하고 이를 재배치하여, 할당받은 커널 공간에서 실행될 수 있도록 한다. 이 는 모듈이 서로 다른 리눅스 시스템에서 똑같은 주소에 로드되거나 두번 모두 같은 주소에 로드된다는 보장이 없기 때문에 반드시 필요하다. 다시 한번, 이렇게 재배치하는 것은 모듈 의 이미지를 올바른 주소로 수정하는 것을 포함한다.

새 모듈은 또한 커널에 심볼들을 익스포트하기 때문에, insmod는 이렇게 익스포트된 이미 지의 테이블을 만든다. 모든 커널 모듈은 모듈 초기화와 모듈 정리 루틴을 가지고 있어야 한다[5]. 이 두 심볼은 익스포트 되진 않지만, insmod는 이 둘의 주소를 알아내어 커널에 넘 겨야 한다. 모든 것이 잘 되었다면, insmod는 이제 모듈을 초기화할 준비가 되어 있고, 특 권 시스템 콜을 불러 커널에 모듈의 초기화 루틴과 정리 루틴의 주소를 넘긴다.

새 모듈이 커널에 추가되면, 커널의 심볼 목록을 갱신하고 새 모듈이 사용하는 모듈들을 수 정해야 한다. 자신에 의존하는 다른 모듈을 가진 모듈은, 자신의 module 자료구조의 포인 터가 가리키고 있는 자신의 심볼 테이블 끝에 참조되는 목록을 관리하여야 한다. 그림 12.1 은 VFAT 파일 시스템 모듈이 FAT 파일 시스템 모듈에 의존하고 있음을 보여준다. 따라서 FAT 모듈은 VFAT 모듈에 대한 참조를 포함하고 있다. 이 참조는 VFAT 모듈이 로드될 때 추가된 것이다. 커널은 모듈의 초기화 루틴을 부르고, 이것이 성공하면 모듈 설치를 계속 하 게 된다. 모듈의 정리 루틴의 주소는 모듈의 module 자료구조에 저장되며, 모듈이 언로드 될 때 커널에 의해 호출된다. 마지막으로 모듈의 상태는 실행중(RUNNING)으로 설정된다.

12.2 모듈을 언로드하기

모듈은 rmmod 명령을 사용하여 제거할 수 있지만, 요구시 로드된 모듈은 더이상 사용되지 않을 때 kerneld에 의해 시스템에서 자동으로 제거된다. kerneld의 타이머가 만료될 때 마다, kerneld는 사용되지 않는 요구시 로드된 모듈을 시스템에서 제거하는 시스템 콜을 부른다. 타이머의 값은 kerneld를 시작할 때 설정되는데,. 필자의 시스템에서는 180초마다 검사하도록 설정되어 있다. 그래서, 예를들어 ISO9660 파일시스템이 모듈로 되어 있는 곳에 서 ISO9660 CDROM을 마운트했다면, CDROM을 언마운트한 후 조금 있으면 ISO9660 모듈 이 커널에서 제거된다.

모듈은 커널의 다른 부분이 자신에 의존하고 있을 때에는 언로드될 수 없다. 예를 들어, 하 나 이상의 VFAT 파일 시스템이 마운트되어 있는 동안에는 VFAT 모듈을 언로드할 수 없다. lsmod의 출력을 눈여겨보면, 모듈에 숫자가 같이 붙어 나오는 것을 볼 수 있을 것이다. 예 를 들어 :

```
Module :        #pages:         Used by
msdos   5                       1
vfat    4                       1 (autoclean)
fat     6       [vfat msdos]    2 (autoclean)
```

카운트는 이 모듈에 의존하고 있는 커널 요소의 개수이다. 위의 예에서는, vfat와 msdos 모듈이 fat 모듈에 의존하고 있으므로 카운트가 2가 된다. vfat와 msdos 모듈은 이 값으 로 1을 갖고 있는데 이것은 마운트된 파일시스템이다. 만약 다른 VFAT 파일 시스템을 읽어 들이면, vfat 모듈의 카운트는 2가 될 것이다. 모듈의 카운트는 그 이미지의 첫번째 longword에 저장된다.

이 항목에는 또한 AUTOCLEAN과 VISITED 플래그가 더 있다. 이 두 플래그는 요구시 로드 된 모듈에서 사용된다. 이들 모듈은 자동으로 언도드 될 수 있다는 것을 시스템이 알 수 있 도록 AUTOCLEAN이라고 표시된다. VISITED 플래그는 모듈이 하나 이상의 다른 시스템 구 성요소에 의해 사용되고 있음을 말한다. 이는 다른 구성요소가 그 모듈을 사용할 때마다 설 정이 된다. kerneld가 시스템에 사용되지 않고 있는 요구시 로드된 모듈을 제거하라고 요 청할 때마다, 시스템은 자신에게 있는 모든 모듈을 뒤져서 그런 후보들을 골라낸다. 이는 단 지 AUTOCLEAN이라고 표시되어 있고 RUNNING 상태에 있는 모듈만을 찾는다. 만약 그 후 보의 VISITED 플래그가 설정되어 있지 않다면 그 모듈을 제거하고, 그렇지 않다면 VISITED 플래그를 지우고 시스템의 다른 모듈을 계속 살펴본다.

한 모듈이 언로드 가능하다고 한다면, 그 모듈이 할당받은 커널의 자원을 해제할 수 있도록 모듈의 정리 루틴이 호출된다. 모듈의 자료구조는 DELETED로 표시되고, 커널 모듈의 리스 트와의 연결을 끊는다. 그 모듈이 의존하고 있는 다른 모듈은 더 이상 자신에 의존하지 않 다는 것을 나타내도록 참조목록이 수정된다. 모듈이 필요로 했던 모든 커널 메모리는 해제 된다.

번역 : 이호
정리 : 이호

역주 1) 커널을 새로 컴파일하는 것을 가리킨다. (flyduck)

역주 2) 이를 module stacking이라고 한다. (flyduck)

역주 3) 모듈을 컴파일할 때 커널의 버전 정보를 넣을 수 있다 이 경우 insmod가 모듈을 로드할 때 버전 검사를 하여, 버전이 맞지 않으면 모듈을 로드할 수 없다. insmod -f 옵 션을 사용하면 버전이 맞지 않더라도 로드하게 할 수는 있지만 안전하진 않을 것이다. 좀 더 좋은 방법으로 모듈이 사용하는 커널 서비스에 넘겨주는 인자들이 달라진 경우에 만 모듈을 로드할 수 없게 할 수 있다. 즉 컴파일 된 모듈이 사용하는 서비스가 현재 커 널에서 제공하는 서비스와 달라진 것이 없다면 문제가 되지 않으며, 이를 위해선 커널과 모듈 양쪽에서 심볼이 인자정보를 체크섬으로 가지고 있도록 해야한다. (flyduck)

역주 4) kernel/ksymc.c에 보면 커널이 익스포트할 심볼들의 목록이 들어 있다. 이와 마찬가지 로 모듈을 만들 때 모듈에 있는 모든 심볼들을 익스포트하지 않고 필요한 것만 익스포트 하도록 할 수 있다. 이는 너무 많은 심볼들이 심볼 테이블에 들어가 발생할 수 있는 문 제를 미리 막기 위한 것이다. (flyduck)

역주 5) 이들의 이름은 각각 init_module(), cleanup_module()로 정해져 있다. 이들은 심볼 테이블에 들어있지 않더라도 전역 함수로 되어 있다면 그 주소를 알아낼 수 있다. (flyduck)