커널메모리 릭 방지를 위한 리눅스 커널 메모리 관리자 구현*

백승재° 박세은* 최종무*
*단국대학교 정보컴퓨터 학부
°단국대학교 전기전자컴퓨터 공학부
ibanez1383@hanmail.net

Implementation of Linux Kernel Memory Protector for Preventing Kernel Memory Leak

Seung-Jae Baek° Se-un Park* Jongmoo Choi*
°Dept. of Electrical & Electronics & Computer Engineering, Dankook University
*Div. of Information and Computer Science, Dankook University

요 약

본 연구는 날로 다양하고 복잡해지고 있는 임베디드 시스템에서 필수로 요구되는 운영체제의 주요 기술적 과제중 하나인 효율적으로 메모리를 사용할 수 있는 메모리 보호기능을 설계하여 리눅스 상에 구현하였다. 이는 현재 사용되고 있는 많은 임베디드 리눅스에 직접 적용하여 실제적인 메모리 관리 성능향상을 가져오며, 또한 차세대 메모리로 주목받고 있는 PRAM등 차세대 NVRAM에서의 필요성이 특히 부각된다는 점에서 그 중요성이 크다.

1. 서 론

정보가전제품과 모바일 기기 등의 보급에 따라 임베디드 시스템에 대한 수요가 폭발적으로 증가하고 있다. 이러한 임베디드 시스템은 서비스가 다양해지고 기술의 복잡성으로 인하여 운영체제를 필요로 하고 있다. 수많은 임베디드 시스템용 운영체제 중 점점 리눅스의 비중이 높아지는 것은 첫째 크기가 작고, 둘째 쉽게 재구성화가 가능하고, 셋째 인터넷과 최상으로 호환되고, 넷째 무료이기 때문이다[1]. 즉, 용도와 구조 및 하드웨어 구성이 다양한 임베디드 시스템에선, 복잡하고 방대한 환경을 모두 지원할수 있는 운영체제가 바로 리눅스이다.

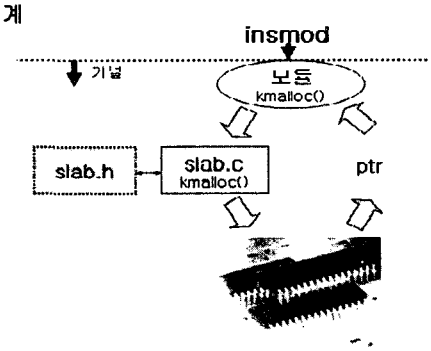
최대한 많은 공간을 사용자의 정보나 자료를 저장하고, 사용자 프로그램의 저장에 할당하기 위해서는 커널의 크기를 최소화시켜야만 한다. 즉 사용하지 않는 기능이나 디바이스 드라이버 등을 커널에서 제외 시켜야 한다. 하지만 사용 빈도가 낮더라도 새로운 장비들을 장착해 사용해야 할 경우가 있다. 이런 경우를 대비해, 필요한 디바이스 드라이버를 모두 포함시켜 놓는다면 사용빈도가 낮은 디바이스 때문에 비용이 높은 메모리의 손실이 생길수 있다. 이러한 문제를 해결하기위해 본래 모놀리틱 커널이던 리눅스는, 모듈 개념을 도입하여 필요시에만 디바이스 드라이버용 모듈을 적재하여 사용하고 필요 없을 때는 다시 커널에서 내릴 수 있도록 했다. 따라서 평상시에는 커널의 크기를 최소로 유지할 수 있다.

그러나 모듈은 커널 내부에 삽입되어 실행되므로, 메모리 관리를 포함한 모든 자원에 대한 할당 / 해제의 책임이 전적으로 모듈 작성자에게 있다. 따라서 적절하게 자원의 할당 / 해제를 고려하지 않고 설계된 모듈이나, 예기치 않은 상황으로 인해 해제 해야 할 자원을 해제해 주지 않은 모듈은 시스템 자원의 낭비를 가져올 수 있다.

본 논문에서는 모듈의 자원사용에 대한 문제 중, 효율적으로 메모리를 사용할 수 있는 메모리 보호기능을 설계하여 리눅스 상에 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서는 커널 메모리 관리자의 설계를 설명한다. 3장에서는 실제 커널 수정을 통한 구현에 대해 기술한다. 4장에서는 실험결과를 기술하며, 5장에서는 결론 및 향후 연구 내용을 정리한다.

2. 설계



<그림 1> 모듈의 메모리 할당 개념도

커널은 모듈이 로딩될때 모듈 자체를 위한 공간과, <그림 1>에서 보여지듯이 모듈이 내부적으로 사용하기위해 요청한 공간을 할당한다. 모듈 자체를 위한 공간 할당 / 해제는 커널이 관리를 해주고 있다. 그러나 상기한 바와 같이 모듈이 내부적으로 사용하기 위해 요청한 공간의 할당 / 해제는 전적으로 모듈

* 이 연구는 2003학년도 단국대학교 대학 연구비의 지원으로 연구되었음.

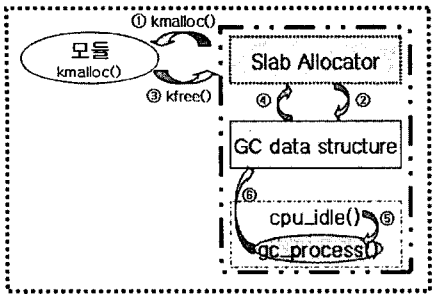
의 작성자에게 책임이 있으며, 만약 모듈이 메모리를 할당 받고 적절히 해제해 주지 않으면 이는 재부팅 전까지 사용이 불가능한 메모리 릭이된다. 이는 비교적 적은 양의 메모리가 가용하므로 효율적인 메모리 관리가 필수적인 임베디드 시스템상에서는 반드시 해결해야 할 문제이다. 차세대 메모리 매체로 각광을 받고 있는 PRAM, FRAM, MRAM등과 같은 Non-Volatile Memory에서는 시스템의 재부팅 후에도 메모리 릭으로 인해 손실된 공간을 사용할수 없다. 따라서 모듈이 사용하려고 요청하는 메모리공간을 효율적으로 관리하기 위한 커널 기능이 필수적이라고 하겠다.

커널 내부에서의 메모리 할당 함수는 kmalloc()과 vmalloc()이 있다. vmalloc()은 유저공간 할당 함수의 방식이며, 대부분의 모듈에서는 여러 이유로 kmalloc()으로 메모리를 할당 받아 사용하게 된다. 이 두 함수의 차이점은 아래 <표 1>과 같다.

	kmalloc()	vmalloc()
연속성	물리적으로 연속된 메모리를 할당	가상 주소 공간에서만 연속임을 보장
추가 플래그	설정 가능	설정 불가

<표 1> kmalloc()와 vmalloc()의 주요 차이점

따라서 본 논문에서는 모듈에서 주로 사용되는 kmalloc()에 대한 메모리 릭 방지 기능을 구현하였다.

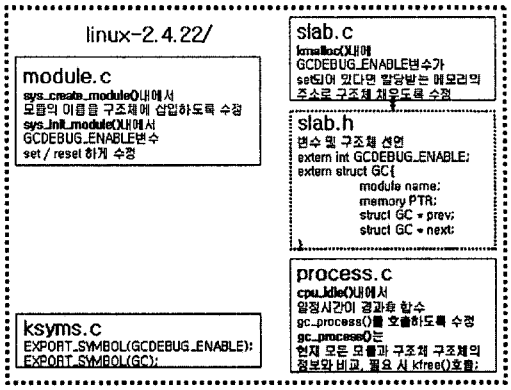


<그림 2> 알고리즘 개요

설계 내용은 <그림 2>와 같다. 모듈은 내부적으로 동적 메모리를 사용하기 위해 kmalloc()을 호출한다(<그림 2>의 ①). 그러면 리눅스 커널의 slab할당자는 어떤 모듈이 얼마 만큼의 공간을 할당 했는지 기록한 후(<그림 2>의 ②), 메모리를 모듈에게 할당해 준다. 언젠가 모듈이 삭제되면 자신이 사용하던 동적 메모리 공간을 반납한다(<그림 2>의 ③). 이때 slab할당자는 기록되어 있던 내용을 삭제한다(<그림 2>의 ④). 만일 모듈이 삭제될때 kfree()를 호출하지 못하거나 또는 내부적인 결함에 의해 종료되면 메모리 릭이 발생하게 된다. 본 논문에서는 위와 같은 메모리 릭을 다음과 같은 방법으로 해결한다. 시스템이 idle상태가 되면 cpu_idle()함수가 수행되는데, 이때 gc_process()를 호출한다. 이 함수는 kmalloc()와 kfree()시에 적절히 유지해 오던 구조체를 순차적으로 검색하여, 현재 활성화 되어 있지 않은 모듈의 정보가 구조체에 들어있다면 이를 kfree()시켜 메모리 릭을 방지한다. <그림 2>에서 GC자료구조와 gc_process()함수 및 ②,④,⑤,⑥과정이 본 논문에서 설계한 내용이다.

3. 구현

본 논문에서는 메모리 릭 방지 기능을 구현하기 위해 리눅스 커널 2.4.22을 선택하였다.



<그림 3> 커널 수정부 요약

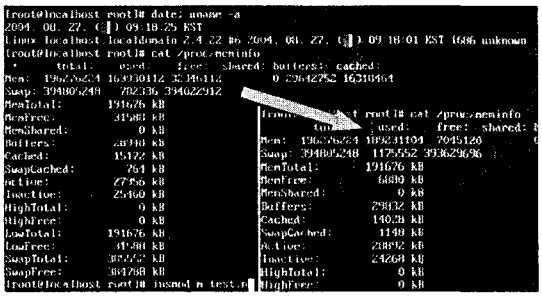
모듈은 insmod 명령을 이용하여 처음 로딩시에, 위 <그림 3>과 같이 내부적으로 sys_create_module()을 호출하며, 이는 다시 sys_init_module()을 호출하게 된다. 이때 본 논문에서는 include/linux/slab.h를 수정하여, 모듈의 이름과, 메모리를 할당해 사용할 경우 할당받은 포인터를 저장할 필드를 가지는 GC 구조체를 커널 내부에 추가한다. mm/slab.c를 수정하여, 모듈이 kmalloc()으로 메모리를 할당받을때 모듈의 이름과 할당받은 메모리의 포인터를 위 GC 구조체에 저장해 놓도록 한다. 또한 kfree()시엔 기 작성한 구조체에서, 해제하는 메모리에 해당되는 구조체 필드를 삭제하도록 한다.

다음으로, 커널이 idle state가 되어 arch/cpu별디렉토리/kernel/process.c내의 cpu_idle()를 호출하게 되면, 작성한 구조체를 검색해 나간다. 현재 로드 되어 있지 않은 모듈의 이름이 구조체 상에 존재한다면, 이는 정상적으로 kfree()되지 않는 메모리가 있음을 의미한다. 따라서 해당 메모리를 kfree()시켜준 뒤 구조체의 필드를 삭제해준다. cpu_idle()이 불려질때 마다 불필요게 여러번 구조체 검색이 생기는 것을 막기 위해 변수를 선언하여 이 변수가 특정 값, 즉 cpu_idle()이 호출된 뒤 일정 시간이 지난 뒤에만 작업이 이뤄지도록 구현하였다[2].

본 논문에서는 1G까지의 메모리 공간할당을 고려하여 구조체를 10000개 할당하였다. 이때 구조체를 위해 사용되는 공간은 240K이나, 실제 구조체는 1500개 이하로 사용되었다. 이는 효율적인 태스크의 관리를 위해 1.7K크기의 task_struct구조체를 사용하는 리눅스의 정책에 미뤄 볼때 충분히 가치있는 사용이다[3]. 일반적으로 이보다 훨씬 적은 메모리가 장착된 임베디드 시스템에선 적절한 구조체 개수 선택이 필요하다.

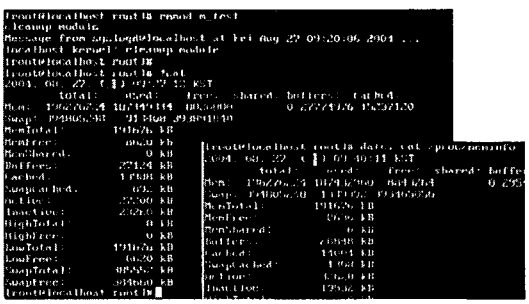
4. 실험 결과

실험을 위해 모듈 삽입시 커널 메모리를 할당 받고, 모듈 해제시 할당받은 공간을 반환하지 않는 모듈을 인위적으로 작성하였다. 이 모듈을 기존 리눅스 커널에 적용시켜 본 결과는 아래 <그림 4>와 <그림 5>에 나타나 있다.



<그림 4> 초기 메모리 사용량, 모듈 삽입

위 <그림 4>의 좌측 화면에서 현재의 메모리 사용량을 확인한다. insmod명령을 사용해 기 작성한 모듈을 삽입하면 우측 화면에 나타나 있듯이 Free메모리 공간이 줄어든 것을 확인 할 수 있다. 다음으로 <그림 5>의 좌측 화면과 같이, rmmod명령을 사용하여 삽입된 모듈을 해제한다. 그러면 좌우측 화면의 메모리 사용량 확인에서 볼수 있듯이, 커널은 해당 모듈이 빠져 나간 뒤에도 모듈에서 kmalloc()으로 할당받은 메모리 공간을 사용하지 못하고 있는 것을 알수 있으며, 따라서 이는 심각한 메모리 릭을 유발한다.

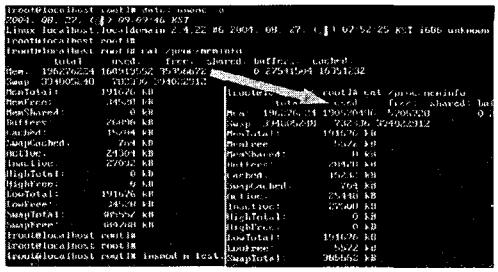


<그림 5> 모듈 해제후 메모리양 확인

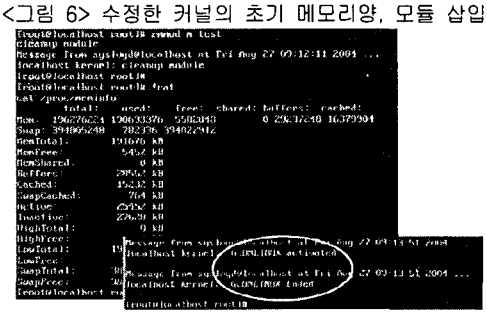
그러나 본 논문에서 설계한 대로 커널을 수정한 뒤, 같은 모듈을 적용시켜보면 결과는 다음과 같다.

우선 <그림 6>의 우측 메모리 사용량 확인 화면에서 보이듯이, 모듈은 이전과 마찬가지로 메모리를 할당받는다. 그러나 <그림 7>의 좌측 화면과 같이 rmmod명령을 통해 모듈이 빠져나간 뒤에, 임의로 정해놓은 시간이 지나면 우측 화면에 보이는 메모리 복구 과정을 거치게 된다. <그림 7>의 'GCONLINUX activated' 메시지가 gc_process()함수의 수행을 의미한다.

능동적으로 동작하는 메모리 복구 과정이 종료된 뒤, 시스템이 안정 상태가 되면, 아래 <그림 8>과 같이 기존 메모리를 다시 사용가능하게 되었음을 확인할 수 있다.



<그림 6> 수정한 커널의 초기 메모리양, 모듈 삽입



<그림 7> 모듈 해제, 메모리 관리자 작동

<그림 8> 메모리 복구 결과 확인

5. 결론

본 연구는 날로 다양하고 복잡해지고 있는 임베디드 시스템에서 필수로 요구되는 운영체제의 주요 기술적 과제중 하나인 '효율적 메모리 사용'을 위한 메모리 보호 기능을 설계 / 구현하였다. 이는 현재 사용되고 있는 많은 임베디드 리눅스에 직접 적용하여 실질적인 메모리 관리 성능 향상을 가져올 수 있다. 본 논문은 모듈의 kmalloc()호출뿐 아니라 다른 모든 커널 쓰레드들의 메모리 할당을 관리 할수 있도록 발전시키는 것을 추후 목표로 하며, 장기간에 걸친 커널 신뢰성 테스트를 통해 보다 완벽한 메모리 관리 도구로 확장할 것이다.

참고 문헌

[1] 박영환, 임베디드 시스템 임베디드 리눅스, 2002

[2] Daniel P. Bovet, Understanding the Linux Kernel, 2nd Edition, 2003

[3] Robert Love, Linux Kernel Development, 2003