

신기술/기능 요구사항 지원 보고서

[제목: 리눅스 메모리 관리 정책]

한국소프트웨어진흥원
공개SW기술지원센터

<Revision 정보>

일자	VERSION	변경내역	작성자
2007.11.11	0.1	초기 작성	김상운

목 차

1. 문서 개요	4
가. 문서의 목적	4
나. 본 문서의 사용방법	4
2. 신기술/기능 요구사항	5
가. 대상 업체	5
나. 요구 사항	5
3. 리눅스의 메모리 관리 정책	6
가. 리눅스 메모리의 개요	6
나. x86 아키텍처의 특성	6
다. 메모리 맵핑(Memory Mapping)	6
라. 하위 레벨 메모리 할당	7
1) 부팅시간 메모리 초기화	7
2) 실행시간 메모리 할당	7
마. 스왑(Swap)	8
1) 커널 스왑 관리 데몬(kswapd)	8
2) 스와핑이 발생하는 경우	9
바. 결론	9

1. 문서 개요

본 문서는 리눅스 커널 2.6에서 새롭게 적용된 메모리 활용 기법에 대해서 설명하는 문서이다.

가. 문서의 목적

다음과 같은 세부적인 목적을 달성하기 위하여 작성되었다.

- 0 커널 2.6 기반에서 x86 시스템의 메모리 관리 기법 설명
- 0 개발자 입장에서 메모리 관리 기법 제시

나. 본 문서의 사용방법

다음과 같은 방법으로 사용할 수 있다.

- 0 본 문서의 리눅스 문서 사이트인 TLDP의 "Linux Memory Management"를 참조하여 번역 및 작성됨
- 0 최신 내용은 <http://www.tldp.org/HOWTO/KernelAnalysis-HOWTO-7.html>를 참조하기 바람

2. 신기술/기능 요구사항

가. 대상 업체

구 분		비 고
기관명	(주)한글과컴퓨터	
담당자	이호성	
연락처	02-3424-3166	
회사 위치	서울 광진구 구의동	

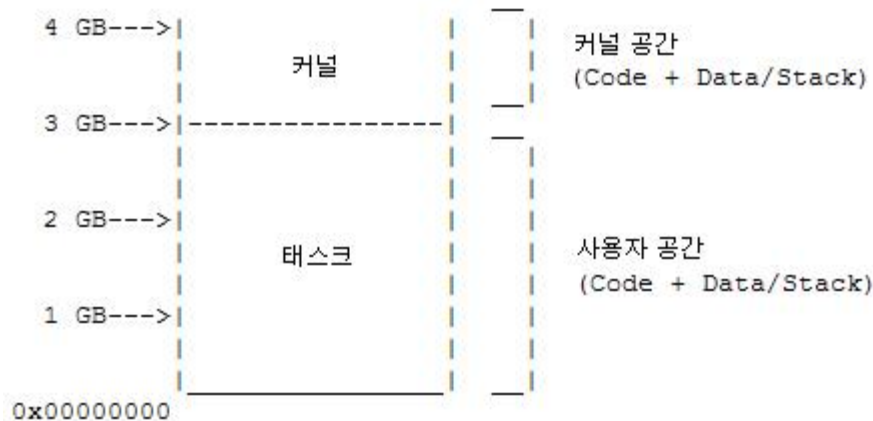
나. 요구 사항

항목	지원 내용	비고
x86 아키텍처와 커널 2.6에서 메모리 관리 기법 요구	커널 2.6에서 새롭게 적용된 메모리 관리기법 설명 및 전달	

3. 리눅스의 메모리 관리 정책

가. 리눅스 메모리의 개요

리눅스는 세그멘테이션 기법과 페이징 기법을 조합하여 사용한다. 리눅스는 4개의 세그먼트를 사용하고, 나머지 메모리관리는 페이징 기법을 사용한다. 2개의 세그먼트는 커널 공간을 위한 코드와 데이터/스택을 위해 사용하며 2개의 세그먼트는 사용자 공간을 위한 프로세스 코드와 데이터/스택을 위해 사용한다. 이 사용자공간 세그먼트는 각 프로세스별로 독립적으로 관리된다. 32 bit x86 버전의 접근 가능한 4GB 선형 메모리 주소 공간에서 3 ~ 4GB 영역은 커널 공간으로 사용하며, 0 ~ 3GB 영역은 사용자 공간으로 사용된다.



[그림1] 커널과 사용자 선형 주소

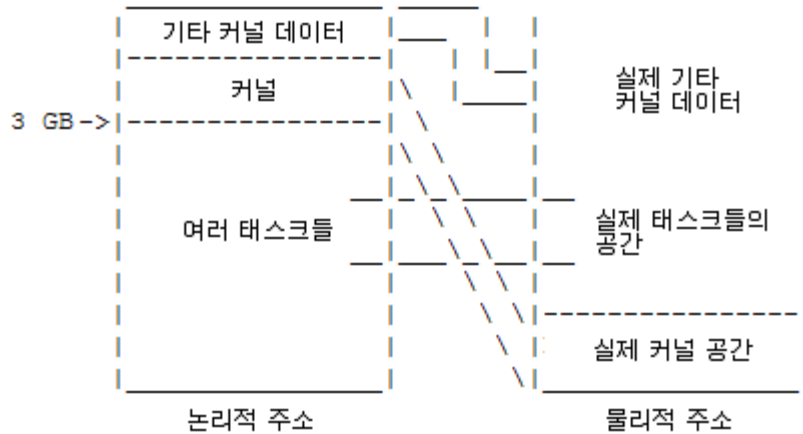
나. x86 아키텍처의 특성

리눅스 커널은 페이징 구조를 3 단계 페이징으로 구현되어 있다. 그러나, 32bit인 i386 아키텍처는 하드웨어 레벨에서 2 단계 페이징만을 지원한다. i386 프로세서에서 동작 시에는 페이지 테이블을 0 bit로 사용하여, 2 단계로 사용한다. 참고로, Pentium Pro 프로세서부터는 PAE (Physical Address Extension)란 기술을 사용하여 64GB까지 확장하여 메모리 접근을 가능하게 지원하며, 커널 컴파일 시 PAE 옵션을 키면 3 단계 페이징으로 사용한다. 현재 내 프로세서가 PAE를 지원하는지 확인은, /proc/cpuinfo의 flags 필드에 "pae"가 있는지로 알 수 있다.

다. 메모리 맵핑(Memory Mapping)

리눅스는 페이징에 대한 접근 제어만을 관리한다. 그래서 다른 태스크가 같은 세그먼트 주소를 가질 수 있지만, 디렉토리 페이지에 저장되어 있는 다른 CR3의 값이 다른 페이지를 가리킨다. 사용자 모드에서 태스크는 3GB 제한을 넘을 수 없으며, 첫 768 개의 페이지 디렉토리 엔트리만이 의미있다. 태스크가 시스템 호출이나 인터럽트에 의해 커널 모드로 진입하

면, 나머지 256 개의 페이지 디렉토리가 중요하게 되고, 모든 다른 태 스크에 의해서 같은 페이지 파일들을 지정하게 된다. (즉, 커널 모드에서는 모든 태스크가 같은 메모리를 접근한다.)



[그림2] 커널 선형 공간과 커널 물리 공간 매핑

라. 하위 레벨 메모리 할당

이 부분에서는 리눅스에서 부팅 시간, 실행 시간 에 메모리를 할당하는 함수 리스트들을 알아보도록 하자. 함수 각각에 대한 코드 내용은 각자 코드를 따라가면서 살펴보도록 하자.

1) 부팅시간 메모리 초기화

메모리 초기화 과정은 부팅 시간에 `start_kernel()` 함수 내에서 `mem_init()` 함수를 호출하여 수행한다. 그리고, 이어서 `kmem_cache_init()` 함수 를 호출하여 커널 캐시 메모리를 초기화 한다. 호출되는 함수의 순서는 다음과 같다. 각 함수의 링크를 따라 가면 리눅스 커널 2.6.18 기준으로 구현되어 있는 소스 코드를 살펴볼 수 있으므로, 참고하기 바란다.

```
| mem_init()
|   | free_all_bootmem()
|   | free_all_bootmem_core()

| kmem_cache_init()
|   | cache_estimate()
```

2) 실행 시간 메모리 할당

리눅스에서 "copy_on_write" 매커니즘과 같이 메모리 할당을 요청할 때, 다음의 함수를

호출하여 수행 한다. "copy_on_write" 매커니즘은 프로세스 관리편에 서도 설명했듯이, 실행 중인 프로세스가 자식 프로세스 를 생성한 경우 프로세스 메모리 영역의 복사가 바로 이루어 지 않고, 부모나 자식 프로세스 중에 하나의 프 로세스가 메모리 write 작업이 발생하는 경우 에 프로세 스 메모리 영역을 복사하는 것이다. fork() 수행 시 프 로세스를 복사하는 copy_process() 함수 내에서 copy_mm() 함수를 호출한다. 호출되는 함수의 순서는 다음과 같다.

```
| copy_mm
| allocate_mm = kmem_cache_alloc
| __cache_alloc
| ____cache_alloc
| cache_alloc_refill
| cache_grow
& | kmem_getpages
& | alloc_pages_node
& | __alloc_pages
```

마. 스왑(Swap)

스왑은 한정된 물리 메모리를 가지고 가상 메모 리를 지원하는데 있어서 필수적인 기능이다. 태스크가 페이지 요청 시 물리적인 메모리가 부족할 경우, 페이지 폴트 예외가 발생하게 된다. 이 경우 스와핑 (swapping)이 발생하며, 물리적인 메모리의 페이지 일 부를 다른 장치(일반적으로 하드 디스크의 스왑영역을 사용)에 저장하고, 필요한 페이지를 할당하여 할당하게 된다.

1) 커널 스왑 관리 데몬(kswapd)

커널 스왑 관리 데몬은 다른 커널 쓰레드들과 같 이 깨어날 때까지 대기 상태로 전환하며, 스왑 요청에 의해 깨어나면 작업을 수행 하고 대기 상태로 다시 전 환되는 작업을 반복한다. 커널 스왑 관리 데몬은 kswapd() 함수로부터 시작된다. 호출되는 함수의 순서 는 다음과 같다.

```
| kswapd
| // 초기화 작업
| for (;;) { // 메인 루프
| 인터럽트 가능한 상태로 설정 및 대기 큐에 넣음
| 페이지 폴트로 커널 스왑 관리 데몬이 깨어 난 경우 대기 큐에서 제거
```



```
| 페이지 데이터에 대해 조절
|}
```

2) 스와핑 이 발생하는 경우

스와핑은 앞서도 간략히 언급했듯이 페이지 폴트 익셉션이 발생하는 경우에 발생하게 된다. 페이지 폴트 예외는 "1. 사용자 페이지"가 "2. 읽기/쓰기 접근 시"에 "3. 페이지가 존재하지 않는 경우" 세가지 조건 이 만족될 때 발생하게 된다. 페이지 폴트 예외가 발생 하면, `do_page_fault()` 함수가 호출되며, 이 함수를 쫓 아가다보면, `wakeup_kswapd()` 함수를 호출하여 인터럽트를 발생시키고, 커널 스왑 관리 데몬을 깨우게 된다. 호출되는 함수의 순서는 다음과 같다.

```
| do_page_fault
|   | handle_mm_fault
|   |   | pte_alloc_map
|   |   |   | __pte_alloc
|   |   |   |   | pte_alloc_one
|   |   |   |   |   | alloc_pages
|   |   |   |   |   |   | alloc_pages_current
|   |   |   |   |   |   |   | __alloc_pages
|   |   |   |   |   |   |   |   | wakeup_kswapd // 커널 스왑 관리 데몬 kswapd를 깨움
```

바. 결론

메모리 관리는 크고, 복잡하고 시간이 많이 소요되는 일이다. 실제 멀티프로그래밍 환경에서 시스템이 어떻게 동작하는지를 모델링하는 것이기 때문에 매우 까다로운 작업이다. 스케줄링, 페이징 작동, 멀티프로세서 인터랙션 같은 컴포넌트들은 상당한 도전 과제이다. d; 글이 여러분께 도움이 되었기를 바란다.