플래시 메모리 기반 리눅스 스왑 시스템에서 기동 시간의 단축

고소향, 전선수, 류연승
명지대학교 컴퓨터소프트웨어학과
e-mail : sohyang01@nate.com

Reducing start-up latency in linux swap system over flash memory

Sohyang Ko, Seonsu Jeon and Yeonseung Ryu
Dept. of Computer Software, Myongji Univ.

요 약

가상 메모리의 스왑 저장 장치로서 플래시 메모리를 사용하는 경우, 시스템을 기동할 때 스왑 영역의 초기화를 위한 삭제 연산이 요구되어 기동 시간이 오래 걸리는 문제점이 있다. 본 논문에서는 스왑 영역의 플래시 메모리 내용을 모두 삭제하지 않고 일부만을 삭제함으로서 기동 시간을 줄일 수 있는 방법을 연구하였다.

1. 서론

최근 NAND 플래시 메모리를 사용하는 SSD (Solid State Disk)의 사용이 매우 빠르게 증가하고 있다. 올해 출시된 256GB SSD 는 동급 HDD 와 비교했을 때 입출력 속도가 약 2.4 배 빠르며 전력 소비가 적어 IOPS/Watt 성능이 우수하다. 향후에 SDD 의 가격이 급격하게 떨어질 것으로 예상되어 HDD 를 점차 대체해나갈 것으로 예상된다.

플래시 메모리는 EEPROM 의 일종인 비 휘발성 저장 매체로서 전원 공급이 단절되어도 데이터는 계속 남아 있다. 플래시 메모리에 대한 읽기와 쓰기는 페이지 단위로 수행되며 삭제는 페이지들의 모임인 블록 단위로 수행된다. 삭제 연산은 페이지가 한 번 쓰여진 후에 다시 쓰기 위한 필수적인 선행 과정이다. 이 삭제 과정은 하드 디스크에는 없는 새로운 특징이다. 또한 한 블록에서 삭제 횟수가 정해진 횟수를 넘을 경우 그 블록은 더 이상 사용할 수 없게 된다.

저장 장치를 HDD 대신 SDD 를 사용하게 되면 운영체제의 파일 시스템 및 가상 메모리 시스템 등 많은 기능이 변경되어야 한다. 그동안 플래시 메모리를 위한 파일 시스템에 대한 연구는 많이 있어왔지만[5-8] 가상 메모리 시스템의 스왑 시스템에 대해서는 연구가 거의 없다[11]. 스왑은 가상 메모리 시스템의 페이지 프레임 회수 알고리즘에 의하여 페이지 프레임을 일시적으로 스왑 장치에 스왑 아웃하여 여유 메모리를 확보하기 위한 가상 메모리 기법이다.

이 논문은 2007 년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2007-313-D00637)

본 논문에서는 스왑 저장 장치로서 플래시 메모리를 사용할 때 기동 시간(start-up time)이 오래 걸리는 문제를 해결하기 위하여 새로운 스왑 공간 관리 기법 및 기동 방법을 연구하였다. 플래시 메모리는 데이터를 기록하기 위해서는 해당 페이지의 내용이 먼저 삭제되어야만 한다. 시스템이 종료되면 수행 중인 프로세스도 함께 종료되므로 스왑 저장 장치에 저장된 내용(프로세스의 페이지들)은 필요 없는 내용이 된다. 따라서, 다시 부팅할 때에 이전에 사용된 플래시 메모리의 블록들을 삭제하여 사용이 가능하도록 해주어야 한다. 플래시 메모리의 삭제 연산은 한 블록을 삭제할 때 약 1.5ms 가 소요된다. 따라서, 스왑 저장 장치의 공간 크기가 커질수록 재 사용에 필요한 초기화에 많은 시간이 소요된다.

본 논문에서 제안한 방법은 스왑 저장 장치의 플래시 메모리 블록을 전부 삭제하지 않도록 스왑 공간을 관리하고 기동 시에 일부 블록만을 삭제하여 기동 시간을 줄인다.

본 논문의 구성은 다음과 같다. 2 장에서는 리눅스 커널의 스왑 시스템에서 스왑 공간의 관리 방법에 대해 살펴보고, 플래시 메모리의 특성을 검토한다. 3 장에서는 제안한 스왑 시스템의 공간 관리 방법과 기동 방법을 설명하고 4 장에서는 성능 평가 결과를 기술한다. 마지막으로 5 장에서 결론과 향후 과제를 기술한다.

2. 관련연구

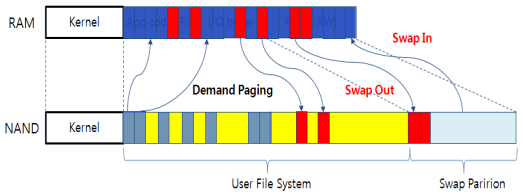
2.1 리눅스 스왑 시스템

스왑 영역은 메모리로부터 스왑 아웃된 페이지를 저장하는 공간이다. 스왑 영역은 독자적인 디스크 파

티션일 수도 있고, 파티션에 포함된 한 파일일 수도 있다. 페이지 프레임이 디스크에 스왑 아웃 될 때에 스왑 영역번호와 슬롯 인덱스로 구성된 식별자를 페이지 테이블 엔트리에 저장한다. 최대 스왑 영역의 개수는 MAX_ SWAPFILES 매크로에 정의되어 있으며 **<표 1> 플래시 메모리의 접근시간[9]**보통 32 개로 정의되어 있다. 하나의 스왑 영역에는 여러 개의 슬롯으로 구성되어 있다. 한 개의 슬롯의 크기는 4KB 이며, 페이지 프레임의 크기와 똑같다.

swap_map 배열은 각 슬롯의 상태를 저장한다. 슬롯에 해당하는 swap_map 배열의 값이 0 이라면 그 슬롯은 할당 가능 상태이다. 이 값이 양수라면 슬롯은 스왑 아웃 된 페이지로 채워져 있으며 해당 페이지를 공유하는 프로세스의 수를 나타낸다. 이 값이 SWAP_MAP_MAX (32,767)이면 페이지 슬롯에 저장된 페이지는 영구적이며 슬롯에서 제거할 수 없다. 이 값이 SWAP_MAP_BAD (32,768)이면 페이지 슬롯에 결함이 있는 것이므로 사용할 수 없다 [1,2].

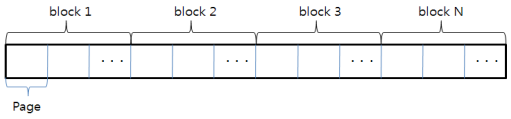
페이지 부재가 발생하면 커널은 페이지 테이블 엔트리(PTE)가 가리키는 슬롯을 읽어 들인다. 커널은 슬롯을 읽을 때마다 스왑 영역의 사용율이 50%가 초과하였는지 검사한다. 만약 초과했다면 커널은 슬롯들을 스왑인 하여 여유 슬롯들을 확보하려 한다 [11].



(그림 1) 스왑 시스템

2.2 플래시 메모리의 특징

NAND 플래시 메모리는 블록들의 집합으로 구성되며 한 블록은 다시 여러 개의 페이지로 구성된다 (그림 2 참조). 페이지의 크기는 제품에 따라 512B, 2KB, 4KB 이다. 페이지는 읽기/쓰기 작업의 단위이다. 그러나 삭제 작업은 블록의 단위로 수행된다. 플래시 메모리는 삭제 연산이 선행되지 않으면 데이터를 저장할 수 없고 삭제 연산은 쓰기/읽기 연산에 비해 많은 시간이 소요된다(표 1 참조). 하나의 블록에 대하여 삭제 연산은 제품에 따라 약 1 만-100 만번의 제한을 갖는다. 이것은 장치의 수명과도 관련 있기 때문에 마모도의 최적화와 관련된 많은 연구가 있다. 특정 블록에 삭제가 집중되지 않고 모든 블록에 삭제 연산이 균등하게 수행되는 것을 마모도 평준화(wear leveling)라고 한다.



(그림 2) 플래시 메모리의 구성

<표 1>에서 볼 수 있듯이 NAND 플래시 메모리의 삭제 연산은 읽기/쓰기 연산에 비교했을 때 매우 느리기 때문에 삭제 연산은 주의깊게 수행되어야 한다.

(표 1) 플래시 메모리의 연산 속도

Operation	Access Time
Read	125μs(2B)
Write	300μs(2B)
Erase	1.5m

즉, 언제 삭제를 해야 하는지와 삭제 작업의 횟수를 줄이는 방법 등을 고려해야 한다. 일반적으로, 데이터의 변경을 원래 데이터에 수행하지 않고 빈 공간에 수행한 후, 이전 데이터는 가비지(무효 페이지, invalid page)로 남겼다가 나중에 이러한 가비지들을 처리하는 가비지 수집 기법을 사용한다[5,6,7,8].

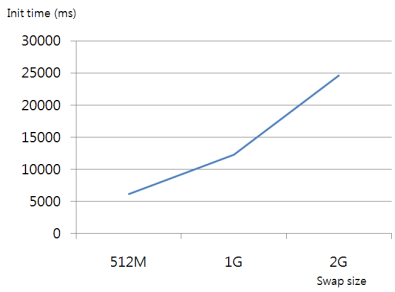
가비지 수집은 삭제할 블록을 결정한 후, 블록 안의 유효 페이지들은 빈 공간으로 복사하고 블록을 삭제한다.

3. 제안하는 기동 기법

3.1 배경

플래시 메모리는 데이터를 기록하기 위해서는 해당 페이지의 내용이 먼저 삭제되어야만 한다. 시스템이 종료되면 수행 중인 프로세스도 함께 종료되므로 스왑 저장 장치에 저장된 내용(프로세스의 페이지들)은 필요 없는 내용이 된다. 따라서, 다시 부팅할 때에 이전에 사용된 플래시 메모리의 블록들을 삭제하여 사용이 가능하도록 해주어야 한다. 플래시 메모리의 삭제 연산은 한 블록을 삭제할 때 약 1.5ms 가 소요된다. 따라서, 스왑 저장 장치의 공간 크기가 커질수록 재사용에 필요한 초기화에 많은 시간이 소요된다.

(그림 3)은 스왑 영역의 크기 당 기동 시간을 보여주고 있다. 이는 스왑 영역의 크기가 커질수록 많은 기동 시간이 소요됨을 보여준다.



(그림 3) 기동 시간

다음 수식은 기동 시간을 계산하는 수식이며 <표 2>는 수식에서 사용한 심볼에 대한 설명을 보여준다.

$$t_{start\_up\_time} = t_{block\_erase} * N$$

$$N = \frac{S_{swap\_area}}{S_{block}}$$

(표 2) 용어

용어	의미
$T_{start\_up\_time}$	스왑 기동시간
t_{blcok_erase}	블록을 삭제하는데 소요되는 시간 (1.5ms)
N	세그먼트 개수
S_{swap_area}	스왑 영역의 크기 (512M, 1G, 2G)
$S_{segment}$	세그먼트의 크기 (128개의 블록을 갖는다고 가정)

스왑 시스템의 기동 시간은 스왑 영역이 512MB 의 크기인 경우 1.5ms * (512M / 128K)가 걸리며, 1GB 의 크기인 경우 1.5ms * (1G / 128K)가 걸리며, 2GB 의 크기인 경우 1.5ms * (2G / 128K)가 걸린다.

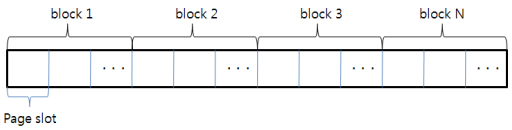
3.2 새로운 스왑 영역의 구조

본 논문에서 제안한 방법에서는 스왑 영역을 여러 개의 세그먼트 집합으로 관리하며 각 세그먼트는 플래시 메모리의 삭제 블록 집합으로 구성한다 (그림 4 참조). 또한, 각 삭제 블록은 스왑 페이지를 저장하기 위한 스왑 페이지 슬롯들로 구성된다. 페이지 슬롯은 리눅스에서와 같이 4KB 의 크기를 가진다.

각 세그먼트는 세그먼트 안에서 첫번째 블록의 첫번째 페이지 슬롯을 세그먼트 헤더로 사용한다. 세그먼트 헤더는 다음과 같은 세그먼트에 대한 정보를 관리한다.

- 세그먼트에 수행된 삭제 연산 횟수
- 세그먼트 상태 (used, free)

<a> 스왑 영역의 구조



<b> 각 세그먼트 구조

(그림 4) 제안하는 스왑 영역의 전체구조

제안한 방법에서 세그먼트는 삭제 단위가 된다. 즉, 세그먼트 내부의 블록들을 삭제할 때는 함께 삭제한다. 세그먼트를 삭제하면 세그먼트 헤더에 세그먼트 상태를 free 로 기록하며 삭제 연산 횟수를 1 증가시킨 값을 기록한다.

3.3 새로운 기동 기법

운영체제가 부팅되어 시작될 때 스왑 영역의 모든 세그먼트 헤더들을 스캔 하여 읽어 세그먼트 정보를 구한다. 이때, 세그먼트 관리를 위한 데이터 구조체를 RAM 에 생성한다. 이 데이터 구조체는 세그먼트 헤더의 정보를 저장한다. 그러나, 세그먼트의 상태 값은 free, old_used, new_used 세 개의 값으로 관리한다. 처음 기동할 때는 세그먼트 헤더에서 읽은 세그먼트의 상태 값이 free 이면 free 로 하며, used 이면 old_used 로 한다. 나중에 스왑 시스템에서 세그먼트에 페이지를 스왑 아웃하게 되면 데이터 구조체의 해당 세그먼트 상태 값을 new_used 로 변경한다.

RAM 상의 데이터 구조체를 완성하고 나면 스왑 시스템은 가용 세그먼트 개수를 고려하여 세그먼트의 삭제가 필요한 지를 결정한다. 가용 세그먼트의 개수가 부족하여 세그먼트의 삭제가 필요하면 삭제할 세그먼트들을 결정하여 삭제해주고 나머지는 세그먼트들은 나중에 가비지 수집으로 삭제한다.

1. 세그먼트 헤더 검사
2. 헤더상태(state) 검사
3. 헤더정보를 가진 구조체 생성
4. 삭제할 세그먼트 결정
5. 결정된 세그먼트 삭제

3.4 가비지 수집

가비지 수집은 스왑 아웃되어 쓸모 없는 페이지 슬롯을 포함하고 있는 세그먼트를 삭제하여 사용 가능한 세그먼트로 만드는 작업이다. 삭제할 세그먼트가 old_used 상태라면 부팅 이전의 페이지들을 가지고 부팅 이후의 페이지는 포함하지 않고 있기 때문에 바로 삭제할 수 있다. 그런데, 삭제할 세그먼트가 new_used 상태라면, 부팅 이후의 페이지를 포함하고 있으므로 유효한(즉, 스왑 인이 안된) 페이지는 다른 세그먼트로 복사한 후에 세그먼트를 삭제해야 한다.

이와 같은 가비지 수집 절차를 설명하면 다음 표와 같다. 회수할 세그먼트 선택 한다. 세그먼트 상태가 old_used 이면 세그먼트를 삭제하고, nuew_use 이면 유효슬롯을 빈 세그먼트로 복사한다. 마지막으로 세그먼트를 삭제한다.

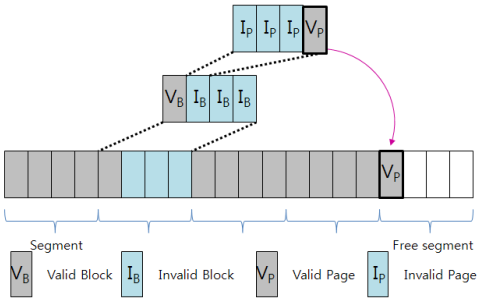
```
Select the victim segment to reclaim

If ( segment_state == old_used)
     erase (segment);
else {  /*segment_state == new_used
     /* for all valid slots in the segment */
          copy to free segment();
     erase (segment);
}
```

가비지 수집 과정을 (그림 5)에서 보여주고 있다.

4. 성능 평가

(그림 6)은 스왑 영역의 세그먼트 크기에 따라 소

요되는 기동 시간을 보여주고 있다. 이는 초기 스왑 영역이 시작될 때 세그먼트를 많이 가질수록 많은 기동 시간이 소요됨을 보여준다.



(그림 5) 가비지 수집 과정

다음 수식은 제안한 스왑 시스템에서 기동 시간을 계산하는 방법이며, <표 3>은 수식에서 사용하는 심볼에 대한 설명을 보여준다.

$$t_{start\_up\_time} = t_{page\_read} * N$$

$$N = \frac{S_{swap\_area}}{S_{block}}$$

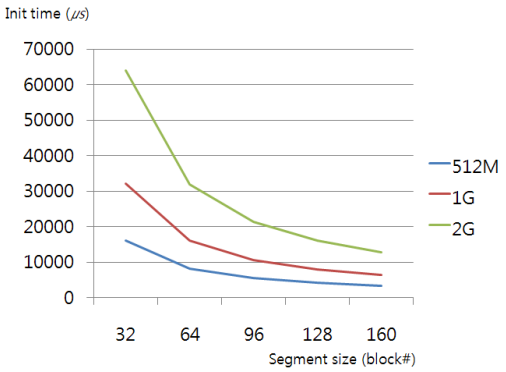
(표 3) 용어

용어	의미
$T_{start\_up\_time}$	스왑 기동시간
T_{page_read}	페이지를 읽는데 소요되는 시간 (125㎲)
N	세그먼트 개수
S_{swap_area}	스왑 영역의 크기 (512M, 1G, 2G)
$S_{segment}$	세그먼트의 크기

(그림 3)과 (그림 6)에서 볼 수 있듯이 새로운 스왑 시스템은 기동 시간이 크게 줄어든 것을 확인할 수 있다. 예를 들어 512MB 스왑 영역을 사용 할 경우 본래의 기동 시간은 6144ms 에 비해 제안된 기동 기법의 기동 시간은 4000 ㎲(=4ms)로 시간이 줄었고, 1GB 스왑 영역을 사용할 경우 본래의 기동 시간은 12288ms 에 비해 제안된 기동 기법의 기동 시간은 8000(=8ms)로 시간이 줄었고, 2GB 스왑 영역을 사용할 경우 본래의 기동시간은 24576ms 에 비해 제안된 기동 기법의 기동 시간은 16000 ㎲(=16ms)로 줄어들었다.

그러나, 제안한 스왑 시스템에서 기동할 때 세그먼트의 삭제가 요구되면 추가의 기동 시간이 소요될 수 있다. 예를 들어, 세그먼트의 크기가 128 블록인 경우 128*1.5ms(=약 192ms)의 시간이 더 소요된다.

5. 결론



(그림 6) 제안한 스왑 시스템의 기동 시간

본 논문에서 제안한 스왑 시스템에서는 기동할 때 모든 무효(invalid) 데이터를 삭제하지 않고 일부만 삭제함으로서 기동 시간을 크게 줄일 수 있다. 향후에는 기동할 때 및 가비지 수집할 때에 수행되는 재생 세그먼트의 선택 정책을 연구할 계획이다.

참고문헌

[1] Daniel P.Bovet & Marco Cresati, Understading the Linux Kernel, O'Reilly, 3rd Edition, 942p, 2005.

[2] Daniel P.Bovet & Marco Cresati; 박장수, 리눅스 커널의 이해, O'Reilly; 한빛미디어, 개정 3 판, 864p, 2006.

[3] M. Chiang and R. Chang, “Cleaning Policies in Mobile Computers Using Flash Memory,” *Journal of Systems and Software*, Vol.48, No.3, pp. 213-231, 1999.

[4] M. Chiang, P. Lee, and R. Chang, “Using Data Clustering to Improve Cleaning Performance for Flash Memory,” *Software Practice and Experience*, Vol.29, No.3, pp. 267-290, 1999.

[5] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," *International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994.

[6] Yaffs (yet another flash filing system), http://www.aleph1.co.uk/yaffs/

[7] D. Woodhouse, "Jffs: The Journalling Flash File System," in *Proceedings of the Ottawa Linux Symposium*, 2001.

[8] A. Kawaguchi, S. Nishioka, and H. Motoda, "Flash Memory Based File System," in *Proceedings of USENIX95*, pp. 155-164, 1995.

[9] 삼성전자, 256m x 8bit / 128 x 16bit nand flash memory. http://www.samsungelectronics.com

[10] Li-Pin Chang, "On Efficient Wear-Leveling for Large-Scale Flash Memory Storage Systems," in *Proceedings of 22nd ACM Symposium on Applied Computing*, 2007

[11] S. Koh, Y. Ryu, et al., "A New Linux Swap System for Flash Memory Storage Devices," in *Proceedings of 3rd International Workshop on Data Storage Devices and Systems*, Jun, 2008.