리눅스 상에서 플래시 메모리를 위한 효율적인 페이지 프레임 회수 기법

김수영 ° 이준원
한국과학기술원 전산학과
{sykim°, joon}@camars.kaist.ac.kr

Efficient Page Frame Reclaiming Mechanism for Flash Memory in Linux

Sooyoung Kim° Joonwon Lee
Dept. of Computer Science, Korea Advanced Institute of Science Technology

요 약

플래시 메모리는 롬(ROM)의 특성과 램(RAM)의 특성, 저전력 등의 이점을 바탕으로 임베디드 시스템의 저장 장치로 많이 사용되고 있다. 그러나 특성상 읽기와 쓰기 연산의 속도가 많이 다르고, 쓰기 연산을 한번 수행한 부분에 다시 쓰기 연산을 하기 위해서는 먼저 지우기 연산을 수행해야 하고, 지울 수 있는 회수도 제한되어 있는 단점을 가지고 있다. 따라서 본 논문에서는 이런 특성들을 고려하여 저장 장치로서 플래시 메모리를 사용할 때 운영체제에서 최적화할 수 있는 부분 중 가상 메모리 시스템의 페이지 프레임 회수 기법을 최적화하여 쓰기와 지우기 연산의 수를 줄일 수 있는 방법을 제시한다.

1. 서론

플래시(Flash) 메모리는 전원이 나가도 내용이 지워지지 않는 롬(ROM)의 특성과, 내용을 수정할 수 있는 램(RAM)의 특성을 동시에 가지고 있고, 크기나 전력 소모 면에서 이점이 있기 때문에 임베디드 시스템의 저장 장치로 많이 사용되고 있다. 특히 NAND 플래시 메모리는 NOR 플래시 메모리에 비해 높은 집적도와 낮은 단가로 인해 대용량 데이터를 저장하기 위한 장치로 주목 받고 있다.

운영체제의 관점에서 플래시 메모리를 데이터 저장 장치로 사용하기 위해 사용하는 대표적인 방법은 FTL(Flash Translation Layer)이라 불리는 모듈을 저장 장치에 내장하여 플래시 메모리의 물리적 블록과 운영 체제가 인식하는 논리적 블록에 대한 맵핑 정보를 관리한다. 이와 같은 방법은 플래시 메모리를 위한 별도의 드라이버 없이, 디스크를 위해 개발된 기존의 블록 디바이스 드라이버를 통해 플래시 메모리의 사용이 가능하여 기존 파일 시스템을 수정 없이 사용할 수 있으므로, 호환성 측면에서 이점을 가진다.

그러나 이러한 방법은 기존 디스크 드라이버와의 호환성을 위해 플래시 메모리의 특성을 운영체제에 숨기게 된다. 따라서 플래시 저장 장치가 디스크와는 매우 다른 특성을 보임에도 불구하고 플래시 메모리의 특성을 운영체제가 충분히 활용할 수 없는 문제점이 발생하게 된다. 즉, 플래시 메모리가 저장 장치로 사용되기 위해서는 플래시 메모리의 특성에 맞게 운영체제를 최적화시켜야 한다. 본 논문에서는 운영체제의 가상 메모리 시스템을 플래시 메모리에 맞게 최적화하여 운영 체제의 성능을 높일 수 있는 방법 중 가상 메모리 시스템의 페이지 프레임 회수 기법(Page Frame Reclaiming Mechanism)을 플래시 메모리의 특성에 맞게 보완한 알고리즘과 이를 리눅스에서 구현한 내용을 소개한다.

2 장에서는 리눅스 커널(Kernel)에서 현재 사용하고 있는 페이지 프레임 회수 기법과 새로 고안한 기법에 대해서 설명하고, 3 장에서는 실제 리눅스 커널에서 어떻게 구현했는지를 설명한다. 4 장에서는 실험 환경과 결과에 대해서 설명하고, 마지막 장에서 결론을 내린다.

2. 페이지 프레임 회수 기법

운영체제의 가상 메모리 시스템은 사용 가능한 메모리 공간이 부족할 경우 앞으로 사용되지 않을 가능성이 높은 페이지 프레임을 스왑 장치로 옮겨 메모리 공간을 확보하는 기능을 수행한다. 이때 어떤 페이지 프레임을 스왑 장치로 옮길지를 결정하기 위해 사용하는 기법이 페이지 프레임 회수 기법이다.

본 장에서는 현재 리눅스 커널이 어떤 방법을 사용하고 있는지 설명하고 (2 단계 큐를 이용한 LRU(Least Recently Used) 기법[1]), 다음으로 본 논문에서 제안하는 플래시 메모리에 맞게 수정한 프레임 회수 기법을 설명한다.

본 연구는 정보통신부 ITRC 프로그램의 지원으로 이루어졌습니다.

2.1. 리눅스 LRU

리눅스 커널은 LRU 리스트를 액티브 리스트(active list)와 인액티브 리스트(inactive list)로 구분하여 관리하고 이 두 리스트 간의 페이지 프레임 이동을 위해서 레퍼런스 비트(reference bit)와 액티브 비트(active bit)를 이용한다. 그림 1은 리눅스 LRU 에서의 페이지 프레임의 상태 전이도을 보여준다. 회수할 페이지 프레임이 인액티브 리스트에만 포함되어 있는 것을 보여준다.

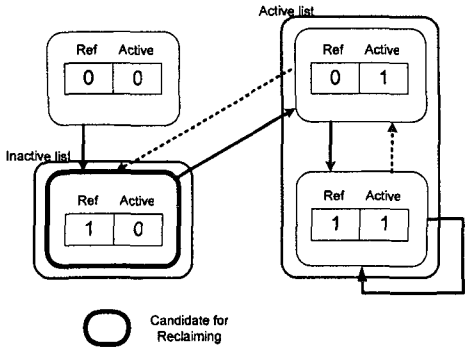

그림 1. LRU 에서의 페이지 프레임의 상태 전이도

2.2. LRU with Direction (LRUD)

기존의 LRU 기법은 앞 절에서 설명한 바와 같이 두 개의 요소만을 고려하고 인액티브 리스트에서만 회수할 페이지 프레임을 선택한다. 그러나 액티브 리스트에는 접근될 가능성이 낮아지는 페이지 프레임(레퍼런스 비트와 액티브 비트가 모두 1 인 상태에서 레퍼런스 비트가 0 이고 액티브 비트가 1 인 상태로 바뀐 페이지 프레임)이 포함되어 있고 인액티브 리스트에는 접근될 가능성이 높아지는 페이지 프레임(레퍼런스 비트와 액티브 비트가 모두 0 인 상태에서 레퍼런스 비트가 1 이고 액티브 비트가 0 인 상태로 바뀐 페이지 프레임)이 포함되어 있다. 회수할 페이지 프레임을 선택할 때, 인액티브 리스트에 포함되어 있는 접근될 가능성이 높아지는 페이지 프레임보다 액티브 리스트에 포함되어 있는 접근될 가능성이 낮아지는 페이지 프레임을 선택하는 것이 보다 좋을 수 있다. 이러한 점을 고려하여 페이지가 액티브 리스트와 인액티브 리스트로 옮겨가는 방향성을 새로운 요소로 추가하였다. 이 요소를 방향(Direction)이라 부르고, 방향은 위(Up)나 아래(Down)의 상태를 가진다. 따라서 회수할 페이지 프레임을 선택할 때 인액티브 리스트에서만 고려하는 기존 방법과 달리 인액티브 리스트와 액티브 리스트에 있는 페이지 프레임들 중 다음과 같은 우선 순위를 고려하여 선택한다.

① 변경 X, 인액티브 리스트, 아래 상태인 페이지 프레임
② 변경 X, 인액티브 리스트, 위 상태인 페이지 프레임
③ 변경 X, 액티브 리스트, 아래 상태인 페이지 프레임
④ 변경 O, 인액티브 리스트, 아래 상태인 페이지 프레임
⑤ 변경 O, 인액티브 리스트, 위 상태인 페이지 프레임
(변경 X = 내용이 변경되지 않음, 변경 O = 내용이 변경됨)

액티브 리스트에 있는 변경되지 않고 아래 상태인 페이지 프레임들을 인액티브 리스트에 있는 변경된 페이지 프레임들보다 먼저 검색함으로써 쓰기 연산의 수를 최대한 줄일 수 있다.

그림 2는 LRUD 에서의 페이지 프레임의 상태 전이도와 회수할 페이지 프레임이 인액티브 리스트에만 있는 것이 아니라 액티브 리스트에도 포함되어 있는 것을 보여준다.

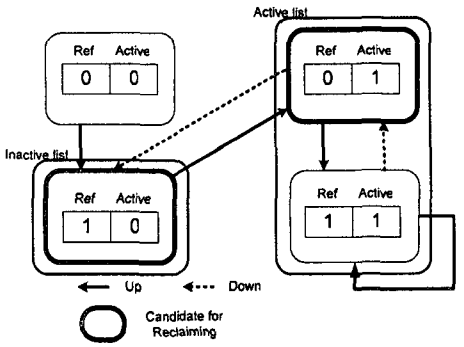

그림 2. LRUD 에서의 페이지 프레임의 상태 전이도

3. 구현

앞 장에서 제안한 방식대로 페이지 프레임 회수 기법을 수정하면 기존 리눅스 커널의 많은 부분이 수정되어야 하고 필요한 정보를 모두 따로 관리해야 하므로 시스템 전체 성능이 저하될 수 도 있다. 따라서 가능한 리눅스 커널의 수정을 최소화하기 위해 제안된 5 가지 종류의 대상 페이지 프레임들을 크게 두 개의 그룹((①,②), (③,④,⑤))으로 구분하였다. 회수할 페이지 프레임은 인액티브 리스트의 가장 오른쪽부터 왼쪽으로 검색하면서 회수할지를 결정한다. 회수할 페이지 프레임을 검색하는 방식은 기존의 방식을 그대로 사용하고, 인액티브 리스트를 채우는 방식을 기존 방식과는 달리 우리의 아이디어가 최대한 반영되도록 수정하였다.

- 우선 순위 ① 항목에 해당하는 페이지 프레임의 경우 기존의 방식에서는 인액티브 리스트의 가장 왼쪽에 넣는데, 이와는 달리 인액티브 리스트의 가장 오른쪽에 넣는다.
- 우선 순위 ② 항목에 해당하는 페이지 프레임의 경우 기존의 방식에서는 액티브 리스트에 넣는데, 이와는 달리 인액티브 리스트의 가장 오른쪽에 넣는다.
- 우선 순위 ③, ④, ⑤ 항목에 해당하는 페이지 프레임의 경우 기존의 방식과 같이 인액티브 리스트의 가장 왼쪽에 넣는다.

이렇게 함으로써 (①,②) 항목에 해당하는 페이지 프레임은 모두 인액티브 리스트의 오른쪽에 위치하게 되고 (③,④,⑤) 항목에 해당하는 페이지 프레임은 모두 왼쪽에 위치하게 된다. 물론 그룹 내에서의 우선 순위는 섞이게 되지만 이와 같은 방식으로 구현함으로써 리눅스 커널의 수정은 최소화하면서 LRUD 를 구현할 수 있다.

4. 실험

이 장에서는 제안한 페이지 프레임 회수 기법에 대한 실험 환경과 실험 결과에 대해서 설명한다.

4.1. 실험 환경

실험 환경은 표 1과 같다.

표 1. 실험 환경

구분	내용
CPU	Intel Pentium 4 2.4GHz
운영체제	Linux 2.4.27
메모리	32 MB
스왑장치	32 MB (mtdram, ftl 을 이용)

실험에서는 실제 플래시 메모리를 사용하지 않고, 리눅스에서 플래시 메모리를 사용하기 위해 제공되는 MTD(Memory Technology Device)[2]를 이용하였다. 그리고 flash_erase 와 ftl_format 과 같은 프로그램은 MTD 공식 사이트에서 최신 버전을 다운받아 사용하였다.

4.2. 실험 방법 및 결과

실험은 아래의 두 가지 경우로 진행하였다.

- 콘솔에서 리눅스 커널을 세 번 컴파일한 후에 읽기, 쓰기, 지우기 연산의 수를 비교
- X 윈도우상에서 아크로벳 리더(Acrobat Reader)를 실행시켜 큰 파일을 열고, 모질라(Mozilla)를 실행한 후에 읽기, 쓰기, 지우기 연산의 수를 비교

기존 기법과 새로운 기법의 실험 결과를 비교하면 그림 3과 그림 4에 나타내었다. 그림 3과 그림 4는 기존 기법의 연산 수를 기준으로 정규화하여 나타낸 것이다.

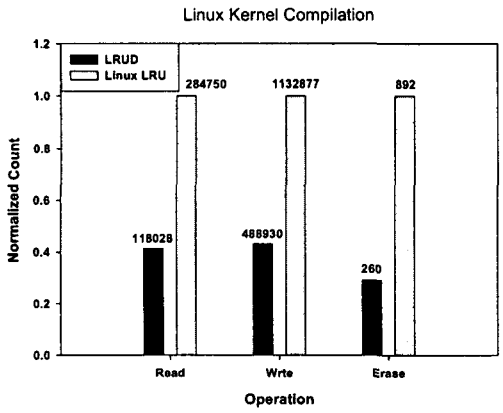

그림 3. 읽기, 쓰기, 지우기 연산 수 비교
(Linux Kernel Compilation)

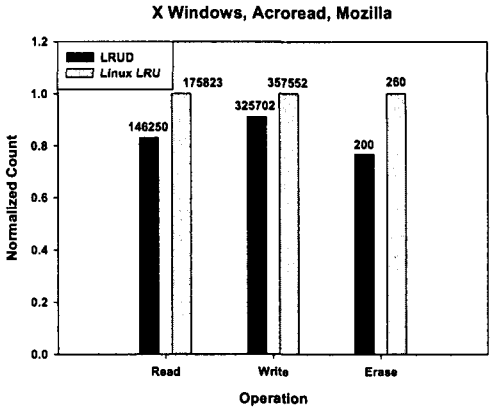

그림 4. 읽기, 쓰기, 지우기 연산 수 비교
(X Windows, Acroread, Mozilla)

그림에서 보는 바와 같이 LRUD 기법의 성능이 리눅스 LRU 기법의 성능보다 향상된 것을 보여주고 있다.

5. 결론

운영체제가 저장 장치로써 플래시 메모리가 사용된다는 것을 알고 그 특성에 맞게 동작한다면 플래시 메모리의 특성을 충분히 활용할 수 있다. 이런 점을 착안하여 스왑 장치로써 플래시 메모리가 사용되는 경우에 초점을 맞추어 가상 메모리 시스템에 새로운 아이디어를 추가하고, 기존의 기법과 성능을 비교하였다.

결과에서 보듯이 새로운 기법이 기존 기법보다 향상된 성능을 보여주고 있다. 그러나 스왑 장치를 사용하는 두 가지 경우만을 실험했기 때문에 보다 많은 경우에 대해 어떤 결과가 나오는지 확인해 볼 필요가 있다. 또한 대용량 저장 장치로 주목 받고 있는 NAND 플래시 메모리를 위한 FTL 인 nftl 을 사용하지 못하고 NOR 플래시 메모리를 위한 ftl 을 사용한 점이 아쉬운 점이다. 하지만 두 방식의 차이가 논리적인 면에서는 미미하므로 결과는 크게 다르지 않을 것이다.

6. 참고문헌

[1] Daniel P. Bovet, Marco Cesati, " Understanding of the Linux Kernel, 2nd Edition" , p561, 2002

[2] David Woodhouse, " Memory Technology Device (MTD) Subsystem for Linux" , http://www.linux-mtd.infradead.org