Journal of Korea Multimedia Society Vol. 10, No. 5, May 2007(pp. 583-593)

NAND 플래시 메모리 저장장치를 위한 요구 페이징 기법 연구

유윤석[†], 류연승[††]

요 약

본 논문에서는 플래시 메모리 기반의 가상 메모리 시스템에서 페이지 부재를 처리하는 과정에 있어서 플래시 메모리에 대한 삭제연산을 줄여 시스템에서의 전력 소모를 줄일 수 있고 플래시 메모리를 균등하게 사용함으로써 플래시 메모리의 마모도 평준화 정도를 좋게 할 수 있는 CFLRU/C, CFLRU/E, DL-CFLRU/E 페이지 교체 알고리즘을 연구하였다. 제안한 기법은 메인 메모리의 페이지를 클린 페이지와 더티 페이지로 구분하고 가장 오랫동안 사용되지 않았던 페이지들 중에서 클린 페이지를 빅팀으로 선택한다. 이때, 클린 페이지가 없다면 CFLRU/C 기법은 정해진 윈도우 내에서 참조 횟수가 가장 적은 더티 페이지를 빅팀으로 선택하고, CFLRU/E 기법은 페이지가 속한 블록의 삭제 연산 횟수가 적은 더티 페이지를 빅팀으로 선택한다. DL-CFLRU/E 기법은 클린 페이지 리스트와 더티 페이지 리스트를 따로 관리하며 페이지 부재가 발생할 때 우선 클린 페이지 리스트에서 클린 페이지를 선택하며, 클린 페이지 리스트가 빈 경우, 더티 페이지 리스트에서 블록 삭제 연산 횟수가 적은 페이지를 선택한다. 본 논문에서는 시뮬레이션을 통해서 제안한 기법이 기존 기법들(LRU, CFLRU)보다 플래시 메모리의 삭제 연산을 줄일 수 있었고, 마모도 평준화 정도를 향상시킬 수 있음을 보였다.

A Study on Demand Paging For NAND Flash Memory Storages

Yoon-suk Yoo[†], Yeonseung Ryu[††]

ABSTRACT

We study the page replacement algorithms for demand paging, called CFLRU/C, CFLRU/E and DL-CFLRU/E, that reduce the number of erase operations and improve the wear-leveling degree of flash memory. Under the CFLRU/C and CFLRU/E algorithms, the victim page is the least recently used clean page within the pre-specified window. However, when there is not any clean page within the window, the CFLRU/C evicts the dirty page with the lowest frequency while the CFLRU/E evicts the dirty page with the highest number of erase operations. The DL-CFLRU/E algorithm maintains two page lists called the clean page list and the dirty page list, and first finds the page within the clean page list when it selects a victim. However, when it can not find any clean page within the clean page list, it evicts the dirty page with the highest number of erase operations within the window of the dirty page list. In this thesis, we show through simulation that the proposed schemes reduce the number of erase operations and improve the wear-leveling than the existing schemes like LRU.

Key words: Flash Memory(플래시메모리), Virtual Memory System(가상 메모리 시스템), LRU, Demand Paging(요구페이징)

※ 교신저자(Corresponding Author) : 류연승, 주소 : 경기도 용인시 처인구 남동 명지대학교(449-728), 전화 : 031)330-6781, FAX : 031)336-6476, E-mail : ysryu@mju.ac.kr
접수일 : 2007년 2월 28일, 완료일 : 2007년 3월 19일
[†] 명지대학교 컴퓨터소프트웨어학과

[††] 종신회원, 명지대학교 컴퓨터소프트웨어학과
(E-mail : swish90@mju.ac.kr)
※ 본 연구는 한국학술진흥재단 젊은과학자연구활동지원사업 (R08-2004-000-10391-0)의 지원으로 수행되었음.

1. 서 론

최근 임베디드 시스템의 수요가 급증하고 있는 가운데 상대적으로 제한적인 하드웨어 자원과 적은 전력으로 사용해야 하는 임베디드 시스템의 특성을 뒷받침하기 위해서 플래시 메모리를 저장 매체로 사용하는 양이 점차 늘어나고 있다[1].

플래시 메모리(Flash memory)는 비휘발성 메모리로 접근 속도가 빠르고 저전력으로 운용되며 크기가 매우 작고 충격에 강하다[2,3]. 이런 플래시 메모리는 ROM과 RAM의 중간 단계에 있는 메모리이다. 평소에는 ROM으로서의 역할을 하지만 필요할 때 내부의 데이터를 수정이 가능하다[4]. 플래시 메모리는 그림 1과 같이 내부방식에 따라 NOR형과 NAND형으로 나눠진다. NOR형이란 셀이 병렬로 연결된 방식이고, NAND형이란 셀이 직렬로 연결된 구조를 가지고 있다. NAND형은 NOR형에 비해 제조단가가 싸고, 대용량이 가능하다는 장점이 있는 반면에, NOR형은 NAND형에 비해 데이터의 접근 시간이 짧고 데이터의 안정성이 우수한 장점을 가지고 있다.

플래시 메모리는 고정된 개수의 블록으로 구성된다. 또한 블록은 고정된 개수의 페이지로 구성된다. 플래시 메모리는 다음과 같은 특성을 갖는다. 읽기/쓰기 연산을 할 때는 페이지 단위로 이루어지며 삭제 연산을 할 때는 블록 단위로 이루어지는 것이다. 즉 연산 단위가 다른 것이다. 그래서 플래시 메모리는 특이한 연산 방법을 갖는다. 페이지에 쓰기 연산을 하기 위해서는 해당 블록을 먼저 삭제를 해야 하는 선 지움 후 쓰기 연산이 이행되는 것이다. 표 1은 플래시 메모리의 연산 시간을 보여준다. 표에서 볼 수 있듯이 쓰기/읽기 시간보다 삭제시간이 최고 7배 이상 느린 것을 볼 수 있다. 그래서 플래시 메모리를 고려한 시스템에서는 되도록 삭제 연산을 지양하는 것이 시스템 성능을 높이는 방법이라 할 수 있다. 또한 플래시 메모리는 삭제연산의 횟수에 제한이 있다. 삭제연산 횟수는 보통 백만 번으로 규정한다. 플래시 메모리의 메카니즘상, 읽고 쓸 때마다 조금씩 절연파괴가 진행된다[5]. 이러한 것들이 조금씩 쌓여서 절연층이 다 깨지면 더 이상 전자상태를 저장할 수가 없고, 더 이상 메모리를 못 쓰게 되는 것이다. 하지만 플래시 메모리 저장장치 기반의 가상메모리 시스템에서 요구 페이징에 대한 기법에 대한 연구는 거의 없다. 최근에 CFLRU(Clean First LRU) 교체 알고리즘아 제안되었으나 플래시 메모리의 삭제연산 횟수가 많고 마모도 균등화에 대한 문제가 여전히 남아 있다.

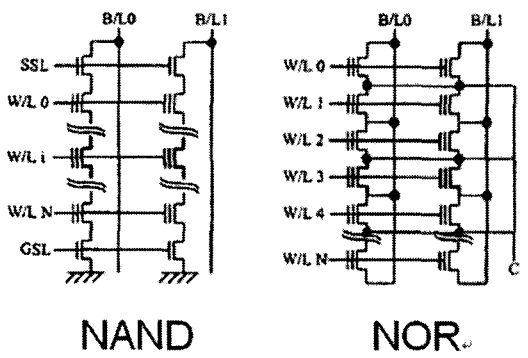


그림 1. 플래시 메모리 내부 구조

본 논문에서는 플래시 메모리에서 사용되는 LRU 기법에 대한 문제점을 논하고 마모도 균등화와 삭제연산 횟수에 대한 문제점을 개선하기 위하여 제안된 CFLRU[6] 기법에 대해 알아보고 문제점을 논한다. CFLRU 기법을 기반으로 버퍼에서의 페이지 참조횟수나 해당 플래시 메모리에서의 삭제연산 횟수를 고려해서 수정한 CFLRU/C[7,8], CFLRU/E[9] 기법에 대해 알아본다. 그리고 클린 LRU 리스트와 더티 LRU 리스트, 즉 두 개의 LRU 리스트를 버퍼에 유지하고 페이지 부재인 경우에 빅팀(Victim)을 먼저 클린 LRU 리스트에서 찾게 함으로써 플래시 메모리에서의 삭제연산을 최소화 하는 DL-CFLRU/E (Double List CFLRU/E) 기법에 대해 설명한다. 이 기법은 보다 좋은 마모도 균등화와 플래시 메모리에서의 낮은 삭제연산 횟수를 갖는다. 요구 페이징 기법에 대한 성능평가는 15가지의 상황에 대한 워크로드를 만들어 실험하였다.

표 1. 플래시 메모리 접근 시간 비교

연산 / 매체	접근 시간		
	읽기	쓰기	삭제
NOR Flash[1)]	14.4 μs	3.53 ms	1.2s (128KB)
NAND Flash[2)]	35.9 μs	226 μs	2ms (16KB)

1) NOR Flash : Intel 28F128J3A-150 [10]
2) NAND Flash : Samsung K9F5608U0M [11]

본 논문의 구성은 다음과 같다. 2장에서는 가상메모리 시스템의 개요와 요구 페이징 기법에 대해 관련 연구를 살펴보고, 3장에서는 플래시 메모리의 특성을 고려한 요구 페이징 기법에 대해 제안한다. 4장에서는 실험 환경 및 워크로드에 대한 개요를 설명한다. 5장에서는 실험 결과에 대한 성능 평가를 하고, 6장에서 결론과 향후 연구 내용을 기술한다.

2. 관련 연구

2.1 가상 메모리

최근의 임베디드 시스템은 크기가 큰 프로그램을 실행시켜야 하고 여러 개의 프로그램을 동시에 실행시켜야 한다. 다중 프로그래밍을 실현하기 위해서는 메모리에 많은 프로세스들을 동시에 유지해야 한다. 가상 메모리는 프로세스 전체가 메모리 내에 올라오지 않더라도 실행이 가능한 기법이다. 일반적으로 가상 메모리 기법은 MMU(memory management unit)에 의해 지원된다[12,13].

프로세서가 생성하는 주소를 가상주소(virtual address)라고 부른다. 가상주소는 메모리 버스에 바로 가지 않고 그림 2에서처럼 가상주소(virtual address)를 물리주소(physical address)에 연결시키는 MMU로 간다. 그림 3은 가상 메모리 시스템을 나타낸다.

MMU는 가상 메모리에서 물리 메모리로 변환에 대한 모든 정보를 나타낸 테이블을 주 메모리 안에 저장한다. 이 테이블을 페이지 테이블이라고 한다. 페이지 테이블 엔트리는 가상 페이지에 대한 정보들, 즉 가상 페이지를 물리 페이지 프레임으로 변환하기 위해 사용되는 물리 베이스 주소, 페이지에 할당되는 접근권한, 페이지를 위한 캐시 및 쓰기 버퍼 설정값

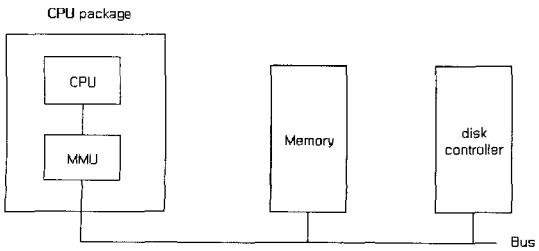


그림 2. MMU를 포함한 시스템

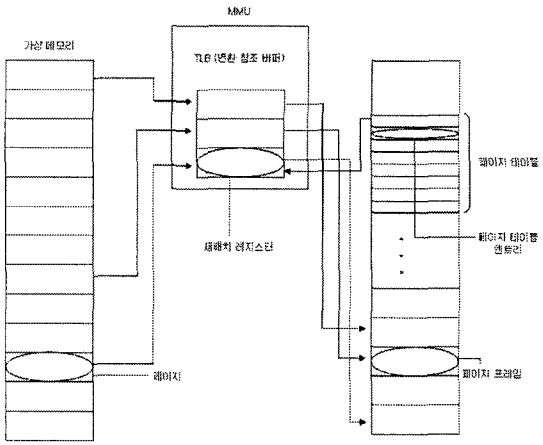


그림 3. 가상 메모리 시스템

등을 포함하고 있다. CPU는 참고할 가상주소를 MMU에 보낸다. MMU는 가상주소를 물리주소로 바꾼 후에 메모리로 보낸다. 가상주소와 물리 메모리 주소간의 관계는 그림 4와 같이 페이지 테이블을 통해 나타낸다.

그림 4는 16비트 주소를 생성할 수 있어 0부터 64KB까지 가상 메모리를 갖고 물리 메모리는 16KB를 가진 시스템이다. 프로세스의 크기가 64KB까지 쓰일 수 있지만 프로세스 전부가 메모리에 상주하기엔 메모리가 작다. 그래서 프로세스의 완전한 이미지는 디스크에 저장이 되고 지금 필요한 부분만 메모리에 올라가게 된다. 가상주소 공간은 페이지로 구성된다. 이 가상주소 공간이 물리 메모리에 대응되는 단위를 페이지 프레임이라고 한다. 여기서 페이지와 페이지 프레임은 같은 크기이다.

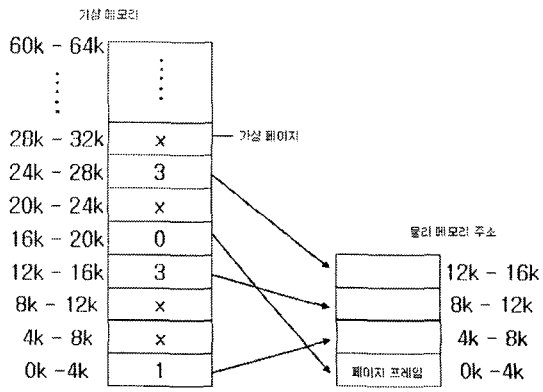


그림 4. 가상 메모리 주소와 물리 메모리 주소간의 관계

2.2 요구 페이징(Demand Paging)

요구 페이징은 프로세스의 필요한 페이지에 대한 요구(demand)가 있으면 주기억장치에 그 페이지를 적재시키는 가상 메모리 기법이다[13]. 이렇게 필요한 페이지만 메모리에 적재함으로써 메모리 공간 낭비와 시간 낭비를 줄일 수 있다. 필요한 페이지만 메모리에 올리기 위해서는 어느 페이지가 메모리에 올라와 있는지 구별 할 수 있어야 한다. 그래서 페이지 테이블에 유효/무효 비트를 표시한다. 메모리에 올라오는 페이지는 유효가 되며 현재 메모리에 없는 페이지는 무효가 된다. 그 대신 페이지가 저장되어 있는 저장장치 주소를 가리킨다. 그림 5는 프로세스가 실행될 때 필요한 페이지만을 메모리에 올린 상황을 나타낸다. 현재 사용되는 프로세스의 페이지는 논리 메모리에서의 A, E, F이다. 그래서 물리 메모리에도 A, E, F가 적재 되어 있고, 페이지 테이블에는 물리 메모리에 적재된 페이지에 대해서만 유효로 세트 되고 나머지는 무효로 세트된다. 만약 메모리에 올라와 있지 않은 페이지를 참조 할 경우 페이지 부재(page-fault)가 발생한다. 페이지 부재가 발생하면 먼저 물리 메모리에 자유 프레임을 찾게 된다.

자유 프레임을 찾고 나서 못 찾은 페이지를 저장장치에서 옮겨온다. 그리고 이 페이지가 메모리에 있다는 것을 알리기 위해 페이지 테이블을 갱신한다. 그림 6은 페이지 C를 참조할 때 페이지 부재가 생겨서 저장 장치에서 물리페이지의 자유 프레임에 옮겨 놓은 그림이다.

2.3 페이지 교체 알고리즘

페이지 부재를 처리할 때 물리 메모리에 자유 프레임이 없을 경우 그림 7과 같이 필요 없는 페이지를 저장장치로 스왑아웃하고 필요한 페이지를 저장장치로부터 스왑인하여 적재한다[13].

여러 페이지 교체 알고리즘이 존재 하지만 본 논문에서는 전통적인 시스템에서 좋은 성능을 보이는 LRU(Latest Recently Used) 알고리즘에 대해 알아본다. LRU 알고리즘은 과거에 가장 오랫동안 사용되지 않은 것을 선택하는 알고리즘이다. 페이지 부재 처리의 경우에 LRU 알고리즘을 사용할 경우 가장 오랫동안 사용되지 않은 페이지 프레임을 빅팀으로 한다. 일반적으로 LRU 알고리즘을 구현하기 위해서는 해당 프레임이 언제 사용되었는지를 알아야 한다. LRU 알고리즘은 리스트를 이용해서 페이지에 대한 사용 순서를 그림 8과 같이 항상 리스트로 유지한다.

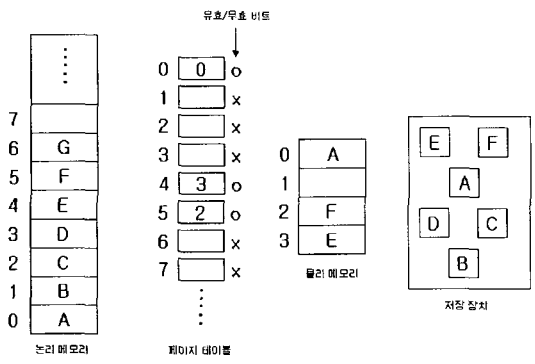


그림 5. 프로세스 논리 메모리에서의 요구 페이징

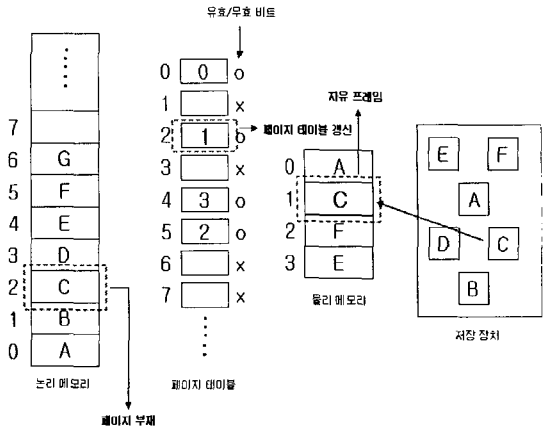


그림 6. 페이지 부재 처리 과정

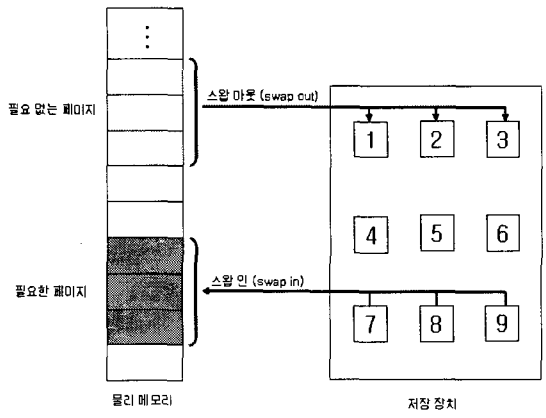


그림 7. 페이지 교체 과정

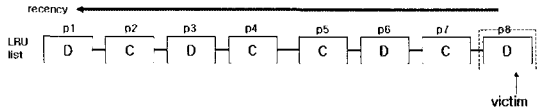


그림 8. LRU 데이터구조

그동안 하드 디스크를 가정하는 데이터 교체 알고리즘들이 연구 되어 왔다. 그러나 플래시 메모리가 저장장치로 사용될 경우 페이지 교체 알고리즘에 따라 시스템 성능이 좌우된다. 플래시 메모리는 표 1에서 볼 수 있듯이 삭제 연산할 때 시간이 가장 오래 걸린다. 그리고 에너지 소모량도 크다[6]. 최근 플래시 메모리의 특성을 고려하는 CFLRU 알고리즘이 제안되었다[6]. 그림 9는 CFLRU 알고리즘을 보여주고 있다.

CFLRU 알고리즘은 기존의 LRU 알고리즘처럼 페이지의 LRU 리스트를 만든다. 페이지 폴트가 발생하여 메모리에서 빅팀을 선택할 때 CFLRU는 플래시 메모리의 삭제 연산을 고려한다. 삭제 연산이 많으면 많을수록 연산시간과 전력 소모가 커지기 때문이다. CFLRU는 페이지의 LRU 리스트에서 가장 오랫동안 사용하지 않은 페이지에서부터 윈도우를 정해놓고 그 안에서 클린(Clean) 페이지를 찾는다. 여기서 클린 페이지는 페이지에 대해 오직 읽기연산만 수행된 페이지를 말한다. 플래시 메모리에서 단지 읽기 연산을 위해 스왑 인(swap in)된 페이지는 버퍼에서 스왑 아웃(swap out)될 때 플래시 메모리의 원래 페이지를 변경할 필요가 없다. 그래서 삭제연산이 필요 없게 된다. 만약 클린 페이지가 없을 경우에는 LRU 알고리즘과 같은 방법으로 윈도우내에 가장 오랫동안 참조되지 않은 더티(Dirty) 페이지를 빅팀으로 선택한다. 그림 9는 CFLRU의 데이터 구조와 빅팀 선택에 대한 그림이다. 그림 9에서는 윈도우를 4개의 페이지로 했고 윈도우 안에서 가장 오랫동안 참조하지 않은 p7 클린 페이지를 빅팀으로 선택하였다. CFLRU 알고리즘은 시스템의 전력 소모를 줄이기 위해서 고안된 알고리즘이다.

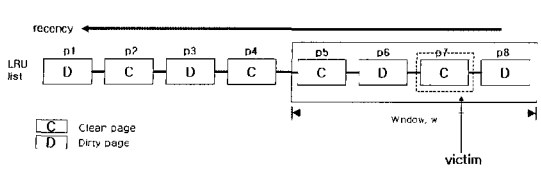


그림 9. CFLRU 데이터구조

3. 제안 기법

플래시 메모리의 삭제 연산을 보다 줄이기 위해서는 위도우 안에 예비 빅팀들에 대해 여러 가지 사항을 고려해야 한다. 본 논문에서는 기존의 CFLRU 알고리즘에서 윈도우에 속하는 빅팀들의 참조 횟수, 빅팀이 속해있는 플래시 메모리에서의 블록 삭제횟수를 고려한 알고리즘을 살펴보고 버퍼 안에서 2개의 리스트를 갖는 새로운 알고리즘에 대해 제안한다.

3.1 CFLRU/C (CFLRU/Count)

첫 번째 알고리즘은 윈도우 안에 예비 빅팀들의 참조 횟수를 고려하는 것이다. 중앙 처리 장치에서 요구하는 데이터와 명령어가 버퍼에서 참조된다면 그것을 히트라고 한다. CFLRU 알고리즘에서 윈도우 안에 빅팀들에 대한 참조 횟수를 가지고 페이지가 얼마나 집중적으로 참조되었는가에 관심을 갖고 가장 적게 사용되거나 집중적으로 참조되지 않은 페이지를 빅팀으로 선택할 수 있다.

이 알고리즘을 CFLRU/C라고 하겠다. 교체할 빅팀을 선택할 때 윈도우 안에 클린 페이지를 찾아 교체하는 것은 기존의 CFLRU와 같지만 클린 페이지가 없을 경우에는 LFU 알고리즘을 이용해서 윈도우 안에 더티 페이지 중에서 가장 참조 횟수가 적은 페이지를 교체 페이지로 선택하게 된다. 그림 10은 CFLRU/C 알고리즘을 나타낸 그림이다. 윈도우 안에 페이지 p5, p6, p7, p8은 모두 더티 페이지이다. 그래서 이 페이지들 중에서 참조 횟수가 가장 적은 p6 페이지가 빅팀이 된다. 참조 횟수는 버퍼에서 쓰기 연산할 경우에만 횟수를 증가 시킨다.

3.2 CFLRU/E (CFLRU/Erase)

두 번째 알고리즘은 윈도우 안에 예비 빅팀들이 속한 플래시 메모리에서의 블록 삭제 연산 횟수를 고려하는 것이다. 플래시 메모리는 블록마다 삭제연

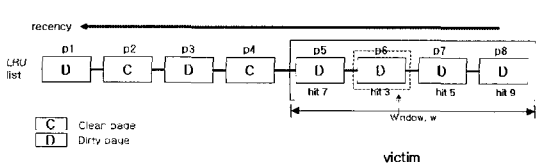


그림 10. CFLRU/C의 예

산 횟수가 정해져 있기 때문에 되도록 삭제를 지양하는 것이 좋다. 그리고 전체적으로 균등하게 삭제연산이 이루어지는 것도 플래시 메모리상에서의 마모도 평준화에 좋다. CFLRU/E 알고리즘은 윈도우 안에 예비 빅팀들 중에서 예비 빅팀들이 속한 플래시 메모리 상의 블록의 삭제연산 횟수를 비교 후에 빅팀을 추출하게 된다[7-9]. 그림 11은 CFLRU/E 알고리즘을 나타낸 그림이다. 윈도우 안에 페이지 p5, p6, p7, p8은 모두 더티 페이지이다. 윈도우 안에 있는 페이지 중에서 p6 페이지가 속한 플래시 메모리에서의 블록 삭제 연산횟수가 가장 적기 때문에 p6 페이지가 빅팀으로 추출된다.

3.3 DL-CFLRU/E (Double List CFLRU/E)

지금까지 살펴본 알고리즘들은 CFLRU에 기반하고 있으며 클린 페이지와 더티 페이지가 윈도우 내에 혼재되어 있다. 그러므로 윈도우 밖에 클린 페이지가 존재 하더라도 윈도우 안에 더티 페이지를 선택하게 되고 스왑 아웃할 때 플래시 메모리 삭제 연산이 수행되게 된다.

클린 페이지가 있음에도 불구하고 더티 페이지에 대한 삭제 연산을 하는 것은 불필요한 연산이다. 불필요한 연산을 줄이기 위해서는 버퍼 안에 클린 페이지를 빅팀으로 선택하는 것이 필요하다.

따라서, 본 절에서는 버퍼들을 2개의 리스트로 관리하는 기법을 제안하였다. 하나는 더티 페이지 리스트이고, 다른 하나는 클린·페이지 리스트이다. 그림 12는 DL-CFLRU/E를 나타낸 그림이다. 제안한 기법은 빅팀을 스왑 아웃할 때 클린 페이지가 있는 리스트를 먼저 검색한다. 클린 페이지 리스트에 클린 페이지가 있을 경우 ①과 같이 첫 번째 클린 페이지

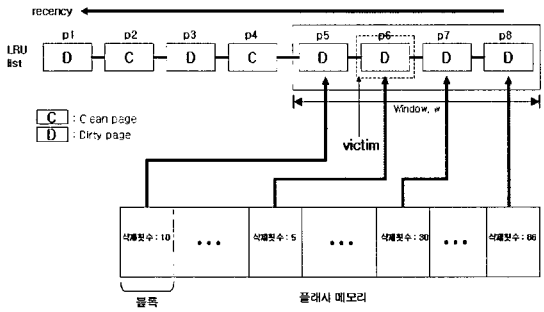


그림 11. CFLRU/E의 예

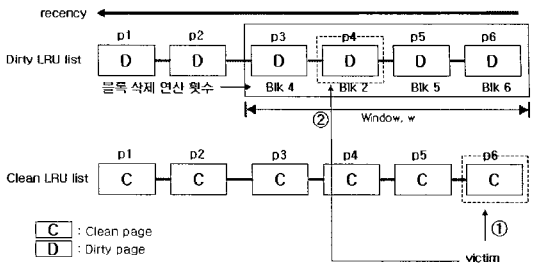


그림 12. DL-CFLRU/E의 예

를 교체하게 된다. 이 클린 페이지는 새로운 데이터를 위한 페이지로 할당되고 쓰기 연산인지 읽기 연산인지에 따라 더티 리스트 또는 클린 리스트에 가게 된다. 만약 클린 리스트에 페이지가 없을 경우에는 더티 리스트에서 CFLRU/E 알고리즘과 같이 윈도우 안에 가장 적은 블록 삭제 연산 횟수를 가진 페이지를 빅팀으로 선택한다. CFLRU/E를 사용하는 이유는 다음 장에서 설명하듯이 삭제 횟수에 대해 좋은 성능을 보이기 때문이다. 그림 13은 DL-CFLRU/E 알고리즘을 보이고 있다.

4. 실험환경 및 워크로드

4.1 실험환경

본 절에서는 제안한 알고리즘의 성능을 검증하기 위해서 구성한 실험환경에 대해 기술한다. 본 연구에서 가상 메모리 시스템을 시뮬레이터로 구현하였으며 가정한 시스템의 구성은 그림 14에 나타내었다.

```
DL-CFLRU/E()
{
    클린 LRU 리스트에 클린 페이지가 있으면,
        클린 LRU 리스트에서 가장 오랫동안 사용하
        지 않은 클린 페이지를 빅팀으로 한다.
    클린 LRU 리스트에 클린 페이지가 없으면,
        더티 LRU 리스트에서 윈도우 안에 있는 페이
        지 중에서 플래시 메모리에서의 해당 블록 삭
        제 연산 횟수가 가장 적은 페이지를 빅팀으로
        한다.
}
```

그림 13. DL-CFLRU/E 알고리즘

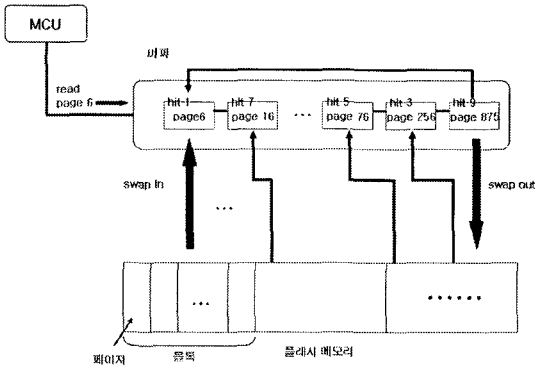


그림 14. 시스템 구성

플래시 메모리는 총 300개의 블록과 각각의 블록에는 32개의 페이지로서 구성되었다. 주 메모리 안에 버퍼는 개수를 변화시키면서 실험하였다.

본 실험에서는 버퍼 페이지 크기와 플래시 메모리 페이지 크기가 같다고 가정하고 실험하였다. MCU에서 페이지를 읽거나 쓰는 명령은 워크로드를 통해서 구현하였다. 워크로드는 다음 절에서 설명한다. 워크로드에 따라 읽기/쓰기 연산을 하면서 사용된 페이지가 가상의 버퍼에 저장된다. 쓰기연산중에 비어 있는 버퍼가 없고 페이지 부재가 발생하면 페이지 교체 알고리즘을 선택하게 된다.

본 논문에서는 페이지 교체 LRU 알고리즘과 플래시 메모리를 고려한 CFLRU 알고리즘과 본 논문에서 제안하는 CFLRU/C, CFLRU/E, DL-CFLRU/E 알고리즘에 대한 실험을 하고 각 알고리즘에 대해 플래시 메모리의 블록 삭제 횟수와 버퍼에서의 참조 횟수를 측정하였다.

4.2 워크로드

실험에서 사용한 워크로드는 인공적으로 만들었다. 일정 개수의 페이지를 메모리에 워크로드에 따라 접근 형태를 변경하면서 실험한다. 그림 15와 같이 접근 빈도 횟수를 90/10으로 하는 것은 플래시 메모리 영역의 10%에 총 읽기와 쓰기 연산의 90%를 집중적으로 실행하고 나머지 영역의 90%에 연산의 10%를 균등하게 실행한다. 이와 같이 90/10, 80/20, 70/30, 60/40, 50/50(uniform access) 유형별로 워크로드를 생성하였다. 읽기와 쓰기도 같은 방법을 사용하여 90/10, 50/50, 10/90으로 각각 접근 빈도 횟수에 맞춰서 워크로드를 생성하였다. 90/10 읽기/쓰기에

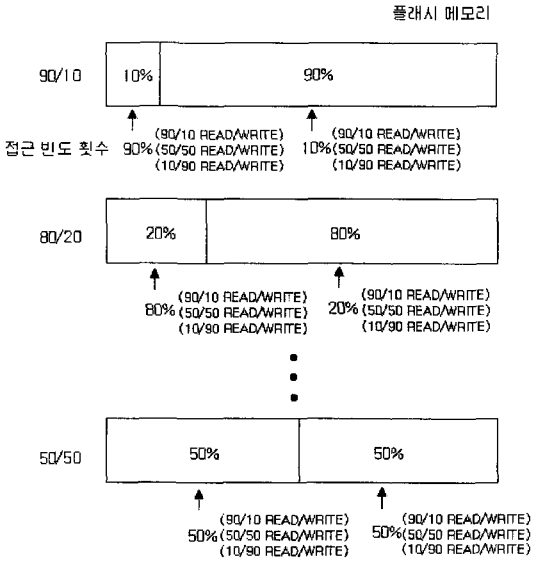


그림 15. 워크로드

대한 접근 빈도는 플래시 메모리에 대한 전체 연산중에서 90%는 읽기 연산을 나머지 10%는 쓰기 연산을 하는 것이다.

5. 실험 결과 및 성능 평가

그림 16에서 그림 18은 버퍼가 1000이고 윈도우가 500인 가상의 시스템에 100만 번 읽기 쓰기 연산을 수행한 성능 평가결과를 보여주고 있다. 그림 16은 Read 10 Write 90 인 경우이고, 그림 17은 Read 50 Write 50, 그림 18은 Read 90 Write 10 인 경우이다. 히트가 많게 되면 버퍼에서의 참조가 많아지며 페이지 교체 횟수가 줄어들게 되므로 플래시 메모리 삭제 연산 횟수가 줄어든다.

그림 16 (a)에서 CFLRU/C가 다른 알고리즘들보다 마모도 평준화 정도가 비교적 좋지 못한 것은 버퍼에서의 참조 횟수가 많을수록 해당 페이지가 버퍼에 오래 남아 있어 해당 블록의 삭제 연산이 거의 일어나지 않기 때문이다. 그래서 특정 블록에 삭제 연산이 집중되거나 집중되지 않기 때문에 마모도 평준화가 좋지 못하다.

그림 16에서는 Read 연산 횟수가 적기 때문에 클린 페이지가 적게 생성된다. 그러므로 각 알고리즘에 따른 성능이나 플래시 메모리에서의 삭제 연산 횟수에 차이가 크게 나타나지 않는다. 그러나, 그림 17과

그림 18에서는 읽기 연산 횟수가 쓰기 연산 횟수보다 많기 때문에 삭제 연산 횟수와 마모도 평준화 정도가 차이나는 것을 확인할 수 있다. 그림 19는 버퍼 크기에 따른 플래시 메모리에서의 삭제 연산 횟수를 나타낸다. 버퍼가 크면 참조 할 수 있는 페이지를 많이 적재 할 수 있기 때문에 플래시 메모리에서의 삭제 연산 횟수도 줄어드는 것을 볼 수 있다. 특히 접근 유형이 90/10과 같이 집중적인 경우에는 버퍼 크기가 클수록 좋은 성능을 보여준다. 버퍼의 크기가 크면 그만큼 클린 페이지를 많이 적재할 수 있기 때문에 LRU 알고리즘을 제외한 클린 페이지를 우선으로 빅팀으로 사용하는 알고리즘들의 성능이 좋아진다. 특히 DL-CFLRU/E 알고리즘이 클린 LRU 리스트를 따로 두고 있기 때문에 클린 페이지에 대한 활용도가 높다. 그러므로 버퍼 크기에 상관없이 다른 알고리즘에 비해 좋은 성능을 보이고 있다. 특히 버퍼 크기가 작고 접근 유형이 한 곳에 집중될 경우 더욱 좋은 성능을 나타낸다.

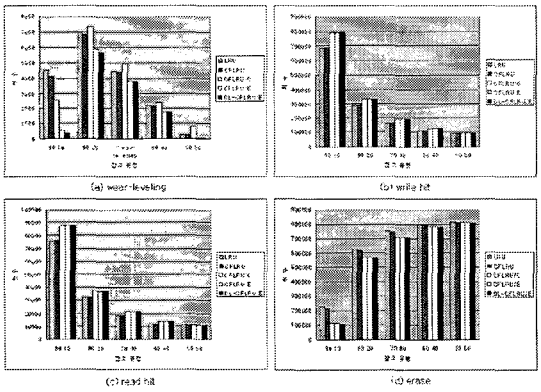


그림 16. read 10, write 90의 워크로드 실험 결과

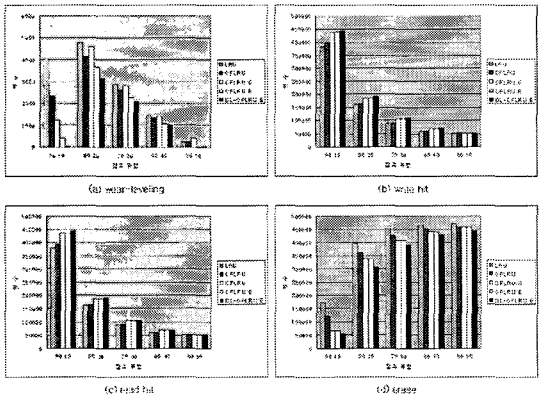


그림 17. read 50, write 50의 워크로드 실험 결과

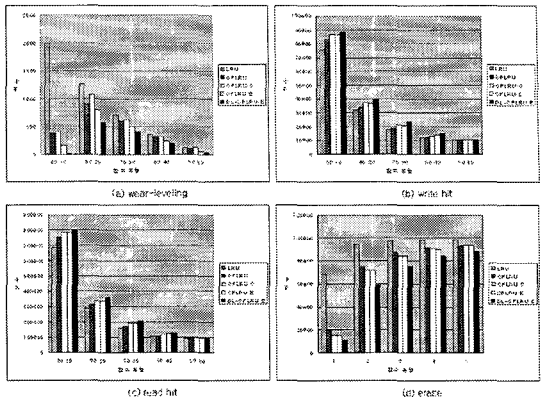


그림 18. read 90, write 10의 워크로드 실험 결과

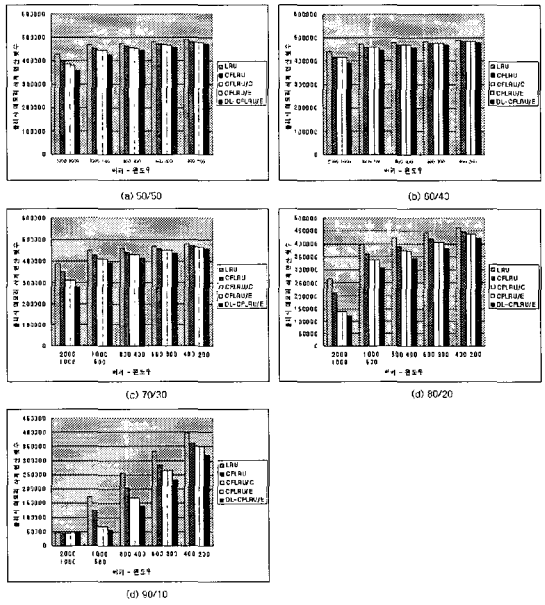


그림 19. 버퍼 크기 vs 삭제 연산 횟수

그림 20은 버퍼 크기가 1000인 시스템에서 윈도우 크기에 대한 플래시 메모리 삭제 연산 횟수와 마모도 평준화 정도를 나타내고 있다. 윈도우 크기가 작아지면 플래시 메모리에서 삭제 연산이 전체적으로 늘어나게 되고 플래시 메모리에서의 특정 부분에 삭제 연산이 늘어나게 되기 때문에 마모도 평준화 정도가 좋지 못하다. 그림 20의 (a)에서 90/10인 경우에는 윈도우 크기가 작을수록 삭제 연산 횟수는 적지만 마모도 평준화 정도는 좋지 않게 나타나게 된다. 이것은 윈도우 크기가 작을수록 플래시 메모리에서의 마모도 평준화에 고려되는 페이지 수가 더욱더 적어

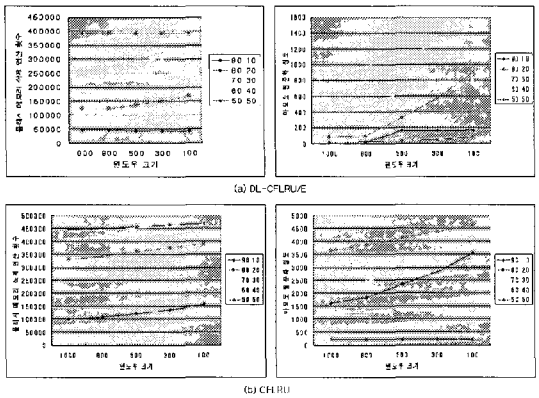


그림 20. 윈도우 크기 VS 삭제 연산 횟수, 마모도 평준화 정도

지기 때문이다. 그림 20의 (a)와 (b)에서 볼 수 있듯이 DL-CFLRU/E 알고리즘은 CFLRU 알고리즘과는 달리 윈도우 크기에 따라 성능이 크게 영향을 받지 않는다. DL-CFLRU/E 알고리즘에서는 클린 페이지가 있을 경우에 빅팀을 바로 클린 리스트에서 선택하기 때문에 CFLRU와 같이 윈도우에서 클린 페이지를 찾는 별도의 연산이 필요 없다. 그리고 윈도우 크기가 크면 빅팀을 선택하는 연산과정에서 윈도우만큼의 페이지를 모두 찾아야하는 오버헤드가 생기게 된다. 하지만 DL-CFLRU/E 알고리즘은 마모도 평준화에 영향을 미치지만 플래시 메모리에서 삭제 연산 횟수에 크게 영향을 미치지 않는다. 그러므로 DL-CFLRU/E 알고리즘이 CFLRU 알고리즘보다 빅팀을 찾는 오버 헤드가 작다.

그림 21은 LRU, CFLRU, DL-CFLRU/E 알고리즘에 대한 플래시 메모리에서의 마모도 평준화 정도를 비교한 것이다. 새로운 알고리즘이 가장 적은 삭제 연산 횟수를 보여주고 있다. 플래시 메모리에서의 어떤 부분에 대한 집중도가 클수록 DL-CFLRU/E 알고리즘 성능은 좋게 나타난다. 그리고 50/50에서는 DL-CFLRU/E 알고리즘이 모든 블록에 대해 일정한 삭제 연산을 횟수를 갖는 것을 볼 수 있다.

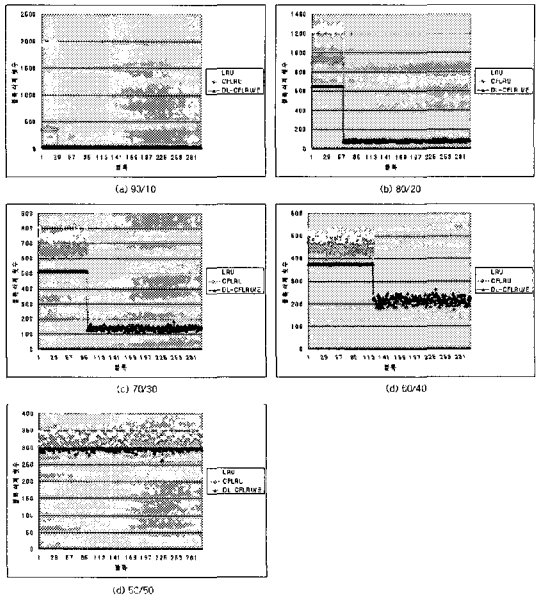


그림 21. 마모도 평준화의 실험 결과

5. 결 론

본 논문에서는 플래시 메모리를 기반으로 한 요구 페이징 기법에서 페이지 교체 알고리즘을 연구하고 CFLRU/C, CFLRU/E, DL-CFLRU/E 알고리즘을 제안하였다. CFLRU/C 알고리즘은 윈도우 안에 예비 빅팀들의 참조 횟수를 고려한 알고리즘으로 참조가 많은 페이지가 버퍼에 더 오래 있게 함으로써 참조 빈도가 많은 페이지에 대해 플래시 메모리 해당 블록의 삭제연산을 줄일 수 있다. CFLRU/E 알고리즘은 윈도우 안에 예비 빅팀들의 플래시 메모리에서의 해당블록에 삭제연산을 고려함으로써 플래시 메모리의 마모도 평준화 정도를 좋게 한다. DL-CFLRU/E 알고리즘은 버퍼내의 클린 페이지 리스트와 더티 페이지 리스트를 따로 가지고 페이지 부재가 발생할 때 클린 페이지 리스트에서 클린 페이지를 우선적으로 스왑아웃(swap out)하여 플래시 메모리의 삭제연산을 줄이고 클린 페이지가 없을 경우에는 더티 페이지 리스트에서 윈도우 내의 예비 빅팀들이 해당하는 플래시 메모리 블록의 삭제 횟수를 고려하여 플래시 메모리에서의 마모도 평준화 정도를 좋게 한다.

향후 연구로는 첫째, 오버헤드를 고려하는 알고리즘 연구가 필요하다. CFLRU/C, CFLRU/E, DL-CFLRU/E 알고리즘을 사용하기 위해서는 계수기, 블록당 삭제연산 횟수 등을 플래시 메모리의 블록이나 버퍼 페이지에 저장해야하는 오버헤드가 있는데 이를 최소화하는 연구가 필요하다. 둘째, 실험에 쓰인 워크로드는 인위적인 워크로드이기 때문에 향후에는 실제 워크로드에 대한 실험을 수행할 계획이다.

참 고 문 헌

[1] H.M. Yang, L.Z Han, Y.S. Yoo, D.S. Lim, and Y.S. Ryu, "Design of Multimedia File System on Flash Memory Storage," *in Proceedings of Korea Multimedia Society Fall Conference*, pp. 405-453, Oct. 2005.

[2] L.Z Han and Y.S. Ryu, "Performance Comparison of File Systems on Flash Disk and Hard Disk," *in Proceedings of Korea Multimedia Society Fall Conference*, Nov. 2004.

[3] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J.A. Tauber, "Storage Alternatives for Mobile Computers," *in Proceedings of the 1st Symposium on Operating System Design and Implementation*, pp. 25-37, Nov. 1994.

[4] Y. Ryu and K. Lee. "Improvement of space utilization in nand flash memory storages," *Lecture Notes in Computer Science*, pp. 766-775, Dec. 2005.

[5] Brian Dipert and Markus Levy, *Designing with Flash Memory*, Annabooks, 1993.

[6] Chanik Park, Jeong-Uk Kang, Seon-Yeong Park, and Jin-Soo Kim, "Energy-Aware Demand Paging on NAND Flash-based Embedded Storages," *in Proceedings of International Symposium on Low Power Electronics and Design*, pp. 338-343, Aug. 2004.

[7] 유윤석, 한룡철, 류연승, "플래시 메모리 기반 캐시 시스템에서 LRU 교체 알고리즘의 성능 평가," 모바일학회 2006 춘계학술 대회, Jun. 2006.

[8] 유윤석, 류연승, "플래시 메모리를 고려한 버퍼 교체 알고리즘의 성능 평가," 모바일학회 2006 추계학술 대회, Nov. 2006.

[9] 유윤석, 류연승, "플래시 메모리를 고려하는 버퍼 교체 알고리즘," 2006 명지 IT 포럼, Oct. 2006.

[10] Intel Corp. *Intel Flash memory data sheets*, 2003.

[11] Samsung Electronics. *NAND flash memory data sheets*, 2003.

[12] Andrew N. Sloss, Dominic Symes, and Chris Wright, *ARM System Developer's Guide*, ELSEVIER Inc, Amsterdam, 2005.

[13] Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, Amsterdam, 2001.

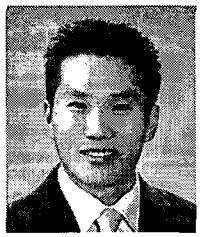
[14] L. P. Chang and T. W. Kuo, "A Real-time Garbage Collection Mechanism for Flash Memory Storage System of real-time Embedded Systems," *in Proceedings of The 8th International Conference on Real-Time Computing Systems and Applications*, pp. 837-863, Mar. 2002.

[15] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," *in Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 86-97, Oct. 1994.

[16] C.Park et al. "A low-cost memory architecture with NAND XIP for mobile embedded systems," *in Proceedings of CODES+ISSS*, pp. 138-143, Oct. 2003.

[17] H. Huang, P. Pillai, and K. G. Shin. "Design and Implementation of power-aware virtual memory," *in Proceedings of USENIX Annual Technical Conference*, pp. 57-70, Jun. 2003.

[18] R. van Riel. "Page replacement in Linux 2.4 memory management," in Proceedings of *USENIX annual Technical Conference*, pp. 165-172, Jun. 2001.



유 윤 석

2005년 2월 명지대학교 전자공학과 졸업(학사)
2007년 2월 명지대학교 컴퓨터소프트웨어학과 졸업(석사)
관심분야 : Embedded System, Storage Software

류 연 승

1990년 2월 서울대학교 계산통계학과 졸업(학사)
1992년 2월 서울대학교 전산과학과 졸업(석사)
1996년 8월 서울대학교 전산과학과 졸업(박사)
1996년 9월~2000년 8월 삼성전자
2003년 3월~현재 명지대학교 컴퓨터소프트웨어학과 부교수
관심분야 : Embedded Software, Operating System, Network System, Storage System