BF 및 PF layer 기능 구현 보고서

2017년 봄 고급데이터베이스 수업
민동문, 전기공학부, 2012-11334
손성호, 컴퓨터공학부, 2016-21213
최성준, 컴퓨터공학부, 2017-28487

BF Layer

BF 레이어는 buffer 프레임의 관리를 담당하는 레이어로 doubly linked list인 buffer 리스트와 linked list인 free list를 관리한다. 기본적인 원리는 buffer 프레임을 할당할 때 free list에서 프레임을 찾아 buffer 리스트로 옮기는 것과 프레임이 방출될 때 프레임을 free list에 다시 넣는 것이다. 이를 위해 사용된 함수들과 함수들의 동작은 아래와 같다.

- **void BF_Init(void)**

BF layer를 init한다. 먼저 hash table을 init하고 BFpage를 BF_MAX_BUFS개 만큼 동적할당하고 이를 Free List에 Linked List로 연결한다. 또 LRU List의 head(LRU_head)와 tail(LRU_tail)을 double linked list의 형태로 연결해준다. 마지막으로 Buffer Pool에 있는 entry의 개수를 나타내는 counter(BF_cnt)를 0으로 설정한다.

- **int BF_AllocBuf(BFreq bq, PFpage **fpage)**

Buffer Pool에 bq에 해당하는 BFpage를 할당하는 함수이다. 파라미터인 fpage는 만들어진 BFpage를 포인팅해서 돌려주기 위한 부분이고, return값은 성공시 BFE_OK, 실패시 BFE_NOBUF(buffer가 핀된 페이지로 꽉 참), BFE_PAGEINBUF(페이지가 이미 buffer pool에 있다)을 출력한다.
victim을 정하는 부분과 LRU에 새로운 페이지를 추가하는 부분은 다른 BF 함수에서도 자주 쓰기에 del_victim과 insert_in_LRU로 따로 구현하여 사용하였다.

- **int BF_GetBuf(BFreq bq, PFpage **fpage)**

bq에 해당하는 BFpage를 찾아 fpage에 포인팅해서 전달하는 함수이다. 파라미터인 fpage는 만들어진 BFpage를 포인팅해서 돌려주는데 쓰고, return값은 성공시 BFE_OK, 실패시 BFE_NOBUF(buffer pool이 핀된 페이지로 꽉 참)을 출력한다.
원하는 페이지가 buffer pool에 없을 때 디스크에서 UNIX read를 이용해서 페이지의 내용을 가져오고 이를 새로운 페이지에 할당한다. 그리고 buffer pool에 추가한다.

- **int BF_UnpinBuf(BFreq bq)**

bq에 해당하는 BFpage를 찾은 후 pin count값을 1만큼 감소시킨다. 성공시 BFE_OK, 실패시 BFE_PAGEUNFIXED(페이지가 이미 unpin되있다), BFE_PAGENOTINBUF(페이지가 buffer pool에 없다)를 return한다.

- **int BF_TouchBuf(BFreq bq)**

bq에 해당하는 BFpage를 찾은 후 dirty flag를 true로 만든다. 성공시 BFE_OK, 실패시 BFE_PAGEUNFIXED(페이지가 unpin되있다), BFE_PAGENOTINBUF(페이지가 buffer pool에 없다)를 return한다.
touch를 하게 되면 해당 BFpage를 MRU 위치로 이동시킨다.

- **int BF_FlushBuf(int fd)**

file descriptor가 fd에 해당하는 모든 BFpage를 buffer pool에서 flush시킨다. 성공시 BFE_OK, 실패시 BFE_PAGEFIXED(페이지가 pinned), BFE_UNIX(UNIX write error)를 return한다.
전체 buffer pool을 LRU_head부터 LRU_tail까지 돌며 file descriptor가 fd인 BFpage들을 Flush한다. 해당 BFpage가 pin되어 있으면 error를 return하고, 해당 BFpage가 dirty하면 디스크 상 file에 페이지의 내용을 갱신하고 Flush한다.
주어진 file descriptor에 해당하는 파일의 총 page number도 함께 주어졌다면 hash table을 이용해서 performance를 향상시킬 수 있었겠지만 page number가 주어지지 않아 전체 LRU를 훑는 약간 느린 방식을 택했다.

- **void BF_ShowBuf(void)**

현재 buffer pool에 있는 BFpage에 관한 모든 정보를 출력한다.
int insert_in_LRU(BFpage* bfpage)
LRU List 상의 head 부분(MRU)에 bfpage를 insert한다. Counter(BF_cnt)를 증가한다.

- **int del_victim(void)**

LRU_tail에서 시작하여 head(MRU)로 이동하면서 unpinned 되어있는 BFpage를 찾는다(=victim). Victim을 buffer pool에서 지운다. 만일 dirty flag가 true일 경우 디스크에 페이지 값을 갱신하고 지운다. hash table에서 victim의 index를 지우고 counter(BF_cnt)를 1 감소시킨다.

Hash Table of BF Layer

해시 테이블은 외부 라이브러리인 uthash(http://troydhanson.github.io/uthash/)를 사용해 만들어졌다. 이를 사용하기 위해 h 폴더에 uthash.h가 추가되었으며 별도의 C 파일은 추가되지 않았다. Uthash는 C에서 사용할 수 있는 해시 테이블 라이브러리이며 함수 대신 매크로를 사용해서 만들어져서 어떠한 타입의 attribute 혹은 attribute의 집합도 키로 사용할 수 있는 장점이 있다. BF_InitHash는 해시 테이블을 초기화하는 함수로서 향후 해시 테이블을 사용하기 위해 키의 길이를 계산하는 코드가 있다. 해시 테이블은 BF_InsertHash, BF_SearchHash, BF_DeleteHash로 이루어지며 각각 해시 테이블에서 엔트리를 삽입, 검색, 삭제하는 함수이다. 이 함수들은 uthash의 매크로인 HASH_ADD, HASH_FIND, HASH_DELETE 등을 사용해 구현되었다. 콜리젼이 발생했을 경우 chaining에 의해 관리되며 이를 위해 해시 테이블의 엔트리는 doubly linked list로 정의되었다.

Makefile of BF Layer

BF 레이어 구현시 PF 레이어의 함수를 참조해야 해서 makefile을 수정하였다. 수정된 makefile은 빌드시 PF 디렉토리에서 make 명령을 수행한 후 자신의 빌드를 수행하게 된다. 사용된 PF 레이어 함수는 해당 파일 디스크립터와 페이지 번호가 유효한지 확인하는 역할을 하는 것으로 PF 레이어에서 정의된 파일 테이블에 access하여 무결성 여부를 검사한다. 다만 이 변경 사항은 make 명령시 자동으로 수행되며 BF 레이어 빌드시 make 외에 추가적인 명령어나 단계가 필요하지 않다.

PF Layer

PF 레이어는 file table을 관리하며 파일의 생성, 삭제, 열기, 닫기 외에도 각 파일의 특정 페이지에 대한 접근, 페이지 추가 할당, dirty marking 등의 작업을 수행한다. 이 과정에서 UNIX system call을 이용하나, PF layer 내부에서는 UNIX의 file descriptor가 아닌 자체 PF file descriptor를 사용하여 각 파일을 관리한다. PF layer의 각 함수는 BF layer에서 구현된 함수들을 이용하여 파일 페이지의 buffer pool 할당, 페이지 접근, pinning / unpinning / dirty marking 등을 수행한다.

1) **PF file table**
   PF file table은 PF_Init가 실행될 때 PFftab_ele를 PF_FTAB_SIZE개만큼 저장하는 배열을 생성하여 PFftab_ele * pft에 배열의 주소를 저장하는 방식으로 할당되고 관리된다. PF_OpenFile이 실행되어 file table에 파일 하나가 할당되면 해당 entry의 valid 값이 TRUE가 되며, 파일이 할당되지 않았거나 파일이 열렸다가 닫힌 file table entry의 경우 valid 값이 FALSE이다. 이미 닫힌 파일이거나 적절하지 않은 file table entry에 대한 접근이 일어나면 먼저 해당 entry의 valid값을 검사하므로 오류를 방지할 수 있다.

2) **PF file header**
   현재 file header에는 numpages, 즉 해당 파일에 할당된 페이지의 수를 저장하는 int 변수만이 저장되며 관리된다. File header의 경우 다른 PFpage처럼 파일의 데이터를 담는 역할은 하지 않으나, 추후 파일 관리에 필요한 정보가 추가될 수 있으며 file header 자체의 크기를 PFpage와 같게 설정하도록 되어 있으므로 PFhdr_str에는 int numpages외에 char hdrrest[PF_PAGE_SIZE]를 포함하게 하여 PFhdr_str의 크기 자체가 PFpage와 같도록 해두었다. 이를 통해 PF_CreateFile, PF_OpenFile, PF_CloseFile등을 실행할 때 파일의 맨 처음 PAGE_SIZE 바이트를 읽어들이거나 쓰도록 간단히 구현할 수 있었다.

3) **UNIX system call의 활용**
   PF_CreateFile, PF_OpenFile, PF_CloseFile, PF_DestroyFile에서는 UNIX system call 함수인 open(), close(), remove(), stat(), read(), write(), pwrite()가 활용된다. open()의 경우 O_CREAT flag를 사용하여 파일을 생성하는 데 사용할 수 있으며, O_RDONLY flag를 사용하여 파일의 존재 여부를 검사할 때에도 쓰인다. stat()은 파일의 inode 값을 받아오는 데 쓰인다. read()와 write(), pwrite()는 파일 헤더 정보를 읽어들이거나 수정된 헤더 정보를 파일에 쓸 때 사용된다.

4) **추가로 구현된 함수들**
   a) **Int PF_IsValidPage(int fd, int pagenum)**
      이 함수는 주어진 PF file descriptor와 pagenum이 유효한 파일의 페이지 숫자인지를 검사하여 부적절한 접근을 예방한다.
   b) **Int PF_GetNumPage(int fd, int * pagenum)**
      이 함수는 PF layer 바깥에서 사용할 목적으로 구현하였다. 이 함수는 주어진 PF file descriptor의 파일에 할당된 페이지가 몇 개인지를 *pagenum에 저장한다.

5) **PF_GetFirstPage, PF_GetNextPage, PF_GetThisPage**
   PF_GetFirstPage는 PF_GetNextPage에 값이 -1인 argument를 넘기는 방식으로, PF_GetNextPage는 PF_GetThisPage에 값이 1 늘어난 *pagenum을 넘기는 방식으로 구현하였다.

custom.h
PF 레이어와 BF 레이어에서 사용될 데이터 타입 및 함수의 프로토타입을 custom.h에 정의하였다. custom.h는 h 폴더에 추가되어 있다.