DataBase Project2 보고서

20180038 박형규

1. 서론

Project2에서는 오디세우스/EduCOSMOS Project의 가장 첫 단계인 EduBfm Project를 진행 할 것이다. 이 프로젝트를 통해 Buffer manager에 대한 데이터 구조와 연산들을 구현하게 되는데, EduBfM에서는 오디세우스/COSMOS BfM의 기능들 중 일부의 기능만을 구현할 것이다. 이번 프로젝트에서 구현하게 될 API Function은 EduBfM_GetTrain(), EduBfM_FreeTrain(), EduBfM_SetDirty(), EduBfM_FlushAll(), EduBfM_DiscardAll()이다. 그리고 Internal Function또한 구현하게 되는데, 이는 edubfm_ReadTrain(), edubfm_AllocTrain(), edubfm_Insert(), edubfm_Delete(), edubfm_Deleteall(), edubfm_LookUp(), edubfm_FlushTrain()이다. 이 함수들을 각각 구현하는데에 있어서 필요한 설명은 아래 '2. 함수 설명'에서 하도록 하겠다. 구현할 때 API Function들에서 Internal Function들이 사용되므로 Internal Function을 먼저 구현하고, API Function을 구현하였다. 따라서 아래 함수 설명에서도 Internal Function에 대한 설명을 먼저 한 뒤 API Function에 대한 설명을 진행하도록 하겠다.

2. 함수 설명

2.1 Internal Functions

- edubfm_ReadTrain()

이 함수는 Page/train을 disk로부터 읽어와서 buffer element에 저장하고, 해당 buffer element에 대한 포인터를 반환하는 함수이다. 이는 RDsM_ReadTrain() 함수를 이용하여 간단히 구현할 수 있다. RDsM_ReadTrain()이 Page/train을 disk로부터 읽어오는 함수이므로 이 함수를 return할 때 실행하여 알맞은 인자와 함께 넘겨주면 정상적으로 실행 된다. 이 구조는 Manual ppt 4page에 나와있는 오디세우스 구조에서 볼 수 있듯이 RDsM이 BfM보다 low level에 존재하기 때문에 BfM이 RDsM의 function을 불러와서 실행하는 것이다.

-edubfm_AllocTrain()

이 함수는 bufferPool에서 page/train을 저장하기 위한 buffer element를 한 개 할당 받고, 해당 buffer element의 array index를 반환하는 함수이다. 먼저 buffer element를 할당받기 위해 second chance buffer replacement algorithm을 사용하였다. 먼저 !BI_FIXED()를 활용하여 Fixed bit이 0인 buffer element를 순차적으로 방문하고 (BI_BITS(type, victim) & REFER)를 이용하여 REFER bit을 검사해서 동일한 buffer element를 2회째 방문한 경우에 해당 buffer element를 할당 대상으로 선정하고, REFER == 1인 경우 REFER을 0으로 설정해 주었다. 그리고 victim을 갱신해 나가며 이 과정을 for문을 통해 반복하였다. 최종 victim을 찾은 후에는 victim의 DIRTY bit가 set되어 있는 경우에는 edubfm_FlushTrain() 함수를 이용하여 기존 buffer element의 내용을 disk로 flush하였고, victim의 bufTable element를 초기화 하고 array index를 hashTable에서 삭제하였다. 그리고 마지막으로 선정된 buffer element의 array index를

반환하였다.

-edubfm_Insert()

이 함수는 hashTable에 buffer element의 array index를 삽입하는 함수이다. edubfm_LookUp() 함수를 이용하여 hashTable에 key에 대응하는 buffer element의 array index가 있는지 찾는다. 없다면 우리가 insert가 가능한 상황이므로 반환값이 NOTFOUND_IN_HTABLE일 경우에만 작업을 진행한다. 먼저 edubfm_Hash.c에 define 되어있는 BFM_HASH 매크로를 이용하여 hashValue를 구한다. BI_HASHTABLEENTRY()를 이용하여 Collision의 발생 유무를 살펴보고, Collision이 발생하였을 경우에는 nextHashEntry에 기존의 hashTable entry를 저장한다. 그 뒤에 우리가 이 함수에서 의도했던 hashTable에 index를 저장한다.

-edubfm_Delete()

이 함수는 hash Table에서 buffer element의 array index를 삭제하는 함수이다. 먼저 BFM_HASH() macro를 이용하여 hashValue를 얻는다. 다음으로 hashValue에 대한 hashtable entry를 BI_HASHTABLEENTRY() 함수를 이용해서 얻는다. 다음으로 해당하는 hashValue에 대한 Hashtableentry를 탐색하는데 key가 인자로 받아온 값과 같을 경우 이 entry가 삭제해야하는 entry이다. 이때, entry가 linked list형태로 이어져 있기 때문에 삭제하는 경우 nextEntry를 잘 지정해주고 가야한다. 따라서, entry가 head일 경우와 그 이외의 경우로 나누어서 BI_NEXTHASHENTRY()를 통해 가져온 next entry를 알맞게 이어준다. i가 NIL이 될 때 까지 for문이 반복되었다면 인자로 받아온 key값과 동일한 key값을 찾지 못한 것이므로 eNOTFOUND_BFM error를 발생시킨다.

-edubfm_DeleteAll()

이 함수는 2개의 hashTable(PAGE_BUF, LOT_LEAF_BUF에 대한 hashTable)의 모든 entry를 삭제하는 함수이다. 2중 for문을 이용하여 각 entry들을 초기화함으로써 각 hashTable에 대한 모든 entry를 삭제하였다.

-edubfm_LookUp()

이 함수는 hash key에 대응하는 buffer element의 array index를 검색하여 반환해주는 함수이다. BFM_HASH macro를 이용하여 hashValue를 찾은 뒤, for문을 이용하여 hash table을 검색하며 동일한 key가 있는지 살펴본다. 이는 EQUALKEY를 이용한다. 동일한 Key를 찾으면 return i를 통하여 array index를 반환해 주었다.

-edubfm_FlushTrain()

이 함수는 수정된 page/train을 disk에 기록하는 함수이다. 먼저 edubfm_LookUp() 함수를 이용하여 Flush할 page/train이 저장된 buffer element의 array index를 가져온다. 그 뒤에 DIRTY BIT가 설정되어 있다면 RDsM_WriteTrain() 함수를 이용하여 disk에 기록하고, Dirty bit를 다시 0으로 만들어 준다.

2.2 API Functions

-EduBfM_GetTrain()

이 함수는 Page/train을 bufferPool에 fix하고 page/train이 저장된 buffer element에 대한 포인터를 반환한다. 먼저 edubfm_LookUp() 함수를 이용하여 해당 page/train이 저장된 buffer element의 array index를 구한다. 만약 index를 찾지 못한다면 우리는 buffer element를 할당 받는 작업을 해야 한다. 따라서 edubfm_AllocTrain() 함수를 호출하여 buffer element를 할당 받고 array index값을 index변수에 저장한다. 만약 반환된 index값이 0보다 작다면 제대로 할당받지 못한 경우이므로 error 메세지를 출력한다. buffer element를 할당 받았으면 edubfm_ReadTrain() 함수를 이용하여 disk에서 page/train을 받아와서 buffer element에 저장했다. 만약 이 과정에서 에러가 발생한다면 error code가 반환값인 e에 저장되므로 e가 eNOERROR가 아니라면 에러를 출력한다. 마지막으로 할당 받은 buffer element의 array index를 hashTable에 삽입하기 위해 edubfm_Insert() 함수를 통해 이 작업을 수행한다. 이 작업에 대한 에러처리도 위와 마찬가지로 진행된다. 다음으로는 처음에 edubfm_LookUP() 함수를 통해 제대로 된 index를 받아 왔든, 받아 오지 못했든 공통으로 수행되는 작업이다. 해당되는 buffer element의 fixed값을 1 증가시키고 해당 buffer element에 대한 포인터를 반환하면 작업이 끝난다.

-EduBfM_FreeTrain()

이 함수는 Page/train을 bufferPool에서 unfix하는 함수이다. 먼저 edubfm_LookUp() 함수를 이용하여 해당 page/train이 저장된 buffer element의 array index를 구한다. index가 정상적으로 찾아졌을 경우에는 Fixed값이 0이상인지 확인하고, 0이상이라면 1을 줄여줌으로써 unfix한다. 0 이하일 경우에는 TestSolution 에서 출력한 모습과 동일하게 만들기 위해서 "fixed counter is less than 0!!!\n"과 trainId를 출력해준다.

-EduBfM_SetDirty()

이 함수는 bufferPool에 저장된 page/train이 수정되었음을 표시하기 위해 DIRTY bit을 1로 setting하는 함수이다. 먼저 edubfm_LookUp() 함수를 이용하여 해당 page/train이 저장된 buffer element의 array index를 구한다. 만약 index값이 0보다 작다면 error가 발생한 것이므로 error code로 index를 return해준다. index가 정상적으로 찾아졌을 경우에는 BI_BITS를 불러와서 DIRTY bit를 bit연산자 or을 이용하여 setting해준다.

-EduBfM_FlushAll()

이 함수는 두 개의 bufferPool에 존재하는 page/train중에서 수정된 page/train들을 disk에 기록하는 함수이다. PAGE_BUF와 LOT_LEAF_BUF 두 개의 bufferPool 모두에 대해 이 작업을 수행하기 위해 2중 for문을 이용하여 작업을 하였다. 바깥의 for문에는 type을 기준으로 작성하였고, 안쪽의 for문은 각 type별로 buffer를 탐색하여 DIRTY bit가 setting된 element가 있는지 확인하였다. DIRTY BIT가 set되어 있다면 edubfm_FlushTrain()함수를 이용하여 해당 page/train을 disk에 기록해주었다.

-EduBfM_DiscardAll()

이 함수는 각 BufferPool에 존재하는 page/train들을 disk에 기록하지 않고 bufferPool에서 삭제하는 함수이다. 이 작업도 위의 EduBfM_FlushAll() function에서 작업했던 것과 비슷하게 이중 for문을 이용하여 작업하였다. PAGE_BUF와 LOT_LEAF_BUF 두 개의 bufferPool 모두에 대해 이 작업을 수행하기 위해 2중 for문을 이용하여 작업을 하였다. 바깥의 for문에는 type을 기준으로 작성하였고, 안쪽의 for문은 각 type별로 buffer를 돌며 각 BufTable의 모든 element들을 초기화해 주었다. 그리고 edubfm_DeleteAll() 함수를 호출하여 각 hashTable에 저장된 모든 entry들을 삭제했다.

3. Discussion

이번 과제를 수행하며 가장 어려웠던 점은 많은 파일에 분할되어 있는 함수들이 어떻게 연관되어 동작하는지 파악하는데 있었다. 또한, 헤더파일에도 많은 매크로들이 설정되어 있어 이것들을 어떻게 하면 함수를 짤 때 최대한 효율적으로 사용할 수 있을 지 고민하는데에도 시간이 많이 소요되었다. 또한, 자잘한 실수 때문에도 시간 낭비를 하였다. 연산자 우선순위를 제대로 고려하지 못하여서 처음에 GetTrain() 함수가 제대로 동작하지 못하였는데 이를 파악하는데 gdb를 사용하여 디버깅을 하는데 시간이 오래 소요되었다. 그래도 이 프로젝트를 수행하고 나니 OS가 어떻게 Buffer을 관리하는지 구체적으로 알게 된 것 같아서 신기하다.