To Implement GetBlk() Algorithm

Step 1.To create BufferCache

1.How to implement BufferCache ?

For BufferCache we will implement doubly linked list having following functions

which has to be implemented :-

--> Structure of node in doubly linked list

1. int data
2. node *next
3. node *prev
4. int status1 (0 free 1 busy)
5 . int status2 (0 valid data , 1 invalid , 2 delayed write)

--> Functions in doubly linked list

1. insert_at_start():-needed when buffer marked as delayed write and kernel performs asynchronous write on disk block, after completion that buffer will be added at front of free list.
2. insert_at_end():- When buffer get unlocked or free then it added at tail of free list
3. delete_from_begin():-When kernel want any free buffer then it will be removed from head of free list.
4. delete_by_value():- When kernel want any specific buffer it may be present in middle of the list so we will search buffer by blocknumber
5. isPresent():-it searches for block whether it is present in buffer cache or not
6. isEmpty():- Used to check whether free list is empty or not
7. getBuffer():- when kernel found the buffer it returns address of that buffer or node in terms of doubly linked list.
8. change_status:-To change status for specific buffer (from free to busy or busy to free)

Initially we are taking 16 buffers (randomly) for better convenience by making 16 objects of node type.

2.Why we choose doubly linked list ?

We choose doubly linked list for following reasons:-

->insertion and deletion takes O(1) time and we usually need insertion and deletion from end points

->Traversing is not possible in another data structures like queue and stacks.For queue we need to maintain extra data structure like Hashset(to check whether block is present or not).

3.How we can maintain hash queue for efficient searching for block in buffer cache ?

For efficient searching we will create array of 4 doubly linked list

eg:-DLL hq[4]

blocknum 0 mod 4 in hq[0]

blocknum 1 mod 4 in hq[1]

blocknum 2 mod 4 in hq[2]

blocknum 3 mod 4 in hq[3]

where hq[i] is the pointer pointing to the respective i^{th} linked list.

4.How we can maintain the sleep and awake calls for block no .(for both general and specific) in buffer cache ?

For efficient implementation we will create a class sleep having data members :-

1. thread no
2. Block no. (for which that thread is waiting)

// if waiting for specific buffer then that block no is assign to this member

// Else if waiting for general then 0 is assign to it.

We create an queue of this sleep type data structure for accessing waiting blocks in FIFO manner. So that whenever the status of an buffer becomes free it give awake signal to all the waiting buffers for it and on the basis of first come first serve it will get an buffer assigned.

Step 2.To Implement GetBlk() algorithm

In this file we will implement following methods:-

getBlock():-This function will handle all five cases of getbllk() algorithm

case1. The kernel finds the block on its hash queue and its buffer is free.

Case2.The kernel cannot find the block on the hash queue,so it allocates a buffer from the free list

Case3.The kernel cannot find the block on the hash queue and in attempting to allocate a buffer from free list , find a buffer on free list that has marked “delayed write”.

Case4.Th ekernel cannot find the block on the hash queue and the free list of buffers is empty

Case5. The kernel finds the block on the hash queue,but its buffer is currently busy.

Handle_write():- use to mark buffer with delayed writei.e.change buffer status

Handle_writeAsync():-if buffer marked with delayed write , remove that buffer from free list write its content to disk block and again add that buffer at front of free list.

Step 3.Driver code

We will create two threads in main function to implement multi programming/multi threading

Thread t1(fun,0)

Thread t2(fun,0)t

Main thread will be terminated after termination of thread t1 and thread t2 implemented as

T1.join();

T2.join();

Why we choose multilplethreads ?

Multi threading is the ability of a process to manage its use by more than one user at a time and to manage multiple requests by the same user without having to have multiple copies of the program.

->lightweight process

->Enhances efficiency in the context of communication.

->easy to handle

->share resources like code section,data section.

Submitted By:--

Divya , Diksha , Cecilia , Sahil , Ashish

Msc computer science (1 year)