Programming Assignment 3

CPS 210, Spring 1999

Due Date: 4/29/99 at midnight(on Thursday)

Implement a multi-threaded buffer manager. A similiar facility is used in implementations of file systems, databases, and virtual memory. The buffer manager provides a cache of 4K data blocks uniquely named by integer IDs. These buffers are allocated from a fixed-size global array of 4K blocks in the process virtual address space. The buffer manager supports five basic operations.

> **Buffer*** GetBlock (**BlockID** bid) -- *Fetch the requested block into the cache if it is not already resident. Return the buffer for use by the caller. Buffers are returned locked.*
> **void** ReleaseBlock (**Buffer*** bp) -- *Inform the buffer manager that the caller no longer needs the specified buffer. The buffer should be locked when ReleaseBlock is called.*
> **Buffer**.SetDirty() -- *Mark a buffer as dirty, forcing a writeback when evicted from the cache.*
> **Buffer**.Lock () -- *Request exclusive access to a buffer (e.g., to examine or modify its contents).*
> **Buffer**.Unlock () -- *Release the lock on a buffer.*

The block cache contains a fixed number of blocks. If the cache is full and the system receives a *GetBlock* for a block that is not in the cache, then some other block must be discarded from the cache so that its buffer can be used to satisfy the request. Any block can be replaced in this fashion, but only if it is not in use by any thread (e.g., all calls to *GetBlock* for that block have been paired with a matching *ReleaseBlock*).

The *SetDirty* method allows updates to data in the buffer cache. *SetDirty* is called by the client (e.g., the test program) to indicate that a buffer has been modified. For example, to modify data in a file, the client would acquire a buffer for the block to be modified (possibly reading the block into the cache), lock the buffer, modify its contents in place, mark it dirty with *SetDirty*, and release it. The client may modify the buffer or call *SetDirty* only when the buffer is locked.

The buffer cache connects to an underlying I/O system through external procedures *FillBlock* and *PushBlock* supplied by the test program. *FillBlock* must be called from *GetBlock* to initialize a buffer given its block ID. The signature for *PushBlock* is the same as *FillBlock*. Your buffer manager must call *PushBlock* to “clean” a dirty block before recycling the buffer that holds it. The buffer must be locked while *PushBlock* is cleaning it, to ensure that no other thread modifies the block or buffer while it is being written out.

Your solution must guarantee that each block appears at most once in the cache, and it must be deadlock-free. Your buffer manager should also be robust. In particular, you should assume that the time for *FillBlock* or *PushBlock* to complete is unbounded. For example, if the buffer manager were used in a network file system, *FillBlock* or *PushBlock* could require network communication with a host or process that has failed. In this case, the buffer manager must block any thread waiting for the particular block or buffer that is being filled or pushed, but other operations should proceed unimpeded. For example, it is acceptable for a *GetBlock* to

wait for a *PushBlock* to complete, but only if the *GetBlock* requests exactly the block that is currently being pushed.

Mechanics

This semester we are publishing basic code for the buffer manager as well as the test program. The goal is to save you the grunt work of implementing the whole thing from scratch, leaving you more time for your projects. The sample code was derived by deleting synchronization-related code from the solution. As with the sthreads assignment, you will need to understand the buffer cache code and reason through various synchronization scenarios in order to implement the synchronization correctly.

The sample code can be found in */usr/project/courses/cps210/src/cache*; the test program is in the same subdirectory. If you've copied over the whole tree on previous assignments it's the same code. Rules for building using *genmake* are the same as for the last assignment. Your solution code should work on *twister* (the multiprocessor Alpha) as in the last assignment. This time we will be using the OS-supported *pthreads* rather than *sthreads*. This should save you from problems stemming from bugs in your code from the last assignment.

Please write up a one page description of how you synchronized your buffer manager and the performance implications of your solution. Your solution should pass all of the tests in the test program; make sure you test your solution adequately.