AOS Assignment

Team Members

1. Ashwani Varshney
2. Supriya Katyal
3. Farheen Siddiqui
4. Simran Mehra
5. Pooja Sikka

Buffer Cache Simulation

A simple buffer cache implementation for simulation of getblk and brelse algorithms.

Overview

In this implementation, the main program act as kernel and manipulates the buffer cache, while processes are being created by using the fork () command. It controls which block to get and release. User can manipulate buffer status. This gives us complete control over the simulation. Also, we can view the buffer cache, free list, buffer list, and sleeping processes at all times.

This is a multi-process synchronous program using fork (), so there is no added complexity of managing threads. We can focus on getblk and buffer cache.

Problem Statement

Buffer cache simulation. The idea was to simulate getblk and brelse algorithms that handle the allocation and release of buffers to processes. The key requirement was to be able to visualize all 5 scenarios of getblk and clearly identify the working of getblk in all of those scenarios.

Suggested Solution

We want a simulation slow enough to analyse the working details of getblk.

Programming Language Used

C++ is the chosen language. Because we are implementing buffer cache, which is a lower level algorithm, it needs to be fast though C provides the fastest system calls but C++ gives us object-oriented approach.

AOS Concepts Implemented

1. Multiprogramming concept using forks
2. Locking unlocking mechanism

Delayed write

-User manually sets a block status to delayed write.

-Then when the getblk algorithm sees this delayed write marked buffer

-it will be removed it from free list it starts a dummy async write to disk (which is just a wait of 5 secs) and the pointer moves on to the next free list buffer.

-The previous buffer which was marked delayed write we will add it to the header of the free list and the delayed write status will be set as False.

Data Structures Used

1. Hash Queues

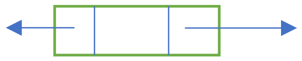
Hash Queues Headers

int *ptr[4]   // array of pointers

initially hash queues are empty and all the buffers are in the free list

Hash Function- Block number mod 4

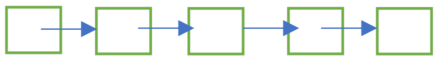
2. Free list pointer

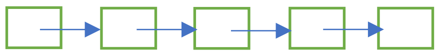


3. Two types of waiting lists

Lists will be implemented as linked lists

1. One for each buffer

   B1 - Processes waiting for Buffer1 

   B2 - Processes waiting for Buffer2 

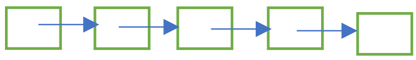
   .

   .

   .

   .

   .

   B16 - Processes waiting for Buffer16 

2. **A common waiting-list**

4. A ready queue

Initial configuration

- A Buffer class with 16 objects (representing 16 buffers)
- Initialize all the flags to False and set all the pointers to null.
- Hash Queues are empty initially and all the Buffers are in free list.
- getblk function with block_no as parameter
- Main function contains 4 to 5 fork () commands to create processes and call getblk (b_no)
- bufferFree () is called when buffer is unlocked by some process

```
class BufferHeader
{
        int device_num;         //variables
        int block_num;
        bool lock;
        bool valid data;
        bool delayed_write;
        bool in_demand;
```

```
    bool read;
    bool write;

    int *data_area;         //pointers
    int *prev_buf_hq;
    int *next_buf_hq;
    int *prev_buf_fl;
    int *next_buf_fl;

}
```

bufferFree()
{

when buffer lock= F
  add buffer to free list
check if in_demand = T
  go to buffer waiting list and common waiting list
call getblk () on all processes in random order

}

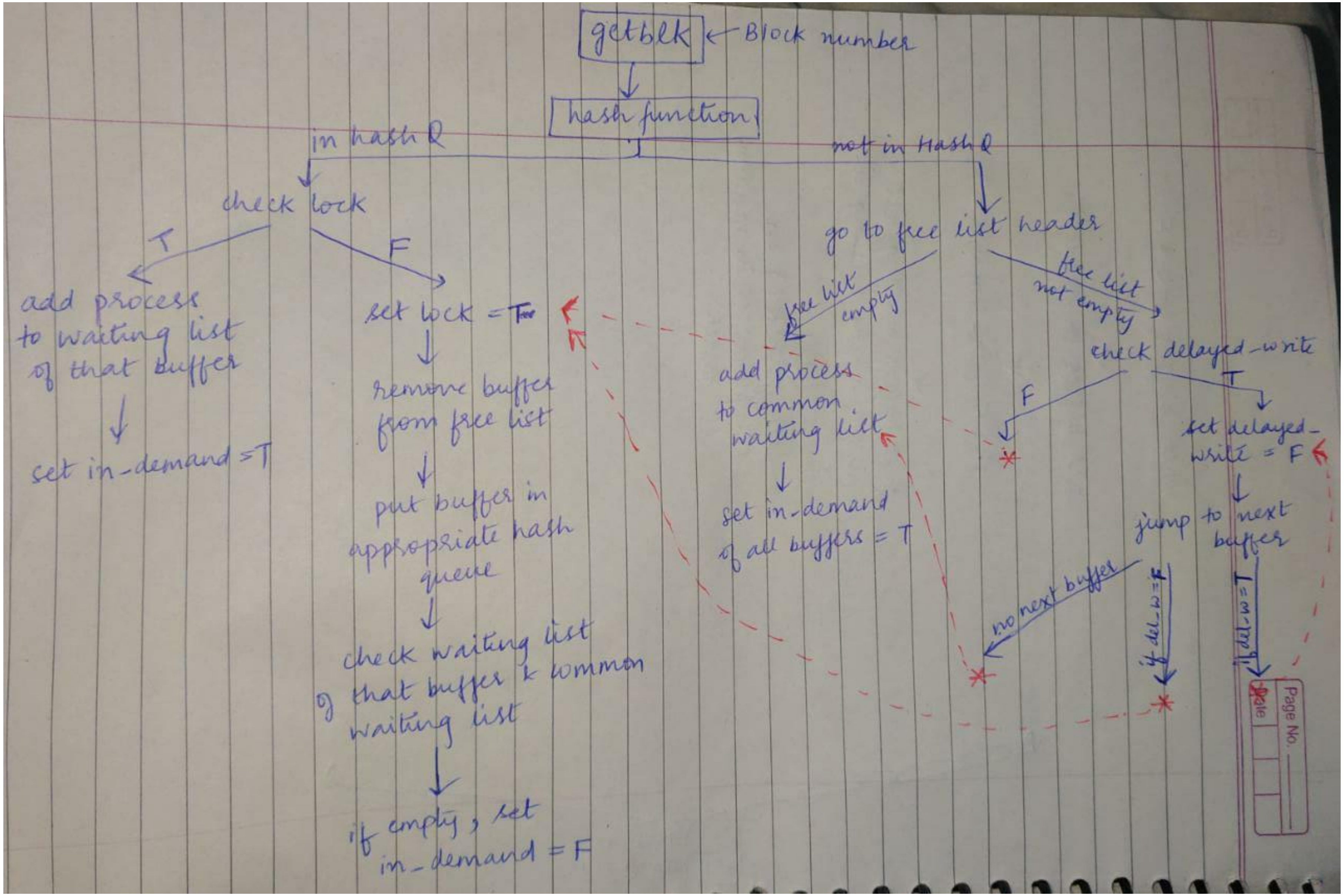
main( )
{

-fork () to create processes
-the created processes will be ordered in ready queue such that all the 5 scenarios of the getblk will be covered.
-the processes will invoke their input files which contain the block numbers they need.
getblk (block_no) is called on the block number which is written in the input file.
After the sleep timer has exhausted the buffer will get unlock and a call to bufferFree () is made.

}

Working –

(In the image below)


getblk
Block number
hash function
in hash Q
not in Hash Q
check lock
T
F
add process to waiting list of that buffer
set in-demand = T
set lock = T
remove buffer from free list
put buffer in appropriate hash queue
check waiting list of that buffer & common waiting list
if empty, set in_demand = F
go to free list header
free list empty
free list not empty
add process to common waiting list
set in-demand of all buffers = T
check delayed-write
set delayed-write = F
jump to next buffer
no next buffer
if del_w = F
if del_w = T
Page No.
Date