BUFFER CACHE SIMULATION

SUBMITTED BY- Himanshu Dabas (22)
Devansh Sharma (16)
Saachi (43)
Kajal Gupta (25)
Rashika Bagri (39)

Modules :-

Module buffer.py

A Buffer is implemented as a class. It contains the metadata/Information corresponding to the block for which it is allocated or requested for. This information is captured as data members of the buffer header part of the node. Since, the buffer has two parts: the memory array that contains the data from the disk and the buffer header that identifies the buffer. But, since we are just simulating a kernel data structure or functionality, thus the memory array part of the buffer would be null and the necessary information regarding the data blocks on the disk will be captured in the status flag/set bit of the buffer header.

Data members of the buffer class data structure are :-

1. **Device number**
2. **Block number -** will be assigned a random number within a specified range from the implementation point of view.
3. This block number will be used in placing the buffer in the appropriate hash queue. The block number will be referenced while searching for a specific buffer.
4. **Buffer number -** to uniquely identify every buffer
5. **Status -** flags/setbits will be maintained for this
    a. **locked/busy and unlocked/free -** set bit will be 1 for locked and 0 for unlocked
    b. **delayed write -** set bit will be 1 otherwise 0
    c. **awaited -** indicates whether any process is waiting for that buffer to become free
    d. **awaited_count -** count of processes waiting for that buffer to become free
6. Pointers to buffers in the hash queue and free list
    a. **Node* hashnext -** Pointer to the next buffer on the hashqueue
    b. **Node* hashprev -** Pointer to the previous buffer on the hashqueue

c. **Node* freenext -** Pointer to the next free buffer on the freelist
d. **Node* freeprev -** Pointer to the previous free buffer on the freelist

We are not maintaining any pointer to the data area since our aim is to just simulate the working of the getblk algorithm. Just maintaining the info about the data in status as valid or invalid.

Member functions -

1. **get_awaited(self) -** to check whether the any process is sleeping for that buffer or not
2. **set_buf_num(self, num) -** assigns the buffer a unique number
3. **get_buf_num(self) -** returns the buffer number
4. **get_status(self) -** returns the status of the buffer
5. **is_locked(self) -** checks whether buffer is locked/busy
6. **set_locked(self, lock=True) -** sets buffer as locked/busy
7. **set_status() -** sets the status of the buffer as passed via the arguments
8. **is_delayed_write(self) -** checks whether the buffer is delayed write or not
9. **set_buffer_delayed(self, delay=True) -** sets status of delayed write of a buffer
10. **get_block_number(self) -** returns the block number currently saved in the buffer
11. **set_block_number(self, blk_num=None) -** sets the block number of the buffer
12. **get_hash_next(self) -** returns the next buffer in hash queue
13. **get_hash_prev(self) -** returns the previous buffer in hash queue
14. **set_hash_next(self, bfr=None) -** sets the next buffer in hash queue
15. **set_hash_prev(self, bfr=None) -** function to set the previous buffer in hash queue -
16. **get_free_next(self) -** returns the next buffer in free list
17. **get_free_prev(self) -** returns the previous buffer in free list
18. **set_free_next(self, bfr=None) -** sets the next buffer in free list
19. **set_free_prev(self, bfr=None) -** sets the previous buffer in free list

Module hash_queue.py

To maintain the hash queue of buffers, we have implemented a class HashQueue which imports the buffer class and specifies important **data members like no. of hash queues, no. of buffers, block range of the buffers, no. of free buffers.**
Now, let's say we have a block number M and no. of hash queues or buckets specified are B. Then, the hash function will be K= M%B. The buffer for the block M will be then inserted at the end of the hash queue at array index K after the completion of the getblk function for the buffer M.

Member functions -

1. **_generate_buffers(self, no, rng) -** generates buffers
2. **get_buf_num_from_blk_num(self, blk_num) -** given a block number, it returns the corresponding buffer number
3. **is_buffer_in_free_list(self, bfr) -** checks whether buffer is in free list or not

4. **rem_buffer_from_free_list(self, bfr=None) -** removes buffer from the free list
5. **is_free_list_empty(self) -** checks whether the free list is empty is empty or not
6. **_generate_hq(self, no) -** initializes the hash queue with the given no. of slots
7. **add_to_free_list_end(self, bfr), add_to_free_list_beg(self, bfr) -** adds a buffer to the free list's end or beginning
8. **search_block_in_hq(self, block_no) -** checks whether a given block number exists in the hash queue or not
9. **add_at_end_hq() -** adds a given buffer at the end of hash queue
10. **rem_buffer_from_hash_queue() -** removes a given buffer from the hash queue
11. **get_bfr_from_bfr_num(self, bfr_num) -** returns the buffer given a buffer number
12. **is_buffer_free(bfr) -** checks whether a buffer is free or not
13. **is_buffer_valid(bfr) -** checks whether a buffer is valid or not

Module buffer_cache.py

Imports hash_queue
BufferCache class will handle all the functions of the buffer cache like it does in the operating system.
hash_queues (default 4): specified no. of slots to create for hash_queue. 4 is easy to visualize
max_buffers (default 10): specifies how many buffers are there in the simulation
In the beginning buffers are added to the hash_queue randomly (i.e. random blocks from the secondary memory). This is done to bootstrap the process of simulation.
This class also contains the implementation of getblock and block release algorithms.

getblk algorithm -

INPUT :-
1. blk_no - block number to search in the cache

RETURNS -
1. temp_bfr - locked buffer which is present in hash queue and not busy
2. False - in case of an error
3. 'buffer_busy' - if buffer is present in hash queue but is being used by some other process
4. 'free_list_empty' - if free list is empty

brelse algorithm -

INPUT :-
1. bfr : locked buffer

RETURNS :-

1. None

Module process.py

Process class implements a process as an instance of a class. There are various possible states for a process. The class has **data members like process name, process status, waiting type, block requested, block assigned.**

Possible status values for the process -

1. idle - 0
2. reading - 1
3. writing - 2
4. writing (delayed) - 3
5. waiting (for a particular block or any block) - 4

Possible values for waiting_type :-

1. **read -** process is waiting to read a block
2. **write -** process is waiting to write to a block

Member functions -

1. **get_status(self) -** returns the current status of the process
2. **get_assigned_buffer(self) -** returns the buffer assigned to the invoking process
3. **set_status() -** sets the status of the process
   INPUT :
   1. **state of the process -** [0,1,2,3,4] different states assigned to the processes
   2. **buffer_assigned -** name of the buffer assigned to the process for performing any task (read/write/delayed write)
   3. **block_requested -** name of the block that the process has requested to read/write data from. (resides in sec memory)

   RETURNS : 0 on success and 1 if an error occurs

Module process_list.py

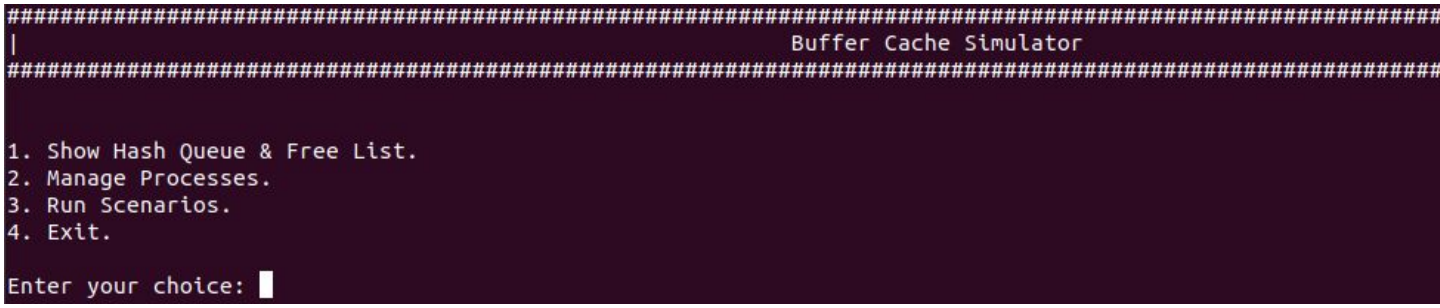
It contains a class ProcessList which is a collection of all the processes created so far. It helps to easily print all the processes and blocks occupied by them. When the system starts by default, it creates 4 processes, none of which holds any of the buffers initially.

Member functions -

1. **__str__ -** prints all the processes in the system along with the buffers and blocks assigned to them.
2. **get_block_requested -** takes the name of the process and returns the block number requested by the process
3. **get_waiting_type -** takes the name of the process and returns the type of event for which the process of sleeping. Returns false if a process is not sleeping.
4. **print_process_list -** function to print the entire list of processes along with the block number and buffer number held by them.
5. **_generate_processes -** generates random processes when the system initializes
6. **_get_unused_process_name -** assigns an unused name to a newly created process
7. **is_name_duplicate -** checks whether a process name is already assigned to another process or not
8. **get_no_of_active_processes -** returns the number of active processes in the system
9. **add_process -** adds a new process in the system
10. **del_process -** deletes a process from the system
11. **is_any_buffer_assigned -** checks whether any buffer is assigned to the process or not
12. **release_buffer -** releases a buffer currently occupied by the invoking process

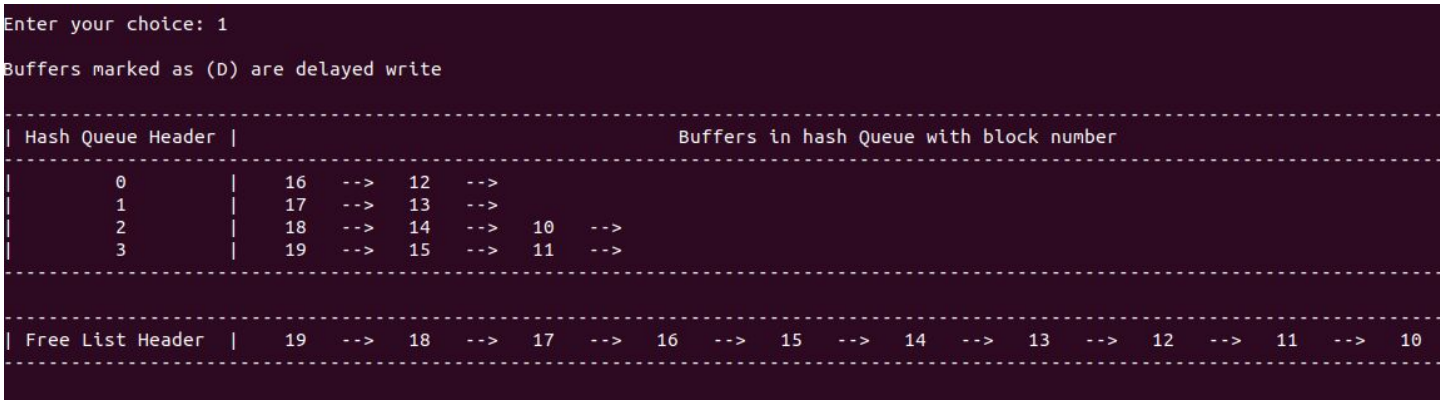
Module main.py

This program binds together all the modules in the project to simulate a buffer cache.
A menu is displayed when this module is run.



User gets to choose from the above given options.

Option 1 : Initially buffers numbered from 1 to 9 are assumed to be not in main memory and hence not displayed in the hash queue. Buffers numbered from 10 to 19 are in the free list since no active process holds them.



Option 2 : Manage Process - allows the user to choose from the following options.

Option Print Process List - displays all the active processes in the system along with the buffers and blocks held by them. Initially 4 random processes were created none of which held any buffer.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Print Process List.
2. Create new Process.
3. Delete a process.
4. Read a Block.
5. (Delayed) Write to a block.
6. Release buffer / Cancel Request for a process.
7. Back to Main Menu.

Enter your choice: 1
Process P0::
        Status:  `idle`.
Process P1::
        Status:  `idle`.
Process P2::
        Status:  `idle`.
Process P3::
        Status:  `idle`.
Process testProcess1::
        Status: `reading` Block: 4
        Buffer Assigned: 0
```

Option Delayed Write - Block 4 can be seen marked ‘delayed write’.

```
Buffers marked as (D) are delayed write

-------------------------------------------------------------------------------------------------------------------
| Hash Queue Header |                                        Buffers in hash Queue with block number
-------------------------------------------------------------------------------------------------------------------
|         0         |    16   -->   12   -->   4(D) -->
|         1         |    17   -->   13   -->
|         2         |    18   -->   14   -->   10   -->
|         3         |    15   -->   11   -->
-------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------
| Free List Header  |    18   -->   17   -->   16   -->   15   -->   14   -->   13   -->   12   -->   11
-------------------------------------------------------------------------------------------------------------------
```