Advanced Operating System

Assignment-Report

(getblk-algorithm simulation)

Submitted-By:

Name : Manisha Paruthi
Roll No: 18

Name : Priyanka Sukhalani
Roll No: 27

Problem Statement :

Simulation of 'getblk-algorithm' which is used for buffer-allocation to processes.The prime requirement is to visualize all five-scenarios of getblk-algorithm.

Programming Language Used :

'C' is the chosen programming language because it supports the feature of both low-level and high-level language.

Also, the compilation and execution time of C-language is fast due to low overheads. It has a capability to access system's low level functions.

AOS-concept Implemented :

1. Signals :

- ➔ A signal is a software generated interrupt that is sent to a process by the kernel.
- ➔ In our 'getblock-algorithm' implementation, a signal is sent from kernel to a process when a process completes its system call execution.

2. 'InterProcess Communication' : 'Client-Server Architecture'

- ➔ InterProcess communication is achieved by 'Socket-Programming'.
- ➔ 'Server' acts as 'Kernel' and 'Clients' act as 'Process'.
- ➔ Socket-Programming is used to create a 'Multiprogramming environment', which allows N processes to reside in Memory(i.e. N-processes can communicate with kernel) but at a time only one process will execute(i.e., 1-Process from N-process will be executed at any given time). No-two processes can compete for execution at a time **('FCFS-Scheduling strategy')**.
- ➔ Each Process can request 'kernel' for a block. Kernel runs 'getblk-algorithm' corresponding to requested block. A block can either be allocated or not allocated to a process, depending on getblk 'scenarios'.
- ➔ When a process wants to release a block, a signal is sent to the process and 'kernel' executes 'brelse-algorithm', insert the released block on free-list.
- ➔ If released buffer is valid and buffer is not old: insert released buffer on the end of free-list. Else, insert released buffer on the front of free-list.

➔ A process can terminates it execution by using command '**EXIT**'.

GetBlock Cases Handled:

Let,

No of **Buffers** in **Buffer Cache** be **K** and,
No of **Processes** in **Ready Queue** be **N**.

Each Process will request for a block. The kernel executes getblk-algorithm corresponding to this request. A block can be either be allocated or not-allocated to a process depending on one of the following scenarios:

Scenario-1:
The requested block is present in both Hash queue and Free List. The block is successfully allocated to the process.

Scenario-2:
The requested block is not present in Hash Queue but Free list is non-empty and selected/deleted buffer from Free List is not marked delayed write. Thus, requested block is successfully allocated to the process.

Scenario-3:
The requested block is not present in Hash Queue and buffer removed from free List is marked Delayed write. This removed buffer will be inserted into 'Hash-Queue-Delayed-Write-Mode' and again getblk algorithm is called on the requested buffer.

Scenario-4:
The requested block is not present in both Hash Queue and Free List . The process is inserted into sleeping queue
('Sleep-Queue-any-buffer') to wait for any buffer to become free.

Scenario-5:
The requested block is present in Hash Queue but it is currently busy with other process. The process is inserted into sleeping queue('Sleep-Queue-specific-buffer') to wait for specific buffer to become free.

'In Scenario-1 and scenario-2,' Process locks the requested block and enters into indefinite sleeping queue(Process-Queue) to execute the system call. Once system call is completely executed, a 'signal' is sent to the Process, that it has completed its system call. Now, the kernel allows the process to 'release/unlock' its locked block using 'brelse algorithm'. This released block can now be acquired by one of the sleeping processes (sleeping processes are present in

Sleep-Queue). This is done by waking up all the sleeping processes waiting for any-buffer and 'this-buffer' to become free.

Kernel will randomly choose one of these woken-up process and call getblk-algorithm on block required by chosen sleeping process.
Once chosen-sleeping process completes getblk execution, the process which initially unlocks the block can demand for another block from the kernel by calling getblk algorithm.
'In scenario-4', The process which requested the block is inserted into sleeping queue waiting for any buffer to become free, which will wake-up when either free-list becomes non-empty or any process releases/unlocks a buffer.

'In scenario-5', The process which requested the block is inserted into sleeping queue waiting for specific buffer to become free, which will wake-up when a process acquiring this block releases/unlocks its buffer.

Delayed-Write-Case(Scenario-3):

If Free buffer removed is marked delayed write, it is inserted into 'Hash-Queue-Delayed-Write-Mode', and getblk will be called again on the requested block.

A buffer will remain in Delayed-Write mode,until it finishes its asynchronous writing. In our program, this is implemented by using clock(). If removed buffer is marked delayed write, this specific buffer is given a 'start-time'(time when buffer is found as delayed-write). Let

Time-taken = (current-time - start-time) / CLOCK_PER_SEC

(Each buffer can stay in delayed write queue for 2-seconds or 200 processor-clock ticks).

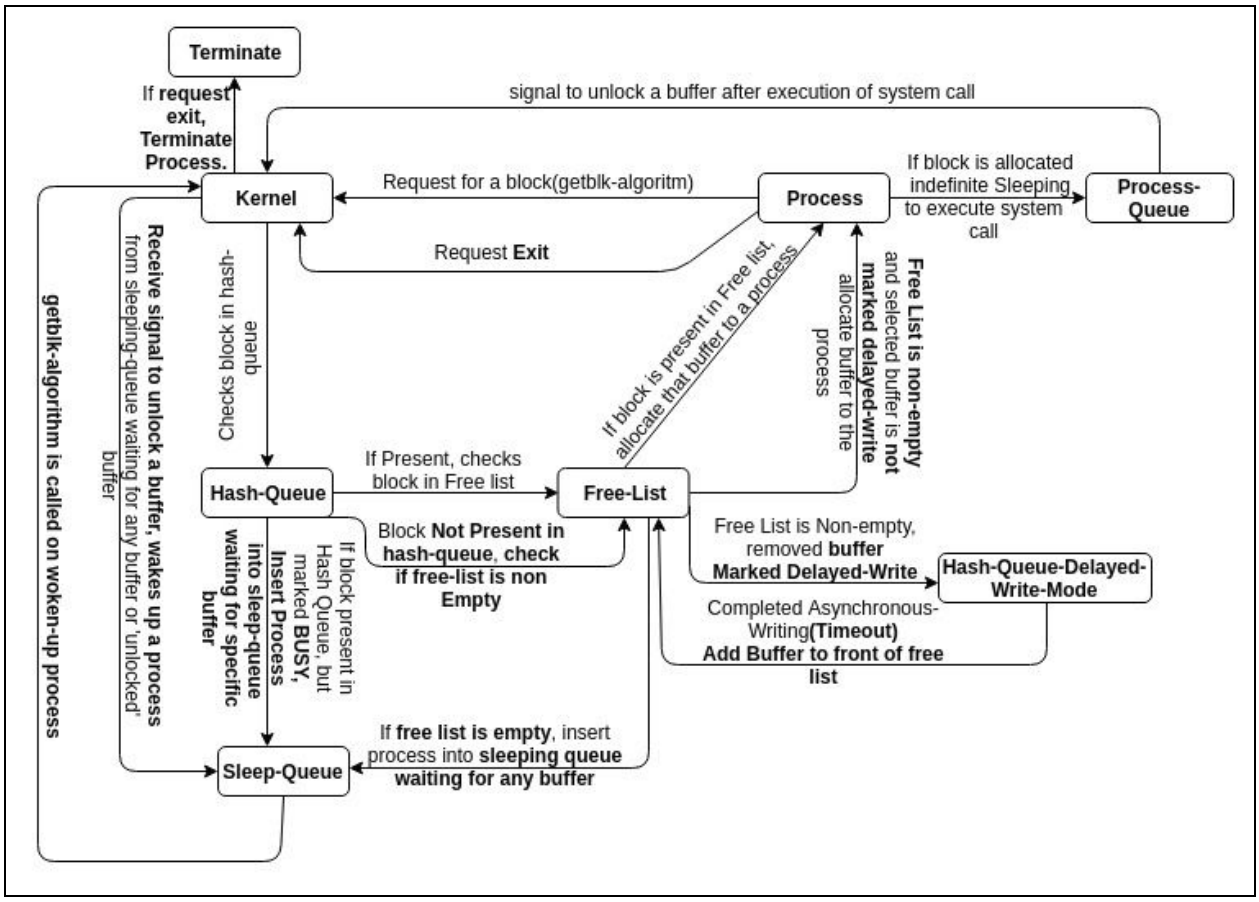
When time-taken becomes greater than '0.0020' or '2-sec', we have assumed that buffer has completed its asynchronous writing. At this time, buffer is inserted at the front of Free-list.

Brelse-Algorithm:

This algorithm allows a process which has completed its system call execution to unlock the buffer acquired by it. Here, we are choosing a random number between 0 and 1, a **random-no '0'**, indicates that released/unlocked buffer will be inserted at the end of free-list and marked delayed-write and **random-no '1'**, indicates that released/unlocked buffer will be inserted at the front of free list as it was previously marked delayed-write(based on assumption).

Brelse algorithm is solely responsible for selecting a process from sleeping queue(any/specific) and calling getblk algorithm on the block required by selected process as described in Scenario-1 and 2.

Overview:


Terminate
If request exit, Terminate Process.
signal to unlock a buffer after execution of system call
Kernel
Request for a block(getblk-algoritm)
Process
If block is allocated indefinite Sleeping to execute system call
Process-Queue
Request Exit
Checks block in hash-queue
Receive signal to unlock a buffer, wakes up a process from sleeping-queue waiting for any buffer or 'unlocked' buffer
getblk-algorithm is called on woken-up process
If block is present in Free list, allocate that buffer to a process
Free List is non-empty and selected buffer is not marked delayed-write
allocate buffer to the process
Hash-Queue
If Present, checks block in Free list
Free-List
Block Not Present in hash-queue, check if free-list is non Empty
If block present in Hash Queue, but marked BUSY, Insert Process into sleep-queue waiting for specific buffer
Free List is Non-empty, removed buffer Marked Delayed-Write
Hash-Queue-Delayed-Write-Mode
Completed Asynchronous-Writing(Timeout) Add Buffer to front of free list
If free list is empty, insert process into sleeping queue waiting for any buffer
Sleep-Queue

Individual Contribution:

Manisha Paruthi

Contribution:

- ➔ Project Brainstorming
- ➔ brelse-algorithm implementation
- ➔ in-building core-concepts of how things to be done using Sockets.
- ➔ ReadMe and Documentation

Learning-Experience:

- ➔ learnt how to make a 'readme.md' file
- ➔ learnt how to create a multiprogramming environment, using socket programming.
- ➔ read about how to create user-defined header files in C and how to compile and link them together.

Priyanka Sukhalani

Contribution:

- ➔ Project Brainstorming
- ➔ getblk-algorithm implementation
- ➔ Signal generation
- ➔ makefile and Documentation

Learning-Experience:

- ➔ learnt how to make a 'makefile' for distributed files.
- ➔ learnt how to create a multiprogramming environment, using socket programming.
- ➔ read about 'signals' and 'named-pipes'.