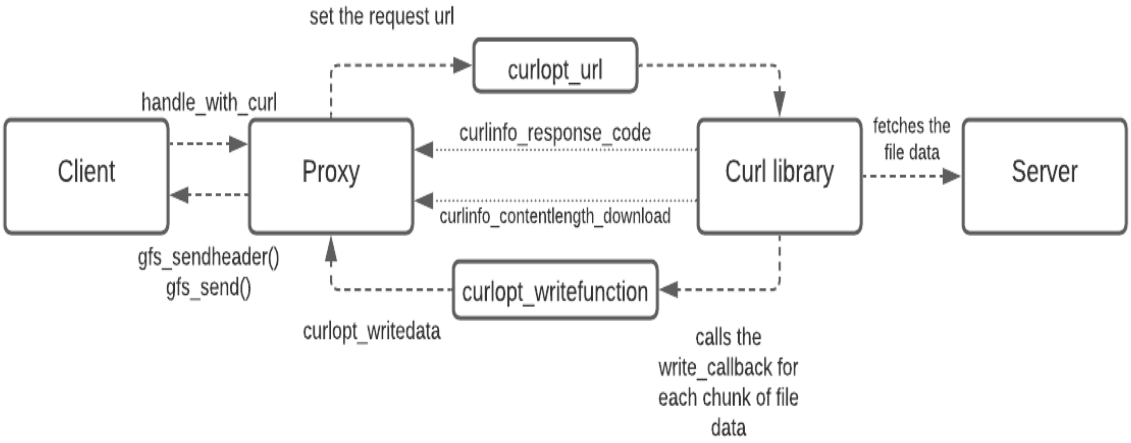
README

Introduction

This Readme file is a consolidated documentation of the two sections of project III.

1. Proxy server
2. Cache server

Part I : Proxy Server



The proxy server implementation has the multithreaded proxy server acting as a mediator that handles the requests sent from client to the server. The handle curl method of the proxy utilizes the libcurl library functions to query the webserver and procure the file. The request URL is registered with the libcurl easy set option CURLOPT_URL. After the curl perform operation is successful (CURLE_OK), the response code and the file length are fetched using the curl handle options CURLINFO_RESPONSE_CODE, CURLINFO_ CONTENT_LENGTH_DOWNLOAD.

Based on the response code from the server, the proxy sends the appropriate header to the client using gfs_sendheader. For each chunk of file data read by libcurl, the write call back that was set in the proxy using the CURLOPT_WRITEFUNCTION is invoked and the buffer registered with CURLOPT_WRITEDATA gets filled will file content. The

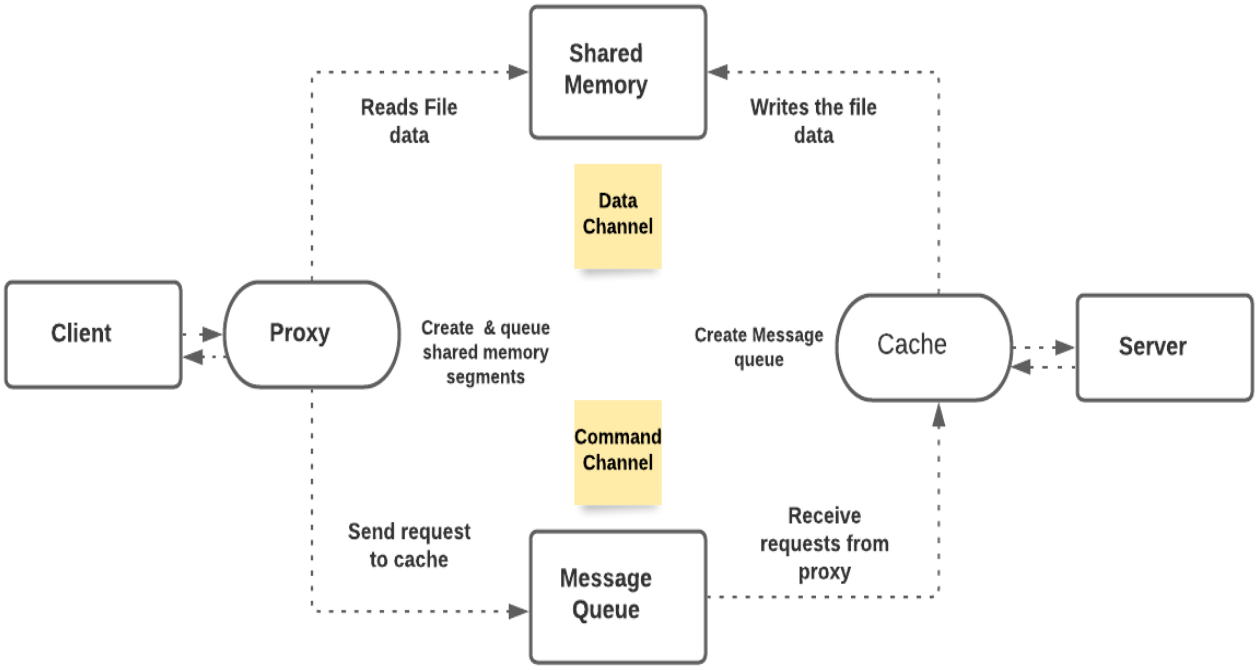
buffer containing the file content is then passed to the client using gfs_send .The process is repeated until the all the file contents are read and delivered.

Tests done locally:

1.The proxy server was tested with varied number of requests from client against a single proxy thread. Appropriate response was sent to the client based on the requested file.

2. Varied number of proxy threads serving varied number of client requests. Files requested were successfully downloaded.

Part 2 : Cache Server



In this part of the project ,implemented file transfer between the client/server via proxy/cache using Inter-Process communication.

The Multithreaded client sends requests to the proxy (Multithreaded) which in turn passes them to the cache via Command channel. The multithreaded cache upon receiving the request, serves the file contents via the Data channel.

Project Design :

Command channel : **POSIX message queue**

Data channel : **POSIX shared memory**

Synchronization between Proxy threads : **Mutex and condition variables**.

Synchronization between Processes (Proxy – Cache) : **Semaphores.**

Implementation

The proxy server upon initiation , creates the shared memory segments (shm_open) and resizes (ftruncate) them to the segment size mentioned. The shared memory is mapped onto the proxy process using mmap. Since pointers created in the address space of one process is not visible to another process , all the metadata and sync constructs are created as part of the shared memory. To fully utilize the segment size for storing file data, the shared memory data structure has a flexible array member along with other metadata information . So, the total size of the segment can be the sum of size of metadata and the provided segment size. Successfully mapped shared memory pointers (casted to struct with distinct segment id) are placed inside a steque. The steque is protected by mutex and condition variable to facilitate synchronization among the proxy threads. Each request from the client is tagged with a shared memory pointer retrieved from the steque and sent to the cache via message queue. After completion of file transfer, the shared memory pointer is enqueued back on to the steque to serve other incoming requests.

Since the number of requests that can be concurrently processed is dependant upon the number of available free segments, at any point of time when all the shared memory segments are busy serving requests, the proxy threads would have to wait for a free memory segment to be added onto the queue before proceeding to send the next request to the cache.

The cache process is implemented as a boss – worker model. The boss thread creates the message queue and the worker threads. Each worker retrieves a request from the

message queue in the worker procedure and begins to process them. The IPC mechanism message queue is multithread safe and therefore there was no need of synchronization between the cache threads when they access it.

Semaphores are used as a signalling tool between cache and the proxy. The cache server implementation uses three semaphores - file status and file size notification , signal to proceed and signal to pause between reading and sending of file data from cache ->shared memory and proxy->client.

Each request utilizes a shared memory segment during the duration of file transfer. The cache threads map to the shared memory segment pointed to by the metadata information from the proxy request. Resizing of the segment is done according to the size mentioned. If a file requested is not found, the file status is updated accordingly in the shared memory and the proxy is notified using semaphore. The fields in the shared memory pointer are cleared and it is enqueued back to the steque and the proxy threads are signalled.

If a requested file is found, the file size and status of the file is updated on the shared memory and the proxy is notified using the file length semaphore. The reading of file data begins on the cache side and a chunk of file content is loaded onto a char buffer and mem copied into the data buffer in shared memory. The cache semaphore is put on wait and the proxy semaphore is signalled to proceed .Once the proxy finishes sending the chunked file data to the client, the cache semaphore is notified to read the next chunk of file content. The process is repeated until the bytes read on the cache side and the bytes sent on the proxy side match the file size.

Upon successful file transfer, the memory pointer is added back to the steque so that they can be reused by proxy threads to process other incoming requests.

Tests done locally:

1. Started the implementation with single threaded client, proxy and cache. Tested if a single request from client is successfully processed for cases GF_OK and GF_FILE_NOT_FOUND and file is downloaded successfully.

2. After implementing the proxy, client and cache multithreaded, tested with varied number of requests and threads, different number of segments and segment sizes.
3. Tested for case where no. of requests > no of segments, so that the segments that are dequeued and re-enqueued back to the steque are successfully reused by the proxy threads.

References:

Proxy server

- Curl.haxx.se. 2020. *Libcurl Example - Ftpuploadresume.C*. [online] Available at: <https://curl.haxx.se/libcurl/c/ftpuploadresume.html> [Accessed 31 October 2020].https://stackoverflow.com/questions/290996/http-status-code-with-libcurl
- C++, S., Isaksson, J., Laagland, M. and Köhler, U., 2020. *Save Curl Content Result Into A String In C++*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/9786150/save-curl-content-result-into-a-string-in-c> [Accessed 31 October 2020]
- Curl.haxx.se. 2020. *Libcurl Example - Getinmemory.C*. [online] Available at: <https://curl.haxx.se/libcurl/c/getinmemory.html> [Accessed 31 October 2020].

Cache server

- O'Reilly Online Learning. 2020. *The Linux Programming Interface*. [online] Available at: <https://learning.oreilly.com/library/view/the-linux-programming/9781593272203/xhtml/ch54.xhtml> [Accessed 31 October 2020].
- Users.pja.edu.pl. 2020. *POSIX.4 Message Queues*. [online] Available at: <https://users.pja.edu.pl/~jms/qnx/help/watcom/clibref/mq_overview.html> /> [Accessed 31 October 2020].
- Cs.kent.edu. 2020. *Message Queues*. [online] Available at: <http://www.cs.kent.edu/~ruttan/sysprog/lectures/shmem/msqueues.html> /> [Accessed 31 October 2020]
- https://www.youtube.com/watch?v=ukM_zzrIeXs
- Generated, m. and WYSOCKI, K., 2020. *Mq_Receive() Doesn't Return After Signal Has Been Generated*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/60062406/mq-receive-doesnt-return-after-signal-has-been-generated>/> [Accessed 31 October 2020]
- Gist. 2020. *POSIX Shared Memory IPC Example (Shm_Open, Mmap), Working On Linux And Macos*. [online] Available at:

<https://gist.github.com/garcia556/8231e844a90457c99cc72e5add8388e4> [Accessed 31 October 2020]

- Kohala.com. 2020. [online] Available at: <http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf> /> [Accessed 31 October 2020].
- array, C., Norum, C., SB, K., Moscow, V. and Pulley, R., 2020. *Copy Struct Into Char Array*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/26430446/copy-struct-into-char-array/> [Accessed 31 October 2020].
- GeeksforGeeks. 2020. *Flexible Array Members In A Structure In C - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/flexible-array-members-structure-c/> [Accessed 31 October 2020].