CS 550

Sanchayan Maity
CWID A20340174
Section 03 - IITC, Bangalore

Assignment3: Design Document

November 5, 2015

1 Server Design

The server is designed using an event based and threaded producer consumer model.

1.1 Server Thread

The main function on start creates a file descriptor by listening on the given IP and port combination. The listen descriptor returned by the *tcp_listen* function is passed on to the server thread. This is the thread in which the main epoll based event loop is run. An epoll descriptor is created and we specify the EPOLLIN to detect incoming connections before adding it to the epoll descriptor. We wait for maximum 16 events as we know for our purposes we have eight nodes so we just specify double this number to be on the safer side. The wait for events is done inside the infinite for loop.

For each event occured, we check for first if an error occured or the connection was closed with the EPOLLERR, EPOLLHUP and !EPOLLIN check. If this has occured then the said descriptor is closed and the relevant variables set accordingly.

Next check is for if the event occured on the listen descriptor which was added before entering the event wait loop. If the event occured on the listening descriptor then we have an incoming connection. We accept this connection with a call to *accept* and then add this descriptor returned by accept to detect incoming request from other nodes. Note that while the listen descriptor was made nonblocking in *tcp_listen*, the descriptor returned by *accept* is not made non blocking. The reasoning for this is that the incoming connections are being waited on and will be signalled by the *epoll_wait* call and the *accept* call should not block which would have been the case had the listen descriptor been blocking. If the *accept* call blocks the *epoll_wait* utilised for the event loop will be useless as the *accept* will indefinitely block in this circumstance waiting for an incoming connection. The descriptor returned by *accept* is not made non blocking as we use non blocking IO calls through out the program. While adding the descriptor returned by *accept* to epoll, we specify the flags EPOLLIN, EPOLLHUP and EPOLLERR as we want event notifications for incoming request, disconnection notification and error notification respectively.

1.2 Workqueues

If it is neither of the two conditions above then we have incoming request from another node. The descriptor related to the incoming request is first extracted and then deleted from epoll wait. This is done because epoll should not wait for any events on a descriptor on which another threads are operating on. This descriptor is then passed on to one of the workers. Memory allocation is done for the job and this is then added to the workqueue which is a linked list maintaining the jobs submitted. Once the job is submitted, a conditional signal is signalled on which the worker threads are waiting. The thread wakes up, removes the job from queue of waiting jobs and calls the job function which was assigned for prcoessing the incoming request. Once the request is processed, the descriptor that was passed as data for reading from/writing to is now added back to the epoll descriptor as we would like to process incoming requests on the same descriptor/connection in the future. The number of workers should be primarily decided based on the number of CPU cores in the system. This can be controlled and changed with the *NUMBER_OF_WQS* variable specified in the *common.h* header file. After changing recompilation should be done for it to take effect. The workqueues and the worker threads are allocated and initialised at start up.

So we have primarily an event based multithreaded producer consumer model. epoll provides the event based implementation while acting as producer from one thread while other worker threads from the workqueue implementation act as the consumer.

2 Peer Design

The peer functionality is run in the main thread providing the functionality of registration, search and obtaining the file from other peers. Files are used as keys and peer id viz. peer ip and port combination are used as value while doing the registration. The replication is done by just replicating the files on the next node. In case the node goes down or is killed a read from an existing connection returns 0. This is used to detect that the node is no longer up and get the required file from the node next to the one where the file actually existed. It is assumed that the next node is up and running.

3 Hash Table

Both the peer and server use the same hash table implementation. Jenkins hash function is used as the hash function. For the peer, the returned hash value is mod with the number of servers paritcipating and for the server it is the hash table size which the server maintains. The hash table is of fixed size and dynamic resizing is not implmented. In case of collision, where two values hash to the same location, chaining is implemented using a singly linked list implementation.

4 Trade Offs

- The current implementation for *register* is sub-optimal. The registration of files was retrofitted into the second assignment implementation. So we iterate over all the files in the directory and call the primary registration function for each file. This results in the registration process taking more time than it should as the complete registration packet is constructed and send each time for a file. The correct way would be to send all the file list in a single registration command message payload.
- The payload length has still been kept same and we send 1500 bytes each time even if we might have only five bytes or so to send, a remnant of the second assignment implementation.

5 Scope for Improvements

- While the multithreaded workqueue based epoll implementation is almost the best server design possible, there is one more level to improve. Use of edge triggered IO using the EPOLLET flag and non blocking IO. Since converting all the read/write calls in the application to non blocking would require some more validation and testing, this has not been carried out at the moment.
- The hash table size has been kept fix in size for simplicity. However the underlying hash table implementation can be easily made better using high performance implementations like tommyds, uthash or concurrency kit.