Key/Value Server

TCP Sockets with Thread Syncronization

Nate Fear
University of Bristol
Email: nf16713@my.bristol.ac.uk
Candidate Number: 32583

***Abstract*—this report documents the way in which mutexes and condition variables were used to address the key/value server problem to provide an efficient and synchronized solution.**

Keywords—TCP Sockets; Server; POSIX Threads; C;

I. Introduction

The key/value server problem requires the safe synchronization of a main thread with four worker threads so that many clients can access and manipulate the data store concurrently. This report will detail the solution.

II. Main Thread

A. Main Design

The main thread is responsible for accepting new connections on the control and data port as well as processing control port commands. It begins this task by creating and binding to the control and data port sockets, this produces a file descriptor for each socket. The four worker threads are then created so that they may process data port commands later. The aforementioned sockets are then set to non-blocking so that the thread will not block until data has been read from a file descriptor, the sockets are then listened to. This allows incoming connections to be accepted on the control and data socket, once in place an instance of poll is created to monitor these sockets for incoming connections. Epoll is used in edge triggered mode so that epoll only wakes when there is a change on the file descriptor, this is because the solution does not pass the accepted file descriptors off to the worker threads to handle the accepted client until they disconnect. It instead uses epoll monitor the file descriptors of all the connected clients as well as the control and data port, so that the server is able to accept and serve more than four clients at a time by using a queue to store the file descriptors of the clients who have data to read and hence command to process. This is much more user friendly because no clients are left waiting to connect.

Upon epoll waking on a control or data port the connection is accepted which produces a new file descriptor, this new file descriptor is then added to epoll as an event for epoll to monitor. When new data is available on a file descriptor that was accepted via the data port, the file descriptor is added to the linked list and a worker thread is signalled to wake and read from the file descriptor to process the command. However if epoll wakes to a file descriptor which was accepted on the control port the data is read and the command is processed as part of the main thread loop, a single command is processed before closing the file descriptor to disconnect the control port client. The commands can consist of count or shutdown where count lets the user know how many items are stored within the data store and shutdown closes the data port and terminates the server when all remaining data port clients have disconnected. Whilst this is easy to perform when the final client closes the terminal rather than using the close command because this causes epoll to wake to a EPOLLHUP event so the main loop can exit, it is more difficult when the final client disconnects via the servers close command because epoll will be waiting with no epoll event left to wake it. To overcome this issue the self-pipe trick has been used where the read end of a pipe has been added to epoll for monitoring, so that if the final client uses the close command to exit a write to the pipe can be performed in order to wake epoll so the main loop can be exited.

B. Main Locking

A mutex is used within the main thread so that the thread can safely update the amount of work available, add work to the list and signal locking. Locking on these operations prevents any workers from trying to read a partially updated value and ensures the worker won't go back to sleep after signalling because the same lock is required by the worker thread when it wakes, which means the worker thread will block whilst it waits for the lock. Locking is also used when the control port commands shutdown and count are executed, within shutdown so a scheduled shutdown global flag can be updated safety and within count so the data store can be accessed safely. Lastly, locking is used to update the final shutdown flag as well as to signal the workers to wake as this prevents any other threads accessing the flag whilst it is being updated and causes the workers to block when they're woken in order to wait for the lock.

III. Worker Thread

A. Worker Design

The worker thread consumes file descriptors which have data to read, the file descriptors are stored in a linked list and the size of the linked list is recorded. This is so that the worker

thread can check if there is work available or not, if there isn't any work available the thread will sleep till it receives a signal from the main thread indicating work is available. However, if there is work available the thread will remove the file descriptor from the linked list, read all the new data on that file descriptor then parse the data and process the command. If the command is invalid the appropriate error message will be written to the client. However, if the command is put, get, count, delete, exists or an empty line the appropriate command will be executed and the result written to the client before the worker goes back to waiting again.

B. Worker Locking

Mutexes are used within the worker thread to protect access to the counter which keeps track of the amount of work available, this is so that two threads cannot check it at the same time and both attempt to proceed with reading the same file descriptor. Access to the linked list is also locked so that when one thread is removing a file descriptor from the linked list the main thread can't add a new file descriptor to it, this is so a data race does not occur. Lastly, locks are also used to prevent data races when performing commands because they may occur within the data store if two users try and perform simultaneous commands on the same key without locking in place.

IV. CONCLUSION

The solution handles connections from up to 60 clients, performs client commands safety, shuts down correctly, uses error handling and is robust to the telnet escape keys which can cause rogue SIGPIPE signals .