Christos Mourouzi
Canditate Number 33747

Server Software CW2-Report

In this coursework a server is implemented by using socket programming and POSIX threads. An administrator is connected to a control port and clients connect to the data port. The program can be executed as *./server CPORT DPORT*. In order the server to be synchronized the methodology that is used is described below:

Main Thread: The main function of the program is firstly responsible for creating a thread pool, creating the sockets (control and data), binding them and then listening for incoming connections. Then the main function enters an infinite loop and by using the poll() function waits (blocks until an event occurs) for incoming events (connections) and handles them according to the port they occured.

- **Data port:** If a client tries to connect to the data port, the main thread accepts the connection and saves it to a fixed-size queue. Afterwards, a signal is sent to a thread informing it that there is an available connection (by setting a connection flag predicate to true and then send a condition signal so only one thread wakes up). If there are no available threads (checked by a global variable (full) handled by the worker threads) at the moment it continues to save the accepted file descriptor to the queue but does not pass it to a worker. All the global variables (full flag, connection flag, fixed-size queue, counter of the queue) are protected by a mutex so they cannot be accessed by the worker threads and lead to a data race. After a data connection is handled, the main thread is waiting for another event to happen.
- **Control port:** If an administrator tries to connect to the control port, the main function accepts it immediately. The administrator can connect whenever he wants (priority) as the main function is not blocked anywhere while accepting a data connection. The administrator then just gives a single command (count, invalid or shut down) to the server and the server responses with the proper message, closes the connection and waits for another event to happen. If the command is COUNT then the KV store is accessed, protected by a data mutex lock with the server writing to the administrator's terminal the current number of items stored in the database. If the server gets a SHUTDOWN command it changes a shutdown predicate to true and broadcasts a signal to all threads to inform them that a shutdown command occurred and then breaks from the infinite loop. Then it just waits all the threads to join after finishing their last job. If some threads are waiting for a connection, they join immediately. Finally the main thread prints a shutdown message and exits.

Worker Thread: When the worker threads are created they enter an infinite loop and wait for upcoming jobs. At first, they all wait (condition wait) on a connection flag predicate and a shutdown predicate. When a signal is sent, a thread wakes up and checks if a shutdown command occurred. If the shutdown flag is true then it breaks from the infinite loop and exits. If the event that woke up the thread was an incoming connection then the worker immediately changes the connection flag to false (so after unlocking the mutex the other threads will be waiting for the next connection) and gets the next available file descriptor from the fixed-size queue by using a file descriptor counter (how many connections were handled by all threads). Each time a thread is accepting a connection it increases another counter that is equal to the number of concurrent data connections. If that counter is equal to the number of the workers then the full flag is changed to true to inform the main thread that no more jobs can be passed to a worker. The access to the flags, to the counters and to the fixed size queue is protected by a mutex lock to avoid data race. Afterwards, a welcome message is printed to the client's terminal and the worker enters an infinite loop to handle KV store's commands. At each time a KV store command is read, the access is protected by a data mutex to avoid data race. The server responses the appropriate message until it receives an empty line. If that happens, the worker thread exits the database infinite loop, closes the connection, decreases the concurrent connections counter and changes the full flag to false. If the main thread was accepting connections while all threads were working concurrently then the connection flag is set to true. By that, if a thread is waiting for a connection it will get it without waiting a signal from the main thread. In that way if more clients trying to connect at the same time they get a connection in a row with a FIFO order.