Lab 2 - Channels and Shared Memory

William Li

Overall Solution Discussion

Abbreviations:

- DFS = depth-first-search
- BFS = breadth-first-search

In this lab we leveraged parallelism to brute force a JWT secret. My solution uses the threadpool Rust crate, instead of spawning threads and assigning work manually. This makes the code simpler.

In my program, an iterative deepening DFS algorithm is used to traverse the solution space for the secret, creating work as it goes (found in *check_all()*). The created work is to call the *is_secret_valid()* function, and to communicate (via channel or shared memory) that the solution has been found in the case that the secret is valid. This work is consumed by the threadpool, which has been initialized to the size of *num_cpus::get()*, parallelizing the many computations of HMAC-SHA256 and signature verification.

During each iteration of the traversal algorithm, and also after the entire solution space has been traversed (all possible secrets of length up to max length provided), there is a check to see if a worker thread has found the correct secret and communicated it (via channel or shared memory). If a solution has been found, the threadpool is joined via *join()* to ensure all threads finish, and *check_all()* returns with the solved secret.

Initially, the recursive DFS algorithm used in the default code was changed to a queue-based BFS algorithm to traverse the solution space. This was because a DFS algorithm would at first miss even the smallest solutions such as the string “b”, in favor of verifying a string of max length first. This seems a little unintuitive to me. More importantly, it also means that if the max length parameter provided is a large over-estimation of the true length of the secret, a DFS algorithm will perform extremely poorly in comparison to a BFS traversal. A BFS traversal would verify the smallest possible secrets first, no matter what the max length estimate provided is.
The BFS algorithm performs quite fast, however I noticed that for longer secret lengths the memory requirements for the program were too intense. This is because BFS has larger space complexity than DFS, which resulted in the queue blowing up in size when secret lengths were larger. Finally, I decided to refactor the traversal to use iterative deepening DFS. Iterative deepening DFS is able to perform well with large over-estimations of max-length similar to BFS, however it reaps the benefits of DFS space complexity. DFS memory requirements scales off of

depth instead of number of nodes, so the iterative deepening DFS traversal results in much more improved memory usage.

Message-passing

To implement message passing, channels from the crossbeam crate were used. Whenever work is created to be executed by the threadpool, a channel send end is cloned and moved into the work. This send end is then used by a thread to communicate when a valid secret solution is found. The message sent through the channel is the solution itself, which is then received by the main thread to output to the command line as the answer.

One thing to note is that while the traversal is still going, the presence of a solution message is checked using the *try_recv()* function. On the other hand, when the traversal is complete and the main thread is waiting for the threadpool to finish executing the rest of the work, the *recv()* function is used instead, which is blocking, ensuring that the code waits for the secret answer to be found.

Shared-memory

For shared-memory, the code is very similar, except for the method of communicating between threads when a solution is found. Instead of using channels, a shared variable is used between all the threads which can be modified. This shared memory is locked behind a mutex and an atomic reference count to ensure safe read and writes between all threads. At first, this variable is initialized to be an empty vector/string. When a thread finds the solution, it unlocks the mutex and writes the answer to the shared variable. Then, when the main thread wants to check if a solution has been found, it unlocks the shared memory itself and checks if it is no longer empty.