ECE 459: Programming for Performance
Assignment 2

Zahin Mohammad

February 22, 2021

1 Problem

The goal of this lab is to compare sequential, and concurrent code execution with information sharing, to solve a computational intensive problem. The computationally intensive problem in this lab is cracking a JWT secret. More on the JWT standard can be found on the Wikipedia page.

2 Solution Space

For a JWT secret solver, the solution space is all the valid combinations of characters that can be a secret. For this lab, the solution space was bounded by only allowing an alphabet with 36 characters and specifying a maximum secret length. At run time, the maximum number of characters that can be used for a solution needs to be provided. With these two pieces of information, the solution space can be bound to

$$n^a \tag{1}$$

unique solutions, where n is the max length of a secret key, and a is the length of the alphabet of valid characters.

3 Solutions

The three methods of solving the JWT secret are sequential, threading with message passing, and threading with shared-memory. All three methods where tested using the same JWT token, with the same alphabet, and same maximum secret key length.

$$\begin{aligned} signature &= shNrMqeoWA3La5bOmJ9rzGtX8rh4M9fR93HVbE3JQTA \\ secret &= 0123 \\ alphabet &= abcdefghijklmnopqrstuvwxyz0123456789 \\ max\_secret\_length &= 5 \end{aligned}$$

The full JWT token can be seen in *test_cases.txt*.

mean (s)	stddev (s)	median (s)	min (s)	max (s)
71.96917534250001	4.939484706466287	70.27418282469999	66.1442981547	78.74343373369999

Table 1: Benchmarking results for sequential

mean (s)	stddev (s)	median (s)	min (s)	max (s)
12.150998533085	1.7482438533646705	12.524463474285	9.656861917785001	14.564476260785

Table 2: Benchmarking results for message passing

3.1 Sequential

The sequential method of solving the JWT secret does not use threading (in other words the program is single-threaded). The solution is to check all possible combinations for a secret by recursively building a secret with a growing window. The solution will start with an empty string, then check the first valid alphabet character (i.e a), then the first valid character twice (i.e aa). The solution will do a DFS on the solution space until it finds a solution.

3.2 Message Passing

In the message passing solution, the solution space had to be split to make use of the performance boost from multiple threads. In this case, each thread started their DFS solution (similar to Sequential), but instead of an empty string, each thread starts starts with a character from the alphabet (no two threads start with the same character).

Additionally, each thread gets a copy of a transmitter and receiver (multiple producer, multiple consumer). In each thread, during the DFS, before a new solution is tried, the threads will check the receiver to see if any other threads have already found the solution, in which case they would exit early. If a given thread finds the solution to the JWT secret, it will send the solution through the transmitter, and exit. With this method, all threads will exit when any single thread finds a solution. The main thread, upon joining the spawned threads, will collect the solved JWT secret from the receiver.

3.3 Shared Memory

In the shared memory solution, the solution space is split the exact same way as the message passing solution. The difference between the two methods is in how they communicate between threads. Instead of a receiver and transmitter channel, each thread communicates using a shared variable holding the JWT Secret (if it has been found yet). Access to the shared memory is guarded using a mutex. Similar to message passing, in each thread, before checking a new solution, the threads will lock on the mutex guarding the shared variable to see if any other threads have found a solution yet, in which case they would exit early. If a given thread finds a solution, it will acquire the lock on the shared variable and update it to hold the correct JWT secret, and then exit. In this solution, all threads will exit when any single thread finds the correct JWT secret. The main thread, upon joining the spawned threads, would lock on the shared variable and return the secret inside it.

mean (s)	stddev (s)	median (s)	min (s)	max (s)
12.26528417234	1.2596351143094493	12.84050926814	10.13797419314	13.34330043414

Table 3: Benchmarking results for shared memory

method	mean (s)	stddev (s)	median (s)
Sequential	71.96917534250001	4.939484706466287	70.27418282469999
Message Passing	12.150998533085	1.7482438533646705	12.524463474285
Shared Memory	12.26528417234	1.2596351143094493	12.84050926814

Table 4: Benchmarking results for all methods

4 Comparison

The full benchmarking result for each method can be seen in their respective sections. Table 4 presents a subset of the data from those benchmarks for all methods for ease of comparison.

The sequential method took the longest to finish, which is expected as each thread (the single thread) has to potentially try every solution in the solution space. The shared memory and message passing solutions finish much faster than sequential and are very close in terms of run time. The message passing solution is slightly faster than shared memory. Message passing is likely faster as each thread does not need to wait on any locks, whereas the shared memory solution requires this.