**Portfolio Project 1**

Jacob W. Palmer

CSU Global

CSC450: Programming III

Reginald Haseltine

February 1, 2025

Concurrency allows multiple threads to be run at the same time, also known as multithreading. Multithreading is great as it enables programs to run faster due to multiple processes being run at the same time, which is where processors with multiple cores tend to shine. Where this becomes an issue for performance is the amount of power being used to run the threads and the number of cores being used. A processor has a finite number of threads and has a limit to the speed at which it can process information, like how an engine has a limit to how many revolutions per minute it can spin. The speed of a processor is the major blocker to how fast a process can be executed and completed, before it can take on a new task, adding to the overall time it will take to run a program. There is some nuance in that we do have the ability to overclock our CPUs, but this comes at the cost of using more power and, if you push it far enough, the life expectancy of your CPU. Where we can run into problems with concurrency is the resources and how they’re allocated. We can run into a resource contention problem, where two threads are trying to use the same space in memory, where the last thread to use the resource will be successful and the others will have to re-run. We can also run into another issue called synchronization overhead, where the threads aren’t synced until some point where the other threads are finished, created a gap in time where the processing must wait. In my program if I didn’t lock down each function with a mutex, and had probably 16 times the functions in there, I could run into some of these issues where there aren’t enough processors to run them concurrently. Without the mutex specifically, I ran into an issue where the countingDown function would start while the countingUp function was running, usually somewhere within the first 30% of it counting up.

A vulnerability not limited to strings, but specific to concurrency is that you’re more susceptible to denial-of-service attacks or DDOS. DDOS is where an attacker will send more data than your program can handle, causing a blockage in the data pipe affect, stopping services all together. The more threads that are running at the same time, the less work it is for someone to overwhelm your system. Previously I touched on how multiple threads might access the same data at the same time, which is another vulnerability that can be exploited. Data sharing the same space can lead to a type of spillage, where the information separately may not be harmful, but together it can be used for malicious intent. An example from the military is how a person’s name may be unclassified, the name of a base may be unclassified, and a date may be unclassified. When they’re added together, we can paint a picture of a specific person at a specific place and time, creating a possible target for adversaries. For this reason, we want to ensure that we avoid race conditions as much as possible.

In my program, the primary data type is an integer. A security concern can be integer overflow or underflow, leading to unsigned integers. This can be harmful because this can be used to circumvent the defined byte limits in a program, allowing more data to pass through undetected.