Rashed Elbaba

Luan Nguyen

Andrew Vick

Project 4 Write Up

Project 4 deals with multithreading and synchronization. This project allowed us to better understand how this works by creating a program to control traffic on an imaginary bridge with two lanes. The vehicles would be coming from 2 directions, north and south, and the type of vehicle would be decided by the choice of random. This is a perfect example of what multithreading is, which is defined as a program that contains 2 or more parts that can run at the same time. Each part of the program is considered a thread, which we can see as being the 2 lanes in this program.

Understanding thread synchronization in C, this is defined as a mechanism that will allow for 2 or more simultaneous threads that don't execute a specific segment known as a critical section. We want to make sure that if one thread is executing a critical section, we want the other to wait its turn, therefore not triggering a race condition.

Listed below are some of the functions and methods that were used to help complete the project at hand. These include:

* Included in the program is the use of appropriate libraries in assisting us to make sure the program runs efficiently as possible. These include:
  + #include stdlib.h, unistd.h, signal.h, and more.
* “Int main() {}” - where the work occurs. Needing the use of multiple threads to represent the vehicles of the north and south. Input from the user will be asked to enter in the number of vehicles.
  + The use of a while loop which will include:
    - Printing the total number
    - Printing the north and south bound ratio
    - Printing the delay time
  + An if loop that will:
    - Run a delay time function and adding to the counter that was called
  + A while loop that deals with the count of north and south bound vehicles and assigned temp variables to them.
  + A for loop that takes in 3 parameters:
    - An int tempId =0; tempId < totalNum[count];
  + A for loop that takes in 3 parameters:
    - Int a = 0; a < tempN; a++;
    - Assigning a temp value to a random call
    - A when a north vehicle has arrived, it will be assigned a tempId
    - If the tempId == 0, then it will be called a car with weight of 200
    - Else, the north vehicle arriving will be a van of weight 300
  + A for loop that follows the same instructions as above, but this time deals with vehicles arriving from the south bound.
* Threads will be created that will equal the same number of vehicles that were entered by the user.
* Threads will be initialized with the use of a for loop
  + 3 input parameters are taken in which include: int i = 0; i < tempN; i++
  + A thread will be created dealing with the number of north vehicles, as well as when they arrive and how many.
  + The for loops will be identical, the only difference is if it is for the north or the south.
* A thread will be created with the mutex lock to eventually destroy the lock of the north and south locks in place.
  + Count will be incremented by 1
* “#define \_\_\_” - our group needed to make sure that we had certain variable names when dealing with the number of cars and vans, and the directions they were going. A few examples of this can be seen as:
  + Need to set the max amount of cars allowed on the bridge at one time. “#define MAX\_WEIGHT 1200”, “#define CAR\_WEIGHT 200”,

“#define VAN\_WEIGHT 300”

* A “struct” that takes in vehicle specifics. Shown below:
  + Char type[10]; int id; int vehicleWeight;
* A static function that takes in parameter “vehicle” allowing if a vehicle can proceed or not. An if-else loop is used.
  + If the current weight of the vehicle is less than the max weight, the vehicle can proceed.
    - Else return the function returns false
* A void function that will take in a vehicle parameter that deals with the arrival of a vehicle. A while loop is used to make this happen.
  + While the vehicle is safe, and not 1, it will arrive at the bridge. A print statement will declare if the vehicle has arrived.
* A void function that will let the user know the vehicle is crossing the bridge.
  + Print vehicle is crossing the bridge, and then there will be a 3 second delay for the next vehicle.
* A void function that will let the user know the vehicle is leaving, and complete the crossing of the bridge.
  + Print vehicle has finished crossing the bridge, and behind the scenes the weight of that vehicle is subtracted from the current weight of the bridge.
* A void function that will deal with cars going northbound on the bridge. An input parameter of void type is taken in.
  + A mutex lock for synchronization to ensure that only 1 of the north and south vehicles can enter the bridge at a time.
  + Print “northBound” to let the user know which way the vehicle is coming, crossing function with vehicle will occur, then leaving vehicle function will occur, then the lock will be released.
  + These same steps and code will be used for the “southBound” function.

In conclusion, the user should get a good understanding of how multithreading and synchronization work. The randomization of vehicles coming to a bridge and making sure it does not go overweight, can be seen in this program. The use of bridge restrictions to make sure the traffic control policy is followed were used.