CS3307 – ASSIGNMENT 2 ANALYSIS

By: Alex MacLean & Ernie Kwan

Design Pattern 1: Singleton

This pattern was used for the debugger. When a call is made to the Debugger class it passes back a single instance of the object that all threads share. This is to ensure that calls to the Debugger are not lost or confused while different threads are using it. The constructor of the Debug class is also hidden so further secure the singleton design pattern.

- Debug
    - This class generates the html file with a report of the hosts and ports that have been scanned and are open
    - It is called from the main method of main.cpp to create the html page after all the ports have been scanned
    - Also used to write debug messages to the correct outputs when methods such as ProcessInformation(), ProcessDebugLine(), and ProcessErrorLine() are called from WriteLine() in the Debug class that is used throughout the other classes
    - AddHost() is called from portScan() in scan.cpp to add a host to be included in the html file that has open ports
    - WriteHtml() and WriteCss() contain the main logic to create the html page
    - WriteLine() takes as a parameter a message and a message type and calls the appropriate method out of ProcessInformation(), ProcessDebugLine(), and ProcessErrorLine()
    - Console output is done with threads, blocked off by mutexes so that they print in the correct order

Design Pattern 2: Consumer-Producer

This pattern was used for the port scanner. It was chosen to efficiently process information regarding the range of hosts and ports to scan given by the user. This info is placed in a data structure for the scanner to pull from when it needs to. The inclusion of the design pattern is helpful because it allows all the information to be passed to the data structure at once instead of passing a host and port to the scanner one at a time and waiting for it to complete the scan before passing the next host and port. For efficiency's sake, the modification was made to the mvc pattern so that everything's produced at one time.

- Host
    - Creates the vectors to hold the list of hosts and the open ports found during the scan, as well the corresponding iterators
- Scan
    - ParseHostRange() determines the list of hosts to scan given from the command line. It does this by scanning the range (e.g. 192.168.100-

200) up until the “-“ then it determines that 100 is the lower bound and 200 is the upper bound. For each IP address from 192.168.100-200 it adds the host to the list of hosts to scan (Line #63).

- ParsePortRange() determines the range of ports similarly. From an input such as “200-400” it will add 200 to the first element of an array and 400 to second element of the array. It will then check that the lower bound is less than higher bound and that they are both less than 35536, the maximum port range. If all is good then it adds the ports from 200-400 to the port list (Line #130).
- The constructor for Scan first checks that the parameter for the host range is either an appropriate range or a single IP address using regex. If it is suitable then ParseHostRange() is called and the list of hosts is populated and the amount of hosts to scan is determined. It then does the same thing for the port range. Finally it sets the maximum amount of threads allowed to what was given on the command line or if the thread max exceeds the actual number of hosts, it sets the thread count to the number of hosts instead. Default is 20.
- scan() is the producer in the consumer-producer pattern. It spawns the threads using a loop that is defined by the user (_hostCount), or by the default, 20 (_threadMax). Each thread spawned then calls scanHosts() from scanHostsDelegate()
- scanHosts() is the consumer. It calls portScan() for each host. Since each thread calls scanHosts we need to use mutex to make sure no two threads call the method concurrently. Only after all the hosts have been consumed do we break out of the loop.
- portScan() contains the main business logic of the port scanner. For each port within the port range it tries to connect to it on the current host. It first creates a socket descriptor by calling socket, then it tries to connect to the IP specified in hostaddr, if it returns -1 then the port is closed, otherwise the port is open. If the port is found to be open it adds the host to the Debug class’ list of hosts that have open ports.

Main.cpp

The main method of the program begins by printing a banner with the project info and if there are not 4 or 5 command line arguments it prints an error message regarding the usage of the program. It then attempts to create a Scan object and call scan() on it. If those steps are successful it then calls WriteHtml() from Debug to generate the report of open ports.

Questions

- How does _scanHostMutex.lock() work in scan.cpp?
- What does emplace_back do in scan.cpp?
- Why is the max amount of threads 20? What are the limitations?