Henry Chang, hechang@ucsc.edu
Andrew De Neve, adeneve@ucsc.edu
Aaron Doubek-Kraft, adoubekk@ucsc.edu
CMPS 109
11/25/2016

MIS Phase 2 Report

Makefiles/Building/Running::

- In order to compile the project, enter the Server and Client directories and build each executable with “make”. Run the Server, which will continue accepting input until terminated. Then, run a client with Client <filename>. The Client will send the file over to the Server, which will run it, and then send the files back. Our server supports multiple clients at a time.

Notes:

- The keywords that were not yet implemented as of our last submission have now been completed, as have the multithreading keywords.

Threading:

- A new Threading keyword parent base class has been added to deal with passing new lists/data structures held in the MIS object (such as the vector of threads, thread id counter, locked variables map). All the new Threading keywords inherit from this class.
- Process of creating a thread:
  1. the parser hits the THREAD BEGIN keyword
  2. Thread Begin object is created, making use of all resources in the MIS
  3. using the parser, the Thread Begin object collects all instructions until hitting Thread End
  4. instructions are placed in a list and a c++11 thread is created linking to a function called runThread.
  5. in runThread the thread strips the first instruction and checks if its locked, if not it calls doInstruction method with the striped arguments
  6. during doInstruction, a mutex is locked to prevent changes to the same variables
- Under the hood, the computer running the MIS will jump between all the different threads the user has declared in the .mis program rather than executing sequentially. Thread jumping is to only happen after full execution of an instruction.
- Additionally, the user must use BARRIER at the end of the .mis file to ensure that background threads are joined before main is terminated

Client/Server:

- Our Client/Server implementation uses TCP for communication. We chose TCP over UDP because we have no guarantees about the size of the files being sent through the sockets, or how much output the server will generate, so the stable, ordered TCP connection is worth the decrease in performance. UDP could be the superior option if the

size of the files was limited to the size of one packet. Our implementation makes use of Prof. Sobh's object-oriented TCP socket classes.

- The Server copies the program from the client into a temporary local .mis file, instantiates and MIS on it, and then runs the MIS. The MIS stores its output into files, which are then read by the server and sent back over the network to the Client. We chose this implementation in order to minimize the amount of re-implementation of the MIS.
- In order to accomplish multi-threading on the server, each socket and corresponding MIS environment is spawned inside a thread, which is then detaches from the main process and runs to completion, returning the output and error of its MIS session to its Client.