Synchronization using Path Expressions

- **Sadan Mallhi & Rahul Shevade**

The flow of the program is as follows: Any problem (such as readers-writers) to be run first calls the init_synchronizer() method with the path expression defined for that problem. Then, it begins spawning threads according to the test cases and each thread starts its assigned operation by calling the enter_operation() function with the operation name as a parameter. Once the thread is done it calls the exit_operation() function.

Synchronizer Implementation & Issues

The synchronizer must parse the given path expression to generate a valid entry and exit into each operation name being called by a thread. We generate a tree of nodes containing information about P(), PP(), V(), VV() operations along with the actual procedure name. For example, the leaf node, "P(s1) A V(s1)" tells us that for a thread to perform 'A' it must decrement 's1' semaphore if possible, and then only proceed. Else, it must wait. And once it's done, it must increment the semaphore. To track the semaphores, we create an array of them, and each node will store the relevant indices of the semaphore to the left and the right of it so that it can use or pass this information to the children.

To parse the actual path expression, we tried to go from left to right, but there are some operators that require us to look ahead where we haven't yet generated the synchronization variables which was causing an issue. Instead, we recursively generate nodes starting with the 'outer-most' operation in the path expression. We find this operation by looking whether '+' or ';' occurred before a pair of '{' & '}', or after, or in between and then decide what should be the root and the left and right child (if any). We assume there are no nested braces. If they are before or after the braces, then that is the root, and the left side is left child and right is the right child. If they are within braces, then braces are root, and it has only one (left) child. Multiple "path…ends" have not been taken care of in this implementation. We also don't handle regular parenthesis.

When a thread enters the enter_operation() with a particular procedure name, we have the parsed path expression stored and we look for the procedure. We

perform the P(), PP(), etc., that occur before this procedure in then tree. If all these synchronization operations succeed, then the thread can perform the operation it wants to. Otherwise, it must wait for another thread to help it proceed.

When a thread enters the exit_operation() with a particular procedure name, we perform synchronization operations that lies to the right of this procedure in the tree.

The leaf nodes contain the procedure names and the left semaphore operations and the right semaphore operations which is used to enter and exit.

Contributions:

Both worked together on the idea for the parser and implementing the code together.