CS4710
SPIN Group Project
Eric Zimmer
Anthony DuPont

Our Promela model is split up into 6 Sections

1) In the first section, a process is selected to initialize the array with distinct values while the other processes skip the section. The distinct values chosen are just array indices from 0 to N, where N is the number of processes. When the array is populated by the selected process, the 'arrayInitialized' variable is set to true and the process then continues to the second section of code.

2) In the second section, all processes wait until the 'arrayInitialized' variable is set to true. Once that happens, all processes move to the third code section. This is not a critical section because the 'arrayInitialized' variable being checked is not modified by any of the processes. Worst case, a race condition here only means a process iterates through the wait loop an extra time and the code is made more efficient by not putting a mutex here.

3) The third section is where the processes try to enter the critical section. It has an atomic block because a variable being checked if a process can enter the critical section is incremented if the critical section is available. If the process cannot enter the critical section, it waits until it can enter the critical section.

4) The fourth section is the critical section. A random number is selected between 0 and N, where N is the number of processes, and is used as an index along with the processes id to swap variables in the array. When the array's variables are swapped, the variable used to check the critical section is decremented and the process exits the critical section.

5) In the fifth section, all processes wait for every other process to reach the fifth section. Once all processes make it to the fifth section, they enter the sixth and final section.

6) The final section is cleanup and checking to make sure everything is still valid. All processes needed to finish their critical sections before the validity of the data in the array could be checked. The check makes sure each value in the original array is in the modified array, and that the order of the modified array is different than the original array.

As far as LTL statements go, we only have three, one of them is to make sure that there is never more than 1 process in the critical section at the same time. The second checks to make sure that all processes complete. The third checks to make sure that there are no “non progress” cycles, that is cycles that do nothing. We then have 3 assertions to make sure that the program is working as intended at specific points in the execution. The first assertion checks to make sure that after swaps the same values are still in the array. The second checks to make sure that the order of the array changed from initialization to where it is at the end. Finally the third checks to make sure that the correct number of swaps was performed, this is in order to make sure that each process executes.

Extra Credit

By adding in multiple swaps per process, our LTL properties are still satisfied. No errors are thrown by SPIN, and the program still functions as intended.

The proctypes do not need any further synchronization when each one goes through the critical section multiple times because no process can access the array while another process is in the critical section. Further synchronization between the processes does not provide any additional value.