BACI Projects for an OS Course

1. **Implementation of Machine Instructions:** Implement the exchange instruction in the text. The implementation of this instruction should be based on an ATOMIC function which returns a BOOLEAN value. You should test your implementation of the machine instruction by building a mutual exclusion protocol on top of your low-level operation.

2. **Implementation of Fair Semaphores (FIFO):** The default semaphore in BACI is implemented with a random wake up order. For non-terminating processes, this random behavior allows the possibility of starvation. A fair semaphore implementation, on the other hand, has a FIFO wake up order. Implement semaphores with this FIFO wake up order. Allow users to define a semaphore as a CONST value in the range [1..13]. This CONST value should be used as a tag for access to a semaphore and its corresponding variables. At least four procedures will be required in your implementation:

   - PROCEDURE `Create_Semaphores`();
     You should place your initialization code here.
   - PROCEDURE `Init_Semaphore`(*sem_index* : INTEGER; *val* : INTEGER);
     In this procedure, you should initialize the semaphore represented by *sem_index*. As in the given BACI semaphore implementation, your FIFO semaphore implementation should not allow the use of a semaphore unless it has been initialized.
   - PROCEDURE `FIFO_P`(*sem_index* : INTEGER);
     This procedure should include the ATOMIC compare-and-swap operation implemented above. If a calling process needs to be put to sleep, use the SUSPEND operation. The WHICH_PROC command helps keep track of sleeping processes.
   - PROCEDURE `FIFO_V`(*sem_index* : INTEGER);
     This procedure should REVIVE a process waiting on the semaphore represented by *sem_index* in a FIFO basis. If no such semaphore exists, this procedure should increment the value associated with the semaphore. Again, use the ATOMIC compare-and-swap operation to obtain mutual exclusive access to variables associated with this semaphore.

   Bear in mind that this code should be written as a system implementation and, as such, should handle all possible errors. You are responsible for producing code which is robust in the presence of ignorant, stupid, or even malicious use by the user community. Lastly, full credit will not be received if your solution has non-required waiting.

3. **Implementation of Unfair Semaphores:** Assume that the semaphore operation given in BACI does not exist. Implement semaphores based on Dijkstra's original proposal by randomly choosing which process to wake up when a signal occurs (E. Dijkstra, "Hierarchical ordering of sequential processes," *Acta Informatica*, vol. 1, no. 2, pp. 115–138, 1971.) The RANDOM(INTEGER) command returns a random number in the range [0..(INTEGER-1)]. Follow the information in the fair semaphores implementation project on the minimal number of procedures required for this implementation.