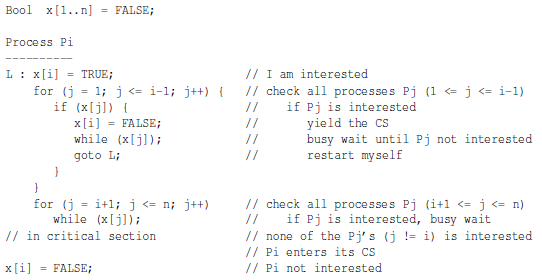
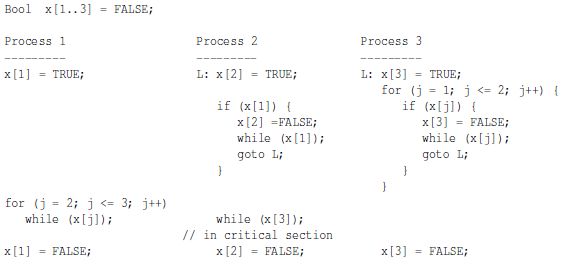
Fall 2015 Final Exam

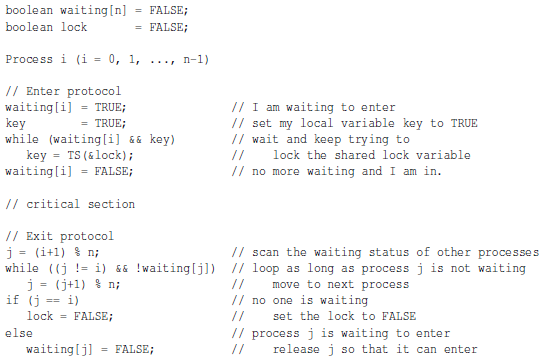
1. Define the meaning of a race condition. Use an execution sequence.
2. The following is a solution to the critical section problem for n processes where is a known integer. The general code for process is below.



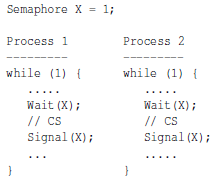
Convert the above code to a 3-processes version for . Prove rigorously that the above algorithm for the tree processes satisfies the mutual exclusion requirement.



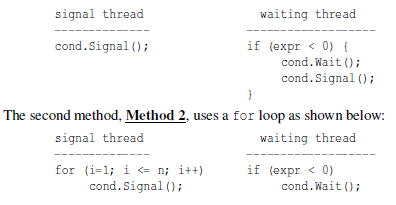
1. Consider the following solution to the critical section problem using the atomic TS instruction. This solution works for processes, where is a know integer. It was shown that this solution satisfies mutual exclusion. Show that this solution also satisfies progress and bounded waiting.



1. Consider the following implementation of mutual exclusion with a semaphore X. show rigorously that the above implementation satisfies mutual exclusion.

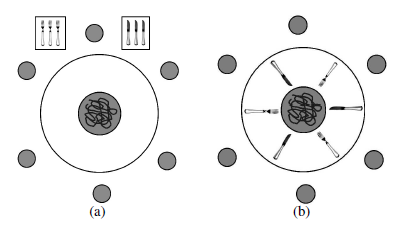


1. Suppose in a Hoare type monitor a thread waits on condition variable cond if the value of an expression expr is negative. The value of expr my be modified by other threads, and the thread that makes expr non-negative signals cond to release a waiting thread. There are two different ways to signal all threads that are waiting on cond. Suppose we know that the total number of waiting threads is n. The first method, Method 1, does not have to know the value of n and uses cascading signaling as follows. In this way, a released thread can signal cond immediately to release another.



Study both methods and answer this question: In a Hoare type monitor, which method or methods can guarantee that the expression exp is non-negative when a waiting thread is released?

1. Enumerate and elaborate all major differences between a semaphore wait/signal and a condition variable wait/signal.
2. Why is calling a monitor procedure from within another monitor (i.e. nested monitor call) not a good programming practice?
3. Three processes share four resource units that can be acquired and released only one at a time. Each processes needs maximum of two units at any time. We also know that these resource units must be used in a mutually exclusive way and that they are non-preemptable. Show that this is a deadlock free system. Hint: Think about the necessary condition of a deadlock.
4. There are six philosophers seated at a circular table. There are three knives and three forks available. The philosophers are quite hungry but require both a fork and a knife to eat.

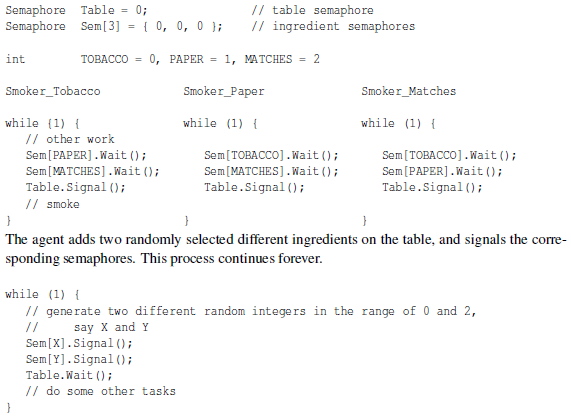


Do the following problems:

* 1. Each philosophers goes to the tray (Figure (a) above) and grabs any knife until he is successful. He then grabs any fork until he is successful. Once he has a fork and a knife, he eats and finally returns the fork and knife.
  2. If the forks and knives are arranged in an alternating way as shown in Figure (b), each philosopher flips a coin to determine if he is going to first try for a fork or for a knife. He grabs his choice until he is successful and grabs the other type of utensil until he is successful. Then he eats and returns the utensils after eating.

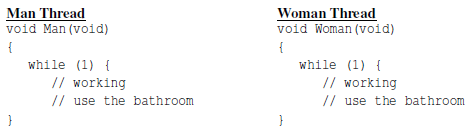
For each problem, answer if a deadlock is possible. If not, clearly explain why. If it can, use an execution sequence to show the existence of a deadlock.

1. Three ingredients are needed to make a cigarette: tobacco, paper, and matches. An agent has an infinite supply of all three. Each of the three smokers has an infinite supply of one ingredient only. The following solution uses three semaphores, each of which represents an ingredient, and a fourth one to control the table. A smoker waits for the needed ingredients on the corresponding semaphores signals the table semaphore to tell the agent that the table has been cleared, and smokes for a while.

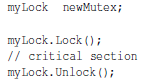


Show using execution sequences that this solution can have a deadlock.

1. What is rendezvous in message passing? You must provide the context of a rendezvous and a clear explanation.
2. A unisex bathroom is shared by men and women. a man or a woman may be using the room, waiting to use the room, or doing something else. They work, use the bathroom, and come back to work. The rule of using the bathroom is very simple: there must never be a man and a woman in the room at the same time; however, people with the same gender can use the room at the same time. Declare semaphores and other variables with initial values and add wait() and signal() calls to the threads so that the man threads a woman threads will run properly and meet the requirements. Your implementation should not have any busy waiting, race condition, and deadlock, and you should aim for maximum parallelism.



1. A party room has a capacity of m people. Each party goer can bring a number of people. However, if the sum of the number of people in a new group and the num of people in the party room exceeds the maximum capacity m, the new group will not be admitted and must wait outside the party room. People in the party room must also exit in a group. Design a Hoare monitor PartyRoom and monitor procedures Admit(u) and Exit(v) where u is the number of people in the group requesting to enter the party room and v is the number of people in the group leaving the party room. When a group of u people arrives, a representative of this group calls Admit(u) to make a request to enter. If this request cannot be granted, all u people wait as a group. When a group of v people leave the party, a representative calls Exit(v). Because the party room has v people less, your monitor must allow some (or none) of those waiting groups to enter and make sure the number of people in the party room will not be larger than m. You may assume that entering and exit groups form automatically and can be considered as a single unit.
2. It is not very difficult to see that the ENTRY and ACCEPT pair in Ada forms a many-to-one channel, where ENTRY and ACCEPT represent blocking send and blocking receive. We also mentioned that semaphores, monitors, and channels are equivalent to each other under the shared memory model. ThreadMentor has a synchronous many-to-one channel type. This channel uses the send and receive methods for sending and receiving messages. Based on the idea you learned from Ada and the many-to-one channel and thread capability of ThreadMentor, implement a mutex lock. More precisely, define a new class newMutex that has two methods Lock() and Unlock(). Then, a thread can use the following for mutual exclusion purposes.



Here are some notes.

* 1. The acutal ThreadMentor syntax is not so important as long as I know what your are doing
  2. You can only use many-to-one channels
  3. You must use the idea of Ada that was discussed in class
  4. Providing a convincing argument to show your implementation is correct