Solution

Series 5: Interprocess Communication

1. Race condition in bidding

a. Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. One strategy to avoid busy waiting temporarily puts the waiting process to sleep and awakens it when the appropriate program state is reached, but this solution incurs the overhead associated with putting the process to sleep and later waking it up.

a. There is a race condition on the variable highestBid. Suppose that two people wish to bid on an old Macintosh computer, and the current highest bid is $1,200. Two people then bid concurrently: the first person bids $1,300, and the second person bids $1,400. The first person invokes the bid() function, which determines that the bid exceeds the current highest bid. But before highestBid can be set, the second person invokes bid(),and highestBid is set to $1,400. Control then returns to the first person, which completes the bid() function and sets highestBid to $1,300. The easiest way of fixing this race condition is to use a mutex lock:

```
void bid(double amount) {
      acquire();
      if (amount > highestBid)
            highestBid = amount;
      release();
      }
```

2. Pizza restaurant

This seems like a fairly easy solution. The three assitants processes will make a pizza and serve it. If they can't make a pizza, then they will go to sleep. The chef process will place two items on the table, and wake up the appropriate assistant, and then go to sleep. All semaphores except lock are initialized to 0. lock is initialized to 1, and is a mutex variable.

Here's the code for the chef process.

```
1   do forever {
2     P( lock );
3     randNum = rand( 1, 3 ); // Pick a random number from 1-3
4     if ( randNum == 1 ) {
5        // Put tomato sauce on table
```

```
 6        // Put dough on table
 7        V( assistant_topping );  // Wake up assistant with cheese
 8      } else if ( randNum == 2 ) {
 9        // Put tomato sauce on table
10        // Put topping on table
11        V( assistant_dough );  // Wake up assistant with dough
12      } else {
13        // Put dough on table
14        // Put topping on table
15        V( assistant_tomato ); } // Wake up assistant with tomato
16      V( lock );
17      P( chef );  //  chef sleeps
18     }  // end forever loop
```

I will give code to one of the assistants. The others are analogous.

```
 1   do forever {
 2      P( assistant_dough );  // Sleep right away
 3      P( lock );
 4      // Pick up tomato sauce
 5      // Pick up topping
 6      V( chef );
 7      V( lock );
 8      // Serve.
 9   }
```

The assistant immediately sleeps. When the chef puts the two items on the table, then the chef will wake up the appropriate assistant. The assistant will then grab the items, and wake the agent. While the assistant is serving the client, the chef can place two items on the table, and wake a different assistant (if the items placed aren't the same). The chef sleeps immediately after placing the items out. This is something like the producer-consumer problem except the producer can only produce 1 item (although a choice of 3 kinds of items) at a time.

3. Mutexes and semaphores

Check solution on moodle.

4. Monitor

Check solution on moodle.