Stamatios Morellas
COM S 352 - Homework 6
Due 3/13/20 @ 11:59pm

6.12 (10 points)

A potential problem that could occur when using the function `getValue()`in this scenario is that if the `sem` variable ever becomes zero, a process has the potential to get blocked. If a process is executing and it waits for sem, it can be prevented from entering the loop again due to the next process modifying the value of sem to be zero, causing in a block of the initial process that was running.

6.26 (10 points)

The difference between the behavior of this operation among monitors and semaphores is that in its association with monitors, the `signal()` operation is not persistent. This means that if the function is called and there are no waiting threads, the operation will be completely disregarded by the operating system. In its association with semaphores, the `signal()` function will increment the value of the semaphore, regardless of if there are any waiting threads.

6.29 (10 points)

```
monitor fileshare {
    /* shared variable declarations */
    int sum = 0; // sum of all unique numbers
    int wait = 0; // num processes waiting on next
    condition m; // mutex
    condition nx; // next

    // called before accessing a file
    void access_file(int u) {
        m.wait();
        // sum < n constraint prevention
        while (sum + u > n) {
            m.signal();
            nx.wait();
            m.wait();
        }
```

```
        sum+=u;
        m.signal();
    }

    void release_file(int u) {
        m.wait();
        sum-=u;
        nx.broadcast(); // resume suspended
        wait = 0; // reset
        m.signal();
    }


    initialization_code() {
        int n;
        access_file(n);
        ///////////////
        release_file(n);
    }
}
```

7.9 (14 Points)

```
monitor bounded_buffer {
    int buffer[MAX_ITEMS];
    int num = 0; // number of buffer elements
    condition fill;
    condition empty;

    // Called by the producer to put item v in the buffer
    void produce(int v) {
        while (num >= MAX_ITEMS) {
            fill.wait();
        }
        buffer[num++] = v;
        empty.signal();
    }
```

```
    // Remove an item in the buffer and return it
    int consume() {
        int item; // return this
        while (num >= 0) {
            empty.wait();
            item = buffer[--num];
        }
        fill.signal();
        return item;
    }
}
```

5. (16 points)

Consider the following sleeping barber problem. There is a barber shop which has one barber, one barber chair, and N chairs for waiting customers. If
there is no customer, then the barber sleeps in his own chair. When a customer arrives, he has to wake up the barber. If there are many customers and the barber is
cutting a customer's hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty. Write a semaphore
solution to the sleeping barber problem. You should write two procedures: barber() and customer().

```
monitor sleeping_barber {
    int N; // number of chairs for waiting customers
    int waiting = 0; // total waiting customers
    semaphore mutex = 1;
    semaphore barber= 0;
    semaphore customers = 0;

    void barber() {
        while (true) {
            wait(customers);
            wait(mutex);
            waiting--;
            signal(barber);
            signal(mutex);
        }
```

```
    }
    void customer() {
        wait(mutex);
        while (waiting < N) {
          waiting++;
          signal(customers);
          signal(mutex);
          wait(barber);
        }
        signal(mutex);
    }
}
```