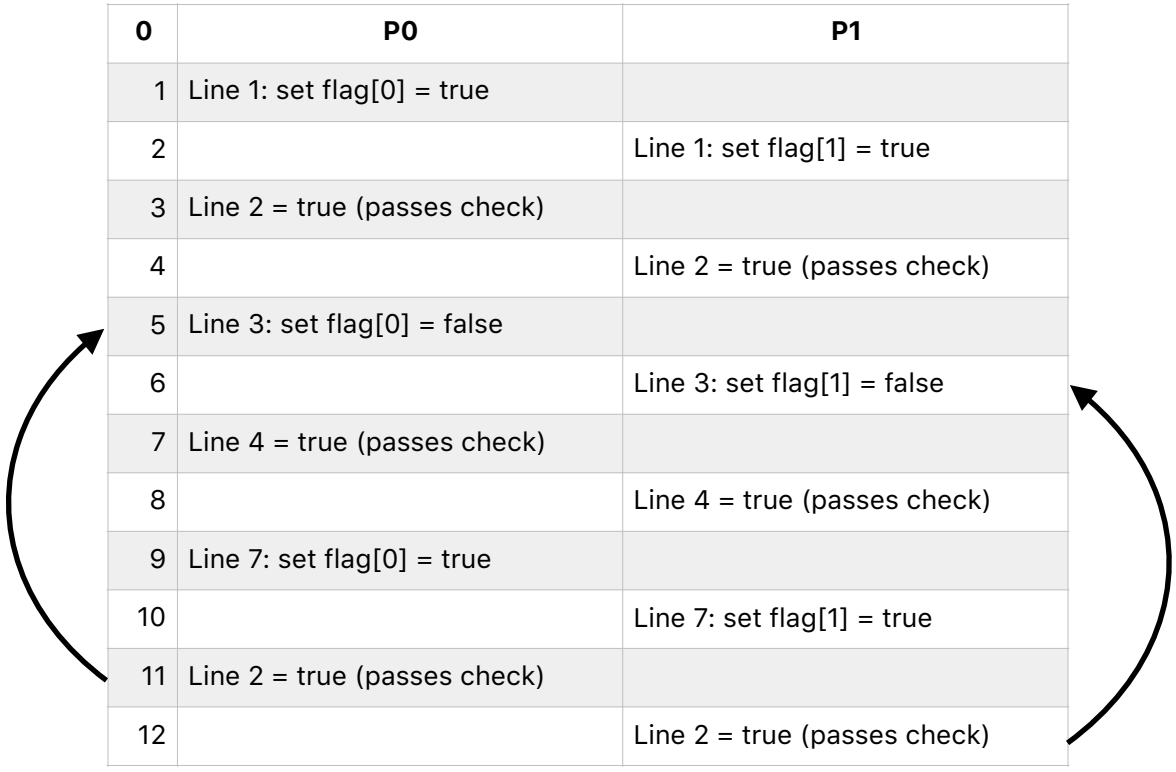
Craig Huff
Homework3
ID #: 009841390

1. Consider the following code example for allocating and releasing processes (i.e. tracking number of processes)
   a. In the allocate_processes() method and the release_processes() method, there is a race condition when reading and writing the number_of_processes variable. When multiple processes are running, there is no way to guarantee the value of number_of_processes will have the same value during the if-else statement.
   b. Mutex locks will be placed above and below the if-else statement to make sure that all of the values of the variables inside of critical section are not changed while conditions are being checked and executed.
   c. Yes. Because only one thread will be able to access that particular atomic value. In order for this to be true, we would still use a mutex lock when calling `atomic_add(1, &number_of_processes)`.

2. Specify which of the following requirements are satisfied or not by this algorithm. If it is satisfied, explain why (with line numbers). If it is not satisfied, come up with a possible scenario
   a. Mutual Exclusion: **Satisfied**
      Each process, i, can only enter the critical section after setting the `flag[i] = true`, it checks to make sure the other process's flag is still false. The second process will be false if it hasn't started executing the remainder section at line 11, or it is currently executing lines 3-6.
   b. Progress: **Not Satisfied**
      Both processes can get stuck checking the flags of the other process if the execution is interleaved.



0	P0	P1
1	Line 1: set flag[0] = true	
2		Line 1: set flag[1] = true
3	Line 2 = true (passes check)	
4		Line 2 = true (passes check)
5	Line 3: set flag[0] = false	
6		Line 3: set flag[1] = false
7	Line 4 = true (passes check)	
8		Line 4 = true (passes check)
9	Line 7: set flag[0] = true	
10		Line 7: set flag[1] = true
11	Line 2 = true (passes check)	
12		Line 2 = true (passes check)

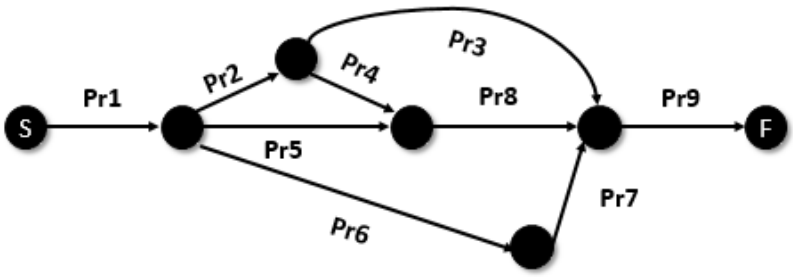
c. Bounded Waiting: **Not Satisfied**
Depending on which process is scheduled first, one of the processes will continuously enter the critical section while the other process will wait forever.

0	P0	P1
1	Line 1: set flag[0] = true	
2	Line 2: P0 will not enter while loop since flag[0] is false	
3	Line 9: Enter critical section	
4		Line 1: set flag[1] = true
5		Line 2 = true (passes check)
6		Line 3: set flag[1] = false
7		Line 4 = true (passes check)
8	Line 10: set flag[0] = false	
9	Line 11: execute remainder section	
10	Line 1: set flag[0] = true	
11	Line 2: P0 will not enter while loop since flag[0] is still false	
12	Line 9: Enter critical section….	
13		Line 5: Still looping since Line 4 is still true. To terminate the loop, flag[0] needs to be false.

3. Assume any condition variable cond has two methods: `cond.wait()` and `cond.signal()`. There are multiple producers that invoke `produce()` and there are multiple consumers that invoke `consume()`. Please implement the `produce()` and `consume()` methods in C

```
monitor bounded_buffer{
      int items[MAX_ITEMS]; /* MAX_ITEM is a constant defined
      elsewhere */
      int numItems = 0; /* # of items in the items array */
      condition full, empty;
      void produce(int v);/*deposit the value v to the items array*/
      int consume(); /* remove the last item from items, and return
      the value */
void produce(int v) {
      while (numItems == MAX ITEMS)
          full.wait();
      items[numItems++] = v;
      empty.signal();
  }
int consume() {
      int tempVal;
      while (numItems == 0)
          empty.wait();
      tempVal = items[--numItems];
      full.signal();
      return tempVal;
      }
```

4. Please use pseudo code (which utilizes semaphores) to enforce execution order of the following process execution diagram.



s1=0; s2=0; s3=0; s4=0;
s5=0; s6=0; s7=0; s8=0;
Pr1: Start; V(s1); V(s1); V(s1);
Pr2: P(s1); body; V(s2); V(s2);
Pr3: P(s2); body; V(s3);
Pr4: P(s2); body; V(s4);
Pr5: P(s1); body; V(s5);
Pr6: P(s1); body; V(s6);
Pr7: P(s6); body; V(s7);
Pr8: P(s4); P(s5); body; V(s8);
Pr9: P(s3); P(s7); P(s8); Finish