HW2 - Solution

1. Another solution is shown in /pub/CIS520/programs/progB.c - using trywait

```
sem_t s[NUM_THREADS][NUM_THREADS];
for(i=0;i<NUM_THREADS;i++)
{
   for(j=0;j<NUM_THREADS;j++)
   {
     sem_init(&waitFor[i][j], 0, 0);  // initialize semaphore to 0
   }
}

void Barrier(int threadId)
{
   int i;
   printf("Thread %d at barrier.\n", threadId);
   for (i = 0; i < NUM_THREADS; i++)
   {
      sem_post(&s[threadId][i]);
   }
   for (i = 0; i < NUM_THREADS;i++)
   {
      sem_wait(&s[i][threadId]);
   }
   printf("Thread %d past barrier.\n", threadId);
}
```

2. If wait() and signal() are not executed atomically, mutual exclusion can be violated. Two processes, P1 and P2 , will violate mutual exclusion if they both call wait() or signal(). With wait(), if there is a semaphore with value 1 and P1 reads the value of the semaphore, then P2 reads the value of the semaphore, they have both read the value as 1 and then they will both decrement. We lose one of these updates and mutual exclusion does not hold. The same is true for signal(), we may lose an update of the semaphore value.

3.
```
monitor alarm {
   condition c;

   void delay (int ticks) {
     int begin_time = read_clock();  // get the current time
     while (read_clock() < begin_time + ticks)
       c.wait();
   }

   void tick() {
     ticks++;
     c.broadcast();
   }

   // optional, if they provide
   int read_clock() {
     return ticks;
   }
```

4.
Semaphore s12 = 0;
Semaphore s13 = 0;
Semaphore 24 = 0;

P1	**P2**	**P3**	**P4**
Do work	Wait(s12)	Wait(13)	Wait(s24)
Signal(s12)	Do work	Do Work	Do Work
Signal(s13)	Signal(s24)		