Problem 6 of 8

Thread Sync

Consider the following scenario:

1) Dr. Evil has an array with n numbers in it and he wants to create a multi-threaded program to print them out. Every thread prints one word at a time and is not allowed to go again until all the other threads have had a chance to print a word. He can only create m threads, with m <n, and he wrote the following implementation. Unfortunately for him, he accidentally activated his Bomb Lab protections and obfuscated some of his code. Can you fix it for him?

```
#define m 5
#define n 10
sem_t mutexes[m];           //Array of mutexes with mutexes[i] corresponding with the mutex of thread i
pthread_t thread_ids[m];    //Array of thread ids
volatile int word_loc = 0;  //Counter to current word to print
int words[n];               //Array of numbers to print out

void *thread_print(void* thread_id);

void P(sem_t *sem)
{
    sem_wait(sem);
}

void V(sem_t *sem)
{
    sem_post(sem);
}

int main(void)
{
    for (long i = 0; i < n; i++)
    {
        words[i] = i;
    }
    for (long i = 0; i < m; i++)
    {
        sem_init(&(mutexes[i]), 0, 0);
        pthread_create(&(thread_ids[i]), NULL, thread_print, (void*)i);
    }
    V(&mutexes[0]);
        for (long i = 0; i < m; i++)
    {
        pthread_join(&(thread_ids[i]), NULL);
    }
    return 0;
}

void *thread_print(void* tid)
{
1.  long thread_id = (long)tid;
2.  while(word_loc != n)
    {
3.      P(          );                          //Grab A Mutex
4.      printf("%d \n", words[word_loc]);
5.          ;                                   //Move to next word
6.      if(word_loc == n) break;            //Termination Condition
7.      V(          );                          //Release A Mutex
    }
8.  exit(1);
}
```

2) Complete the shared variable analysis below. Use `Y` for 'Yes' and `N` for 'No'. Case does not matter. Assume that m > 2.

Referenced by	Thread 0?	Thread m-1?
`mutexes[0]`		
`mutexes[1]`		
`mutexes[m-1]`		
`word_loc`		

3) Will this program always terminate?

- ◯ Yes
- ◯ No