```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>
bool turn = 1;//I had to change this to bool since it is really just TRUE or FALSE.   
  Should give performance boost/memory boost.
bool flag[2] = { 1, 0 };//Boolean, same logic supra. This will help us make sure that 
  all the "11111" print first. That is, we want to make sure we do not
//print "2222" first. Always good to init variables.
unsigned char charCount;//We change this to unsigned char for speed/memory boost Value
   goes up to 255 which is plenty for what we need.
pthread_mutex_t t1_lock;
pthread_mutex_t t2_lock;
pthread_mutex_t print_lock;
/*Online Resources on Thread Syn
https://stackoverflow.com/questions/29529476/c-print-ping-pong-using-semaphores-and-   
  threads
*/

void* thread1()
{
    int i;
    int num = 5000;
    for (i = 0; i < num; i++) {
        pthread_mutex_lock(&t1_lock);//Start of the Critical Section
        flag[0] = 1;
        turn = 1;
        while ((flag[1]) && (turn == 1));
        if (charCount >= 30) {
            charCount = 0;
            printf("\n");
        }
        printf("11111");
        charCount += 1;
        flag[0] = 0;
        pthread_mutex_unlock(&t2_lock);//End of the Critical Section
    }
    pthread_exit(0);
}

void* thread2()
{
    int i;
    int num = 5000;
    for (i = 0; i < num; i++) {
        pthread_mutex_lock(&t2_lock);//Start of the Critical Section
        flag[1] = 1;
        turn = 0;
        while ((flag[0]) && (turn == 0));
        if (charCount >= 30) {
            charCount = 0;
            printf("\n");
        }
        printf("22222");
        charCount += 1;
        flag[1] = 0;
        pthread_mutex_unlock(&t1_lock);//End of the Critical Section
    }
    pthread_exit(0);
}

int main(int argc, char** argv)
{
    int i;
    pthread_t threads[2];

    pthread_mutex_init(&t1_lock, NULL);
    pthread_mutex_init(&t2_lock, NULL);
    pthread_mutex_init(&print_lock, NULL);

    pthread_create(&threads[0], NULL, thread1, NULL);
    sleep(1);//Add the sleep here because we need to make sure thread 2 doesn't kick
      off too fast.
    //Tried to figure out a more sexy solution since this really slows down the
      process. But failed. N00b.
    pthread_create(&threads[1], NULL, thread2, NULL);
    for (i = 0; i < 2; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("\n");

    pthread_mutex_destroy(&t1_lock);
    pthread_mutex_destroy(&t2_lock);
    pthread_mutex_destroy(&print_lock);
    return 0;
}
```