```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>

#define NUM_T1       4
#define NUM_T2             4

bool done = false;
int count = 0 ;
pthread_mutex_t count_mutex;
pthread_cond_t count_cond;

#define COUNT_MAX 15
#define MSGLEN 32
struct thread_dat {
        int id ;
        char message[MSGLEN];
};

void *T1(void *t)
{
   struct thread_dat *pData = (struct thread_dat*)t ;
   printf("%s.\nThread %d started.\n", pData->message, pData->id);
        while(!done) {
                pthread_mutex_lock(&count_mutex);
                count++ ;
                                if (count == COUNT_MAX) {
                                        pthread_cond_broadcast(&count_cond);
                        printf("T1 [%d] count = %d : reached threshold.\n", pData->id, count);
                                }
                printf("T1 [%d] count = %d\n", pData->id, count);
                pthread_mutex_unlock(&count_mutex);
                sleep(1);
        }
   printf("T1 thread %d done.\n",pData->id);
   pthread_exit(NULL);
}

void *T2(void *t)
{
   struct thread_dat *pData = (struct thread_dat*)t ;
   printf("%s.\nThread %d started.\n", pData->message, pData->id);
        while(!done)
                {
                pthread_mutex_lock(&count_mutex);
                                if(count< COUNT_MAX) {
                                        pthread_cond_wait(&count_cond, &count_mutex);
                        count = pData->id ; // reset
                                }
                printf("T2 [%d] count = %d\n", pData->id, count);
                pthread_mutex_unlock(&count_mutex);
                sleep(2);
        }
   printf("T2 thread %d done.\n",pData->id);
   pthread_exit(NULL);
}


int main (int argc, char *argv[])
{
   pthread_t thread[NUM_T1 + NUM_T2];
   pthread_attr_t attr;
   int t;
   void *status;

   struct thread_dat *pthreadData = malloc(sizeof(struct thread_dat) * (NUM_T1 +
NUM_T2));
   if(!pthreadData) {
           fprintf(stderr, "Cannot allocate thread data!\r\n");
           return 255 ;
```

```
   }

   /* Initialize mutex and condition variable objects */
   pthread_mutex_init(&count_mutex, NULL);
   pthread_cond_init (&count_cond, NULL);

   /* Initialize and set thread detached attribute */
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

   for(t=0; t<NUM_T1; t++) {
          pthreadData[t].id = t ;
          snprintf(pthreadData[t].message, MSGLEN, "Welcome to thread %c", 'A'+t) ;
      pthread_create(&thread[t], &attr, T1, (void *)&pthreadData[t]);
   }

   for(t=0; t<NUM_T2; t++) {
          pthreadData[t+NUM_T1].id = t+NUM_T1 ;
          snprintf(pthreadData[t+NUM_T1].message, MSGLEN, "Welcome to thread %c", 'Z'-t) ;
      pthread_create(&thread[t+NUM_T1], &attr, T2, (void *)&pthreadData[t+NUM_T1]);
   }

   sleep(5);
   done = true ;
   /* Free attribute and wait for the other threads */
   for(t=0; t<NUM_T1+NUM_T2; t++)
      pthread_join(thread[t], &status);

   printf("Main: program completed. Exiting. Count = %d\n",count);
   /* Clean up and exit */
   if(pthreadData) free(pthreadData);
   pthread_attr_destroy(&attr);
   pthread_mutex_destroy(&count_mutex);
   pthread_cond_destroy(&count_cond);
   pthread_exit(NULL);

}
```

```
recondite> cc -lpthread thread_ex3.c
recondite> a.out
Welcome to thread A.
Thread 0 started.
T1 [0] count = 1
Welcome to thread B.
Thread 1 started.
T1 [1] count = 2
Welcome to thread C.
Thread 2 started.
T1 [2] count = 3
Welcome to thread D.
Thread 3 started.
T1 [3] count = 4
Welcome to thread Z.
Thread 4 started.
Welcome to thread Y.
Thread 5 started.
Welcome to thread X.
Thread 6 started.
Welcome to thread W.
Thread 7 started.
T1 [0] count = 5
T1 [2] count = 6
T1 [3] count = 7
T1 [1] count = 8
T1 [3] count = 9
T1 [2] count = 10
T1 [0] count = 11
T1 [1] count = 12
T1 [3] count = 13
T1 [1] count = 14
T1 [0] count = 15 : reached threshold.
T1 [0] count = 15
T2 [5] count = 5
T2 [6] count = 6
T2 [7] count = 7
T2 [4] count = 4
T1 [2] count = 5
T1 [3] count = 6
T1 [1] count = 7
T1 [0] count = 8
T1 [2] count = 9
T2 thread 5 done.
T2 thread 4 done.
T1 thread 1 done.
T1 thread 2 done.
T2 thread 6 done.
T2 thread 7 done.
T1 thread 3 done.
T1 thread 0 done.
Main: program completed. Exiting. Count = 9
recondite>
```