```
/* Disciplina: Computacao Concorrente */
/* Prof.: Silvana Rossetto */
/* Barreira usando semaforos */

/***** Condicao logica da aplicacao: a cada iteracao, as threads
incrementam seu contador, imprimem o valor atual, e só podem
continuar depois que todas as threads completaram o passo. Apos N
passos, as threads terminam  ****/

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>

#define NTHREADS  5
#define PASSOS  4

/* Variaveis globais */
int chegaram = 0;
sem_t mutex, cond;

/* Barreira (incorreta!)*/
void barreira_v1(int numThreads) {
    sem_wait(&mutex);
    chegaram++;
    if (chegaram < numThreads) {
      sem_post(&mutex);
      sem_wait(&cond);
    } else {
      printf("\n");
      for(int i=1; i<numThreads; i++)
         { sem_post(&cond); }
      chegaram = 0;
      sem_post(&mutex);
    }
}


/* Barreira (versao 2) (incorreta!) */
void barreira_v2(int numThreads) {
    sem_wait(&mutex);
    chegaram++;
    if (chegaram < numThreads) {
      sem_post(&mutex);
      sem_wait(&cond);
      chegaram--;
      if (chegaram==0) sem_post(&mutex);
      else sem_post(&cond);
    } else {
      printf("\n");
      sem_post(&cond);
      chegaram--; //deveria vir antes da linha anterior!
    }
}

/* Barreira (solucao correta) */
```

```
void barreira(int numThreads) {
    sem_wait(&mutex);
    chegaram++;
    if (chegaram < numThreads) {
      sem_post(&mutex);
      sem_wait(&cond);
      chegaram--;
      if (chegaram==0) sem_post(&mutex);
      else sem_post(&cond);
    } else {
      printf("\n");
      chegaram--;
      sem_post(&cond);
    }
}

/* Funcao das threads */
void *A (void *t) {
  int my_id = *(int*)t;
  int boba = 0, boba1, boba2;

  for (int i=0; i < PASSOS; i++) {
    boba++;
    printf("Thread %d: passo=%d\n", my_id, i);
    /* barreira */
    //barreira_v1(NTHREADS); ///versao incorreta->fura a barreira e
pode levar a deadlock
    barreira(NTHREADS);  ///versao correta (até que se prove o
contrario ;-)
    /* faz alguma coisa... */
    boba1=100; boba2=-100; while (boba2 < boba1) boba2++;
  }
  pthread_exit(NULL);
}

/* Funcao principal */
int main(int argc, char *argv[]) {
  int i;
  pthread_t threads[NTHREADS];
  int id[NTHREADS];
  /* Inicializa os semaforos */
  sem_init(&mutex, 0, 1);
  sem_init(&cond, 0, 0);

  /* Cria as threads */
  for(i=0;i<NTHREADS;i++) {
     id[i]=i;
     pthread_create(&threads[i], NULL, A, (void *) &id[i]);
  }

  /* Espera todas as threads completarem */
  for (i = 0; i < NTHREADS; i++) {
    pthread_join(threads[i], NULL);
  }

  sem_destroy(&mutex);
  sem_destroy(&cond);
```

```
  printf ("FIM.\n");
  pthread_exit (NULL);
}
```