```
/* Disciplina: Computacao Concorrente */
/* Prof.: Silvana Rossetto */
/* Codigo: Implementação e uso de sincronização por barreira */

/***** Condicao logica da aplicacao: a cada iteracao, as threads fazem
uma parte do processamento e só podem continuar depois que todas as
threads completaram essa iteração. ****/

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

#define NTHREADS  5
#define PASSOS  5

/* Variaveis globais */
int bloqueadas = 0;
pthread_mutex_t x_mutex;
pthread_cond_t x_cond;

//funcao barreira
void barreira(int nthreads) {
    pthread_mutex_lock(&x_mutex); //inicio secao critica
    if (bloqueadas == (nthreads-1)) {
      //ultima thread a chegar na barreira
      pthread_cond_broadcast(&x_cond);
      bloqueadas=0;
    } else {
      bloqueadas++;
      pthread_cond_wait(&x_cond, &x_mutex);
    }
    pthread_mutex_unlock(&x_mutex); //fim secao critica
}

//funcao das threads
void *tarefa (void *arg) {
  int id = *(int*)arg;
  int boba1, boba2;

  for (int passo=0; passo < PASSOS; passo++) {
    printf("Thread %d: passo=%d\n", id, passo);

    /* faz alguma coisa... */
    boba1=100; boba2=-100; while (boba2 < boba1) boba2++;

    //sincronizacao condicional
    barreira(NTHREADS);
  }
  pthread_exit(NULL);
}

/* Funcao principal */
int main(int argc, char *argv[]) {
  pthread_t threads[NTHREADS];
  int id[NTHREADS];
  /* Inicilaiza o mutex (lock de exclusao mutua) e a variavel de
```

```
condicao */
  pthread_mutex_init(&x_mutex, NULL);
  pthread_cond_init (&x_cond, NULL);

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

  /* Espera todas as threads completarem */
  for (int i = 0; i < NTHREADS; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("FIM.\n");

  /* Desaloca variaveis e termina */
  pthread_mutex_destroy(&x_mutex);
  pthread_cond_destroy(&x_cond);
  return 0;
}
```