```
/*
Programa: pc.c
Descricao: implementa  o problema dos produtores/consumidores usando
variaveis de condicao da biblioteca Pthread
Autor: Silvana Rossetto
*/

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

#define N 5 //tamanho do buffer
#define P 2 //qtde de threads produtoras
#define C 4 //qtde de threads consumidoras

//variaveis do problema
int Buffer[N]; //espaco de dados compartilhados
int count=0, in=0, out=0; //variaveis de estado

//variaveis para sincronizacao
pthread_mutex_t mutex;
pthread_cond_t cond_cons, cond_prod;

//inicializa o buffer
void IniciaBuffer(int n) {
  int i;
  for(i=0; i<n; i++)
    Buffer[i] = 0;
}

//imprime o buffer
void ImprimeBuffer(int n) {
  int i;
  for(i=0; i<n; i++)
    printf("%d ", Buffer[i]);
  printf("\n");
}

//insere um elemento no Buffer ou bloqueia a thread caso o Buffer
esteja cheio
void Insere (int item, int id) {
   pthread_mutex_lock(&mutex);
   printf("P[%d] quer inserir\n", id);
   while(count == N) {
     printf("P[%d] bloqueou\n", id);
     pthread_cond_wait(&cond_prod, &mutex);
     printf("P[%d] desbloqueou\n", id);
   }
   Buffer[in] = item;
   in = (in + 1)%N;
   count++;
   printf("P[%d] inseriu\n", id);
   ImprimeBuffer(N);
   pthread_mutex_unlock(&mutex);
   pthread_cond_signal(&cond_cons);
```

```
}

//retira um elemento no Buffer ou bloqueia a thread caso o Buffer
esteja vazio
int Retira (int id) {
   int item;
   pthread_mutex_lock(&mutex);
   printf("C[%d] quer consumir\n", id);
   while(count == 0) {
     printf("C[%d] bloqueou\n", id);
     pthread_cond_wait(&cond_cons, &mutex);
     printf("C[%d] desbloqueou\n", id);
   }
   item = Buffer[out];
   Buffer[out] = 0;
   out = (out + 1)%N;
   count--;
   printf("C[%d] consumiu %d\n", id, item);
   ImprimeBuffer(N);
   pthread_mutex_unlock(&mutex);
   pthread_cond_signal(&cond_prod);
   return item;
}

//thread produtora
void * produtor(void * arg) {
  int *id = (int *) arg;
  printf("Sou a thread produtora %d\n", *id);
  while(1) {
    Insere(*id, *id);
    sleep(1);
  }
  free(arg);
  pthread_exit(NULL);
}

//thread consumidora
void * consumidor(void * arg) {
  int *id = (int *) arg;
  int item;
  printf("Sou a thread consumidora %d\n", *id);
  while(1) {
    item = Retira(*id);
    sleep(1);
  }
  free(arg);
  pthread_exit(NULL);
}


//funcao principal
int main(void) {
  //variaveis auxiliares
  int i;

  //identificadores das threads
  pthread_t tid[P+C];
```

```
  int *id[P+C];

  //aloca espaco para os IDs das threads
  for(i=0; i<P+C;i++) {
    id[i] = malloc(sizeof(int));
    if(id[i] == NULL) exit(-1);
    *id[i] = i+1;
  }

  //inicializa o Buffer
  IniciaBuffer(N);

  //inicializa as variaveis de sincronizacao
  pthread_mutex_init(&mutex, NULL);
  pthread_cond_init(&cond_cons, NULL);
  pthread_cond_init(&cond_prod, NULL);

  //cria as threads produtoras
  for(i=0; i<P; i++) {
    if(pthread_create(&tid[i], NULL, produtor, (void *) id[i]))
exit(-1);
  }

  //cria as threads consumidoras
  for(i=0; i<C; i++) {
    if(pthread_create(&tid[i+P], NULL, consumidor, (void *) id[i+P]))
exit(-1);
  }

  pthread_exit(NULL);
  return 1;
}
```