Problema do Produtor – Consumidor

```
#define N 100                                   /* número de lugares no buffer */
typedef int semaphore;                          /* semáforos são um tipo especial de int */
semaphore mutex = 1;                            /* controla o acesso à região crítica */
semaphore empty = N;                            /* conta os lugares vazios no buffer */
semaphore full = 0 ;                            /* conta os lugares preenchidos no buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE é a constante 1 */
          item = produce_item( );               /* gera algo para pôr no buffer */
          down(&empty);                         /* decresce o contador empty */
          down(&mutex);                         /* entra na região crítica */
          insert_item(item);                    /* põe novo item no buffer */
          up(&mutex);                           /* sai da região crítica */
          up(&full);                            /* incrementa o contador de lugares preenchidos */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* laço infinito */
          down(&full);                          /* decresce o contador full */
          down(&mutex);                         /* entra na região crítica */
          item = remove_item( );                /* pega o item do buffer */
          up(&mutex);                           /* deixa a região crítica */
          up(&empty);                           /* incrementa o contador de lugares vazios */
          consume_item(item);                   /* faz algo com o item */
     }
}
```

Problema dos Leitores e Escritores

```
typedef int semaphore;                  /* use sua imaginação */
semaphore mutex = 1;                    /* controla o acesso a 'rc' */
semaphore db = 1;                       /* controla o acesso a base de dados */
int rc = 0;                             /* número de processos lendo ou querendo ler */

void reader(void)
{
     while (TRUE) {                     /* repete para sempre */
          down(&mutex);                 /* obtém acesso exclusivo a 'rc' */
          rc = rc + 1;                  /* um leitor a mais agora */
          if (rc == 1) down(&db);       /* se este for o primeiro leitor ... */
          up(&mutex);                   /* libera o acesso exclusivo a 'rc' */
          read_data_base( );            /* acesso aos dados */
          down(&mutex);                 /* obtém acesso exclusivo a 'rc' */
          rc = rc – 1;                  /* um leitor a menos agora */
          if (rc == 0) up(&db);         /* se este for o último leitor ... */
          up(&mutex);                   /* libera o acesso exclusivo a 'rc' */
          use_data_read( );             /* região não crítica */
     }
}


void writer(void)
{
     while (TRUE) {                     /* repete para sempre */
          think_up_data( );             /* região não crítica */
          down(&db);                    /* obtém acesso exclusivo */
          write_data_base( );           /* atualiza os dados */
          up(&db);                      /* libera o acesso exclusivo */
     }
}
```

Problema do Barbeiro Dorminhoco

```
#define CHAIRS 5                    /* número de cadeiras para os clientes à espera */

typedef int semaphore;              /* use sua imaginação */

semaphore customers = 0;            /* número de clientes à espera de atendimento*/
semaphore barbers = 0;              /* número de barbeiros à espera de clientes */
semaphore mutex = 1;                /* para exclusão mútua */
int waiting = 0;                    /* clientes estão esperando (não estão cortando) */

void barber(void)
{
     while (TRUE) {
          down(&customers);         /* vai dormir se o número de clientes for 0 */
          down(&mutex);             /* obtém acesso a 'waiting' */
          waiting = waiting – 1;    /* decresce de um o contador de clientes à espera */
          up(&barbers);             /* um barbeiro está agora pronto para cortar cabelo */
          up(&mutex);               /* libera 'waiting' */
          cut_hair( );              /* corta o cabelo (fora da região crítica) */
     }
}

void customer(void)
{
     down(&mutex);                  /* entra na região crítica */
     if (waiting < CHAIRS) {        /* se não houver cadeiras livres, saia */
          waiting = waiting + 1;    /* incrementa o contador de clientes à espera */
          up(&customers);           /* acorda o barbeiro se necessário */
          up(&mutex);               /* libera o acesso a 'waiting' */
          down(&barbers);           /* vai dormir se o número de barbeiros livres for 0 */
          get_haircut( );           /* sentado e sendo servido */
     } else {
          up(&mutex);               /* a barbearia está cheia; não espere */
     }
}
```