

```
/** por Engenheiro Anilton Silva Fernandes **/
```

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

```
#define FALSE 0
#define TRUE 1
```

```
#define N 10
```

```
sem_t mutex, empty, full; //empty: semaforo utilizado pra controlar os sinais relativos as posicoes
vazias no array //full: semaforo utilizado pra controlar os sinais relativos as posicoes
preenchidas no array //mutex: semaforo binário utilizado p/ garantir exclusao mutua na regioao
critica
```

```
int array[N], pch, pvz, count; //array: utilizado para armazenar os dados produzidos pelo produtor e
consumidos pelo consumidor //pch: procima posicao cheia
//pvz: proxima posicao vazia
//count: usado para controlar a quantidade de dados presentes no array
```

```
int iniarray() // inicializar todas as possicoes
{ // do array com -1
    int i;
    for(i=0;i<N;i++)
        array[i] = -1;

    return 1;
}
```

```
// funcao para produzir itens
int produce_item()
{
    int val = rand()%90;
    printf("\nProduzindo item: %d", val);
    return val;
}
```

```
// funcao que mostra o item consumido
void consume_item(int item)
{
    printf("\nCosumindo item: %d", item);
}
```

```

//funcao que insere o item dentro do array
void insert_item(int val)
{
    if(count < N)
    {
        array[pvz] = val;
        // A utilizacao da divisao em modulo implementa um comportamento circular da
        // utilizacao do array
        // ou seja, qdo o contador chegar no valor de N (N % N = 0) o valor da variavel voltara
        // ao inicio do array
        pvz = (pvz + 1) % N;
        count = count + 1;
        if(count == N)
            printf("\n----->array cheio<-----");
    }
}

```

```

// funcao que remove o item do array
int remove_item()
{
    int val;
    if(count > 0)
    {
        val = array[pch];
        array[pch] = -1;
        pch = (pch + 1) % N;
        count = count - 1;
        return val;
    }
}

```

```

void *producer(void *p_arg)
{
    int item;

    while(TRUE)
    {
        item = produce_item();

        // sem_wait (realiza o down no semaforo (ver pag. 81 do livro Sistemas Operacionais -
        // 2a edicao - Tanenbaum ))
        // sem_post (realiza o up no semaforo)
        sem_wait(&empty);
        sem_wait(&mutex);
        insert_item(item);
        sem_post(&mutex);
        sem_post(&full);
    }
}

```

```

}

void *consumer(void *p_arg)
{
    int item;

    while(TRUE)
    {
        sem_wait(&full);
        sem_wait(&mutex);
        item = remove_item();
        sem_post(&mutex);
        sem_post(&empty);
        consume_item(item);
    }
}

```

```

int main(void)
{
    // inicializacao do array a -1
    iniarray();

    //Seta a semente da funcao geradora de numeros aleatorios
    //srand(time(NULL));

    count = 0;
    pch = 0;
    pvz = 0;

    /*Inicializa os semaforos
    1o parametro: variavel semaforo
    2o parametro: indica se um semaforo sera compartilhado entre as threads de um processo ou
entre processos
        o valor 0 indica q/ o semaforo sera compartilhado entre as threads de um processo
(digit o comando
        "man sem_init" no shell do linux p/ ver os detalhes)
    3o parametro: valor inicial do semaforo
    */
    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, N);
    sem_init(&full, 0, 0);

    pthread_t thd0, thd1;

    /*
    Inicializa as threads
    1o parametro: variavel thread
    2o parametro: indica se uma thread é "joinable", ou seja, se a thread nao sera finalizada ate

```

chegar a uma chamada de função

```
pthread_join().
```

3o parametro: indica o nome do método que irá compor o trecho de código q/ será executado pela thread

4o parametro: utilizado qdo se necessita passar algum parametro a thread. Pode se passar quaisquer tipos de dados,

inclusive uma estrutura de dados qdo houver a necessidade de passar mais de um parametro.

(dentro do método chamado realiza-se um cast p/ recuperar os dados)

```
*/
```

```
pthread_create(&thd0, 0, (void *) producer, NULL);
```

```
pthread_create(&thd1, 0, (void *) consumer, NULL);
```

//Esses dois métodos indicam q/ a thread não será finalizada até ocorrer a chamada dos mesmos (como mencionado anteriormente)

```
pthread_join(thd0,0);
```

```
pthread_join(thd1,0);
```

```
exit(0);
```

```
}
```