

# Programação Concorrente (IC/CCMN/UFRJ)

Paralelismo de tarefas com os padrões produtores/escritores e  
leitores/escritores

Profa. Silvana Rossetto

Na **computação distribuída** (uma ramificação da computação concorrente) é comum encontrarmos problemas com **paralelismo de tarefas**:

- cada fluxo de execução realiza uma subtarefa distinta
- os fluxos distintos interagem (se comunicam e se sincronizam) para realizar a tarefa completa

## Exemplos

- (micro)serviços que interagem por meio de um serviço de mensageria
- componentes que publicam eventos /  
componentes que registram interesse em eventos

# Exemplo: arquitetura publish/subscribe



1

o **canal** é o componente de interceção que possibilita a interação entre os fluxos distintos

<sup>1</sup><https://inside.contabilizei.com.br/por-que-usar-pub-sub-378f0e212e67>



- Os **produtores** **produzem/geram** novos elementos
- Os **consumidores** **consomem/processam** esses elementos
- O **canal de comunicação** oferece operações de **inserção** e **remoção** de elementos

Uma **área de dados** ou **canal de comunicação** é compartilhada entre **produtores e consumidores**

- Produtores **depositam/inserem** os elementos gerados na área de dados
- Consumidores **retiram/removem** os elementos que devem ser processados da área de dados

# Condições do problema produtor/consumidor

- 1 Os produtores não podem inserir novos elementos quando a área de dados já está cheia
- 2 Os consumidores não podem retirar elementos quando a área de dados já está vazia
- 3 Os elementos devem ser retirados na mesma ordem em que foram inseridos
- 4 Os elementos inseridos não podem ser perdidos (sobreescritos por novos elementos)
- 5 Um elemento só pode ser retirado uma única vez

Implementar uma solução para o problema dos **produtores/consumidores** (função principal para as threads **produtoras** e **consumidoras**):

# Produtor

```
void * produtor(void * arg) {  
    tElemento elemento;  
    while(REPETE) {  
        elemento = produzElemento();  
        Insere(elemento); //pode bloquear!  
    }  
    pthread_exit(NULL);  
}
```



```
void * consumidor(void * arg) {  
    tElemento elemento;  
    while(REPETE) {  
        elemento = Retira(); //pode bloquear!  
        processaElemento(elemento);  
    }  
    pthread_exit(NULL);  
}
```

# Operações para inserir e retirar elementos

```
void Insere (tElemento elem) {  
    ...  
}
```

```
tElemento Retira (void) {  
    ...  
}
```

# Variáveis globais

```
#define N 5 //tamanho do canal

//variaveis do problema
int Canal[N];
int contador=0, posInsere=0, posRemove=0;

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

# Função para inserir elemento

```
void Insere (tElemento elem) {  
    pthread_mutex_lock(&mutex);  
    while(contador == N) {  
        pthread_cond_wait(&cond_prod, &mutex);  
    }  
    contador++;  
    Canal[posInsere] = elem;  
    posInsere = (posInsere + 1) % N;  
    pthread_mutex_unlock(&mutex);  
    pthread_cond_signal(&cond_cons);  
}
```

# Função para retirar elemento

```
tElemento Retira (void) {  
    tElemento elem;  
    pthread_mutex_lock(&mutex);  
    while(contador == 0) {  
        pthread_cond_wait(&cond_cons, &mutex);  
    }  
    contador--;  
    elem = Canal[posRemove];  
    posRemove = (posRemove + 1) % N;  
    pthread_mutex_unlock(&mutex);  
    pthread_cond_signal(&cond_prod);  
    return elem;  
}
```

## O que muda nessa função Insere?

```
void Insere (int item[]) {  
    pthread_mutex_lock(&mutex);  
    while(contador > 0) {  
        pthread_cond_wait(&cond_prod, &mutex);  
    }  
    for(int i=0; i<N; i++)  
        Canal[i] = itens[i];  
    contador = N;  
    pthread_mutex_broadcast(&cond_cons);  
    pthread_mutex_unlock(&mutex);  
}
```

## O que muda nessa função Retira?

```
int Retira (void) {  
    int item;  
    pthread_mutex_lock(&mutex);  
    while(contador == 0) {  
        pthread_cond_wait(&cond_cons, &mutex);  
    }  
    item = vetor[posRemove];  
    posRemove = (posRemove + 1) % N;  
    contador--;  
    if(contador==0) pthread_cond_signal(&cond_prod);  
    pthread_mutex_unlock(&mutex);  
    return item;  
}
```

# O problema dos leitores e escritores





Uma área de dados (ex., arquivo, bloco da memória, tabela de um banco de dados) é compartilhada entre diferentes threads que executam duas operações:

- **leitura**: lêem o conteúdo da área de dados
- **escrita**: escrevem conteúdo na área de dados

Condições do problema:

- ① os **leitores podem ler simultaneamente** uma região de dados compartilhada
- ② **apenas um escritor pode escrever a cada instante** em uma região de dados compartilhada
- ③ se **um escritor está escrevendo, nenhum leitor pode ler** a mesma região de dados compartilhada

# Código das threads

```
void *leitor (void *arg) {  
    while(1) {  
        EntraLeitura();  
        //le algo...  
        SaiLeitura();  
        //faz outra coisa...  
    }  
}
```

```
void *escritor (void *arg) {  
    while(1) {  
        EntraEscrita();  
        //escreve algo...  
        SaiEscrita();  
        //faz outra coisa...  
    }  
}
```

# Funções para leitura

```
int leit=0, escr=0; //globais
void EntraLeitura() {
    pthread_mutex_lock(&mutex);
    while(escr > 0) {
        pthread_cond_wait(&cond_leit, &mutex);
    }
    leit++;
    pthread_mutex_unlock(&mutex);
}

void SaiLeitura() {
    pthread_mutex_lock(&mutex);
    leit--;
    if(leit==0) pthread_cond_signal(&cond_escr);
    pthread_mutex_unlock(&mutex);
}
```

# Funções para escrita

```
int leit=0, escr=0; //globais
void EntraEscrita () {
    pthread_mutex_lock(&mutex);
    while((leit>0) || (escr>0)) {
        pthread_cond_wait(&cond_escr, &mutex);
    }
    escr++;
    pthread_mutex_unlock(&mutex);
}

void SaiEscrita () {
    pthread_mutex_lock(&mutex);
    escr--;
    pthread_cond_signal(&cond_escr);
    pthread_cond_broadcast(&cond_leit);
    pthread_mutex_unlock(&mutex);
}
```

## O que muda nessa implementação de leitores/escritores?

```
int leitores=0; + variaveis sincronização
void AntesLeitura () {
    lock(&mutex);  leitores++;  unlock(&mutex);
}
void DepoisLeitura () {
    lock(&mutex);  leitores--;
    if(leitores==0)
        pthread_cond_signal(&cond_escr);
    unlock(&mutex);
}
void Escreve(void * args) {
    lock(&mutex);
    while(leitores>0)
        pthread_cond_wait(&cond_escr, &mutex);
    //realiza a escrita de args (...)
    pthread_cond_signal(&cond_escr);    unlock(&mutex);
}
```