

Universidade Federal de Santa Catarina
INE5645 - Programação Paralela e Distribuída
T1: Padrões de Projeto - Programação Concorrente

Cozinha Paralela

Uma Simulação Concorrente com Produtor-Consumidor
e Suspensão Controlada

Arthur Lorenzetti da Rosa	19200621
Jacqueline Correia Beber	19200634
Vinicius Araujo	18205320

Apresentação do Problema

Simulamos o funcionamento de uma cozinha com múltiplos clientes fazendo pedidos e vários cozinheiros preparando os pratos de forma concorrente e em paralelo.

- Clientes (threads) fazem pedidos e colocam em uma fila compartilhada.
- Cozinheiros (threads) retiram os pedidos da fila e os processam em paralelo.



Objetivo

Desenvolver uma aplicação concorrente que simule de forma realista o funcionamento de uma cozinha, com:

- Múltiplos clientes - que são os produtores;
- Múltiplos cozinheiros - que são os consumidores;
- Sincronização segura para evitar condições de corrida, garantindo consistência;
- Número variável de clientes e cozinheiros.



Padrões de Projeto

Produtor-Consumidor:

- Desacoplamento:
 - Clientes e cozinheiros operam independentemente
- Escalabilidade:
 - Número variável de produtores e consumidores
- Buffer seguro:
 - Fila protegida por mutex, evitando condições de corrida
- Clientes produzem pedidos.
 - Inserem pedidos na fila compartilhada.
- Cozinheiros consomem pedidos.
 - Retiram pedidos e os processam.

Produtor-Consumidor

```
// Thread do cliente
void* cliente_thread(void* arg) {
    int id = *((int*)arg);
    free(arg);

    pthread_mutex_lock(&mutex_fila);
    inserir_pedido(id);
    printf("Cliente %d fez um pedido.\n", id);
    pthread_cond_broadcast(&cond_novo_pedido);
    pthread_mutex_unlock(&mutex_fila);

    pthread_exit(NULL);
}
```

```
// Thread do cozinheiro
void* cozinheiro_thread(void* arg) {
    int id = *((int*)arg);
    free(arg);

    while (1) {
        printf("Cozinheiro %d deu o lock na mutex_fila.\n", id);
        pthread_mutex_lock(&mutex_fila);

        // Enquanto não há pedidos, espera ou verifica se deve terminar
        while (fila.total == 0) {
            printf("Fila vazia no momento, Cozinheiro %d verificando se acabou o serviço (todos pedidos consumidos).\n", id);

            // Se todos os pedidos foram feitos e processados, termina
            if (todos_pedidos_feitos && pedidos_processados >= num_clientes) {
                pthread_mutex_unlock(&mutex_fila);
                printf("Cozinheiro %d finalizou o trabalho.\n", id);
                pthread_exit(NULL);
            }
            printf("Cozinheiro %d dormiu.\n", id);
            pthread_cond_wait(&cond_novo_pedido, &mutex_fila);
            printf("Cozinheiro %d acordou.\n", id);
        }

        // Processa o pedido
        int pedido = retirar_pedido();
        pthread_mutex_unlock(&mutex_fila);

        printf("Cozinheiro %d está preparando o pedido do cliente %d...\n", id, pedido);
        sleep(1); // Simula tempo de preparo

        pthread_mutex_lock(&mutex_pedidos);
        pedidos_processados++;
        printf("Cozinheiro %d finalizou o pedido do cliente %d. Pedidos processados: %d/%d\n",
               id, pedido, pedidos_processados, num_clientes);
        pthread_mutex_unlock(&mutex_pedidos);

        // Verifica novamente se deve terminar após processar o pedido
        if (todos_pedidos_feitos && pedidos_processados >= num_clientes) {
            pthread_mutex_lock(&mutex_fila);
            pthread_cond_broadcast(&cond_novo_pedido); // Acorda outros cozinheiros
            pthread_mutex_unlock(&mutex_fila);
            printf("Cozinheiro %d finalizou o trabalho.\n", id);
            pthread_exit(NULL);
        }
    }
}
```

Padrões de Projeto

Suspensão Controlada:

- Eficiência:
 - Evita busy-waiting (espera ocupando CPU)
 - Sincronização precisa:
 - Cozinheiros dormem até haver pedidos
 - Broadcast inteligente:
 - Notifica múltiplos cozinheiros de uma vez
 - Evita laços de espera ativa.
- ***pthread_cond_wait***
 - ***pthread_cond_broadcast***

Suspensão Controlada

```
// Cozinheiro (espera eficiente)
while (fila.total == 0) {
    pthread_cond_wait(&cond_novo_pedido, &mutex_fila); // Dorme até ser acordado
}

// Cliente (acorda cozinheiros)
pthread_cond_broadcast(&cond_novo_pedido); // Acorda todos os cozinheiros
```

Estruturas de Sincronização

Mutex (Exclusão Mútua):

- Proteger acesso à fila de pedidos.
- Proteger o contador de pedidos.
- Garante que apenas uma thread acesse a região crítica por vez.

Variável de Condição:

- Coordena threads eficientemente:
 - Cozinheiros dormem quando a fila está vazia.
 - Clientes acordam cozinheiros com broadcast.

```
15  pthread_mutex_t mutex_fila;      // Protege a fila de pedidos
16  pthread_cond_t cond_novo_pedido; // Sincroniza cozinheiros
17  pthread_mutex_t mutex_pedidos;   // Protege o contador de pedidos
18
```

Demonstração da Aplicação

Agradecemos pela atenção!

