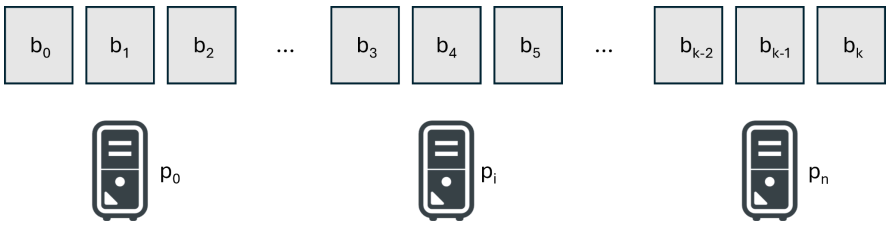
Universidade Federal de Santa Catarina – UFSC
Departamento em Informática e Estatística
INE 5645 – Programação Paralela e Distribuída
Semestre 2025/1

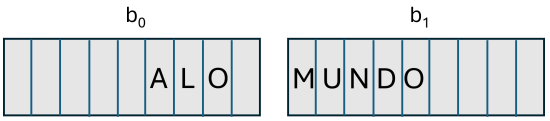
Definição do Trabalho 2: Programação Distribuída

Este trabalho visa o desenvolvimento de uma aplicação distribuída com condições de corrida e necessidade de resolução de conflitos para garantia de consistência de dados. O objetivo é desenvolver um protótipo para a abstração de *memória compartilhada distribuída*, permitindo que o espaço de endereçamento, que é dado em nível de *blocos*, possa ser utilizado por todos os processos de um grupo.

Os blocos são distribuídos entre n processos de um grupo. Qualquer processo do grupo pode acessar qualquer bloco de memória para executar operações de leitura ou escrita. Caso um processo p_i detenha o bloco b_k, este acessa o bloco localmente. Se um processo p_j, onde $p_j \neq p_i$ acessar b_k, este deve obter b_k de p_i. O número total de blocos compondo o espaço de endereçamento é dado por k. A figura a seguir exibe blocos b_0 a b_k dispostos entre os processos p_0 a p_n.



Cada bloco armazena t bytes, sendo o tamanho do espaço de endereçamento dado por $t \times k$. Por exemplo, um sistema configurado com t = 4092 e $k = 10^6$ representaria um espaço de armazenamento de 4GB. Se este espaço for dividido entre 4 processos, cada um seria responsável por manter 1GB de informação localmente. A figura a seguir ilustra a palavra “ALO MUNDO” armazenada em posições contíguas dos blocos b_0 e b_1.



Terminologia:

n: número de processos
k: número de blocos
t: tamanho dos blocos, em bytes
p_i: o i-ésimo processo
b_j: o j-ésimo bloco do espaço de endereçamento

Além da criação do espaço de endereçamento, que pode ser baseado em arquivo de configuração ou alguma função de inicialização com parametrização passada por linha de comando, o serviço deve disponibilizar funções de leitura e escrita. A API para essas primitivas é:

```
int le(int posicao, byte *buffer,int tamanho):
```

- o valor de retorno inteiro (int) deve representar códigos de erro, na impossibilidade de execução da operação;

- `posicao` indica a posição inicial da memória de onde se pretende ler algum conteúdo;
- `buffer` indica o endereço da variável que receberá o conteúdo da leitura
- `tamanho` indica o número em bytes a serem lidos na operação (ou seja, o número de bytes a partir da posição `posicao`).

```
int escreve(int posicao, byte *buffer, int tamanho):
```

- o valor de retorno inteiro (int) deve representar códigos de erro, na impossibilidade de execução da operação;
- `posicao` indica a posição inicial da memória de onde se pretende escrever algum conteúdo;
- `buffer` indica o endereço da variável que contém o conteúdo a ser escrito no espaço de endereçamento
- `tamanho` indica o número em bytes a serem escritos na operação (ou seja, o número de bytes em `buffer`, a partir da posição `posicao`).

Distribuição dos blocos entre os processos

Você pode decidir a forma como os blocos são distribuídos entre os processos. Por exemplo, pode usar uma distribuição por intervalo ou por mapeamento com o uso de uma hash. O conhecimento sobre a pertinência de bloco por processo pode ser obtida de um processo proxy, que tenha o conhecimento sobre o mapeamento ou o conhecimento pode ser estático e imutável, de forma que cada processo pode determinar onde cada bloco está localizado. Fica a critério do grupo implementar a estratégia.

Coerência de cache

Operações tanto de leitura quanto de escrita podem acessar blocos locais ou remotos, ou, ainda, locais e remotos. Para acessos remotos, o processo solicitante deve fazer uma cópia do(s) bloco(s) de interesse e mantê-la em uma cache local. Dessa forma, leituras sobre o bloco podem ser feitas pela cache, caso haja uma cópia do bloco de interesse disponível.

No caso de leituras e leituras sucessivas, se o processo tiver uma cópia válida do bloco, basta ler da sua cache. No caso de escritas, o processo deve atualizar o valor do bloco no conteúdo do dono do bloco e gerar uma invalidação das cópias daquele bloco em caches de outro processos. Este procedimento é comum na implementação de mecanismos para coerência de cache e chama-se *invalidação na escrita*.

Requisitos e avaliação

Os requisitos específicos são:

- Implementar um protótipo para o serviço de memória compartilhada distribuída com coerência de cache; Deve ser entregue: o código com descrição (ex. arquivo readme) com detalhes para compilação, implantação e execução da aplicação;
- Elaborar um relatório com casos de teste gerais e também para a verificação do mecanismo de coerência de cache. O grupo deve pensar em situações que causem o aproveitamento da cache (ex. leituras sucessivas) e a necessidade de invalidação de blocos em cache;
- A defesa dos códigos será feita em aula, com a defesa de código e avaliação individual de cada grupo.

Entrega

O trabalho pode ser realizado em grupos de até 4 participantes.

O código-fonte e relatório devem ser enviados pelo Moodle para análise e avaliação.

O nome dos membros do grupo deve aparecer nos artefatos. Nomes que não estejam explícitos nos artefatos entregues não serão considerados na avaliação.