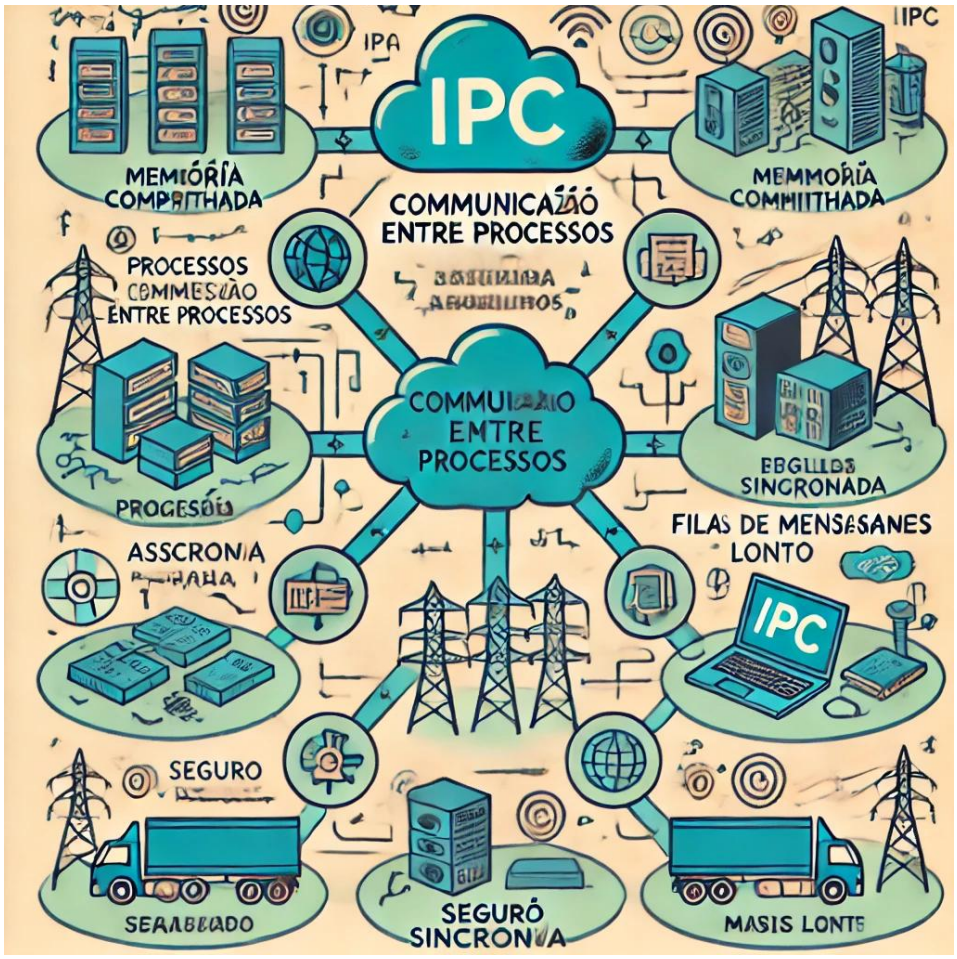
Programação concorrente e distribuída

Alunos: Guilherme Henrique, João Omati, Isaque, Ézio, Diogo

Professor: Paulo

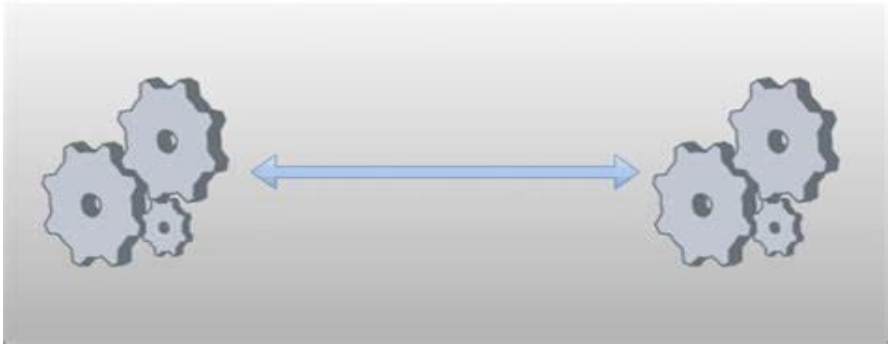
UNIEURO
CENTRO UNIVERSITÁRIO
GRUPO EDUCACIONAL CEUMA

Comunicação entre processos



1 O que é a comunicação entre processos (IPC)?

Inter-Process Communication (IPC). Refere-se à troca de informações e dados entre diferentes processos em um sistema operacional ou em uma rede de computadores. é usado para troca de dados entre vários threads em um ou mais processos ou programas. Os Processos podem estar sendo executados em um ou vários computadores conectados por uma rede.



2 Como funciona a comunicação entre os processos?

Processos e threads interagem para trabalhar conjuntamente em um sistema. Trocam dados / mensagens. Utilizam os serviços de comunicação fornecidos pela máquina e pelo S.O. Seguem protocolos de comunicação para que possam entender uns entender uns aos outros

Dependendo do sistema operacional e do tipo de IPC utilizado, essa comunicação pode ocorrer de várias formas:

1.1 Memória Compartilhada

2.1 Filas de Mensagens

3.1 Pipes (Tubos)

a. Pipes anônimos: Unidirecionais, geralmente usados entre processos pai e filho.

b. Named Pipes (FIFO): Podem ser bidirecionais e usados entre processos independentes.

4.1 Sockets

5.1 Sinalização (Signals)

6.1 RMI (Remote Method Invocation)

7.1 RPC (Remote Procedure Call)

3 Qual é a utilidade da comunicação entre processos?

1. Compartilhamento de Dados: Permite que processos troquem informações, como em **bancos de dados**, **sistemas de arquivos distribuídos** e **aplicações colaborativas**.

2. Modularização de Sistemas: Facilita o desenvolvimento de sistemas grandes e complexos dividindo-os em **múltiplos processos menores e independentes**, tornando o código mais **organizado e escalável**.

3. Execução Concorrente: Permite que múltiplos processos executem **simultaneamente**, otimizando o uso de **recursos do sistema** e melhorando a **performance** de aplicações.

4. Comunicação entre Aplicações Distribuídas: Usado em **microserviços**, **sistemas em nuvem** e **arquiteturas cliente-servidor** para comunicação entre diferentes máquinas, permitindo a construção de **sistemas escaláveis**.

5. Sincronização de Processos: Processos que dependem uns dos outros precisam se coordenar. Por exemplo, **semáforos e mutex** ajudam a evitar **condições de corrida** e **deadlocks**.

6. Balanceamento de Carga e Distribuição de Tarefas: Permite que sistemas distribuam tarefas entre múltiplos processos ou servidores, melhorando o desempenho e evitando sobrecarga em um único ponto.

7. Comunicação entre Componentes do SO: Sistemas operacionais usam IPC para interação entre seus componentes, como **drivers de dispositivos, chamadas de sistema e gerenciamento de processos**.

4 Tipos de IPC e suas especificações:

4.1 **PIPE:** Pipe é um meio de comunicação entre dois ou mais processos relacionados ou inter-relacionados. Pode ocorrer dentro de um único processo ou entre um processo pai e um processo filho. A comunicação também pode ser em múltiplos níveis, como entre o processo pai, o filho e o neto, etc. A comunicação é realizada quando um processo escreve no pipe e outro lê do pipe. Para utilizar a chamada de sistema pipe, crie dois arquivos: um para escrever no arquivo e outro para ler do arquivo. Os pipes podem ser unidirecionais ou bidirecionais. Em um pipe unidirecional, um aplicativo escreve dados no pipe e o outro aplicativo os lê. Em um pipe bidirecional, ambos os aplicativos podem ler e escrever dados. Os pipes podem ser nomeados ou não nomeados

Vantagens:

- **Simplicidade:** Pipes são uma maneira simples e direta para que os processos se comuniquem.
- **Eficiência:** Pipes permitem uma comunicação rápida entre processos, com baixo consumo de recursos.
- **Confiabilidade:** Pipes garantem uma transmissão de dados confiável, detectando erros e assegurando a entrega correta das informações.
- **Flexibilidade:** Pipes podem ser usados para implementar diferentes protocolos de comunicação, incluindo comunicação unidirecional e bidirecional.

Desvantagens:

- **Capacidade limitada:** Pipes possuem uma capacidade limitada, o que restringe a quantidade de dados que pode ser transferida de uma vez entre processos.
- **Unidirecionalidade:** Em um pipe unidirecional, apenas um processo pode enviar dados por vez, o que pode ser uma limitação em alguns cenários.

- **Sincronização:** Em pipes bidirecionais, os processos precisam ser sincronizados para garantir que os dados sejam transmitidos na ordem correta.
- **Escalabilidade limitada:** Pipes são limitados à comunicação entre um pequeno número de processos no mesmo computador, o que pode ser uma desvantagem em sistemas distribuídos de grande escala.

Unanmed Pipes: Um ***pipe anônimo*** é um pipe unidirecional sem nome que normalmente transfere dados entre um processo pai e um processo filho. Pipes anônimos são sempre locais; eles não podem ser usados para comunicação em uma rede

Exemplo:

```
#include <stdio.h>
/* O índice da extremidade "leitura" do pipe */
#define READ 0
/* O índice da extremidade "escrita" do pipe */
#define WRITE 1
char *phrase = "Coloque isso no seu pipe e fume";
main () {
int fd[2], bytesRead;
char message [100]; /* Buffer de mensagem do processo pai */
pipe ( fd ); /* Cria um pipe não nomeado */
if ( fork ( ) == 0 ) {
/* Processo filho - Escritor */
close (fd[READ]); /* Fecha a extremidade não utilizada */
write (fd[WRITE], phrase, strlen ( phrase) +1); /* Inclui o caractere NULL */
close (fd[WRITE]); /* Fecha a extremidade utilizada */
printf("Filho: Escreveu '%s' no pipe!\n", phrase);
} else {
/* Processo pai - Leitor */
```

```
close (fd[WRITE]); /* Fecha a extremidade não utilizada */
bytesRead = read ( fd[READ], message, 100);
printf ( "Pai: Leu %d bytes do pipe: %s\n", bytesRead, message);
close ( fd[READ]); /* Fecha a extremidade utilizada */
}
}
```

Named Pipes: Um **pipe nomeado** é um pipe identificado por um nome, podendo ser unidirecional ou bidirecional, utilizado para comunicação entre um servidor de pipe e um ou mais clientes de pipe. Todas as instâncias de um pipe nomeado compartilham o mesmo nome, mas cada instância possui seus próprios buffers e identificadores, proporcionando um canal separado para a comunicação entre cliente e servidor. O uso de instâncias permite que vários clientes de pipe utilizem o mesmo pipe nomeado simultaneamente.

Exemplo:

```
// Programa em C para implementar um dos lados de um FIFO
// Este lado escreve primeiro, depois lê
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
int fd;
// Caminho do arquivo FIFO
char * myfifo = "/tmp/myfifo";
// Criando o arquivo nomeado (FIFO)
```

```
// mkfifo(<caminho>, <permissões>)
mkfifo(myfifo, 0666);
char arr1[80], arr2[80];
while (1)
{
// Abre o FIFO apenas para escrita
fd = open(myfifo, O_WRONLY);
// Obtém uma entrada do usuário.
// 80 é o comprimento máximo
fgets(arr2, 80, stdin);

// Escreve a entrada no FIFO
```

4.2 **SOCKETS:** Sockets é um tipo de IPC muito especial, para não dizer o melhor, esse IPC permite a comunicação entre dois processos na mesma máquina, bem como a comunicação entre dois processos em máquinas diferentes através de uma rede, ou seja, dois processos rodando em computadores fisicamente separados. Um socket é um dispositivo de comunicação bidirecional, sendo possível enviar e receber mensagens. Para entender de forma fácil o que seria o socket, é fazer uma analogia com uma ligação telefônica, você deseja ligar para uma determinado número, então disca esse número e quando a conexão é estabelecida começa a tocar no telefone na outra ponta, a pessoa atende e começam a conversar, o mesmo ocorre para o socket, porém ao invés de usar número usamos IP e porta para estabelecer a comunicação. Este IPC possui duas formas de comunicação(não irei mencionar as outras devido não serem tão utilizadas) conhecidas como TCP(Transmission Control Protocol) nesse caso as mensagens são enviadas na forma de stream de mensagens, e UDP(User Datagram Protocol) onde as mensagens são enviadas em blocos de mensagens.

O que é TCP?

O protocolo de controle de transmissão (TCP) é orientado à conexão, o que significa que, uma vez que a conexão foi estabelecida, os dados podem ser transmitidos em duas direções. Ele tem sistemas integrados para verificar se há erros e garantir que os dados sejam entregues na ordem em que foram enviados, tornando-o o protocolo perfeito para a transferência de informações como imagens estáticas, arquivos de dados e páginas da web.

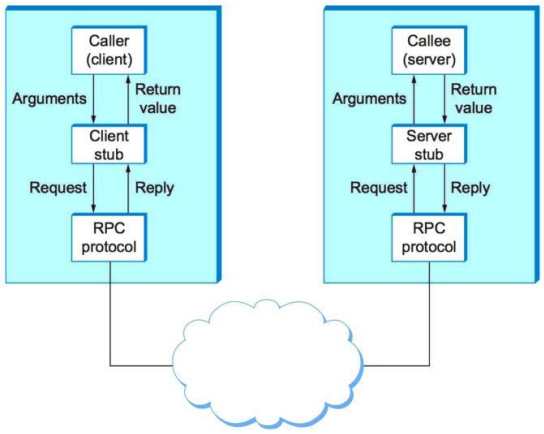
O que é UDP?

O User Datagram Protocol (UDP) é um protocolo de Internet mais simples e

sem conexão, no qual os serviços de verificação e recuperação de erros não são necessários. Com o UDP, não há sobrecarga para abrir, manter ou encerrar uma conexão — os dados são continuamente enviados para o destinatário, quer ele os receba ou não. Qual a diferença de TCP para UDP? TCP é um protocolo orientado à conexão e fornece a comunicação confiável com vários recursos. Por outro, o UDP é um protocolo de transporte em tempo real sem conexão, que oferece um processo de comunicação mais rápido, mas com pouca confiabilidade.

4.3 **RPC: REMOTE PROCEDURE CALL** ou chamada de procedimento remoto, é um tipo de IPC que permite que um sistema chame uma função ou procedimento em outro sistema como se estivesse no local, mesmo que estejam em locais remotos. Abordagem bastante útil em **sistemas distribuídos**, já que diferentes partes do sistema precisam trocar informações de forma rápida e eficaz.

Funciona da seguinte forma: vamos imaginar dois computadores conectados por uma rede. Um computador faz uma solicitação de uma função(procedimento) o outro computador responde, executa essa função e devolve o resultado. Com isso temos dois componentes principais: O **Cliente RPC e o Servidor RPC.** O cliente RPC é a máquina que faz a solicitação e o servidor RPC processa essa solicitação e envia a resposta. Quando o cliente faz a chamada, o **RPC protocol** traduz essa solicitação para um formato que o servidor entenda, assim o servidor executa o procedimento solicitado e envia a resposta de volta para o cliente.



Remote Procedure Call (RPC)

4.4 RMI: RMI **(Remote Method Invocation) Remote Method Invocation (RMI)** é um mecanismo que permite a invocação de métodos em objetos remotos, promovendo a comunicação entre processos em diferentes máquinas na rede. Diferente do RPC, o RMI trabalha diretamente com objetos Java, permitindo a transferência de referências a objetos e não apenas dados primitivos.

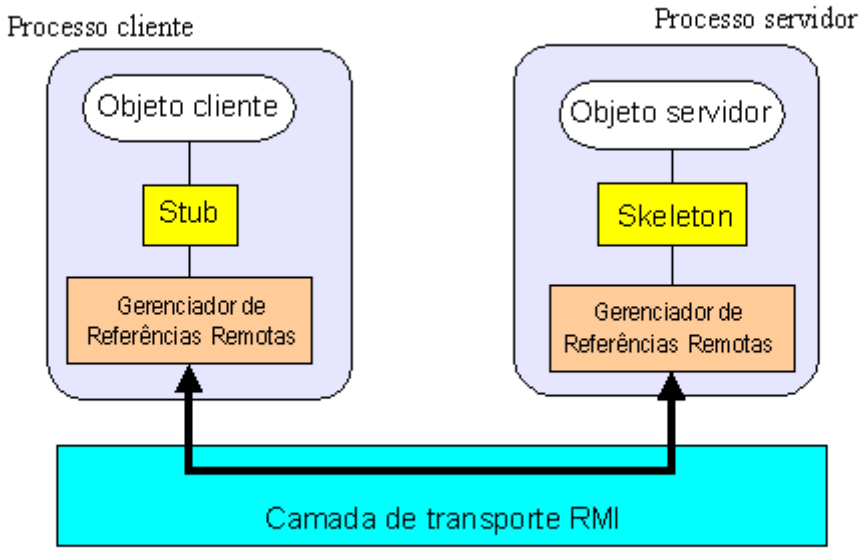
Funcionamento do RMI: Cliente e Servidor: Um cliente faz uma chamada a um método remoto e o servidor executa essa chamada.

Registro do Servidor: O objeto remoto precisa ser registrado em um serviço de nomes (RMI Registry).

Stub e Skeleton: O cliente usa um stub para enviar a requisição ao servidor, que possui um skeleton para processar a chamada e retornar o resultado.

Vantagens do RMI:

- Permite a comunicação orientada a objetos.
- Facilidade na criação de aplicações distribuídas em Java.
- Abstrai detalhes de comunicação de rede.



5 Referências:

- O que é: Comunicação Inter-Processos • Napoleon data: 08/03/2025
- Comunicação entre processos (IPC) no sistema operacional data: 08/03/2025
  Comunicação entre Processos.ppt data: 08/03/2025
- O que é RPC (Remote Procedure Call) e como ele funciona? data: 08/03/2025
- Qual a diferença entre TCP e UDP? Entenda o que são esses protocolos! - Gaea data: 08/03/2025
- https://learn.microsoft.com/pt-br/windows/win32/ipc/about-pipes data:08/03/2025
  https://www.geeksforgeeks.org/ipc-technique-pipes/ data: 08/03/2025

- https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/ data:08/03/2025
- https://www.sfu.ca/sasdoc/sashtml/os2/zipeover.htm data: 08/03/2025
- https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681 data:08/03/2025
- https://www.devmedia.com.br/tutorial-rmi-remote-method-invocation/6442

data: 08/03/2025

- https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2017_2/rmi/o_que_e.html data: 08/03/2025