Programação com Objetos Distribuídos

Comunicação entre processos: Sockets

Março/ 2022

Comunicação Entre Processos

A **comunicação entre processos: Inter-Process Communication** (IPC) é o grupo de mecanismos que permite aos processos transferirem informações entre si.

Durante a execução de um processo, o sistema operacional abstrai, entre outras coisas, a criação de um contexto de execução próprio que, de certa forma, oculta do processo os componentes reais do sistema.

Devido a esta abstração de recursos, **o processo não tem conhecimento acerca dos outros processos** e, como tal, não consegue trocar informação.

Comunicação Entre Processos

Sincronização entre processos:

- Permite gerenciar o acesso concorrente a recursos do sistema operacional de forma controlada por parte dos multiplos processos, de maneira que um recurso não seja modificado em simultâneo, ou que os processos não fiquem *em espera* que o recurso seja libertado. São usados:
  - Relógios
  - Sinais
  - Semáforos

Comunicação Entre Processos

Sincronização entre Processos

- **Memória Compartilhada**:
  - uma região de memória é compartilhada entre os processos (Threads).
- **Troca de Mensagem**:
  - Ocorre por meio de troca de mensagens entre os processos (Pipe, Sockets, RPC).

Troca de Mensagem: Socket

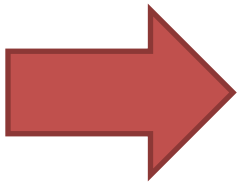
O que é um socket?

- É uma interface com qual processos em diferentes hosts se comunicam através da rede;

- Um socket é a interface entre a camada de aplicação e a de transporte da rede.

O que é um socket?

Em outras palavras, os sockets foram a forma de permitir que dois processos se comuniquem (Inter-process communication).



Esses processos podem

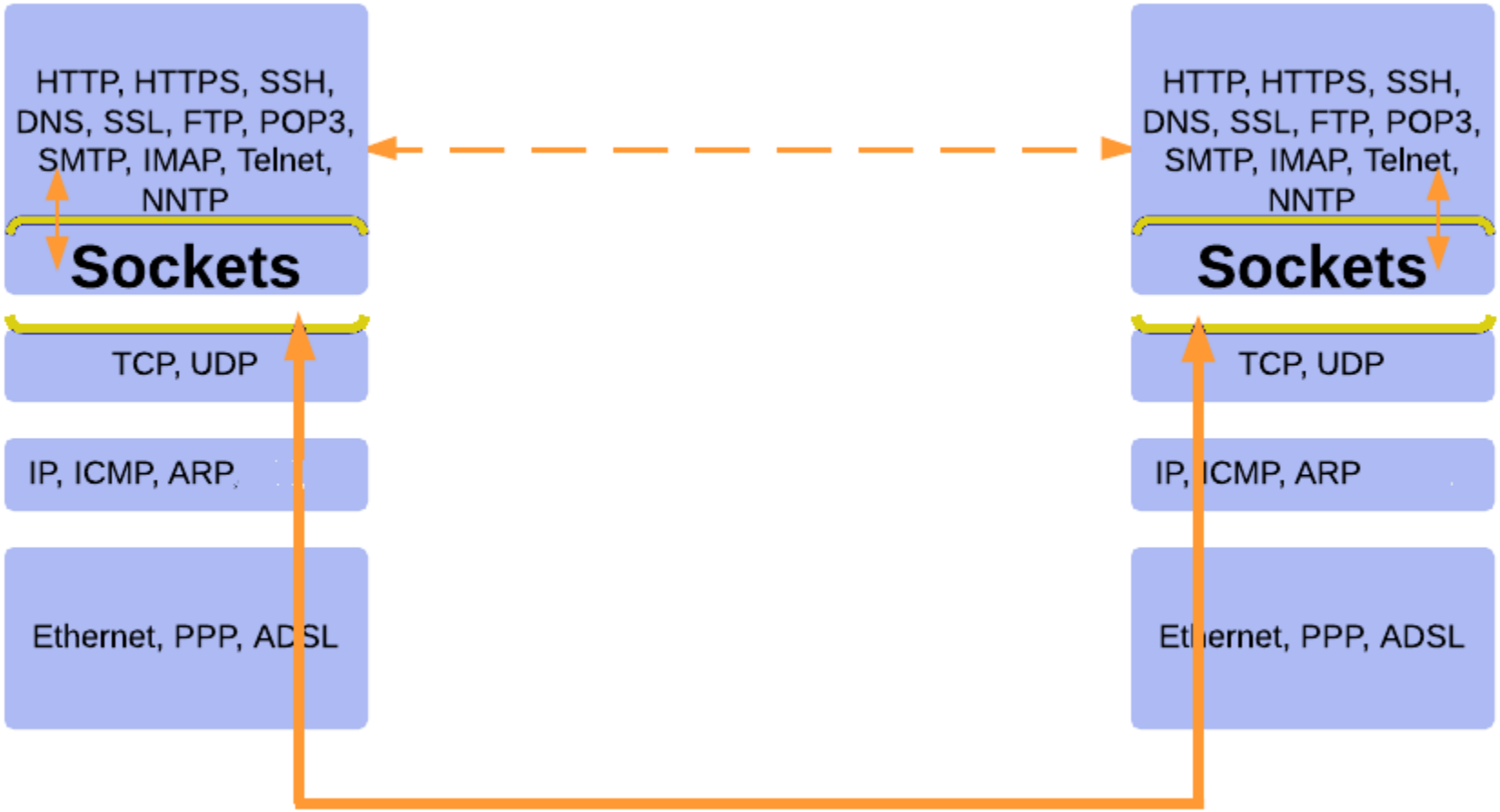
ou não estar na mesma máquina.

O que é um socket?

Diversas aplicações que utilizamos no dia-a-dia fazem uso de sockets pra se comunicar:

- Nosso navegador web utiliza sockets pra requisitar páginas.
- Quando um sistema se integra com um banco de dados ele abre um socket.
- Quando fazemos um ssh em um servidor estamos abrindo e utilizando um socket.

O que é um socket?



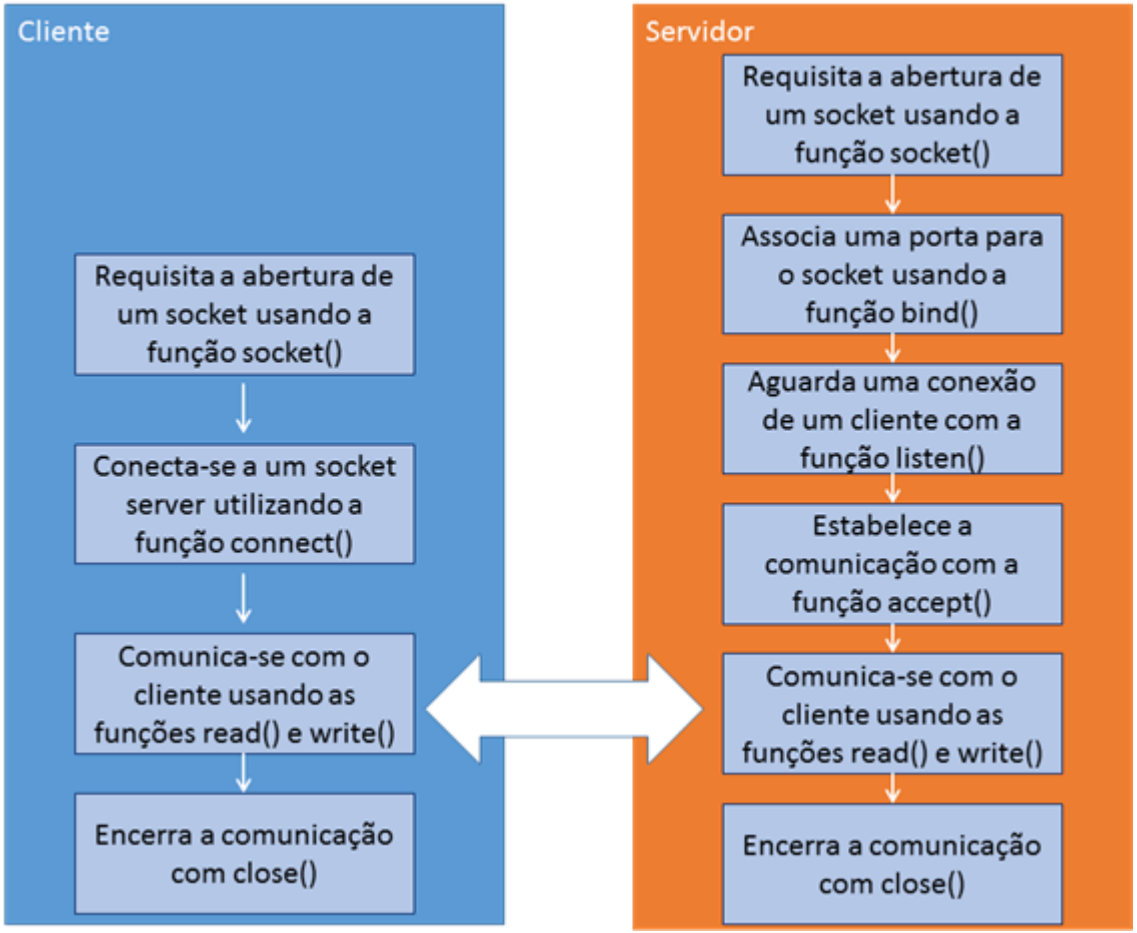
O que é um socket?

1) O programa cliente primeiro cria um *socket* através da função *socket()*.

2) Em seguida ele se conecta ao servidor através da função *connect()* e inicia um *loop* (laço) que fica fazendo *send()* (envio) e *recv()* (recebimento) com as mensagens específicas da aplicação.

3) É no par *send (write), recv (read)* que temos a comunicação lógica.

4) Quando alguma mensagem da aplicação diz que é o momento de terminar a conexão, o programa chama a função *close()* para finalizar o *socket*.



1) O programa servidor também utiliza a mesma API de *sockets.* Ou seja, inicialmente ele também cria um *socket.* No entanto, diferentemente do cliente, o servidor precisa fazer um *bind(),* que associa o *socket* a uma porta do sistema operacional, e depois utilizar o *listen()* para escutar novas conexões de clientes nessa porta.

2) Quando um novo cliente faz uma nova conexão, a chamada *accept()* é utilizada para começar a se comunicar.

3) Da mesma forma que no cliente, o servidor fica em um *loop* (laço) recebendo e enviando mensagens através do par de funções *send()* e *recv()*.

4) Quando a comunicação com o cliente termina, o servidor volta a aguardar novas conexões de clientes.

O que é um socket?

- Existe diversos tipos de Socket porém os dois mais importantes (conhecidos) são: "**Stream Sockets**" e os "**Datagram Sockets**”.

- **Stream Sockets (TCP)**
  - Usados aplicações como telnet, www, etc. Os pacotes são sequenciais e seguem nos dois sentidos.
- **Datagrams Sockets (UDP).**
  - Os pacotes não são sequenciais. Envia em sentido único.

Há outros tipos conhecidos de sockets (https://sites.uclouvain.be/SystInfo/usr/include/bits/**socket.h**.html).

Estrutura de um Socket

Estrutura de endereçamento de um socket:

```
struct sockaddr_in{
    short int sin_family; (define o tipo de família do protocolo)
    unsigned short int sin_port; (número da porta TCP ou UDP)
    struct in_addr sin_addr; (endereço IP do host de destino)
    unsigned char sin_zero[8]; (limpar a estrutura nao usada)
}
```

Criação do socket – Função socket()

int socket(int family, int type, int protocol);

int family:

indica a família de protocolos que será utilizada (PF_INET).

int type:

define o tipo de socket a ser criado (para UDP, SOCK_DGRAM e para TCP, SOCK_STREAM).

int protocol:

identifica o protocolo específico a ser usado. Neste caso será nulo, já que os dois primeiros argumentos já identificam exclusivamente o protocolo.

Se o socket é criado, retorna o descritor de arquivos para este socket, caso contrário retorna um valor negativo.

Criação do socket – Função socket()

int family:

AF_INET: Arpa Internet Protocols
AF_UNIX: Unix Internet Protocols
AF_ISSO: Iso Protocols
AF_NS: Xerox Network System Protocols

Criação do socket – Função socket()

int protocol:

IPPROTO_IP: Internet Protocol (0)
IPPROTO_ICMP: Internet Control Message Protocol (1)
IPPROTO_IGMP: Internet Group Multicast Protocol (2)
IPPROTO_GGP: Gateway-Gateway Protocol (3)
IPPROTO_TCP: Transmission Control Protocol (6)
IPPROTO_UDP: User Datagrama Protocol (17)

Associação do socket a uma porta – Função bind()

int bind(int socket, struct sockaddr *address, int addr_len);

int socket:

é o socket criado pela função socket().

struct sockaddr *address:

é a estrutura de endereçamento que contém as informações necessárias para o estabelecimento da associação.

int addr_len:

é o tamanho dessa estrutura, pois, dependendo da família e do protocolo utilizados, pode mudar o valor.

Retorna um valor negativo em caso de insucesso.

Envio de mensagens usando UDP – Função sendto()

ssize_t = sendto(int socket, char *message, int msg_len, int flags, struct sockaddr *address, int addr_len);

char *message:

é o endereço da variável onde se encontra a mensagem que se deseja transmitir pelo socket.

int msg_len:

tamanho dessa mensagem.

int flags:

é um conjunto de flags que controlam certos detalhes da operação mas que podem receber um valor nulo.

struct sockaddr *address:

estrutura de endereçamento de destino.

int addr_len:

Tamanho da estrutura de endereçamento (sizeof address).

Retorna o número de bytes enviados ou –1, em caso de erro.

Recepção de mensagens usando UDP – Função recvfrom()

ssize_t = recvfrom(int socket, char *buffer, int buffer_len, int flags, struct sockaddr *address, int *addr_len);

A função retorna, além do datagrama recebido (no segundo argumento), a estrutura de endereçamento da origem de forma que o destino possa enviar-lhe uma resposta (quinto argumento) e o número de bytes da área de dados recebidos ou –1 em caso de erro.

Fechando o socket – Função close()

int close(int socket);

A função retorna um valor nulo em caso de sucesso.

Criando um socket no Windows

Criando um socket no windows

WSAStartup()

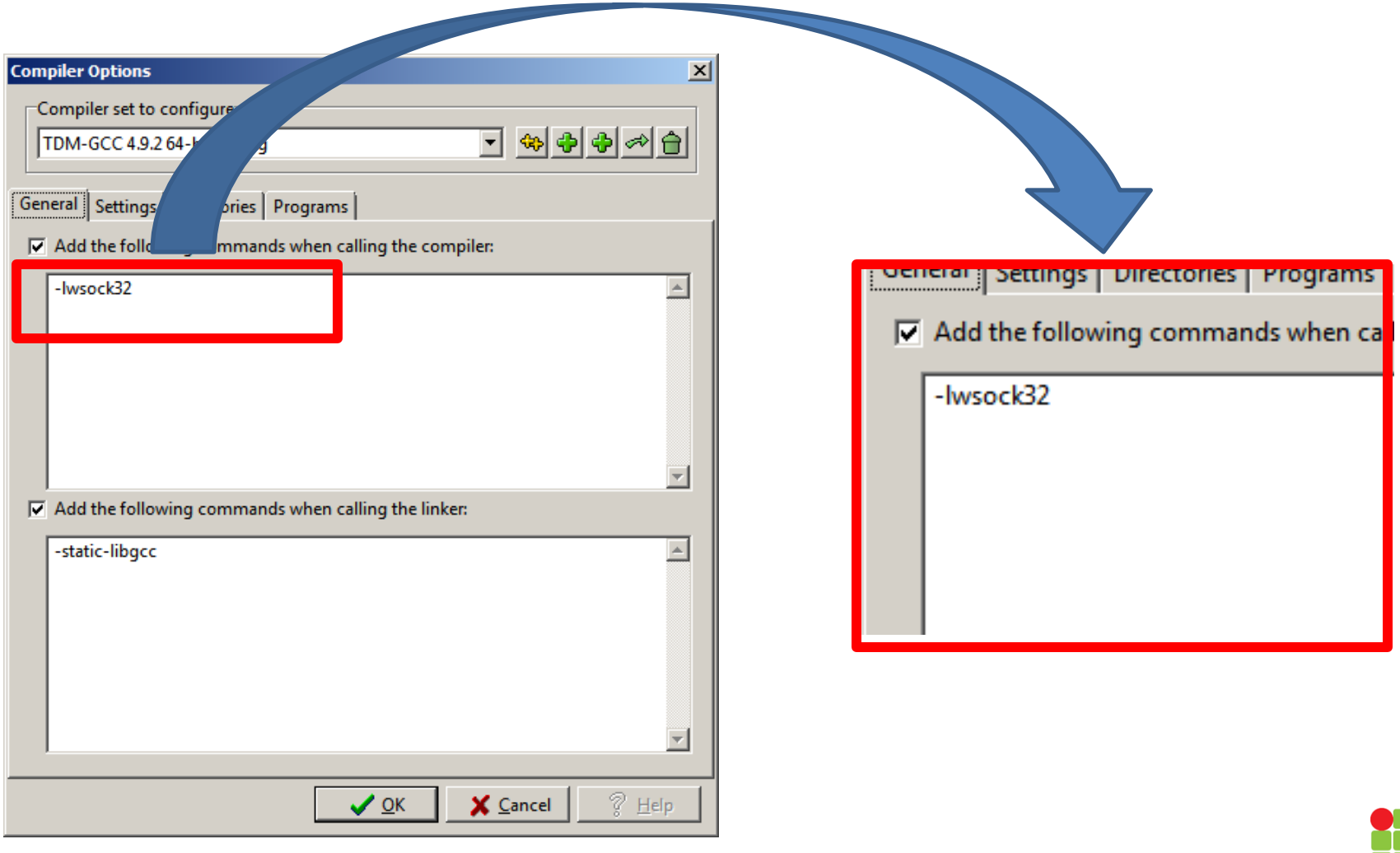
A função **WSAStartup()** inicia o *Windows Sockets Dynamic Link* (WinSock DLL).

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

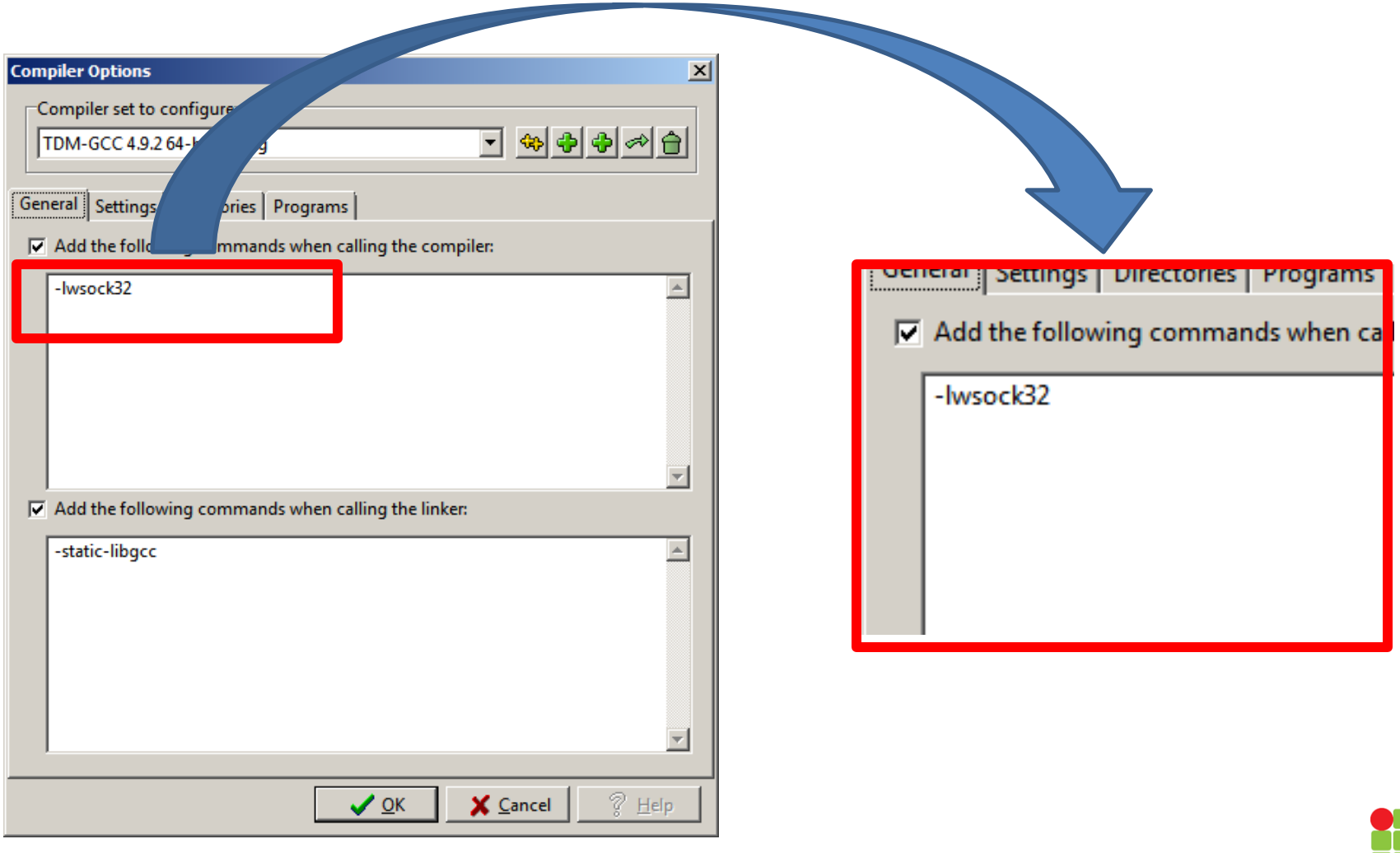
O parâmetro **wVersionRequested** é número da maior versão que a aplicação pode usar. Repare que ele é uma **WORD** onde o byte de maior ordem especifica a número da minor version e o byte de ordem menor indica a major version.

O parâmetro **lpWSAData** é um ponteiro para um estrutura **WSADATA** que receberá os detalhes da implementação do WinSock.

Criando um socket no windows (DEVC++)



Criando um socket no windows (DEVC++)



Código servidor.cpp

Código cliente.cpp