Sockets

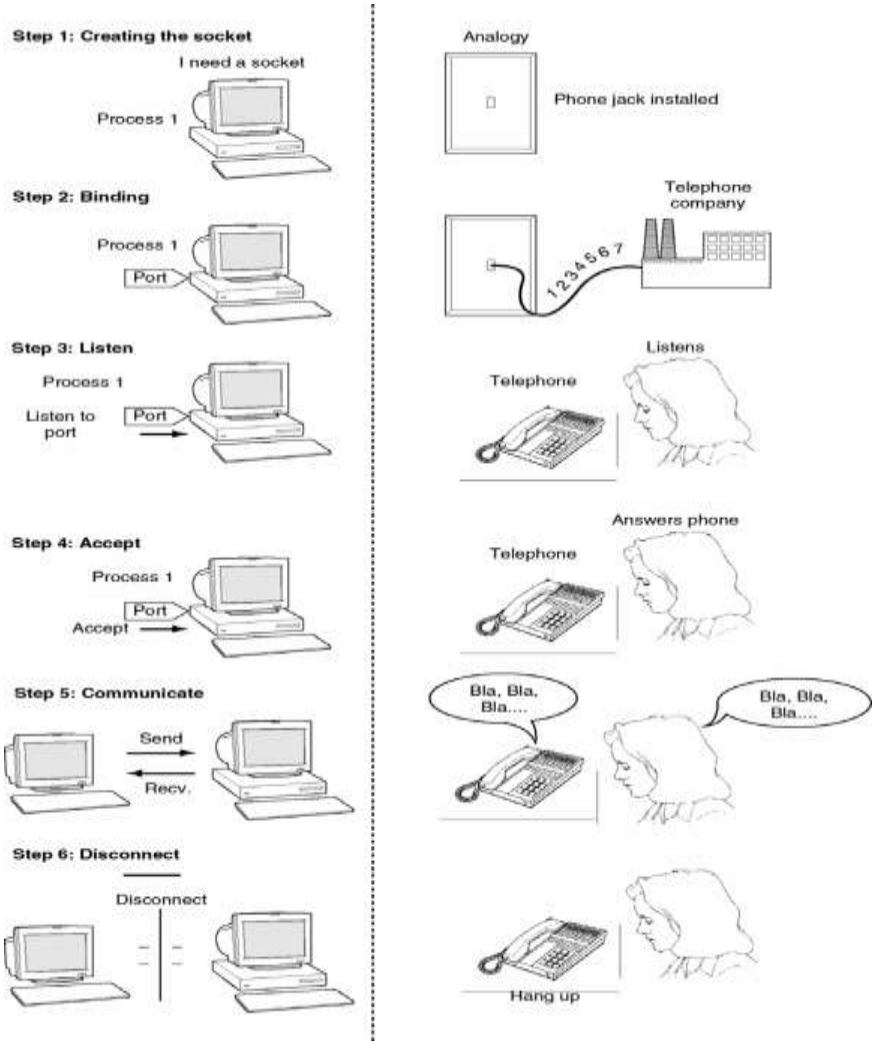
Definição e características:

- Um socket é um ponto final (endpoint) de um canal bidirecional de comunicação entre dois programas rodando em uma rede.
- Dados escritos em um socket por um processo são enviados e lidos pelo outro processo.
- Cada socket tem os seguintes endereços:
  - Endereço local (número da porta)
  - Endereço global (IP) do computador (host) na rede

Formas de comunicação:

- O socket encapsula (esconde) os detalhes de transmissão e recepção de dados.
- Existem várias formas de fazer a comunicação, obedecendo a diferentes protocolos:
  - UDP (*User Datagram Protocol*): protocolo rápido, orientado a datagramas, mas sem garantias de entrega de dados. Socket se comunica com qualquer outro socket.
  - TCP (*Transmission Control Protocol*): protocolo orientado a conexão com garantia contra perdas e desordenamento de pacotes. O socket só se comunica com o socket com o qual está conectado. Para haver a transmissão dos dados, uma fase de conexão entre as duas entidades que se comunicam precisa ser feita.

ANALOGIA COM O PROTOCOLO TCP



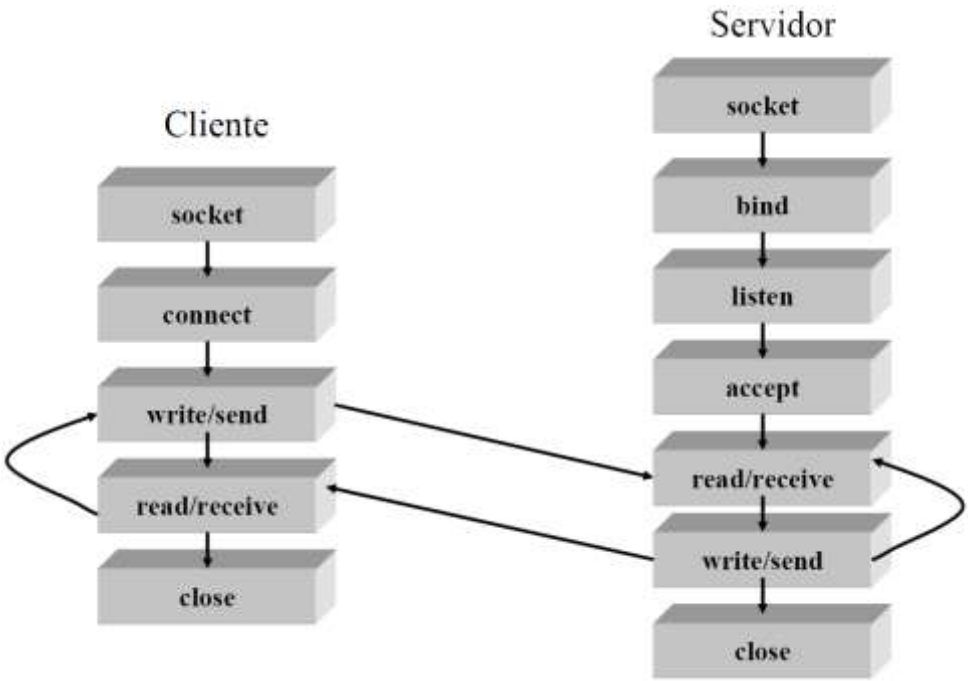
Modelo cliente-servidor (no caso dos protocolos orientados a conexão):

- Para qualquer par de aplicações que se comunicam, um dos lados deve iniciar a execução e esperar até ser contatado pelo outro lado para iniciar a comunicação.
- Um servidor é um programa que inicia a execução e apenas fica "ouvindo" o Socket, aguardando um pedido de conexão do cliente.
- Uma aplicação que inicia a comunicação par-a-par é geralmente chamada cliente. O cliente sabe o nome do host (endereço IP) e qual porta está associada à aplicação servidora.
- Assim que o servidor aceita a conexão, cria um novo socket e pode ficar esperando novas conexões no socket original, enquanto atende às requisições do cliente pelo novo socket.

Tipos de funções dos sockets cliente-servidor:

- O servidor possui dois tipos de sockets:
  - Um socket de recebimento de conexões, permanentemente aberto.
  - Um ou mais sockets de comunicação, sendo um para cada cliente.
- O cliente possui um único socket de comunicação, similar aos existentes no servidor, através do qual o cliente se comunicará com o servidor.

FUNÇÕES DO CLIENTE	FUNÇÕES DO SERVIDOR
Efetua a criação do socket	Efetua a criação de um socket
	Associa o socket a um endereço local
	Aguarda por conexões da parte cliente
Estabelece a conexão	Aceita conexões (cria um novo socket)
Envia requisições	Lê requisições (no novo socket)
Opcionalmente aguarda resposta	Opcionalmente envia resposta (no novo socket)
Fecha o socket	Fecha o novo socket



Operações iniciais com sockets:

- `getaddrinfo(host_name, port_name, in, out)`: Determina o endereço (local e global) a ser utilizado na criação de um socket.
  - `host_name`: endereço do computador a se comunicar (char* com IP ou nome do servidor, no caso do cliente, ou `NULL` para o socket de conexões no servidor).

- `port_name`: char* com número ou nome da porta.
- `in`: struct do tipo `addrinfo` com o tipo de socket desejado:
  - `in.ai_family` = AF_INET (IPv4), AF_INET6 (IPv6), AF_UNSPEC (IPv4 ou IPv6)
  - `in.ai_socktype` = SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
  - `in.ai_protocol` = IPPROTO_TCP, IPPROTO_UDP, 0=padrão
  - `in.ai_flags` = AI_PASSIVE se `host_name` for NULL (servidor)
- `out`: struct do tipo `addrinfo` que retorna os endereços do socket criado. São os campos dessa struct que devem ser usados para criar o socket.

- `id = socket(out->ai_family, out->ai_socktype, out->ai_protocol)`: Cria um socket com as características desejadas e retorna um identificador ou `INVALID_SOCKET` em caso de erro. Os parâmetros da função devem ter sido calculados através de uma chamada prévia a `getaddrinfo`.
- `freeaddrinfo(out)`: libera a memória alocada para conter o resultado da chamada à função `getaddrinfo`. Deve ser chamada quando a struct não for mais necessária.

Operações no servidor:

- `bind(id, out->ai_addr, out->ai_addrlen)`: Associa o socket a um número da porta no qual o servidor espera contato. O primeiro parâmetro da função é o identificador retornado pela função `socket`. Os dois últimos parâmetros da função devem ter sido calculados através de uma chamada prévia a `getaddrinfo`. Retorna 0 em caso de sucesso ou `SOCKET_ERROR` em caso de erro (por exemplo, porta já sendo utilizada).
- `listen(id, num_conex)`: Aguarda por conexões da parte cliente. O primeiro parâmetro é o identificador do socket de recebimento de conexões; o segundo é o número máximo de conexões pendentes. Retorna 0 ou `SOCKET_ERROR`.
- `new_id = accept(id, addr, addrlen)`: Aceita uma conexão pendente no socket de recebimento de conexões `id` e cria um socket conectado ao cliente, retornando o identificador desse novo socket ou `INVALID_SOCKET` em caso de erro. Os dois últimos parâmetros geralmente são `NULL`; caso não sejam, retornam o endereço e o tamanho do endereço do socket que está se conectando. **Essa função é bloqueante**: a execução será interrompida até que haja uma conexão pendente para ser aceita.
- `select(nfds, *readfds, *writefds, *exceptfds, *timeout)`: testa se conjuntos de sockets estão em um determinado estado. O parâmetro `nfds` é ignorado. Os parâmetros `readfds`, `writefds` e `exceptfds` são ponteiros para conjuntos de sockets a serem testados respectivamente sobre:
  - disponibilidade de dados para leitura;
  - capacidade de escrita de dados; ou
  - ocorrência de erro.

  O parâmetro `timeout` determina o tempo máximo que o `select` deve esperar, fornecido através de um ponteiro para uma estrutura do tipo `timeval` ou `NULL`, caso deva esperar indefinidamente (operação bloqueante). A estrutura `timeval` tem dois campos: `tv_sec` (segundos) e `tv_usec` (microssegundos), sendo o período máximo de espera dado pela soma dos dois tempos.

  Os parâmetros `readfds`, `writefds` e `exceptfds` são ponteiros para estruturas do tipo `fd_set`. Se o valor for `NULL`, o aspecto correspondente não será testado. Geralmente, o `select` é chamado com apenas um dos três parâmetros não sendo `NULL`. Como a operação mais frequente é o teste de disponibilidade de dados para leitura, geralmente `readfds` aponta para um conjunto e `writefds` e `exceptfds` são `NULL`.

  Um socket incluído no conjunto `readfds` fará o `select` retornar se uma das seguintes condições acontecer:
  - Para sockets no estado de escuta, se uma solicitação de conexão tiver sido recebida, de modo que seja garantido que um `accept` será concluído sem bloqueio.

- Para outros sockets, se houver dados disponíveis para leitura, de modo que uma chamada para `recv` tenha a garantia de não bloquear.
- A conexão foi fechada.

Para manipular os conjuntos de sockets do tipo `fd_set`, existem algumas macros:

- `FD_SET(s, *set)` : inclui o socket `s` no conjunto `set`.
- `FD_CLR(s, *set)` : remove o socket `s` do conjunto `set`.
- `FD_ISSET(s, *set)` : retorna `true` se socket `s` faz parte do conjunto `set`.
- `FD_ZERO(*set)` : limpa o conjunto `set` (remove todos os sockets).

A função `select` retorna um dos seguintes valores:

- o número total de sockets que estão prontos e contidos nas estruturas `fd_set`;
- zero se o limite de tempo expirou; ou
- `SOCKET_ERROR` se ocorreu um erro.

Ao retornar, as estruturas passadas como parâmetro são alteradas para conterem apenas os sockets que estão prontos. Portanto, para testar se existe dado disponível para leitura em um determinado socket que fazia parte do conjunto `readfds` quando a função `select` foi chamada, deve-se testar se o socket em questão ainda faz parte desse conjunto após a execução da função, usando a macro `FD_ISSET`: se sim, indica que ele tem dados a serem lidos; se não, uma eventual operação de leitura ficaria bloqueada (não há dados).

Operações de leitura e escrita:

- `recv(id,dado,len,flag)`: lê dados do socket `id`. Os parâmetros `dado` e `len` contêm um ponteiro para a área de memória onde os bytes lidos devem ser armazenados e o número de bytes, respectivamente. O último parâmetro, muitas vezes igual a 0, é um flag que controla alguns aspectos da operação de leitura. Por exemplo, se esse parâmetro for igual a `MSG_WAITALL`, a função só retornará quando todos os `len` bytes forem recebidos. **Essa função é bloqueante**: a execução será interrompida até que haja dados a serem lidos. Retorna o número de bytes lidos (que pode ser menor ou igual que `len`), 0 caso a conexão tenha sido fechada corretamente ou `SOCKET_ERROR` em caso de erro.
- `send(id,dado,len,flag)`: escreve dados no socket `id`. Os parâmetros `dado` e `len` contêm um ponteiro para a área de memória onde estão os bytes que devem ser enviados e o número de bytes, respectivamente. O último parâmetro, muitas vezes igual a 0, é um flag que controla alguns aspectos da operação de escrita. Essa função geralmente não é bloqueante. Retorna o número de bytes enviados (que pode ser menor ou igual que `len`) ou `SOCKET_ERROR` em caso de erro.

Operações finais

- `close(id)`: informa ao sistema operacional para terminar o uso do socket `id`.

A biblioteca mySocket

Definição e características:

- Trata-se de uma pequena biblioteca desenvolvida pelo professor **para fins didáticos**. Para uso profissional, sugere-se utilizar alguma das bibliotecas de sockets que podem ser encontradas na rede.

Tipos definidos:

- `MYSOCKET_STATUS` – valor de retorno de várias funções:
  - `SOCKET_ERROR` – valor de erro (-1 ou qualquer valor <0).
  - `SOCKET_OK` – valor correto (=0).
  - Valor positivo – indica um valor correto.
- `MYSOCKET_STATE` – tipo enumerado com os estados possíveis de um socket:
  - `MYSOCKET_IDLE` – socket inativo (não inicializado ou fechado).
  - `MYSOCKET_ACCEPTING` – socket servidor aceitando conexões.
  - `MYSOCKET_CONNECTED` – socket de comunicação conectado.
- `mysocket` – classe base para todos os sockets (não deve ser utilizada para criar objetos).
- `tcp_mysocket` – socket TCP de comunicação.
- `tcp_mysocket_server` – socket TCP servidor (para aceitar conexões).
- `mysocket_queue` – fila de sockets para esperar por eventos (utilizando select).

Classe `mysocket`:

- `shutdown` – fecha o socket para novos envios, mas ainda permite leitura de mensagens que estejam em trânsito. Retorna `MYSOCKET_STATUS`.
- `swap(sock)` – permuta dois sockets. A ser utilizado ao invés do operador de atribuição.
  - `sock` – socket a ser permutado com o objeto em questão.
- `close` – desconecta o socket.
- `closed` – retorna true se o socket está inativo (estado `MYSOCKET_IDLE`).
- `accepting` – retorna true se o socket servidor está aceitando conexões (estado `MYSOCKET_ACCEPTING`).
- `connected` – retorna true se o socket de comunicação está conectado (estado `MYSOCKET_CONNECTED`).

Classe `tcp_mysocket`:

- `connect(name, port)` – conecta-se a um socket servidor que esteja em estado de espera. Retorna `MYSOCKET_STATUS`.
  - `name` – array de char com o número IP do host servidor.
  - `port` – array de char com o número da porta.
- `read(buff, len, time)` – Lê bytes de um socket. Retorna `MYSOCKET_STATUS`: o número de bytes lidos; zero, se retornou por timeout; ou `SOCKET_ERROR`.
  - `buff` – ponteiro para o início do buffer no qual os bytes recebidos serão salvos.
  - `len` – número de bytes a serem lidos.
  - `time` – timeout em milissegundos (negativo se esperar indefinidamente).
- `write(buff, len)` – Escreve bytes em um socket. Retorna `MYSOCKET_STATUS`: o número de bytes enviados ou `SOCKET_ERROR`.
  - `buff` – ponteiro para o início do buffer de bytes a serem lidos e enviados.
  - `len` – número de bytes a serem transmitidos.
- `read_int(num, time)` – Lê um inteiro de 4 bytes (`int32_t`) de um socket. Retorna `MYSOCKET_STATUS`: o número de bytes lidos (4) ; zero, se retornou por timeout; ou `SOCKET_ERROR`.

    - num – referência para o número inteiro a ser lido.
    - time – timeout em milissegundos (negativo se esperar indefinidamente).
- write_int(num) – Escreve um inteiro de 4 bytes (int32_t) em um socket. Retorna MYSOCKET_STATUS: o número de bytes enviados (4) ou SOCKET_ERROR.
    - num – número inteiro a ser enviado.
- read_string(str, time) – Lê uma string C++ de um socket. Retorna MYSOCKET_STATUS: o número de bytes lidos (comprimento da string) ; zero, se retornou por timeout; ou SOCKET_ERROR.
    - str – referência para a string a ser lida.
    - time – timeout em milissegundos (negativo se esperar indefinidamente).
- write_string(str) – Escreve uma string C++ em um socket. Retorna MYSOCKET_STATUS: o número de bytes enviados (comprimento da string) ou SOCKET_ERROR.
    - str: string a ser enviada.

Classe tcp_mysocket_server:

- listen(port, nconex) – coloca o socket servidor em estado de escuta, esperando conexões. Retorna MYSOCKET_STATUS.
    - port – array de char com o número da porta.
    - nconex – número máximo de conexões pendentes que o socket vai aceitar.
- accept(sock) – aceita uma nova conexão. Retorna MYSOCKET_STATUS.
    - sock – referência para o novo socket criado (parâmetro de retorno).