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.

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 espera por requisições de um cliente.
- Uma aplicação que inicia a comunicação par-a-par é geralmente chamada cliente.

Tipos de funções dos sockets:

- 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 ao(s) existente(s) no servidor, através do qual o cliente se comunicará com o servidor.

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
    - `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. 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`.
- `listen(id, num_conex)`: Aguarda por conexões da parte cliente. O primeiro parâmetro é o identificador do socket; o segundo é o número máximo de conexões pendentes.
- `new_id = accept(id, addr, addrlen)`: Aceita uma conexão pendente no socket `id` e cria um socket conectado ao cliente, retornando o identificador desse novo socket. 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.

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. **Essa função é bloqueante**: a execução será interrompida até que haja dados a serem lidos. Retorna o número de bytes lidos, 0 caso a conexão tenha sido fechada ou SOCKET_ERROR em caso de erro.
- `send(id,dado,len,flag)`: escreve dados do 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`.