Aula 3 - Comunicação entre processos

Um processo é um programa em execução. Quando dizemos que dois computadores estão se comunicando, nós estamos dizendo que dois processos destes computadores estão se comunicando. Normalmente, em sistemas distribuidos, dois processos executando em computadores diferentes precisam se comunicar para atingir alguma meta em comum. Desta forma o S.O.D precisa fornecer alguma forma de comunicação entre processos (IPC) para facilitar estas atividades de comunicação.
Os processos que estão se comunicando precisam de alguma forma de compartilhar informações. Existem basicamente dois métodos para tal:

- Memória compartilhada
- Troca de mensagens

Como computadores em um rede não compartilham memória, processos em um S.O.D geralmente se comunicam através de troca de mensagens. Veremos como processos podem utilizar memória compartilhada para comunicação mais adiante.
Veremos três formas de fazer comunicação entre processos já disponivel nos sistemas atuais:

1. Sockets
2. Remote Procedure Call (RPC)
3. Remote Method Invocation (RMI/CORBA/...)

Unix BSD Sockets

Sockets são a base para comunição em sistemas Unix, e se tornaram um padrão de comunicação sobre a Internet. Sockets podem ser vistos como uma abstração de mecanismos de acesso a arquivos que provêem a funcionalidade de comunicação. Asim como com acesso a arquivos, aplicações podem requisitar ao sistema operacional a criação de um socket quando elas necessitarem. O sistem então retorna um descritor para o socket (um inteiro), que a aplicação utiliza para referenciar este novo socket. A principal diferença entre um descritor de arquivo e um descritor de socket é que o sistema operacional associa um descritor de arquivo a um arquivo específico ou dispositivo quando a aplicação usa a rotina *open*, enquanto que sockets podem ser criados sem associação com um endereço específico. A aplicação pode fornecer um endereço cada vez que a mesma utiliza o socket (e.g. quando estiver enviando uma mensagem UDP (*datagram*)), ou pode escolher associar um endereço ao socket e evitar ter que informar o endereço repetidamente (e.g. quando faz uma conexão TCP).

As operações sobre sockets são muito similares as operações sobre arquivos, pode-se, quando necessário, escrever (*write*) ou ler (*read*) de um socket. Por exemplo, após a aplicação ter criado um socket e criado uma conexão TCP daquele socket para um endereço remoto, a aplicação pode escrever (*write*) naquele socket para enviar informações através da conexão (a aplicação no outro lado pode ler (*read*) estas informações).

Criando um socket - *socket*

A chamada de sistema para criar um socket é *socket*, que necessita de três parâmetros:
`result = socket(pf, type, protocol)`

- `pf` especifica a família do protocolo a ser usado com o socket, isto é, como interpretar os endereços quando eles são passados para o socket. Alguns exemplos são: PF_INET (para TCP/IP), PF_PUP (para Xerox PUP), PF_UNIX (para sistema Unix), ...
- `type` especifica o tipo de comunicação desejada. Tipos possíveis são: comunicação confiável de entrega de mensagem, baseado em conexão (SOCK_STREAM); serviço de entrega de mensagem sem conexão e sem garantia de entrega (SOCK_DGRAM), e outros.
- `protocol` especifica qual o protocolo sendo utilizado, e.g. TCP/IP ou UDP/IP. Caso o valor for zero então o sistema escolhe o protocolo a ser utilizado.

Associando um endereço a um socket - *bind*

Inicialmente um socket é criado sem ser associado a qualquer endereço local ou de destino. Para o protocolo TCP/IP, isto significa que nenhuma porta local ou endereço IP foram especificados. Em muitos casos, aplicações não se importam em escolher portas ou endereços, deixando que os protocolos façam isto por elas. Entretanto, servidores que esperam conexões ou informações em determinadas portas, devem especificar esta porta para o sistema operacional. Assim que o socket for criado, um servidor deve usar a chamada de sistema *bind* para estabelecer um endereço local para ele. *bind* tem o seguinte formato:
`bind(socket,localaddr,addrlen)`

- `socket` é o descritor do socket a ser associado ao endereço e porta.
- `localaddr` é uma estrutura à qual o socket deve ser associado. Possui campos para informar o endereço (`sin_addr`), a porta (`sin_port`), e a família de protocolos (`sin_family`).
- `addrlen` especifica o tamanho da estrutura de dados do endereço.

Conectando sockets -*connect*

As duas formas de comunicação mais comuns utilizadas no Unix é a comunicação baseada em conexão (*stream* - TCP) e comunicação sem conexão (*datagram* - UDP). Em uma comunicação baseada em conexão, primeiramente dois processos estabelecem uma conexão via um par de sockets. O estabelecimento da conexão é assimétrico, pois um processo permanece esperando por uma requisição de conexão e o outro processo faz tal requisição. Uma vez que a conexão esteja estabelecida, informações podem ser transmitidas entre os dois processos na duas direções. Este tipo de comunicação é utilizada em aplicações cliente-servidor. Um servidor cria um socket, associa um nome e porta a este socket, e torna-o público. O servidor então espera por uma requisição de conexão de um processo cliente. Quando a conexão estiver sido estabelecida, o cliente e servidor podem começar a troca de informações. Comunicação baseada em conexão suporta troca de mensagens de maneira confiável.
Em comunicação sem conexão, um par de sockets é identificado a cada troca de informações. Para isto o processo que está enviando a mensagem especifica seu descritor de socket local e o descritor de socket do processo que estará recebendo a mensagem. Este tipo de comunicação não é confiável.
A chamada de sistema para estabelecimento de conexão é a *connect* e possui o seguinte formato:
`connect(socket,destaddr,addrlen)`

- `socket` é o descritor do socket local.
- `destaddr` é uma estrutura que contem as informações sobre o socket do processo destino.
- `addrlen` é o tamanho da estrutura de dados de `destaddr`.

A chamada de sistema *connect* é utilizada por um processo cliente para requisitar o estabelecimento de uma conexão entre o seu socket e o socket do processo servidor com o qual o cliente deseja se comunicar. A chamada à *connect* automaticamente associa um endereço de socket ao socket do cliente. Desta forma, associação via *bind* não é necessária.

Especificando o tamanho da fila para um servidor - *listen*

Um servidor cria um socket, associa-o a uma porta conhecida, e espera por requisições. Se o servidor usa um processo de entrega confiável de mensagem (*stream*), ou se o cálculo da resposta de uma requisição leva um tempo considerável para ser computada, então pode acontecer de uma nova requisião chegar antes da anterior ter sido processada. De forma a evitar que o sistema de suporte simplesmente desconsidere tais requisições, o servidor deve informar o sistema de suporte que ele deseja que tais requisições sejam enfileiradas até que ele tenha tempo de processá-las.

A chamada de sistema *listen* permite servidores preparar um socket para solicitações de conexão.
O formato desta rotina é o seguinte:
`listen(socket,qlength)`
listen funciona somente com sockets que foram criados para comunicação confiável, ou seja, SOCK_STREAM.

Como um servidor aceita conexões - *accept*

Como visto anteriormente, um servidor usa as chamadas de sistema *socket*, *bind* e *listen* para cirar um socket, associa-lo a uma porta, e especificar o tamanho da fila para espera de requisições. Repare que a chamada a *bind* associa uma porta ao socket, mas que o socket não esta conectado a nenhum endereço específico. Deve-se especificar um endereço qualquer como endereço a ser passado para a chamada *bind*, por exemplo, INADDR_ANY.
Assim que o socket tiver sido configurado, conforme descrito no parágrafo acima, o servidor precisa esperar por uma conexão. Isto é realizado através da chamada *accept*. Uma chamada à *accept* bloqueia até que uma requisição de conexão chegue. Esta chamada possui o seguinte formato:
`newsock = accept(socket, addr, addrlen)`

- `socket` especifica o descritor do socket onde deve-se esperar a conexão.
- `addr` é um apontador para uma estrutura do tipo *sockaddr*. Quando uma requisição chega, o sistema preenche o argumento `addr` com o endereço do cliente que solicitou uma conexão, e define o tamanho em *addrlen*.

Após a conexão ter sido estabelecida, o sistema cria um novo socket que tem o endereço do socket do cliente e retorna este descritor para a aplicação. O socket original continua como estava anteriormente. Desta forma o servidor pode continuar a acietar conexões neste socket original.
Quando uma requisição de conexão chega ao servidor, a chamada de sistema *accept* desbloqueia e o controle retorna ao servidor. O servidor pode atender esta requisição interativamente, ou concorrentemente. No modo interativo, o servidor trata a requisição, fecha o socket recem-criado, e então chama *accept* para atender novas requisições. No modo concorrente, após *accept* ter retornado, o servidor cria um escravo (novo processo ou thread) para atender a requisição.