Sistemas Distribuídos

Comunicação entre processos

Sistema centralizado

- Comunicação entre processos
  - Arquivo
  - Memória compartilhada
  - Sinal
  - Fila de mensagem
- SO gerencia comunicação

Sistema distribuído

- Idéia do modelo cliente-servidor: estruturar o sistema de forma que servidores oferecem serviços a seus clientes.
- Servidores e clientes executam como processos do usuário sobre o *kernel* do S.O.

Comunicação entre processos

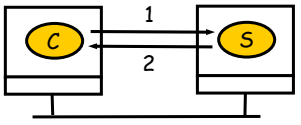
- Solução mais simples: modelo cliente-servidor baseado em protocolo sem conexão do tipo requisição/resposta (*request/reply)*
- Vantagens: estrutura simples, eficiência, evita sobrecarga

Comunicação entre processos

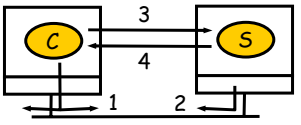
- Serviço de comunicação com duas chamadas de sistema - enviar e receber mensagens:
  - *send(dest, &mptr)* - envia a mensagem cujo endereço é dado em "*mptr*" para um processo identificado por "*dest*", mantendo-se bloqueado enquanto a mensagem está sendo enviada
  - *receive(addr, &mptr)* - recebe a mensagem de um endereço "*addr*" armazenando-a no endereço dado por "*mptr*", mantendo-se bloqueado enquanto espera a mensagem
- Muitas variações dos procedimentos, assim como de seus parâmetros são possíveis

Endereçamento

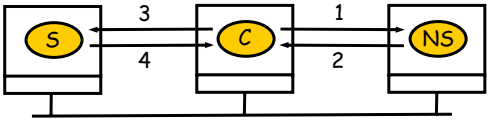
- Para um cliente enviar uma mensagem para um servidor, ele precisa conhecer o seu endereço



1: Requisição para 243.29
2: Resposta para 199.34



1: Broadcast 2: Eu estou aqui
3: Requisição... 4: Resposta...



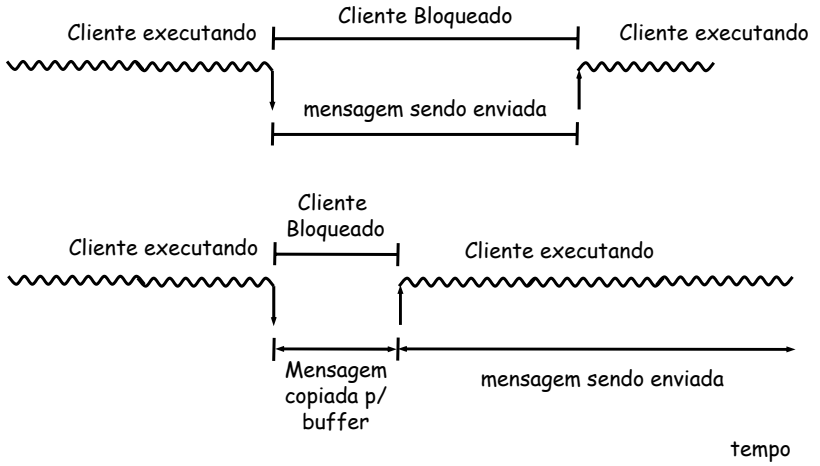
1: Lookup 2: Resposta do servidor de nome
3: Requisição... 4: Resposta...

Endereçamento

Métodos de endereçamento de processos:

1. Colocar *máquina.número* no código cliente
   - Não transparente
2. Deixar que o processo escolha endereços e localizá-los por broadcast
   - Gera carga extra na rede
3. Colocar nomes de servidores em ASCII nos clientes e procurá-los em tempo de processamento
   - Precisa de um componente centralizado (servidor de nomes)

Primitiva bloqueante (síncrona) e não bloqueante (assíncrona)



Primitiva bloqueante (síncrona) e não bloqueante (assíncrona)

Primitivas de envio/recebimento de mensagens:

1. Envio bloqueante
   - Processador fica ocioso durante a transmissão da mensagem
2. Envio não bloqueante com cópia
   - Processador gasta tempo para fazer cópia
3. Envio não bloqueante com interrupção
   - Torna a programação difícil

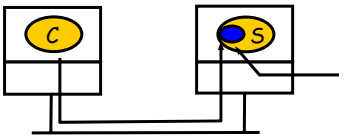
Primitiva bloqueante (síncrona) e não bloqueante (assíncrona)

- Critério distinto:
  - Síncrona: transmissor fica bloqueado até que o receptor tenha recebido a mensagem (msg de reconhecimento)
- Receive bloqueante x não-bloqueante
  - Receive condicional: testa se existe msg e retorna imediatamente
- Temporizadores:
  - Pode-se definir um período até quando a primitiva ficará bloqueada. Quando este tempo atingido retorna-se um código de erro.

Primitivas não bufferizadas

- *receive(addr,&mptr)* - informa o kernel da máquina que o processo chamado está esperando por mensagens no endereço "*addr*" e preparado para receber mensagens enviada para aquele endereço
- quando a mensagem chega, o kernel copia a mensagem p/ um buffer e desbloqueia o processo
- este esquema funciona bem desde que o servidor chame "*receive*" antes do cliente chamar "*send*"

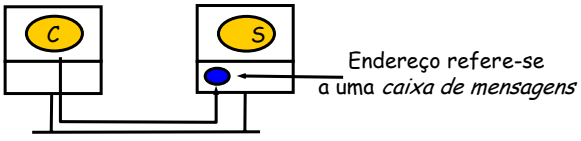
Troca de Mensagem Não Bufferizada



Primitivas bufferizadas

- um processo interessado em receber mensagens solicita ao kernel que crie uma caixa de mensagens (*mailbox)* para ele
- uma chamada "*receive*" apenas remove uma mensagem do "*mailbox*", ou bloqueia se não há nada presente
- assim, o *kernel* sabe o que fazer com mensagens que chegam e tem um local para colocá-las

Troca de Mensagem Bufferizada



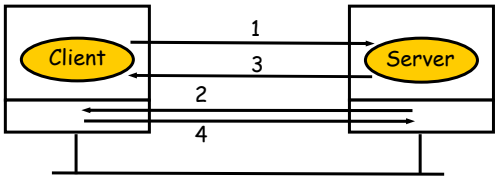
Primitivas não bufferizadas e bufferizadas

- Primitivas de envio/recebimento de mensagens:

1. Não bufferizadas descartando mensagens não esperadas
2. Não bufferizadas com armazenamento temporário das mensagens não esperadas
3. Caixas postais

Primitivas confiáveis e não confiáveis

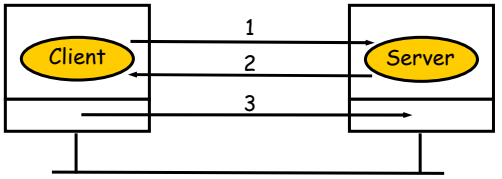
- o kernel da máquina que recebe envia um "reconhecimento" (acknowledgment) para o kernel da máquina que envia
- o "reconhecimento" vai de um kernel para outro
- *requisição* e *resposta* se tornam 4 mensagens, em decorrência dos dois "reconhecimentos"



1: requisição (cliente p/ servidor)
2: ACK (kernel p/ kernel)
3: Resposta (servidor p/ cliente)
4: ACK (kernel p/ kernel)

Primitivas confiáveis e não confiáveis

- o cliente é bloqueado após enviar uma mensagem;
- o kernel do servidor não envia um reconhecimento, ao invés disso a própria *resposta* faz esse papel;
- se levar muito tempo, o kernel do cliente solicita novamente, salvaguardando a perda de mensagens.

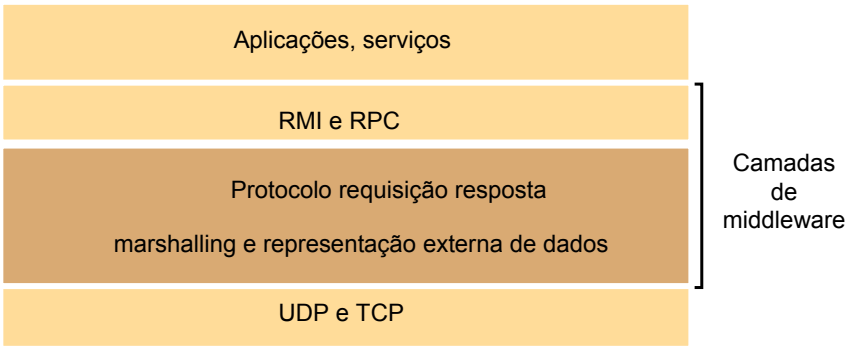


1: Requisição (cliente p/ servidor)
2: Resposta (servidor p/ cliente)
3: ACK (kernel p/ kernel)

Primitivas confiáveis e não confiáveis

- Primitivas de envio/recebimento de mensagens:
  1. Não confiável
  2. Requisição-confirmação-resposta-confirmação
  3. Requisição-resposta-confirmação

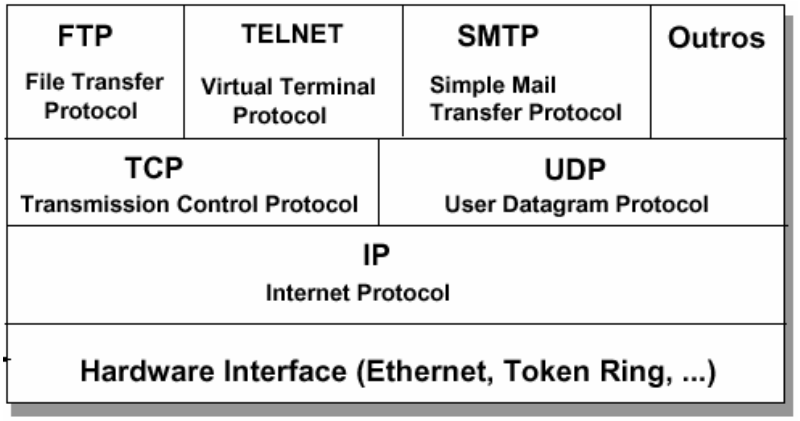
Camadas do middleware



Sockets

- **BSD Sockets**: um conjunto de protocolos e APIs para comunicação entre processos em máquinas distintas e heterogêneas
- Suporta protocolos de comunicação entre processos em vários domínios:
  - Domínio UNIX (usado por processos na mesma máquina)
  - Domínio Internet (TCP/IP ou UDP/IP) (processos em máquinas distintas)
  - Domínio Xerox Ns (não implementado em muitos sistemas)
  - Domínio IPX

Sockets



Características TCP/IP e UDP/IP

- TCP/IP – serviço com conexão (circuito-virtual)
  - Connection-oriented - Circuito virtual
    - Estabelecimento da ligação
    - Transferência de Dados
    - Terminação da ligação
  - Seqüencialidade de dados – garante que a ordem de emissão é a ordem de recepção
  - Controle de erros: retransmite mensagens corrompida ou perdidos
  - Controle de fluxo: controla os ritmos de transferência entre o emissor e o receptor
  - Eliminação de duplicados
  - Fronteiras nos dados: orientado ao byte (stream oriented)
  - Bi-direcional

Características TCP/IP e UDP/IP

- UDP/IP - Serviço Datagrama
  - Connectionless – Datagramas
    - Cada mensagem (datagrama) é transmitida independentemente das outras
  - Fronteira nos dados: orientado à mensagem (message oriented)
  - Permite broadcast e multicast

IP endpoint

- Endereço da aplicação = endereço da máquina na rede (endereço IP) + endereço do serviço na máquina (port)
  - em português: porta ou porto
- Um IP endpoint é um par TCP/port ou UDP/port
  - Os ports TCP e UDP são ports distintos

IP endpoint

- Número de port - Port number (16 bits)
- Gama de ports IP (TCP ou UDP)*
  - Reservados: 1 – 1023 (well known ports)
  - Registrados: 1024 – 49151
  - Dinâmicos ou privados: 49152 – 65535
- Conexão completamente definida:

  { protocolo, endereço local, port local, endereço remoto, port remoto}

* www.iana.org/assignments/port-numbers

Conversão de endereços IP

- DOTTEDASCII para formato de rede

  unsigned long **inet_addr**(char *ptr)

  Exemplo:

  serv_addr.sin_addr.s_addr=inet_addr("10.64.11.101")

- Formato de rede para DOTTEDASCII

  char ***inet_ntoa**(struct in_addr inaddr)

Byte-order - heterogeneidade

- Big endian: IBM 370, Motorola 68000
- Little endian: Intel x86, Dec Vax

- Formato de rede: big endian

Conversão de Byte-Order

- Formato de host para formato de rede

u_long **htonl**(u_long hostlong) - host to network long

u_short **htons**(u_short hostshort) - host to network short

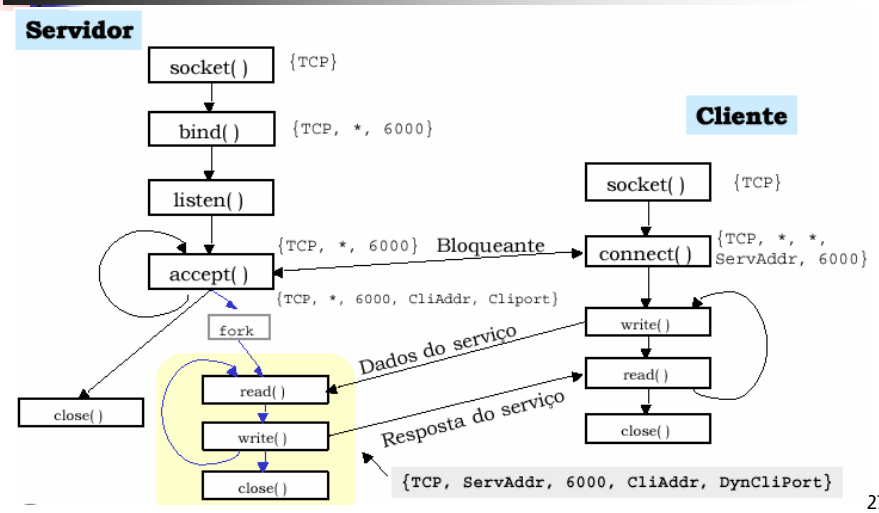
- Formato de rede para formato de host

u_long **ntohl**(u_long netlong) - net to host long

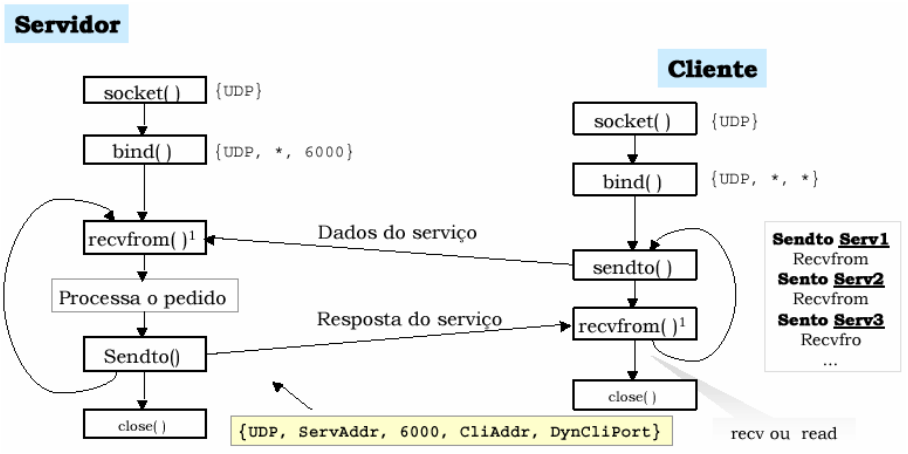
u_short **ntohs**(u_short netshort) - net to host short

- Exemplo: serv_addr.sin_port = htons(500)
  - Port do servidor: 500

Chamadas de sistema para ligação com TCP



Chamadas de sistema para ligação com UDP



[1] Bloqueia até haver uma mensagem

Chamadas de sistema - socket

- Socket – cria um socket com um determinado protocolo associado

  int **socket**(int dominio, int type, int protocolo)
- dominio:
  - AF_UNIX - domínio dos protocolos inertnos UNIX;
  - AF_INET - domínio dos protocolos Internet (TCP ou UDP)
  - AF_IPX - domínio dos protocolos IPX
  - AF_NS - domínio dos protocolos Xerox NS
- type:
  - SOCK_STREAM - stream socket ou circuito virtual (TCP)
  - SOCK_DGRAM - datagram socket (UDP)
- protocolo:
  - 0 – o sistema escolhe o protocolo adequado

Chamadas de sistema - bind

- bind – registra um endereço para que possa receber mensagens

  int **bind**(int sockfd, struct sockaddr *myaddr, int addrlen)
- Especifica os elementos <local address> e <local port> da associação. Obrigatório numa ligação por UDP, facultativo num cliente TCP (em que é utilizado um port dinâmico)

  Sockets address template

```
struct sockaddr {
  u_short sa_family; /* dominio AF_XXX */
  char sa_data[14]; /* especifico do protocolo */
};
```

Chamadas de sistema - bind

Socket address domínio Internet

```
struct sockaddr_in {
  short sin_family; /* AF_INET */
  u_short sin_port; /* 16 bits port number – formato de rede */
  struct in_addr sin_addr; /* formato de rede */
  char sin_zero[8]; /* não usado */
}

struct in_addr { /* endereço host IP*/
  u_long s_addr; /* 32 bits net id */
};

#include <netinet/in.h>
```


Chamadas de sistema - listen

- listen – prepara o circuito virtual para aceitar N conexões

  int **listen**(int sockfd, int backlog)

- Backlog – dimensão da fila de pedidos pendentes

Chamadas de sistema - accept

- accept – abertura passiva por uma conexão

  int **accept**(int sockfd, struct sockaddr *PeerAddr, int *addrlen)

- Quando chegar um pedido de conexão cria e devolve um outro socket com as mesmas propriedades de sockfd. Se não houver pedidos de conexão pendentes, bloqueia o processo PeerAddr e addrlen - retornam o endereço do par que estabeleceu a conexão

Chamadas de sistema - connect

- connect – abertura ativa de uma conexão

  int **connect**(int sockfd, struct sockaddr *RemoteAddr, int addrlen)

- Especifica os dois ou quatro últimos elementos da tupla de associação. No caso dos elementos locais não estarem já definidos assume-os como INADDR_ANY

Acesso a dados

int **send**(int sockfd, char *buff, int nbytes, int flags)
int **recv**(int sockfd, char *buff, int nbytes, int flags)
int **sento**(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *destino, int addrlen)
int **recvfrom**(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *origem, int *addrlen)

- Os três primeiros parâmetros são idênticos às chamadas de sistema read() e write()
- Flags:
  - MSG_OOB - dados Out-Of-Band (send, receive)
  - MSG_PEEK - retorna os dados sem os consumir (recv, recvfrom)
  - MSG_DONTROUTE - bypass routing (send, sendto)
- Retornam o número de bytes lidos ou escritos
- Sendto / recvfrom – recebe / devolve o endereço do destinatário / remetente

Chamadas de sistema - close

int **close**(int sockfd)

- Fecha o socket e se houver dados pendentes ou confirmações (acknowledges) pendentes o sistema garante que são tratados

Biblioteca

- socket, bind, connect, accept, sendto, recvfrom, send, recv

```
#include <sys/types.h>
#include <sys/sockets.h>
```


Exercícios

- "Sistemas operacionais modernos" Andrew S. TANENBAUM Prentice-Hall, 1995
  - Capítulo 10 Exercícios 1-6 (pág. 314)

Bibliografia

- "Sistemas operacionais modernos" Andrew S. TANENBAUM Prentice-Hall, 1995
  - Seção 10.2 pág. 275-285