Comunicação nos Sistemas Distribuídos

1. Protocolos
   - Camadas do Modelo OSI
2. Redes *Assynchronous Transfer Mode*
   - O que é *Assynchronous Transfer Mode* ?
   - Camadas e Comutação em ATM
   - Algumas Aplicações ATM para Sistemas Distribuídos
3. Modelo Cliente/Servidor
   - Exemplos de Clientes e Servidores
   - Primitivas Bloqueantes e Não Bloqueantes
   - Primitivas Bufferizadas e Não Bufferizadas
   - Primitivas Confiáveis e Não Confiáveis
   - Implementação do Modelo Cliente/Servidor

... Comunicação nos Sistemas Distribuídos

4. *Remote Procedure Call*
   - Operações Básica do RPC
   - Passagem de Parâmetros
   - Ligação Dinâmica
   - Semântica do RPC na presença de Falhas
   - Aspectos de Implementação e Problemas
5. Comunicação em Grupo
   - Introdução de Comunicação em Grupo
   - Aspectos de Projetos de Comunicação em Grupo
   - Design Issues to Group Communication no ISIS

... Comunicação nos Sistemas Distribuídos

⋆ **Comunicação nos Sistemas Distribuídos:** é o aspecto mais importante que os diferencia dos Sistemas Monoprocessados.

• ... um exemplo típico, é o problema produtor-consumidor utilizando-se de algum mecanismos de sincronização, que na sua essência compartilha a memória.

⋆ Já nos Sistemas Distribuídos, não há memória compartilhada e, assim, a natureza da comunicação entre processos precisa ser repensada.

⋆ Iniciaremos nossa discussão apresentando conjuntos de regras que governam a comunicação entre processos, conhecidas como protocolos.

⋆ Na seqüência, descrevemos o Modelo Cliente/Servidor e algumas opções particulares de comunicação entre os processos.

1 - Protocolos do Modelo OSI

⋆ Dada a ausência de memória compartilhada, toda a comunicação nos sistemas distribuídos é baseada na troca de mensagens;

• ... no entanto, o problema da comunicação envolve muitos detalhes, tanto aqueles referente à transmissão quanto aqueles referente a semântica da informação.

⋆ ISO (*Internation Standards Organization*) propôs o Modelo de Referência *Open Systems Interconnection* para possibilitar a comunicação de sistemas abertos.

• **sistema aberto:** é aquele preparado para se comunicar com outro sistema aberto, utilizando-se para isso de regras padrões que governem o formato, conteúdo, e significado das mensagens enviadas e recebidas;

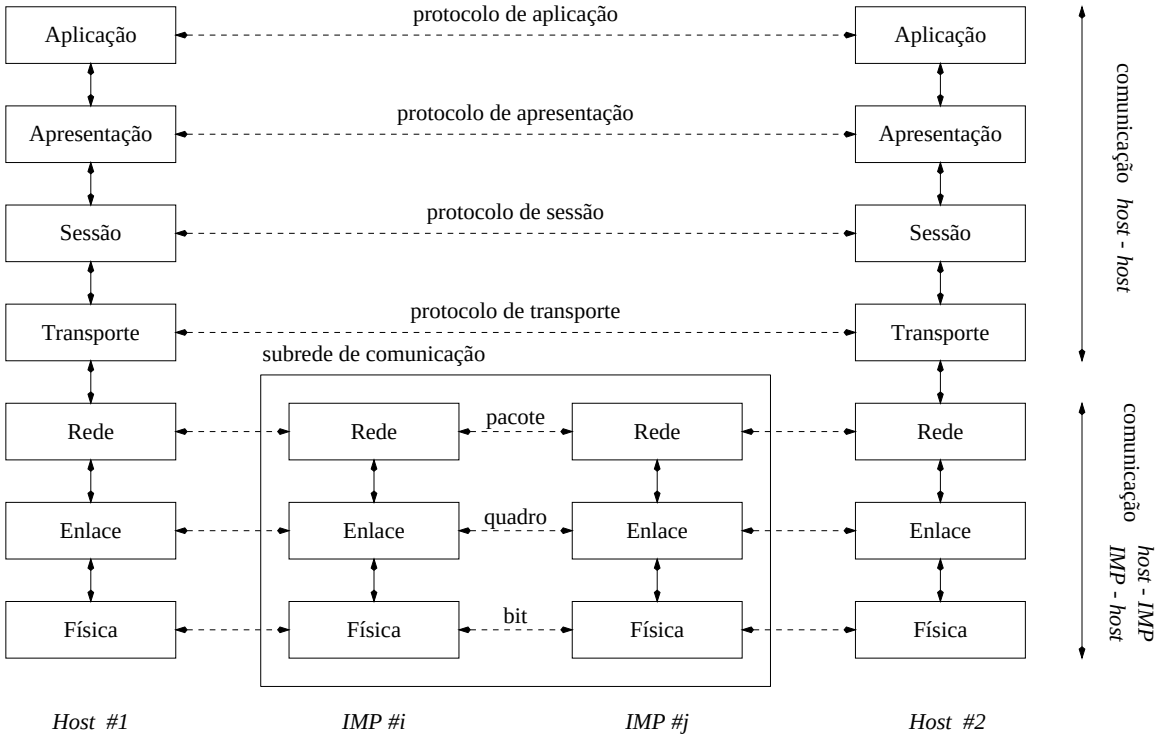
• tais regras são formalizadas no que chamamos de **protocolos**, ou seja, concordância das partes comunicantes em como a comunicação deve ocorrer.

... 1 - Protocolos do Modelo OSI

✯ Modelo OSI distingue dois tipos gerais de protocolos: **orientado a conexão** (*connection oriented*) e **não orientado a conexão** (*connectionless*);

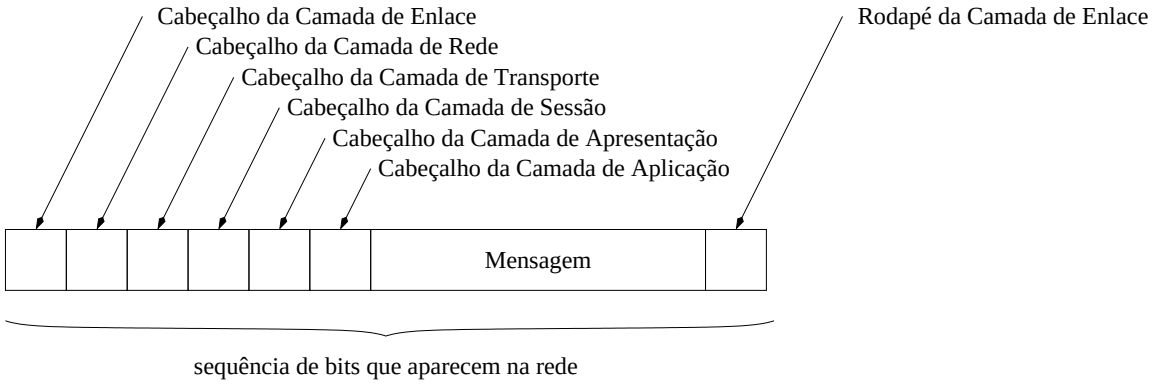
- ... a comunicação é dividida em sete camadas, cada qual com suas especifidades de comunicação, ou seja, seus protocolos;
- ... isto abre o precedente para que o problema de comunicação seja subdividido em problemas menores resolvíveis, cada qual com suas particularidades;
- ... cada camada provê uma **interface** para a camada acima e que consiste de um conjunto de operações que em conjunto definem o serviços da camada.

... 1 - Protocolos do Modelo OSI



... 1 - Protocolos do Modelo OSI

- ... algumas camadas não só adicionam informações no cabeçalho ($header$), mas também no final da mensagem ($trailer$);

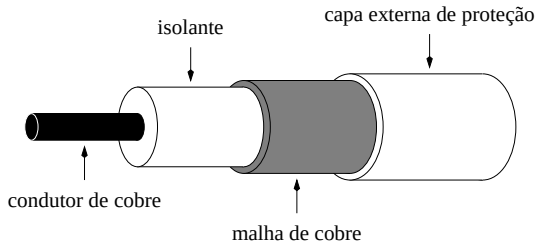


1.1 - Camada Física

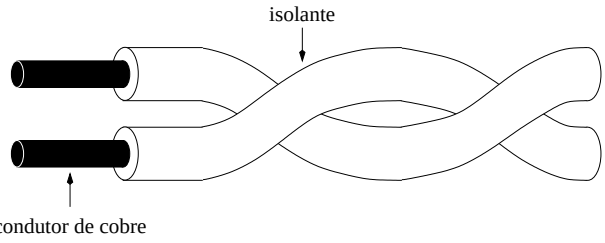
✲ Intimamente relacionada com o meio físico empregado, é responsável pela geração de sinais elétricos, óticos ou eletromagnéticos para serem propagados;

✲ ... algumas características do protocolo:

- especificar qual a duração e intensidade do sinal;
- técnica de multiplexação;
- pinagem, ... ;



Cabo Coaxial



Par Trançado

1.2 - Camada de Enlace

- ... além de agrupar os bits em unidades ($frames$), esta camada provê detecção e correção de erros gerados na troca de dados (p.ex., código CRC);

- **data frames:** agrupamento de bits delimitado por seqüencias de bits pré-estabelecidas (p.ex., 01111110);

- utilizando-se do serviço de transmissão da camada física, transmite (recebe) $data\ frames$ aguardando (enviando) o respectivo quadro de reconhecimento;

✲ ... algumas características do protocolo:

  - transmissão não confiável (mesmo com reconhecimento de recepção);
  - $data\ frames$ podem ser duplicados ao chegar fora de ordem;
  - duplicações geralmente ocorrem quando o $data\ frame$ de reconhecimento é deformado e/ou perdido na transmissão;
  - controla o fluxo de $data\ frames$, evitando que um $host$ envie quadros em uma taxa superior a que o receptor é capaz de processar.

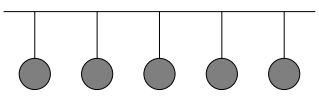
1.3 - Camada de Rede

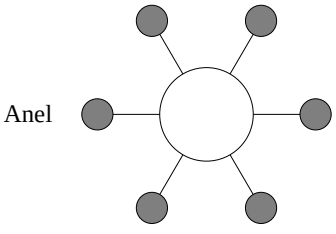
✲ ... sua principal função é controlar a operação da subrede de comunicação;

✲ ... algumas características do protocolo:

  - roteamento de pacotes do $host$ origem ao $host$ destino;
  - o roteamento pode apresentar características dinâmicas ou estáticas;
  - fragmentação e remontagem de pacotes para atender limites impostos.

✲ Nota: em subredes de difusão esta camada é extremamente simples, uma vez que a principal função (roteamento) é inexistente.


Barramento



1.4 - Camada de Transporte

✲ ... algumas das funções da camada de transporte:

- receber dados da camada de sessão;
- particionar estes dados em unidades menores;
- garantir envio de dados sem duplicação e na ordem correta;

✲ ... oferece dois tipos de serviços:

- **serviço sem conexão:** serviço rápido com mensagens de tamanho limitado e sem garantia de entrega, ordem ou ausência de duplicação;
- **serviço orientado a conexão:** serviço mais lento, porém altamente confiável e sem limites de tamanho de mensagens;

✲ É a primeira camada a promover comunicação $host$-$host$, assim controla o fluxo de dados entre dois processos comunicantes.

1.5 - Camada de Sessão

✲ É essencialmente uma versão melhorada da Camada de Transporte, com a função de prover o controle do diálogo bem como rastrear as partes durante a comunicação no que se refere a sincronização da mesma.

✲ ... possibilitar a dois processos de aplicação estabeleçam sessões entre si a fim de organizar e sincronizar a troca de informações.

✲ conexão de sessão $=>$ definição das regras de diálogo entre 2 APs

- $Two\ Way\ Simultaneous$ - TWS;
- $Two\ Way\ Alternate$ - TWA;
- $One\ Way$ - OW.

1.6 - Camada de Apresentação

- ... dentre os serviços oferecidos, destacamos:
  - **representação canônica de dados**
  - **compressão de dados**
  - **criptografia**
- Representação Canônica de Dados .
- Compressão é útil para o envio de grandes massas de dados.
- Criptografia é utilizada quando os dados são confidencias.

1.7 - Camada de Aplicação

- De fato, a Camada de Aplicação constitui-se de uma coleção de protocolos que se prestam às atividades comuns tais como:
  - correio eletrônico - X.400;
  - transferência de arquivo;
  - serviços de diretório - X.500;
  - *login* remoto; submissão de *jobs* remotos, ...
- Também se constitui em ponto de acesso à rede por Processos de Aplicação.
- *Application Program Interface - APIs* (em vias de padronização)
  - são bibliotecas de funções para envio/recepção de mensagens, ...

2 - Redes ATM - Assynchronous Transfer Mode

✯ Modelo OSI, apresentado na seção anterior, foi proposto nos anos 70 e parcialmente implementado nos anos 80.

- ... novos desenvolvimentos na década de 90 propiciaram melhorias no modelo, principalmente nas camadas dependentes de tecnologias.

✯ Outro aspecto é que, na metade do século passado, computadores cresceram em performance muito mais que as redes, ou seja:

- ... ARPANET, inaugurada em 1969, utilizou linhas de 56 Kbps para comunicação;

- ... final da década de 70 e início da década de 80, muitas dessas linhas foram substituídas por linhas de 1,5 Mbps ($T1\ lines$);

- ... eventualmente, $backbones$ evoluíram para redes a 45 Mbps ($T3\ networks$), embora a maioria das linhas tenha se mantido em 1,5 Mbps.

2 - Redes ATM - Assynchronous Transfer Mode

✯ Novos avanços possibilitaram taxas de transmissão mais baixas de 155 Mbps e troncos principais atingindo 1 Gbps ou até mais;

- ... nesse contexto, sistemas distribuídos tiveram um grande impacto tanto no que se refere às novas dimensões das aplicações como também nos novos desafios.

✯ Quando as Companhias Telefônicas decidiram projetar as redes para o Século 21, elas se depararam com o dilema de quais serviços atender:

- ... serviço de voz exigindo largura de banda baixa e constante;

- ... serviço de dados exigindo alta largura de banda em breves períodos de tempo, e nenhuma largura de banda na ausência de tráfego.

- **conclusão:** ... nem os tradicionais circuitos chaveados (Redes PSTN) nem a comutação de pacotes (Internet) acomodava ambos os tráfegos.

2.1 - O que é Assynchronous Transfer Mode ?

✲ Surge, então, a Rede Híbrida (*Assynchronous Transfer Mode*) que, utilizando blocos de tamanho fixo em cima de circuitos virtuais, tinha por proposta atender com boa performance ambos tipos de tráfego.

✲ Modelo ATM pressupõe algumas etapas, quais sejam:

- ... estabelecimento de circuito virtual entre transmissor/receptor, ou seja, um rota é estabelecida e informações de roteamento são armazenadas nas chaves;
- ... usando esta conexão, pacotes são subdivididos em unidades denominadas *cells* e enviadas pelos circuitos virtuais segundo o caminho armazenado nas chaves;
- ... quando não mais fosse necessário a conexão, a mesma era liberada removendo-se das chaves as informações de roteamento.

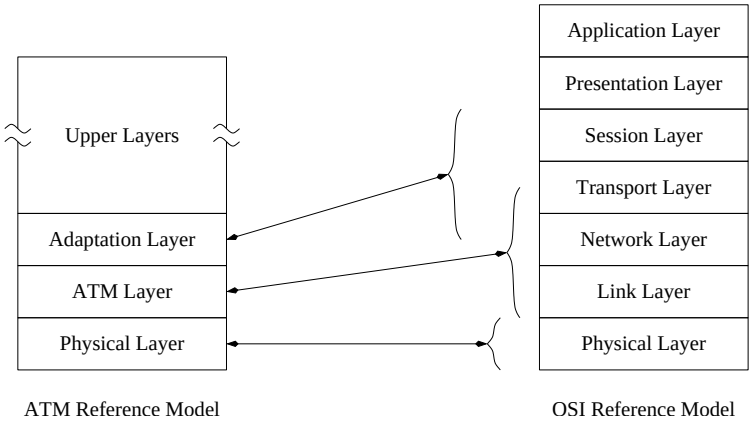
... 2.1 - O que é Assynchronous Transfer Mode ?

✲ Vejamos algumas **vantagens** dessa proposta sobre as anteriores:

- ... uma única rede pode atender a uma mistura arbitrária de informações, quais sejam: voz, dados, *tv broadcast*, rádio, *videotapes*, etc.
- ... esta integração representa uma enorme economia de custos bem como simplificação de projeto tornando possível aos usuários residenciais e comerciais uma única infraestrutura para comunicação e informação;
- ... **células** (ATM *cells*) acomodam tanto conexões ponto-ponto quanto conexões *multicasting* sem perda de performance;
- ... células de tamanho fixo possibilitam rápido chaveamento, algo difícil de ser conseguido na comutação de pacotes *store-and-forward*.

... 2.1 - O que é Assynchronous Transfer Mode ?

- *physical layer:* a mesma funcionalidade que a camada física do Modelo OSI;
- *ATM layer:* trata das células e transporte incluindo o roteamento, assim ela cobre as camadas 2 e partes da 3 do Modelo OSI;
- *adaptation layer:* segmentação dos pacotes em células e sua remontagem no lado aposto, algo não aparece explicitamente no Modelo OSI antes da camada 4.



2.2 - Aplicações ATM em Sistemas Distribuídos

✭ As maiores implicações no projeto de Sistemas Distribuídos advém da disponibilidade de Redes ATM de 155 Mbps, 622 Mbps e potencialmente de 2,5 Mbps:

✍ Os efeitos são decorrentes principalmente da largura de banda disponível e não das propriedades e especificidades das Redes ATM;

- ... considere o envio de um arquivo de 1 Mbits por 3000 Km e a espera do reconhecimento que chega sem erros em diferentes redes:
- ... a 64 Kbps todo o processo toma 15,6 s para dispor os bits na rede e 30 ms para que se propaguem, considerando o reconhecimento sem erros;
- ... a 622 Mbps todo o processo toma 1,6 ms para dispor os bits na rede e 30 ms para que se propaguem, considerando o reconhecimento sem erros;
- ... qual é percentual de uso da infraestrutura de comunicação ?

... 2.2 - Aplicações ATM em Sistemas Distribuídos

✍ Por outro lado, altas taxas de transferência exigem uma outra abordagem para controle de fluxo e controle de congestionamento nas Redes ATM.

- ... considere o envio de um arquivo de *videotape* de 10 Gbits a uma taxa de trasmissão de 622 Mbps, mas o receptor percebendo que não pode acormodar tamanha informação, envia um sinal solicitando a interrupção na transmissão;

- ... quanto de informação foi transmitida quando do recebimento por parte do trasnmissor da solicitação de interrupção do receptor ?

- ... qual é a essência do problema nessas redes ?

✍ **Alguns Artigos:** Eckberg, 1992; Hong and Suda, 1991; Frajkovic and Golestani, 1992; Nikolaidis and Onvural, 1992.

... 2.2 - Aplicações ATM em Sistemas Distribuídos

✯ **conclusão:** o acréscimo considerável de ganho de performance nas taxas de transmissão, não trouxe ganho de performance das aplicações;

- ... o atraso de propagação em grandes distâncias e acima dos atrasos perceptíveis pelos usuários (p.ex., 200 ms) conduz a insatisfação do mesmo.

> *As a consequence of these and other problems, while high-speed networks in general and ATM in particular introduce new opportunities, taking advantage of them will not be simple. Considerable research will be needed before we know how to deal with them effectively.*
>
> *Andrew S. Tanenbaum*

3 - Modelo Cliente/Servidor

✲ Como visto anteriormente, as redes estratificadas em camadas constituem uma alternativa razoável para sistemas distribuídos;

- ... entretanto, esta alternativa não é válida para todo sistema distribuído, uma vez que o $overhead$ gerado pelos cabeçalhos dos protocolos pode ser grande;

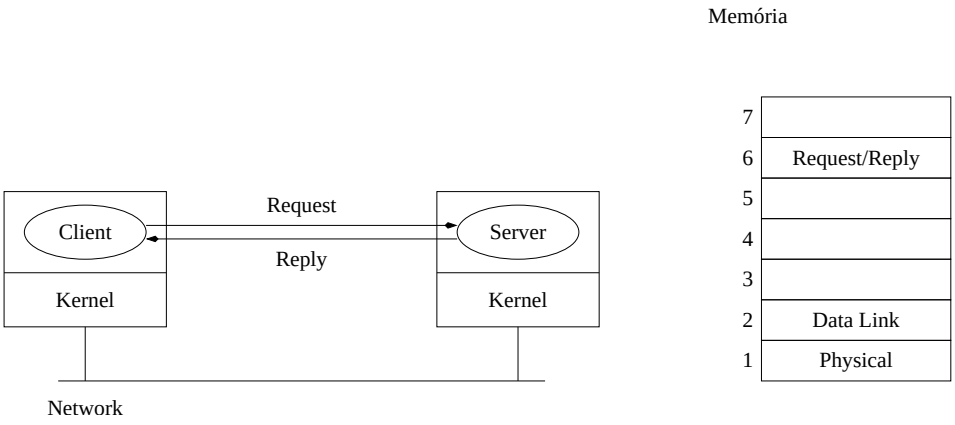
- **exemplo:** em sistemas distribuídos baseados em redes locais, o $overhead$ do cabeçalho é considerável, ou seja, muito tempo do processador é gasto executando protocolos que efetivamente utilizam-se de um pequeno percentual da rede.

- **consequência:** muitos sistemas distribuídos baseados em redes locais não utilizam toda a pilha de comunicação, apenas parte dela.

✲ Ainda assim, o modelo de comunicação trata apenas de alguns dos aspectos do projeto de Sistemas Distribuídos, o que sugere algo mais para ser discutido.

3.1 - Introdução ao Modelo Cliente/Servidor

✲ **idéia:** estruturar o sistema operacional como um grupo de processos cooperantes, chamados servidores, que oferecem serviços aos usuários, chamados clientes.



- ... normalmente, as máquinas executam o mesmo $kernel$, com ambos os clientes e servidores executando como processos do usuário.

... 3.1 - Introdução ao Modelo Cliente/Servidor

✲ Para evitar $overhead$ de protocolos orientados a conexão, o modelo é usualmente baseado em protocolos sem conexão do tipo **request/reply**:

❶ **simplicidade:** nenhuma conexão é estabelecida e, adicionalmente, a mensagem de **reply** serve como reconhecimento da mensagem de **request**;

❷ **eficiência:** como não há roteamento e estabelecimento de conexão, a pilha do protocolo é menor e, assim, mais eficiente.

✲ Dada a simplicidade da estrutura, os serviços de comunicação oferecidos pelo $(micro)kernel$ são reduzidos a 02 chamadas, uma para enviar mensagens **send(dest,&mptr)** e outra pra receber mensagens **read(addr,&mptr)**;

- ... muitas variações destes procedimentos e de seus parâmetros são possíveis, mas inicialmente vamos apresentar um exemplo deste modelo.

3.2 - Exemplo do Modelo Cliente/Servidor

- ... definição de constantes no arquivo de $header$:

```
/* Definitions needed by clients and servers. */
#define MAX_PATH         255    /* maximum length of a file name */
#define BUF_SIZE         1024   /* how much data to transfer at once */
#define FILE_SERVER      243    /* file server's network address */

/* Definitions of the allowed operations */
#define CREATE           1      /* create a new file */
#define READ             2      /* read a piece of a file and return it */
#define WRITE            3      /* write a piece of a file */
#define DELETE           4      /* delete an existing file */

/* Error codes */
#define OK               0      /* operation performed correctly */
#define E_BAD_OPCODE    -1      /* unknown operation requested */
#define E_BAD_PARAM     -2      /* error in a parameter */
#define E_IO            -3      /* disk error or other I/O error */
```

.

... 3.2 - Exemplo do Modelo Cliente/Servidor

```
  .
  .
  .
  .

/* Definition of the message format */

struct message {
  long source;               /* sender's identity */
  long dest;                 /* receiver's identity */
  long opcode;               /* which operation: CREATE, READ, etc. */
  long count;                /* how many bytes to transfer */
  long offset;               /* where in file to start reading or writing */
  long extra1;               /* extra field */
  long extra2;               /* extra field */
  long result;               /* result of the operation reported here */
  char name[MAX_PATH];       /* name of the file being operated on */
  char data[BUF_SIZE];       /* data to be read or written */
};
```


... 3.2 - Exemplo do Modelo Cliente/Servidor

✲ Código de um Servidor:

```
#include <header.h>

void main( int argc, char[] *argv )  {
  struct message m1, m2;             /* incoming and outgoing messages */
  int r;                             /* result code */

  while( 1 )  {                      /* server runs forever */
    receive( FILE_SERVER, &m1 );     /* block waiting for a message */
    switch( m1.opcode )  {           /* dispatch on type of request */
      case CREATE:  r = do_create(&m1, &m2);  break;
      case READ:  r = do_read(&m1, &m2);  break;
      case WRITE:  r = do_write(&m1, &m2);  break;
      case DELETE:  r = do_delete(&m1, &m2);  break;
      default:  r = E_BAD_OPCODE;
    }
    m2.result = r;                   /* return result to client */
    send( m1.source, &m2 );          /* send reply */
  }
}
```

... 3.2 - Exemplo do Modelo Cliente/Servidor

✲ Procedimento Cliente:

```
#include <header.h>

int copy( char *src, char *dst )    /* procedure to copy file using the server */
{
  struct message m1;                /* message buffer */
  long position;                    /* current file position */
  long client = 110;                /* client's address */

  initialize();                     /* prepare for execution */
  position = 0;
  do  {
    /* Get a block of data from the source file */
    m1.opcode = READ;               /* operation is a read */
    m1.offset = position;           /* current position in the file */
    m1.count = BUF_SIZE;            /* how many bytes to read */
    strcpy( &m1.name, src );        /* copy name of file to be read to message */
    send(FILE_SERVER, &m1 );        /* send the message to the file server */
    receive( client, &m1 );         /* block waiting for the reply */
    .
    .
```


... 3.2 - Exemplo do Modelo Cliente/Servidor

✲ continuação do Procedimento Cliente:

```
    .
    .
    .
    .
    .

    /* Write the data just received to the destination file */
    m1.opcode = WRITE;              /* operation is a write */
    m1.offset = position;           /* current position in the file */
    m1.count = m1.result;           /* how many bytes to write */
    strcpy( &m1.name, dst );        /* copy name of file to be written to buf */
    send(FILE_SERVER, &m1);         /* send the message to the file server */
    receive(client, &m1 );          /* block waiting for the reply */
    position += m1.result;          /* m1.result is number of bytes written */
  }  while( m1.result > 0 );        /* iterate until done */
  return( m1.result >= 0 ? OK : m1.result );   /* return OK or error code */
}
```

3.3 - Endereçamento

- ... no nosso exemplo, o servidor foi endereçado por um número inteiro, mas sem especificar o que na realidade representa, ou seja, a máquina, o processo, ... ?

- ... se representar a máquina destino, temos um problema: para qual processo na máquina a chamada deverá ser direcionada?

✍ **conclusão:** este esquema permite que somente um processo esteja executando na máquina, o que pode algumas vezes constitui uma séria restrição.

✍ **alternativa:** embora não seja a melhor solução, identificar os processos e não as máquinas resolve o problema anteriormente descrito.

✲ Ainda assim, esta abordagem exige que o usuário especifique a localização do servidor, algo nada transparente em Sistemas Distribuídos!

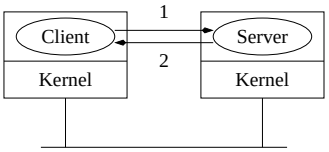
... 3.3 - Endereçamento

✲ Em resumo, o endereçamento pode se dar de três formas:

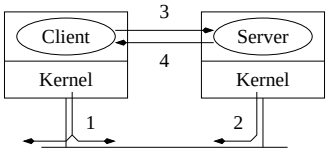
❶ *hardwired* no código do cliente;

❷ localização do servidor por mensagem *broadcast*;

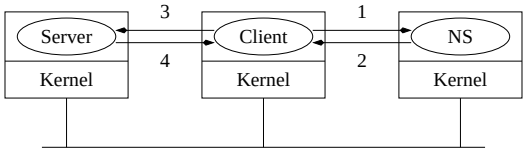
❸ um servidor de nomes passa a fornecer endereços de servidores que tenham sido previamente cadastrados.



1: Request to 243.0
2: Reply to 199.0



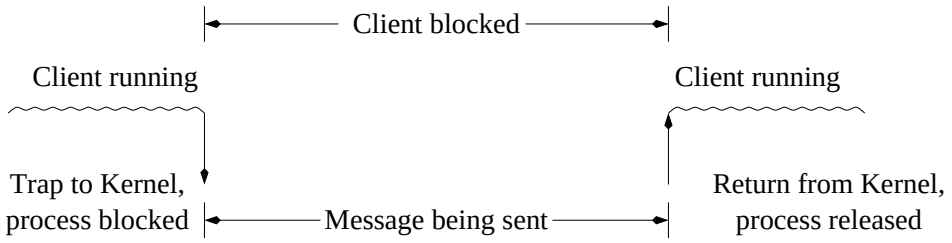
1: Broadcast
2: Here I am
3: Request
4: Reply



3: Request
4: Reply
1: Lookup
2: NS Reply

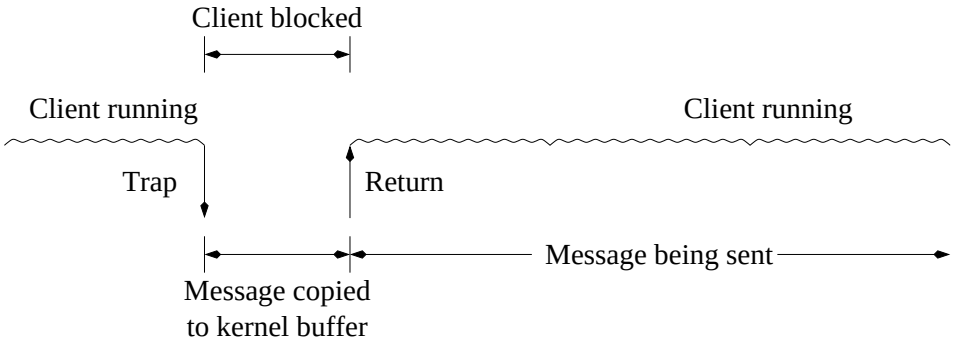
3.4 - Primitivas Bloqueantes e Não Bloqueantes

✲ **primitivas bloqueantes** ou **primitivas síncronas:** bloqueiam o processo que requisitou o envio da mensagem, mantendo-o neste estado até que a mensagem seja completamente enviada;



... 3.4 - Primitivas Bloqueantes e Não Bloqueantes

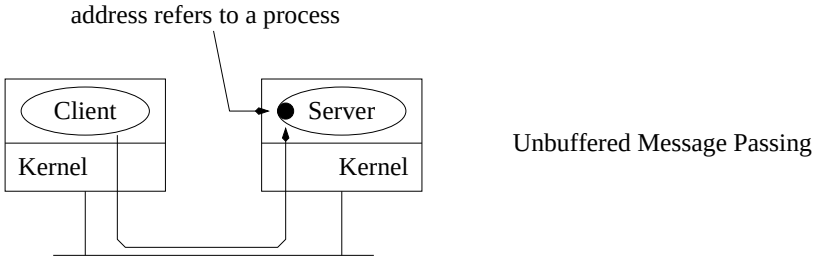
✲ **primitivas não bloqueantes** ou **primitivas assíncronas:** retornam o controle imediatamente ao processo que requisitou o envio da mensagem, antes da mensagem ter sido enviada pelo *kernel* do sistema.



- **desvantagem:** o transmissor não pode modificar a mensagem no buffer até ter sido completamente enviada, o que trás muitas complicações.

3.5 - Primitivas Buferizadas e Não Buferizadas

✯ **primitivas não bufferizadas:** nas descrições anteriores, por exemplo, a chamada *receive* informa ao *kernel* a máquina da qual se espera um mensagem bem como o endereço para o qual a mensagem será copiada.



- ... este esquema funciona bem enquanto o servidor invocar *receive(...)* antes do cliente invocar *send(...)*, pois este mecanismo informa ao *kernel* do servidor qual o endereço que está usando, bem como o local para armazenar as mensagens;

- ... se o *send(...)* é executado primeiro do que o *receive(...)*, como o *kernel* do servidor sabe a quem entregar a mensagem?

... 3.5 - Primitivas Buferizadas e Não Buferizadas

✯ Considere agora, vários clientes acessando um único servidor:

- ... **implicações:** enquanto estiver processando uma mensagem recebida, o servidor não estará pronto para receber/processar outras mensagens;

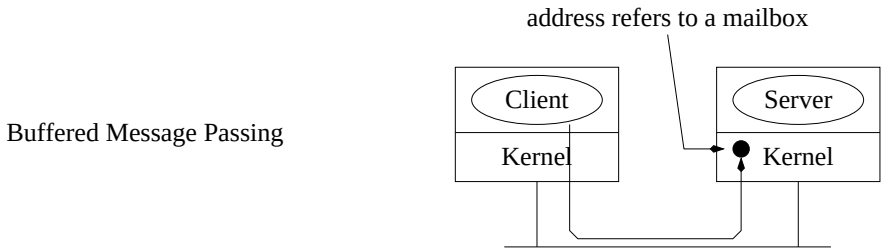
- ... mas não há como impedir que outros clientes enviem mensagens para este servidor, assim, como devemos tratar este problema?

  ❶ contemplar requisições por parte do cliente até que sejam respondidas;
  ❷ permitir que o *kernel* recebe as mensagens, redirecionando-as logo em seguida.

- ... embora este método reduza a chance de descarte da mensagem, ele introduz o problema do armazenamento e gerenciamento prematuro de mensagens.

... 3.5 - Primitivas Buferizadas e Não Buferizadas

✲ **solução:** definir uma nova estrutura de dados (**mailbox**) para tratar o gerenciamento do *buffer*, assim o *kernel* poderá receber as mensagens.



- ... deste modo, o *kernel* sabe o que fazer com as mensagens que estão chegando bem como onde armazená-las, ou seja, **primitivas bufferizadas**.

3.6 - Primitivas Confiáveis e Não Confiáveis

- Até agora, assumimos que na semântica do modelo de troca de mensagens, quando um cliente envia uma mensagem o servidor a recebe;

- ... mas se a mensagem se perder, a semântica do modelo é afetada!

✲ Três abordagens diferentes são possíveis para este problema:

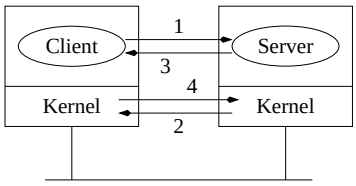
❶ redefinir a semântica do *send(...)* para não confiável;

❷ exigir que o *kernel* da máquina receptora envie uma mensagem de reconhecimento ao *kernel* da máquina transmissora.

❸ tirar proveito da comunicação estrutura como *request/reply*, adicionando uma mensagem de reconhecimento ao conjunto.

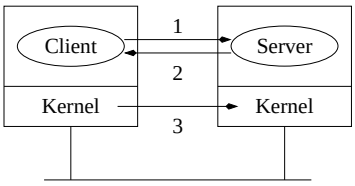
... 3.6 - Primitivas Confiáveis e Não Confiáveis

✲ Alternativas ❷ e ❸:



1. Request (client to server) 2. ACK (kernel to kernel)
3. Reply (server to client) 4. ACK (kernel to kernel)

Individually Acknowledged Messages



1. Request (client to server) 2. Reply (server to client)
3. ACK (kernel to kernel)

Reply being used as the Acknowledgment of the Request

- ... na alternativa ❸, embora o *reply* funcione como reconhecimento do *request*, a omissão do reconhecimento do *reply* pode ser ou não preocupante e/ou séria dependendo da natureza do *request*.

3.7 - Implementação do Modelo Cliente/Servidor

✲ Nas seções precedentes, discutimos aspectos de projetos, endereçamento, bloqueio, bufferização e confiabilidade que a seguir foram resumidas:

Item	Option 1	Option 2	Option 3
Addressing	Machine number	Sparse process address	ASCII names looked up via server
Blocking	Blocking primitives	Nonblocking with copy to kernel	Nonblocking with interrupt
Buffering	Unbuffered, discarding unexpected messages	Unbuffered, temporarily keeping unexpected messages	Mailboxes
Reliability	Unreliable	Request-Ack-Reply Ack	Request-Reply-Ack

- ... enquanto os detalhes de implementação de passagem por mensagem dependem de algum modo das escolhas feitas, ainda assim é possível descrevermos comentários acerca da implementação, dos protocolos e do *software*.

... 3.7 - Implementação do Modelo Cliente/Servidor

❶ Mensagens de tamanho superios que o tamanho máximo de pacote suportado, devem ser quebradas em múltiplos pacotes e enviadas separadamente;

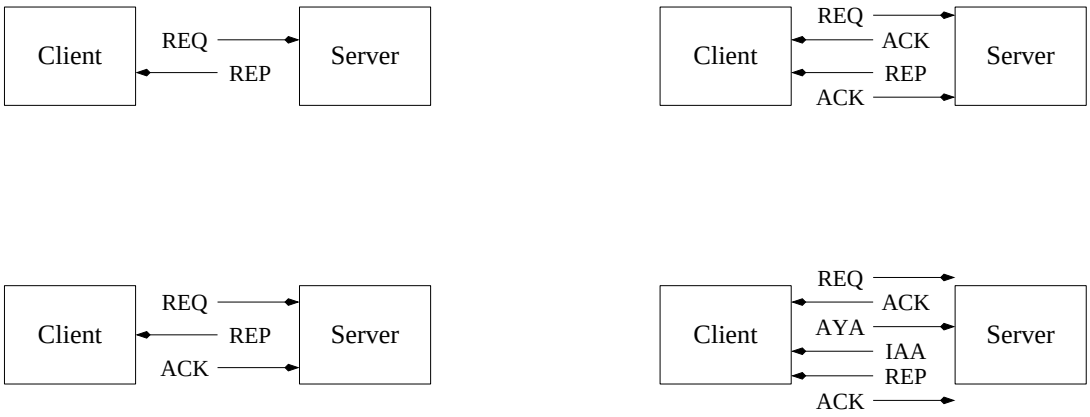
- ... por outro lado, a segmentação pode exigir não só o sequenciamento, como também o reconhecimento de cada pacote componente da mensagem.

❷ Quanto aos protocolos, a tabela abaixo relaciona os 06 tipos de pacotes comumente utilizados na implementação do protocolo Cliente/Servidor:

Code	Packet Type	From	To	Description
REQ	Request	Client	Server	The client wants service
REP	Reply	Server	Client	Reply from the server to the client
ACK	Ack	Either	Other	The previous packet arrived
AYA	Are you alive?	Client	Server	Probe to see if the server has crashed
IAA	I am alive!	Server	Client	The server has not crashed
TA	Try again	Server	Client	The server has no room
AU	Address unknown	Server	Client	No process is using this address

... 3.7 - Implementação do Modelo Cliente/Servidor

❸ Quanto ao $Software$, as seqüências comuns para troca de pacotes na comunicação Cliente/Servidor estão resumidas abaixo:



Some examples of packet exchanges for client−server communication

4 - Chamada de Procedimento Remoto

✲ Vimos na Seção 3 que o Modelo Cliente/Servidor oferece de modo satisfatório um modo que possibilita a estruturação dos Sistemas Distribuídos;

- ... entretanto, o paradigma do modelo repousa no fato de que toda a comunicação seja construída na forma de entrada/saída.

✲ Entrada/Saída não é nem mesmo o conceito chave nos Sistemas Centralizados, mas torná-lo base na Computação Distribuída constitui-se num eqüívoco para muitos pesquisadores/desenvolvedores;

- ... de fato, a intenção é fazer com que a computação distribuída se pareça como se fosse centralizada, ou seja, *Single-Processor Timesharing System*.

✲ Birrel & Nelson (1984) propuseram uma abordagem completamente diferente para este problema: possibilitar que processos invocassem procedimentos localizados em processos remotos (*Remote Procedure Call*).

... 4 - Chamada de Procedimento Remoto

✲ BIRRELL, A.D., and NELSON, B.J.: "Implementing Remote Procedure Calls", ACM Trans. on Computer Systems, Vol. 2, pp.39-59, Feb. 1984.

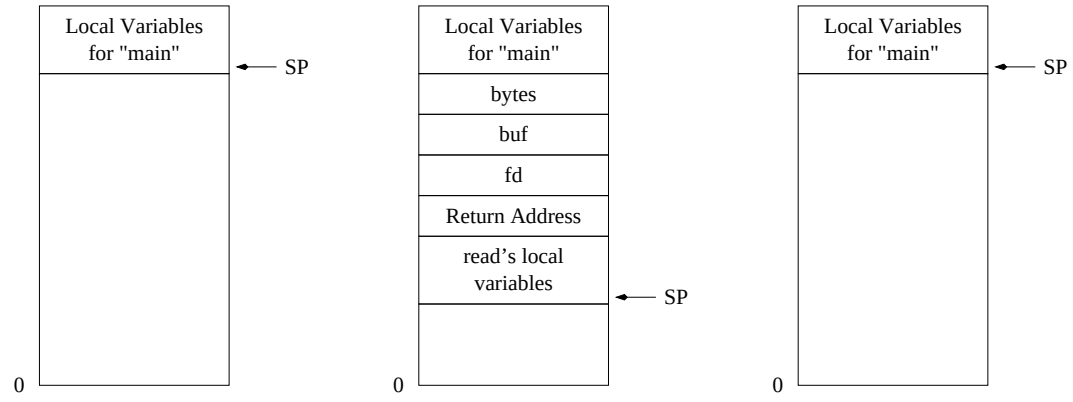
✲ Embora o princípio seja simples e elegante, alguns problemas surgiram:

❶ chamada de procedimento em espaços de endereçamento diferentes;
❷ passagem de parâmetros e bem como o retorno em máquinas de arquiteturas diferentes poderá exigir padronização no envio dos dados;
❸ ambas as máquinas podem falhar causando diferentes problemas.

✲ Ainda assim, muitos desses problemas são tratáveis, possibilitanto que esta Técnica seja largamente usada para suportar muitos Sistemas Distribuídos.

4.1 - Operação Básica no RPC

✲ Antes de prosseguirmos, vejamos como um procedimento convencional funciona, para isto, considere o chamada: **count = red(fd, buf, nbytes);**



The Stack before the call to read. | The Stack while the called procedure is active. | The Stack after the return to the caller.

✍ ... a razão pela qual o compilador C empilha os parâmetros na ordem reversa tem a ver com a localização do primeiro parâmetro da função **fprint(...)**.

... 4.1 - Operação Básica no RPC

✲ **Operação Básica:** fazer com que a chamada de procedimento remoto se pareça como uma chamada local, ou seja, transparência.

- ... o processo requisitando a chamada não se dá conta que o procedimento chamado está sendo executado em uma máquina remota.

- ... **exemplo de chamada local:** a chamada de sistema **read(...)** constitui-se numa interface entre o usuário e o sistema operacional, que quando executada passa parâmetros p/ o $kernel$ através da pilha como visto anteriormente.

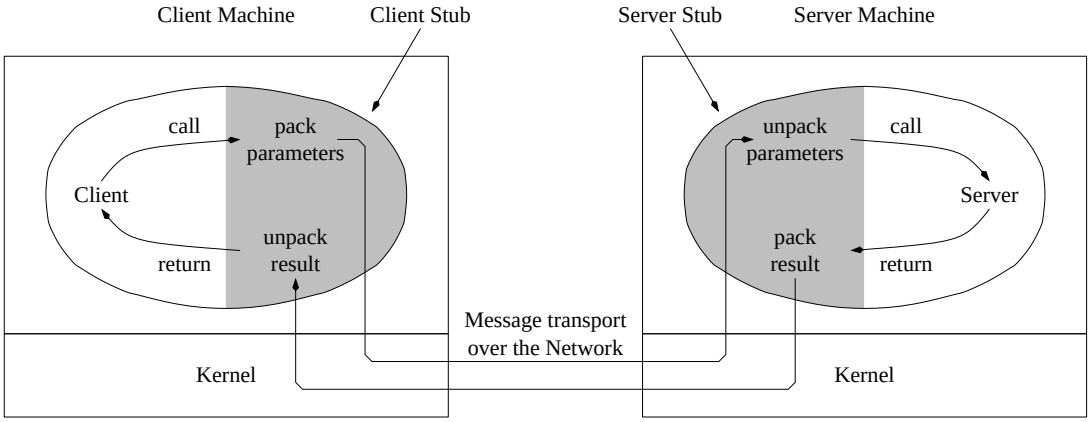
✲ RPC proporciona transparência de modo análogo:

- como procedimento remoto, a chamda **read(...)**, agora denominada **client stub**, também interrompe o $kernel$, sem no entanto passar os parâmetros por pilha;

- ... ao invés, os parâmetros são passados por mensagem ao $kernel$, que por sua vez, se encarrega de encaminhá-la ao $kernel$ remoto.

... 4.1 - Operação Básica no RPC

- ... quanto a mensagem chega ao servidor, o *kernel* a entrega ao **server stub**, que a desempacota e chama o procedimento no servidor no modo usual.



- ... quando o **server stub** readquire o controle, ele empacota o resultado numa mensagem e chama **send(...)** para enviá-la ao cliente.

... 4.1 - Operação Básica no RPC

- ... quanto a mensagem retorna à máquina cliente, o *kernel* identifica a qual processo entregar e copia a mensagem para o seu buffer, desbloqueando-o;

- ... o **client stub** inspeciona a mensagem, desempacota o resultado e faz uma cópia p/ o chamador e retorna o modo usual.

- ... quando o chamador readquire o controle após a chamada **read(...)**, a única coisa que sabe é que os dados estão disponíveis.

✍ Processo Cliente não tem idéia de que todo o trabalho foi realizado remotamente, e não localmente pelo *kernel*.

> *This blissful ignorance on the part of the client is the beauty of the whole scheme. As far as it is concerned, remote services are accessed by making ordinary procedure calls.*
>
> *Andrew S. Tanenbaum*

... 4.1 - Operação Básica no RPC

✲ Todos os detalhes da passagem de mensagens são escondidos em dois procedimentos de biblioteca, assim como os detalhes das interrupções no $kernel$ pelas chamadas de sistema nas bilbiotecas tradicionais.

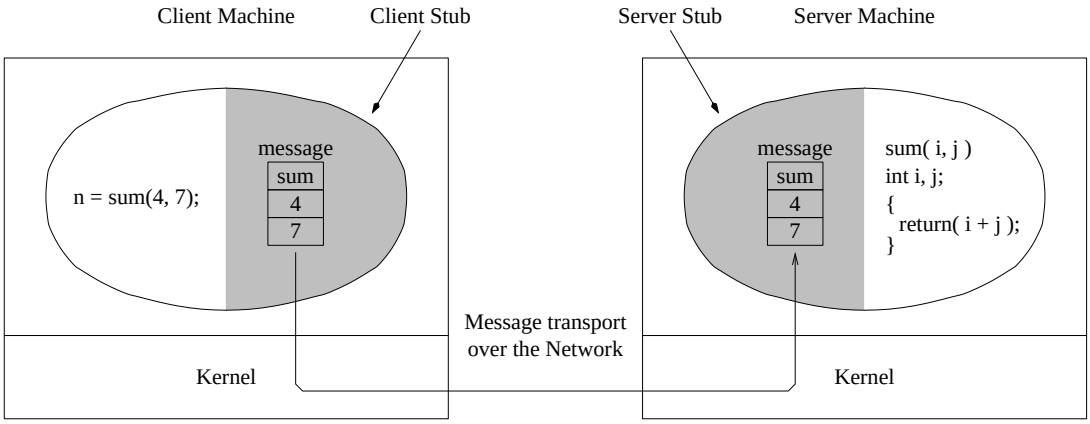
✲ Passos em uma chamada RPC:

❶ o procedimento cliente invoca o stub cliente no modo usual;
❷ o stub cliente monta a mensagem e interrompe o $kernel$;
❸ o $kernel$ envia a mensagem para o $kernel$ remoto;
❹ o $kernel$ remoto entrega a mensagem ao stub do servidor;
❺ o stub do servidor desempacota a mensagem e chama servidor;
❻ o servidor faz o trabalho e retorna o resultado ao stub;
❼ o stub do servidor empacota a mensagem e interrompe o $kernel$;
❽ o $kernel$ remoto envia a mensagem para o $kernel$ cliente;
❾ o $kernel$ cliente entrega a mensagem ao stub cliente;
❿ o stub desempacota o resultado e returna para o cliente.

4.2 - Passagem de Parâmetros no RPC

✲ **parameter marshaling:** empacotamento de parâmetros em mensagem.

• ... **exemplo:** considere o procedimento remote **sum(i,j)** que recebe dois inteiros e retorna a soma aritmétrica.



... 4.2 - Passagem de Parâmetros no RPC

- ... enquanto as máquinas cliente e servidor são idênticas e todas as variáveis são escalares, tais como inteiro, caracteres, etc., este modelo funciona bem;

- ... entretanto, em sistemas distribuídos onde diferentes arquiteturas estão presentes não é possível utilizar-se de um mecanismo tão simples;

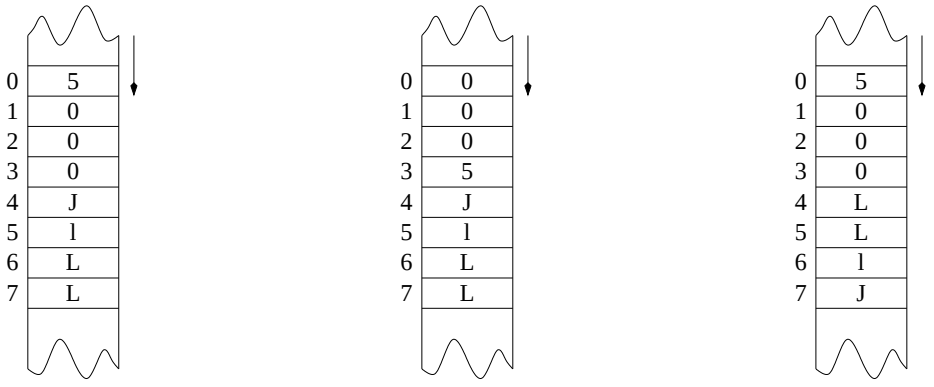
- ... problemas similares são encontrados na representação de inteiros (complemento de um ou de dois) bem como com números de ponto flutuante;

- ... além disso, algumas máquinas utilizam-se de diferentes maneiras para armazenar seus dados em memória, ou seja:

  - **little endian:** numeração começa no lado de baixa ordem;
  - **big endian**: numeração começa no lado de alta ordem;

... 4.2 - Passagem de Parâmetros no RPC

✲ **exemplo:** transferência byte a byte na rede de um inteiro (ex., 5) e de uma string (ex., JILL) de caracteres de uma máquina Intel 486 para um Sparc:



Original Message on the Intel 486 | Message after receipt ont the Sparc | Message after been inverted

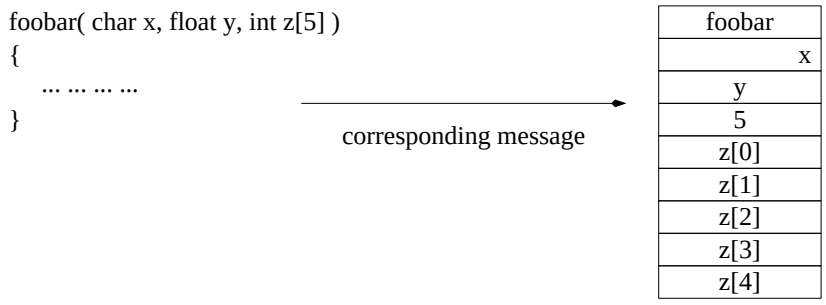
- ... o problema é que inteiros são invertidos, mas strings não, assim, uma solução possível requer a adição de informações sobre o que é string e o que é inteiro.

... 4.2 - Passagem de Parâmetros no RPC

✲ Desde modo, um mensagem correspondente a um procedimento remoto com "n" parâmetros irá contemplar "n+1" campos, um identificando o procedimento e os demais identificando os parâmetros;

- ... entretanto, isto exige a concordância de ambas as partes para a representação dos tipos de dados, da lista de parâmetros e da mensagem.

- **exemplo:** considere um procedimento com três parâmetros, um inteiro, uma string de caracteres e um ponto flutuante.



... 4.2 - Passagem de Parâmetros no RPC

✲ Mesmo com informações adicionais, ainda há outros aspectos para discutir, por exemplo, a representação das mensagens:

- ... um maneira é padronização das mensagens, ou seja, utilizar uma representação canônica para inteiros, caracteres, booleanos, números de ponto flutuante, etc.

- ... como consequência, o stub do servidor não mais se preocupará com a ordem dos bytes, posto que esta ordenação na mensagem passa a ser fixa;

- ... este método no entanto pode ser ineficiente, por exemplo, quando duas máquinas $big\ endians$ estão trocando mensagens.

✲ **alternativa:** o cliente utiliza sua representação e a informa no primeiro byte, assim, um cliente $little\ endian$ monta uma mensagem $little\ endian$;

- ... quando a mensagem chega, o stub do servidor examina o primeiro byte para identificar o cliente e, assim, realiza a conversão quando for necessário.

... 4.2 - Passagem de Parâmetros no RPC

✲ Nos Sistemas baseados em RPC, *stubs* são gerados automaticamente a partir de uma especificação única, facilitando a vida do programador, reduzindo a chance de erros e tornando o sistema transparente com respeito a representação interna.

✲ Finalmente, o nosso último e mais difícil problema: como passar ponteiros?

❶ simplesmente esqueça ponteiros e parâmetros por referência;
❷ substituir a chamada por referência (*call by referentce*) por *copy/restore*;
❸ otimização da solução anterior de modo a dobrar sua eficiência, o que é conseguido através do gerador automático dos *stubs*;
... evitar cópias desnecessárias das variáveis trocadas entre cliente e stub do cliente, bem como entre servidor e stub do servidor.

4.3 - Ligação Dinâmica no RPC

✲ **dynamic binding:** oferecer flexibilidade para o cliente localizar o servidor.

- ... para caracterizarmos tal funcionalidade, considere uma especificação formal para um servidor, contemplando sua versão e procedimentos oferecidos:

```
#include <header.h>

specification of file_server, version 3.1:

  long read( in char name[MAX_PATH], out char buf[BUF_SIZE], in long bytes, in long position );

  long write( in char name[MAX_PATH], in char buf[BUF_SIZE], in long bytes, in long position );

  long create( in char name[MAX_PATH], in int mode );

  long delete( in char name[MAX_PATH] );

end;
```

- ... como já discutido, este servidor não armazena estado, não obstante, RPC é neutro, possibilitando o desenvolvedor considerar o que lhe for solicitado.

... 4.3 - Ligação Dinâmica no RPC

✲ **especificação formal:** seu principal uso reside no gerador de *stubs*, que com base nas informações ali contidas, gera os *stubs* cliente e servidor;

- ... quando o servidor é instanciado, a chamada *initialize(...)* **exports** a sua interface, ou seja, o servidor envia uma mensagem para o programa denominado *binder*, informando a sua existência;

- ... neste processo, referenciado como **registering**, o servidor informa seu nome, versão, um identificador único e um identificador dependente do sistema;

- ... um servidor, pode também, se desregistrar de um *binder*, quanto não mais está preparado para oferecer serviços;

Register	Name, Version, Handle, Unique ID	
Deregister	Name, Version, Unique ID	
Lookup	Name, Version	Handle, Unique ID

... 4.3 - Ligação Dinâmica no RPC

✲ Vejamos agora, como o cliente localiza o servidor:

- quando o cliente invoca um procedimento remoto pela primeira vez, o *stub* percebe que ele ainda não se ligou ao servidor, então ele envia uma mensagem para o *binder* requisitando a interface do servidor na versão 3.1;

- ... o *binder* verifica se um ou mais servidores já exportaram aquela interface com aquele nome e versão, e responde à requisição do cliente informando seu identificador único e seu endereço de máquina;

- ... o *stub* cliente utiliza este *handle* com endereço e envia um mensagem de requisição contendo parâmetros e o identificador único ao servidor;

... 4.3 - Ligação Dinâmica no RPC

✲ **flexibilidade:** este método de exportar e importar interfaces é altamente flexível, dado que atende múltiplos servidores que oferecem a mesma interface;

- ... o $binder$ poderá distribuir os clientes entre os servidores de forma aleatória para balancear a carga, se for o caso;

- ... por exemplo, o servidor pode exigir que seja referenciado somente por um conjunto de usuários previamente cadastrados, caso no qual irá recusar requisições de usuários que não se encontram na lista;

- ... o $binder$ poderá retirar peridicamente os servidores, desregistrando aqueles que falharam em responder, de modo a atingir um grau de tolerância a falhas;

- ... o $binder$ poderá verificar se cliente e servidor estão utilizando-se da mesma versão de interface, e/ou encaminhar a chamada a servidores de mesma versão.

... 4.3 - Ligação Dinâmica no RPC

✲ Entretanto, esta forma de ligação apresenta **desvantagens:**

- $overhead$ de exportar e importar interfaces toma tempo;

- se muitos processos clientes tem tempo de vida curto e se cada processo deve ser iniciado novamente, o efeito pode ser significante;

- em sistemas distribuídos grandes, o $binder$ pode se constituir num gargalo, o que sugere múltiplos $binders$ p/ amenizar o problema;

- consequentemente, quando uma interface é registrada e desregistrada, um número razoável de mensagens se fará necessário para manter todos os $binders$ sincronizados e atualizados, criando ainda mais $overhead$.

4.4 - RPC na presença de Falhas

✲ O objetivo do RPC é esconder a comunicação fazendo com que as chamadas de procedimentos remotos se pareçam como se fossem locais, o que de fato se confirma enquanto Cliente e Servidor estejam funcionando perfeitamente;

• ... mas na presença de falhas, problemas decorrentes das diferenças entre prodimento remoto e local são evidenciados.

✲ Para abordarmos este tema, 05 diferentes classes de problemas serão discutidas:

❶ cliente não está apto a localizar o servidor;
❷ mensagem **request(...)** do cliente para o servidor é perdida;
❸ mensage de **reply(...)** do servidor p/ o cliente é perdida;
❹ servidor morre depois de receber um mensagem de **request(...)**;
❺ cliente morre depois de receber um mensagem de **reply(...)**.

4.4.1 - Cliente não está apto a localizar o Servidor

• **cenário:** considere que o cliente foi compilado gerando uma particular versão de *stub* e que o binário não mais tem sido usado por um longo período;

• ... se nesse período o servidor for modificado p/ agregar novas funcionalidades, novas versões de interfaces serão instaladas e novos *stubs* serão gerados;

• ... quando o cliente finalmente for executado, não será possível localizar o servidor ou o servidor não concordará com os termos impostos pelo cliente na comunicação, dado que são versões diferentes;

✲ Nos 02 exemplos já dicutidos, quais sejam, servidor de arquivo e procedimento remoto *sum(i,j)*, o código de retorno -1 ou a introdução de um novo tipo de erro (p. ex., *Cannot locate Server*) não são gerais o suficiente.

• **alternativa:** possibilitar que erros conduzam/sejam tratados por rotinas específicas de exceção, ou seja, *raise an exception*.

4.4.2 - Mensagens de request(...) Perdidas

- este é um caso simples de ser tratado, adicionando-se um $timer$ no $kernel$ quando do envio da mensagem de **request(...)**;

- ... se o tempo expirar antes do reconhecimento ou **reply(...)** retornar, o $kernel$ envia a mensagem novamente, mas não saberá reconhecer a diferença entre retransmissões e a mensagem original;

- ... não obstante, se várias mensagens de **request(...)** forem perdidas, o $kernel$ poderá abandonar e falsamente concluir que o servidor está fora, retornando ao caso $Cannot\ locate\ Server$, caso ❶.

4.4.3 - Mensagens de reply(...) Perdidas

- $replys$ são consideravelmente mais difíceis de serem tratados, dado que o $kernel$ do cliente não tem certeza da razão de não ter tido resposta;

- **solução:** utilizar-se do $timer$ para repetir operações tantas vezes quanto necessárias e de forma segura, ou seja, **request(...) idempotent**;

- ... mas o que fazer quando não podem ser repetidas de forma segura, por exemplo, numa operação bancária solicitando a transferência de dinheiro;

- ... estruturar as requisições no modo $idempotent$ nem sempre é possível, pois na prática, muitas delas são inerentemente não $idempotent$;

... 4.4.3 - Mensagens de reply(...) Perdidas

- **alternativa:** estruturação de todas as requisições no modo $idempotent$, quando possível, com a adição do número sequencial nas requisições;

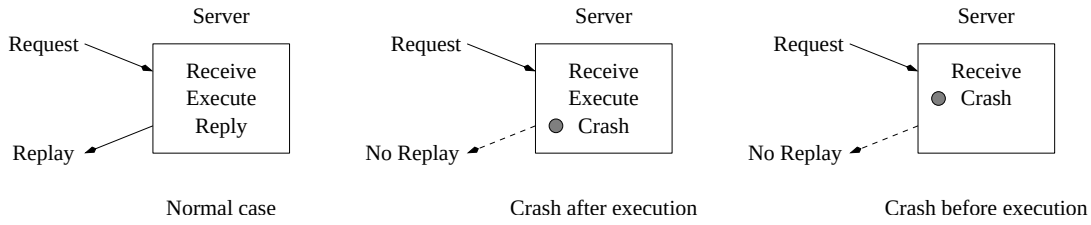
- ... este mecanismo possibilita o rastreamento por parte do $kernel$ do servidor das mensagens recebidas do $kernel$ do cliente, dando condições para o $kernel$ diferenciar uma requisição original daquelas retransmitidas;

- ... um salvaguarda adicional pode ser conseguida com um bit a mais no cabeçalho da mensagem para distinguir requisições iniciais daquelas retransmitidas;

- ... a razão se deve ao fato de que devemos tomar um maior cuidado com as retransmissões, se comparadas com a requisição original.

4.4.4 - Servidor finaliza após receber mensagem de request(...)

- embora relacionado a $idempotency$, infelizmente não pode ser resolvido utilizando-se dos números sequenciais;



- ... o tratamento das duas últimas condições se difere, na segunda o sistema do servidor deve reportar uma falha ao cliente (p. ex., $raise\ an\ exception$) e na terceira o cliente apenas retransmite a requisição;

... 4.4.4 - Servidor finaliza após receber mensagem de request(...)

✲ Várias são as alternativas propostas:

- **at least once semantics:** continuar tentanto até receber um $reply$, ou seja, garantia de que a chamada do procedimento remoto foi requisitada ao menos uma vez, possivelmente várias vezes.

- **at most once semantics:** garantia de que a chamada do procedimento remoto foi requisitada uma vez, possivelmente nenhuma mais.

- **guarantee nothing:** quando o servidor morre, o cliente não recebe nada.

✲ **Conclusão:** a morte repentina do servidor muda radicalmente a natureza da chamada de procedimento remoto, pois claramente é diferente nos sistemas de um único processador quando comparada nos sistemas distribuídos;

- ... no primeiro caso, a morte do servidor implica a morte do cliente, assim nem a recuperação não é nem mesmo possível ou necessária.

4.4.5 - Cliente finaliza após receber mensagem de reply(...)

- **definição:** quando o cliente não mais está esperando pelo resultado de sua requisição, o servidor é dito ser um **orfão** ($orphan$).

- ... orfãos causam uma variedade de problemas, como por exemplo, gasto desnecessário de ciclos de processador, retenção de recursos, etc.

✲ NELSON, B.J.: "Remote Procedure Calls", PhD Thesis, Carniegie-Mellon University, 1981. propôs quatro soluções para este problema:

❶ antes do $stub$ cliente enviar uma mensagem ao procedimento remoto, ele atualiza o log informando o que irá fazer de modo que seja recuperável;

- ... após o $reboot$ do sistema o log de mensagens é avaliado/verificado e orfãos são finalizados, uma solução denominada **extermination**.

- ... esta solução apresenta como devantagens o esforço grotesco de escrita em disco de cada registro de chamada a procedimento remoto.

... 4.4.5 - Cliente finaliza após receber mensagem de reply(...)

❷ **reincarnation:** nessa proposta o tempo é subdividido em uma sequência de marcas numeradas e nenhum dado armazenado em disco;

- ... quando um cliente $reboots$, o mesmo envia um mensagem $broadcast$ para todas as máquinas declarando o ínicio de uma marca;
- ... quando a mensagem de $broadcast$ for recebida, todas as computações remotas serão finalizadas, embora possam restar alguns orfãos caso a rede tenha sido particionada durante o período em que o cliente falhou;
- ... quando os pares que receberam devolvem a resposta, suas mensagens irão conter uma marca obsoleta, tornando-as fáceis de serem detectadas.

... 4.4.5 - Cliente finaliza após receber mensagem de reply(...)

❸ **gentle reincarnation:** quando uma mensagem de $broadcast$ de marca é recebida, cada máquina verifica se há alguma computação remota, localizando o seu requisitor;

- ... somente se o requisitor não puder ser achado é que o cômputo é destruído.

❹ **expiration:** para cada chamada de procedimento remoto atribui-se um tempo para que seja concluída e, em caso de não ser completada, uma requisição explícita de um outro $quantum$ se faz necessária;

- ... o problema maior é escolher um valor razoável para o $quantum$ face a chamadas de procedimentos remotos com os mais diferentes requisitos.

✭ Na prática, nenhum desses métodos é desejável, até mesmo porque eliminar um orfão poderá gerar consequência imprevisíveis.

5 - Comunicação de Grupo

✲ Até agora, assumimos a comunicação RPC entre duas participantes, entretanto, algumas circunstâncias requerem a participação de múltiplos processos;

- ... pode ser desejável que um cliente envie mensagens a todos os servidores, de modo a garantir que a requisição seja atendida mesmo se um deles finalizar;

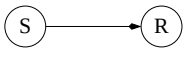
- ... RPC não suporta a comunicação de um transmissor para vários receptores, a não ser que várias chamadas em separado sejam consideradas.

✲ Nesta seção iremos discutir mecanismos de comunicação nos quais uma mensagem pode ser enviada para múltiplos receptores numa única operação.

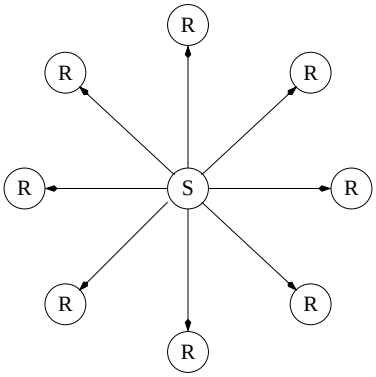
5.1 - Introdução da Comunicação de Grupo

✲ **definição de grupo:** coleção de processos que agem em conjunto em algum sistema ou de modo específico do usuário.

- ... a propriedade chave de todos os grupos é a de que se uma mensagem é enviada para o grupo, todos os membros do grupo a recebem.



Point−to−Point communication is from one sender to one receiver.



One−to−Many communication is from one sender mulitple receivers

... 5.1 - Introdução da Comunicação de Grupo

✲ Mecanismos de Gerenciamento são necessários por um série de razões:

❶ grupos são dinâmicos, ou seja, novos grupos são criados e outros são destruídos;
❷ um processo pode se juntar a um grupo ou solicatar exclusão de um grupo;
❸ um processo pode fazer parte de vários grupos ao mesmo tempo.

- ... ou seja, grupos podem rudimentarmente serem comparados com as organizações sociais, principalmente no que se refere a autonomia de suas entidades em deixar ou entrar em um grupo.

✲ Propósito da Comunicação de Grupos:

❶ permitir que um processo estabeleça comunicação com um grupo de processos;
❷ tratar esta negociação como uma abstração simples de comunicação.

... 5.1 - Introdução da Comunicação de Grupo

✲ Modelos de Comunicação de Grupo:

❶ **multicasting:** em algumas redes, é possível criar endereços especiais nos quais múltiplas máquinas podem receber mensagens;

❷ **broadcasting:** se não houver suporte para o $multicast$, ainda assim a rede poderá entregar para todos as máquinas da rede pacotes com endereços específicos (p.ex., FF:FF:FF:FF:FF:FF), ou seja, difusão para todos na rede;

❸ **unicasting:** se nenhum dos dois anteriores é suportado, ainda assim a comunicação de grupo poderá ser suportada se cada transmissor enviar a mensagem para cada membro do grupo com o qual se está comunicando.

5.2 - Aspectos de Projeto

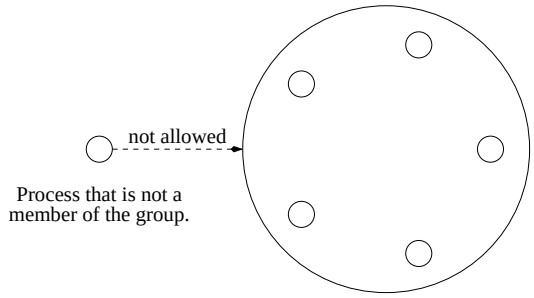
✲ Muitos dos aspectos anteriormente vistos como passagem de mensagens bloqueantes e não bloqueantes, bufferizadas e não bufferizadas, e outras constituem aspectos de projeto da comunicação de grupo.

- ... entretanto, há muitos outros aspectos a considerar em decorrência da natureza da comunicação de grupo ser inerentemente diferente do envio de uma mensagem a um único processo.

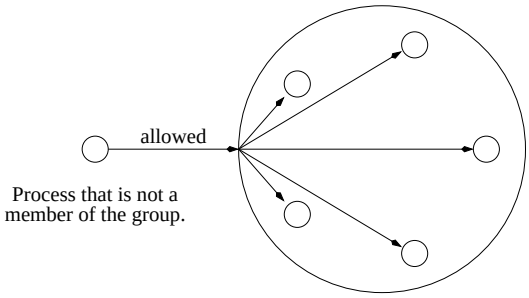
5.2.1 - Closed Groups *versus* Open Grupos

✲ Sistemas que suportam comunicação de grupo podem ser divididos em duas ou mais categorias dependendo de quem envia para quem:

- **closed groups:** somente os membros podem enviar para outros membros do grupo, mas mensagens podem ser enviadas para indivíduos membros;
- **open groups:** qualquer processo no sistema pode enviar para qualquer grupo, ou seja, não precisa ser membro do grupo.



Outsiders may not send to closed group.



Outsiders may send to an open group.

... 5.2.1 - Closed Groups *versus* Open Grupos

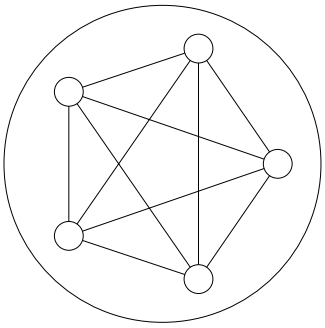
✲ A decisão acerca do suporte a **closed groups** ou *open groups* está usualmente relacionada no porquê da necessidade de se usar grupos, ou seja:

- **processamento paralelo:** constituem-se um grupo fechado, pois os processos interagem entre si para resolverem um problema, mas não precisam trocar informações/interagir com o mundo exterior;

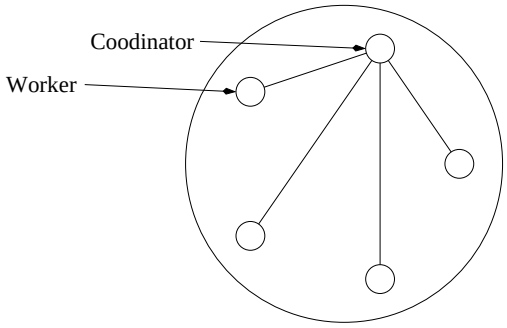
- **servidores replicados:** neste caso é importante que processos não membros possam trocar informações com o grupo e, adicionalmente, os membros possam fazer uso da comunicação do grupo.

5.2.2 - Peer Groups *versus* Hierarchical Groups

✲ Outro aspecto importante reside na forma como um grupo é estruturado, ou seja, todos os processos são iguais (decisões são tomadas coletivamente) ou há um coordenador (decisões são centralizadas):



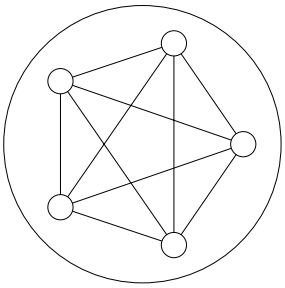
Communication in Per−Group



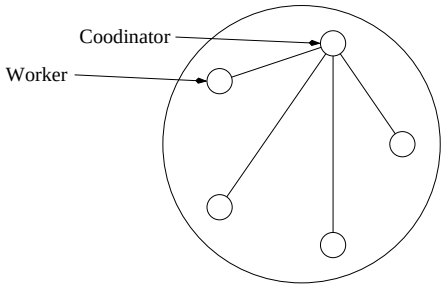
Communication in a Simple Hierarchical Group

... 5.2.2 - Peer Groups *versus* Hierarchical Groups

✲ **peer group:** por suportar comunicação simétrica, o ponto de falha não é único, mas a tomada de decisão é complicada.



Communication in Per−Group



Communication in a Simple Hierarchical Group

✲ **hierarchical group:** embora a perda da coordenação possa levar todo o grupo ao estado de suspensão, enquanto executando tomas as decisões sem importunar qualquer outro membro do grupo.

5.2.3 - Group Membership

✲ Para gerenciar comunicação de grupo, antes é necessário gerenciar o próprio grupo, ou seja, oferecer métodos para criar e remover grupos, possibilitar a um processo se juntar a um grupo bem como deixá-lo, etc.

- ... uma alternativa é ter um servidor de grupo, capaz de manter uma base de dados completa acerca dos grupos e de seus membros;

- ... infelizmente, embora seja direto, eficiente e fácil de implementar, compartilha a mesma técnica dos sistemas centralizados que traz como desvantagem a baixa confiabilidade, posto que, a quebra do servidor de grupos para todo o sistema;

✲ **gerenciamento distribuído:** embora aumente a confiabilidade (ponto de falha não mais é único), exige maior troca de mensagens para se manter o coorência das informações nas bases de dados.

... 5.2.3 - Group Membership

✲ **gerenciamento distribuído** é direto, enquanto tudo funciona bem, como em:

- ... no gerenciamento distribuído em grupos abertos, um candidato a membro envia uma mensagem para todos os membros do grupo anunciando sua presença;
- ... algo similar está presente nos grupos fechados, na verdade, mesmo estes devem ser abertos quando da inclusão de novos membros;
- ... para deixar um grupo, o membro simplesmente anuncia sua saída;

✲ Entretanto, há dois aspectos que complicam o gerenciamento:

❶ se um membro termina de forma anormal, ele efetivamente deixa o grupo, no entanto os demais membros não tem idéia do que ocorreu;
❷ operações de tornar-se membro e deixar um grupo devem ser síncronas enquanto mensagens estão sendo enviadas pelos membros e para os membros, senão, alguns recebem mensagens e outros não;

5.2.4 - Group Addressing

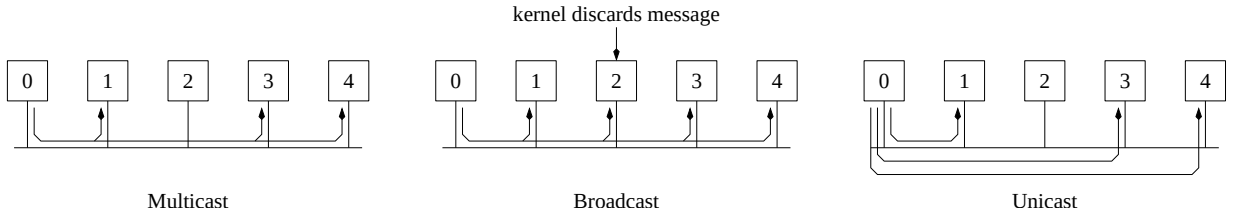
✲ Assim como processos, grupos também necessitam ser endereçados como uma única entidade, assim todos os seus membros receberão as mensagens.

✲ **primeira abordagem:** endereçar os grupos como entidades únicas, o que gera os seguintes desdobramentos:

- ... para isto, uma solução é atribuir um endereço $multicast$ para o grupo, desde modo, as mensagens são enviadas para todos os processos associados ao endereço $multicast$ do grupo e a nenhum outro mais;
- ... se o suporte for $broadcast$, caberá ao $kernel$ extrair as mensagens para os membros do grupo, caso haja algum processo naquela máquina de um dado grupo que esteja recebendo a informação;
- ... se o suporte for $unicast$, o $kernel$ de cada máquina deverá contemplar um lista de todas as máquinas acomodando processos de um dado grupo.

... 5.2.4 - Group Addressing

✲ Cabe ressaltar que o transmissor não está a par do tamanho do grupo ou se a comunicação implementada é por *multicasting*, *broadcasting* ou *unicasting*.



... 5.2.4 - Group Addressing

✲ **segunda abordagem:** requer que o transmissor contemple uma lista explícita de todos os destinos (p.ex., endereço IP);

- ... este método obriga que todos os processos do usuário e membros de um grupo saibam quem é membro de que grupo, ou seja, não é transparente;

- ... adicionalmente, quando ocorre mudanças, processos do usuário precisam atualizar suas listas, o que pode constituir-se numa tarefa árdua.

... 5.2.4 - Group Addressing

- **terceira abordagem:** denominado **predicate addressing**, as mensagens contendo um predicado (a ser avaliado pelo receptor) são enviadas para todos os membros do grupo usando um dos métodos descritos anteriormente.

- ... predicado poderá conter o identificador da máquina do receptor, tamanho da memória, variáveis locais ou outros fatores;

- ... se o predicado é avaliado como TRUE, a mensagem é aceita, caso contrário (predicado é avaliado como FALSE) a mensagem será descartada;

- ... esta abordagem possibilita enviar as mensagens para máquinas que satisfaçam as restrições impostas no predicado, p. ex., a mensagem será processada somente por máquinas com pelo menos 4 MB de memória livre.

5.2.5 - Send and Receive Primitive

- Idealmente, comunicação ponto-a-ponto e de grupo devem convergir para o memso conjunto de primitivas, entretanto, se o rpc for o mecanismo de comunicação, como modelar uma chamada em grupo como uma chamada RPC?

- ... em comunicação de grupo, há potencialmente **n** diferentes respostas e, assim, como tratar com as **n** respostas ($replys$) ?

- **abordagem comum:** abandonar o modelo $request/reply$ como infraestrutura e reutilizar as chamadas explícitas de envio e recepção ($one\text{-}way\ model$).

- ... assim, os procedimentos de biblioteca que os processos invocam podem ser do tipo ponto-a-ponto ou podem ser diferentes, como no RPC.

... 5.2.5 - Send and Receive Primitive

- ... se o sistema é baseado no RPC, processos não invocam *send* e *receive* diretamente, assim há um menor interesse na junção entre os dois conjuntos de primitivas;

- ... se os processos invocam diretamente *send* e *receive*, embora possa haver a junção entre os dois conjuntos, cabe a ressalva de que a comunicação é *one-way*;

- ... isto é, *replys* são mensagens independentes e não estão associadas às requisições prévias que indiretamente as geraram.

5.2.6 - Atomicity

✲ **Atomicity:** *all-or-nothing property.*

- ... esta propriedade é de interesse dos sistemas distribuídos, pois torna mais simples a sua programação, posto que os processos não tem que se preocupar se algum outro processo não recebeu a mensagem;

- ... entretanto, implementar a atomicidade não é tão simples quanto possa parecer, posto que, para garantir que todos receberam é necessário que cada um envie um reconhecimento atestando a recepção.

✲ Entretanto, como manter esta propriedade na presença de falhas?

- ... por exemplo, sob algumas circunstâncias alguns membros do grupo irão receber a mensagem, enquanto outros não, algo inaceitável;

... 5.2.6 - Atomicity

- ... ou pior, aqueles que não receberam a mensagem, não tem como pedir por retransmissão e, mesmo que tivessem como pedir, se o servidor termina anormalmente, não há mais o que fazer;

✭ **Solução:** o transmissor inicia a comunicação com os memsbros do grupo, iniciando antes um $timer$ e retransmitindo quando necessário;

- ... ao receber uma mensagem, cada processo verifica se é a primeira vez que a recebeu, se for, envia a mensagem para todos os membros do grupo;

- ... se ele já recebeu esta mensagem, não há mais nada para fazer;

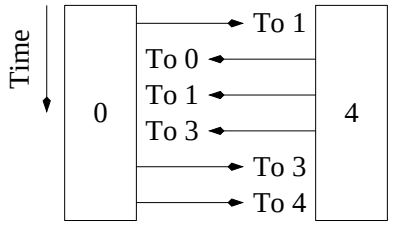
- ... seja qual for o número de máquinas que venham a falhar, todos os processos que sobreviverem irão receber a mensagem;

5.2.7 - Ordenação das Mensagens

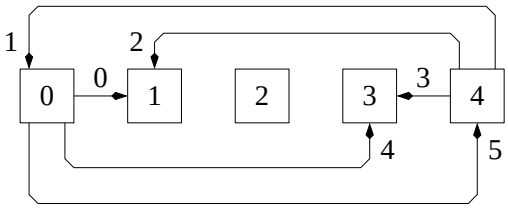
✭ Para tornar um sistema com suporte a comunicação de grupo inteligível e fácil de usar, duas propriedades precisam ser contempladas:

❶ **atomicity broadcast:** garante que a mensagem enviada ao grupo chega a todos os seus membros ou não chega a nenhum deles;

❷ **message ordering:** considere o cenário abaixo com 6 máquinas, cada uma com um processo e com o conjunto 0, 1, 3 e 4 formando um grupo.



The three messages sent by processes 0 and 4 interleaved in time.



Graphical representation of the six messages, showing the arrival order.

... 5.2.7 - Ordenação das Mensagens

- ... o problema é que quando dois processos disputando o acesso na LAN, a ordem na qual estas mensagens são enviadas é não determinística;
- ... processo 1 recebe mensagem de 0 e depois de 4, enquanto o processo 3 recebe mensagem de 4 e de 0 nesta ordem logo depois;
- ... se ambos os processos 0 e 4 estão tentando atualizar o mesmo registro em sua base de dado, os processos 1 e 3 irão finalizar com diferentes valores;
- ... é desnecessário dizer que esta situação é tão indesejável quanto aquela de mensagens serem recebidas por alguns e não por outros (falha de atomicidade);
- ... assim, se o sistema contemplar uma semântica bem definida no que se refere a ordem em que mensagens são entregues, podemos resolver estes problemas.

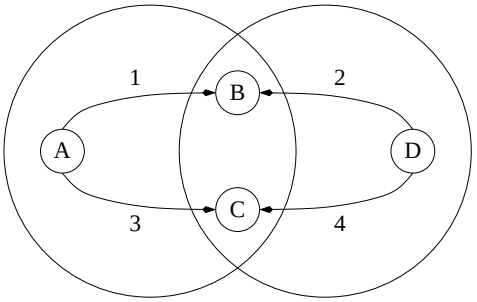
... 5.2.7 - Ordenação das Mensagens

- A melhor garantia de que todas as mensagens irão chegar instantaneamente e na ordem que foram geradas é ordená-las no tempo - **global time ordering**.
- ... no entanto, a ordenação de tempo absoluto não é fácil de ser implementada, face as variações e escolhas de medição no sistema;
- ... uma delas é **consistent time order**, na qual duas mensagens (p. ex., A e B) enviadas juntas no tempo são, numa escolha arbitrária, entregues a todos os membros do grupo numa dada ordem;
- ... pode acontecer que a escolhida para ser entregue primeiro não seja a primeira, mas neste casos o sistema não poderá depender disso;
- ... na verdade, a garantia é de que as mensagens serão entregues a todos os membros na ordem em que foram capturadas, não necessariamente na ordem que foram geradas;

5.2.8 - Overlapping Groups

✯ Como já mencionado, um processo pode ser membro de múltiplos grupos ao mesmo tempo, o que pode levar a um tipo de inconsistência;

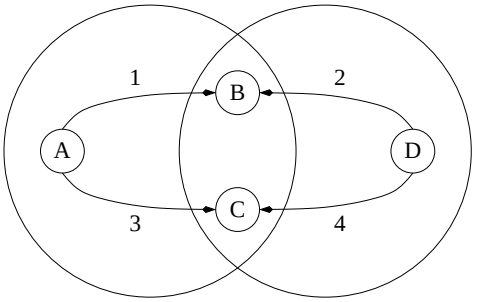
- ... considere dois grupos (p. ex., 1 e 2) e os processos A, B e C membros do grupo 1 e B, C e D membros do grupo 2.



Four processes, A, B, C and D, and four messages. Processes B and C get messages from A and D in a different order.

... 5.2.8 - Overlapping Groups

- ... suponha que os processos A e D decidam simultaneamente enviar uma mensagem para os seus respectivos grupos, e que o sistema utilize **global time ordering** dentro de cada um dos grupo;



Four processes, A, B, C and D, and four messages. Processes B and C get messages from A and D in a different order.

- ... como no exemplo prévio, a troca de mensagens se dá por comunicação *unicasting* na ordem apresentada na figura;

... 5.2.8 - Overlapping Groups

- ... novamente nos deparamos com a situação onde dois processos, no caso B e C recebem mensagens numa ordem diferente daquela prevista ou desejada;

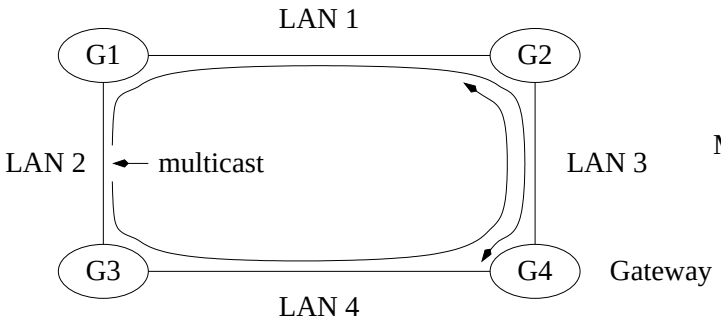
- ... embora tenhamos ordenação de tempo em cada grupo, não há necessariamente uma coordenação entre os vários grupos, daí o problema.

✯ Embora alguns sistemas ofereçam suporte de ordenação de tempo entre grupos que se sobrepõem, a sua implementação é frequentemente difícil de se obter.

5.2.9 - Scalability

✯ Muitos algoritmos funcionam bem quando os grupos contemplam poucos membros, mas o que acontece quando tem-se dezenas, centenas ou até mesmo milhares de membros por grupo? ... ou milhares de grupos?

- ... a presença de *gateways* pode afetar muitas das propriedades de implementação, para começar, *multicasting* torna-se mais complicado;



Multicasting in an internetwork causes trouble.

... 5.2.9 - Scalability

- Considere uma máquina da LAN 2 emitindo um $multicast$, precisamos então de um política nos $gateways$ acerca do tratamento a ser dado neste casos;

- ... se simplesmente, descartamos o $multicast$ nos $gateways$ G1 e G3, muitas das máquinas jamais o verão;

- ... se por outro lado, deixamos-o passar, o mesmo será replicado muitas e muitas vezes com decorrência do anel formado pelas redes;

- ... claramente, um algoritmo mais sofisticado se faz necessário, de modo que os pacotes possam ser rastreados a fim de evitar o crescimento exponencial do número de pacotes $multicast$ na rede;

... 5.2.9 - Scalability

- Outro aspecto é que alguns métodos tiram vantagem do fato de somente um pacote percorre a LAN num dado instante;

- ... com $gateways$ é possível termos dois pacotes em curso, simultâneos, destruíndo assim está propriedade tão útil.

- Finalmente, alguns algoritmos podem não ser escaláveis em decorrência da sua complexidade e do uso de componentes centralizadores, dentre outros fatores.