COMUNICAÇÃO ENTRE PROCESSOS
Remote Procedure Call - RPC

Comunicação entre processos (RPC)

- IPC por troca de mensagens
  - cada aplicação tem um protocolo específico
    - formato de mensagens; forma de tratamento de erros;
    - ex.: servidor de operações matemáticas
      - mensagens: operandos e operação
  - construção de outro cliente: tem que conheces estes detalhes
  - necessidade de um protocolo genérico para IPC para o projeto de aplicações distribuídas

Comunicação entre processos (RPC)

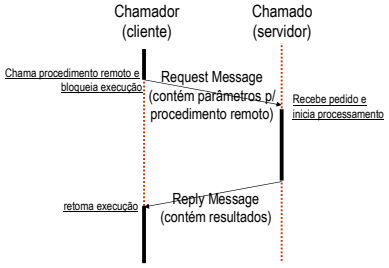
- Razões da aceitação de RPC - Remote Procedure Call
  - sintaxe simples
  - semântica familiar - similar a chamadas locais a procedimentos
  - serviço tem interface bem definida
    - verificações são possíveis em tempo de compilação
  - eficiência
  - independência de localização: pode ser usado para IPC local ou remoto
  - modelo cliente/servidor
    - um processo, ou um grupo de processos cooperantes, fornecem serviços
    - clientes fazem requests
    - servidores ficam a espera de requests, processam e dão a resposta

Comunicação entre processos (RPC)

- Modelo RPC
  - similar ao modelo de chamada de procedimentos usado para transferência de controle em um programa
  - Chamador coloca argumentos para o procedimento em algum local especificado
  - Controle é passado ao procedimento chamado
  - Corpo do procedimento é executado
    - pode incluir cópia de parâmetros
  - Após o final, o controle retorna ao ponto de chamada do procedimento, podendo envolver retorno de resultados
  - Procedimento chamado pode estar na mesma ou em outra máquina
  - Espaços de endereçamento separados
    - procedimento chamado não tem acesso a dados e variáveis do ambiente do chamador.

Comunicação entre processos (RPC)

- Modelo RPC - modelo básico de sincronismo
  - somente um processo ativo em determinado tempo



Comunicação entre processos (RPC)

- Modelo RPC - outros modelos de sincronismo são possíveis
  - por exemplo, chamadas assíncronas (não bloqueantes) são possíveis
  - cliente pode processar enquanto espera resposta
  - servidor pode processar requests em paralelo
    - ex.: lançar threads

Comunicação entre processos (RPC)

- Modelo RPC – análise
  - transparência sintática:
    - chamada remota ter mesma sintaxe que chamada local
  - transparência semântica
    - aspectos de sincronização: ok
    - diferentes espaços de endereçamento:
      - não há sentido no uso de ponteiros – exceção: memória compartilhada
    - vulnerabilidade a falhas:
      - mais de uma máquina → tipos de falhas que não aconteceriam em *local procedure call* tem que ser tratados
    - latência da rede:
      - RPC consome muito mais tempo que chamada local: 100 a 1000 vezes mais tempo

Comunicação entre processos (RPC)

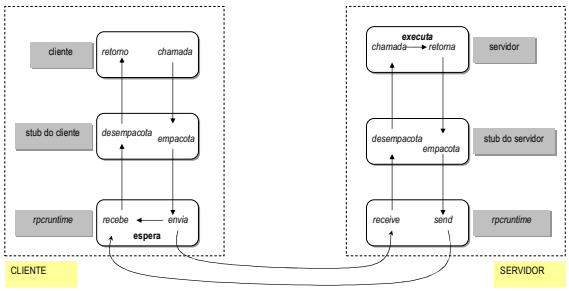
- Modelo RPC – Discussão
  - transparência semântica é impossível
  - alguns pesquisadores: RPC deve ser uma facilidade não transparente
  - usuários/programadores tem os benefícios mas devem estar "cientes" de que um procedimento é remoto e dispor de mecanismos para tratamento de maneira dependente da aplicação de
    - atrasos demasiados
    - falhas

Comunicação entre processos (RPC)

- Implementação de RPC – componentes
  - **Cliente:** faz a chamada ao procedimento remoto
  - ***Stub* do cliente:** faz a interface com o *runtime system* (esconde chamadas de "baixo nível" da aplicação)
  - **RPC *Runtime*:** comunicação entre dois computadores, esconde os detalhes da comunicação de rede
    - retransmissões, confirmações, etc ...
  - ***Stub* do servidor:** mesmo do *stub* do cliente
  - **Servidor**

Comunicação entre processos (RPC)

- Implementação de RPC – funcionamento:



Comunicação entre processos (RPC)

- Implementação de RPC - definição do Serviço
  - conforme IDL (*Interface Definition Language*)
  - independente de ambiente e linguagem
    - possibilidade inclusive de fazer RPC em infra-estrutura heterogênea (SO/HW)
  - especifica características do servidor visíveis aos clientes
    - procedimentos do servidor
    - parâmetros, tipos, se são de entrada, saída ou entrada e saída

```
Specification of file_server version 2.0
   long read(in char fname[n_size], out char buffer[b_size], in long bytes, in long position);
   long write(in char fname[n_size], in char buffer[b_size], in long bytes, in long position);
   int create(in char fname[n_size], in int mode);
   int delete(in char fname[n_size]);
end_specification;
```


Comunicação entre processos (RPC)

- Implementação de RPC
  - obtenção dos stubs
    - manual
    - automática → através do processo de compilação da IDL do serviço
  - compilação da IDL
    - compilador para cada ambiente:
      - IDL → C
      - IDL → Pascal
  - empacotamento/desempacotamento
    - operação de *marshalling* (também chamada *serialização*)
    - realizada pelos stubs cliente e servidor
    - transforma parâmetros (estruturas de dados) em formato para envio na rede que possa ser decodificado no destino, obtendo a mesma estrutura
      - XDR da SUN
      - ASN.1 da ISO

Comunicação entre processos (RPC)

- Implementação de RPC – Exemplo
  - IDL para serviço ADDIT

```
/* this code will be translated into the needed stubs and headers */
/* use: rpcgen addit.x; */

struct record {      /* arguments for RPC must be one single */
int first_num;       /* value or a structure of values */
int second_num;      /* first_num and second_num are addends */
};


program ADDITPROG {                 /* value to register the program */
    version ADDITVERS {             /* version must be assigned a value */
        int ADD_ARGS(record) = 1;   /* this is the service function */
    } =1;                           /* version value */
} = 0x20000003;                     /* program value */
```


Comunicação entre processos (RPC)

- Implementação de RPC – Exemplo
  - Cliente do serviço ADDIT

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "addit.h"

main(int argc, char *argv[]) {
    CLIENT *cl;
    int answer;

    record *rec = (record *) malloc(sizeof(record));
    if (argc != 4) {
        printf("Usage: %s server arg1 arg2\n", argv[0]); exit (1);}
    if (!(cl = clnt_create(argv[1], ADDITPROG,ADDITVERS,"tcp"))) {
        clnt_pcreateerror(argv[1]); exit(1);}
    rec->first_num = atoi(argv[2]);
    rec->second_num= atoi(argv[3]);
    answer = *add_args_1(rec,cl);
    if (answer <= 0) {
        printf("error: no meaningful results"); exit(1);}
    printf("%s + %s = %d\n", argv[2], argv[3], answer);

}
```


Comunicação entre processos (RPC)

- Implementação de RPC – Exemplo
  - Servidor ADDIT

```
/* add_svc.c server code for the example application */

#include <stdio.h>
#include <string.h>
#include <rpc/rpc.h>
#include "addit.h"

int *add_args_1(record *rec, CLIENT *clnt) {

    static int result;

    result = rec->first_num + rec->second_num;

    return ((int *) &result);
}
```


Comunicação entre processos (RPC)

- Implementação de RPC - Mensagens de chamada e retorno
- Mensagens de chamadas
  - componentes básicos:
    - identificação do procedimento remoto a executar
    - parâmetros para a execução
  - componentes adicionais:
    - número de seqüência da mensagem
      - identifica mensagens perdidas e duplicadas
      - possibilita fazer "*matching*" de resposta referente a qual request
    - tipo: call ou reply
    - Identificação do cliente:
      - possibilita servidor identificar cliente para mandar resposta
      - servidor pode autenticar cliente antes de realizar serviço (serviço seletivo)

Comunicação entre processos (RPC)

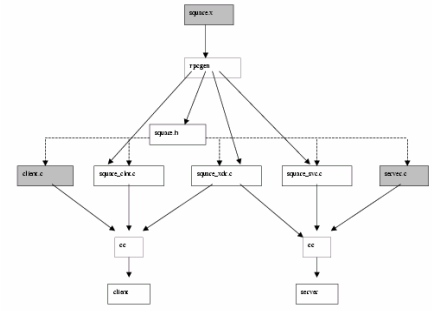
- Implementação de RPC - Mensagens de chamada e retorno
- Reply messages (retorno)
  - situações de falha
    - chamada não inteligível - violação do protocolo RPC - implem. Errada ...
    - Cliente não autorizado para o serviço
    - programa remoto, versão e número do procedimento não disponíveis
    - exceção durante execução do serviço (divisão por zero)
  - identificador da mensagem: mesmo do call correspondente
  - tipo: reply
  - resultado
  - identificação do cliente

Comunicação entre processos (RPC)

- RPC SUN
  - Desenvolvimento
    - Geração automática de stubs
    - interface descrita em IDL
    - compilador "rpcgen" gera, a partir da IDL:
      - arquivo header (tipos e constantes - arq.h)
      - arquivo XDR (*marshal*, *unmarshal* - arq_xdr.c)
      - *stub* cliente (arq_clnt.c)
      - *stub* servidor (arq_scv.c)

Comunicação entre processos (RPC)

- RPC SUN



Comunicação entre processos (RPC)

- RPC SUN
  - IDL
    - aceita 1 argumento de entrada
    - procedimentos com mais de 1 argumento → estruturas
    - sem argumentos: passar NULL
    - chamada RPC tem sempre 2 argumentos: entrada e "handler" (tratador)
    - retorno é um único resultado

Comunicação entre processos (RPC)

- RPC SUN
  - Semântica das chamadas no RPC - SUN
    - *at least once*
    - em caso de *timeout*, retransmite
    - nro_tentativas = tempo total % timeout (defaults 25 e 5 segundos)
    - retorna erro caso não obtenha resposta após nro_tentativas
  - broadcast: suportado, modo datagrama, retransmissões por default; enviado aos portmapper de todos os nodos
  - *binding*: local, usando *portmapper*
    - servidor registra prog, versao, e porta com portmapper
    - cliente deve descobir o port (**clnt_create**)

Comunicação entre processos (RPC)

- RPC SUN
  - segurança
    - sem autenticação,
    - autenticação UNIX - cada mensagem carrega UID e GID do usuário cliente
    - cada mensagem carrega um identificador criptografado (*netname*) do usuário, servidor decriptografa e decide execução (*secure* RPC)
    - uso do DES - *Data Encription Standard*

Comunicação entre processos (RPC)

- RPC SUN
  - críticas
    - não tem transparência de localização
    - IDL não permite especificar argumentos
    - não é independente de protocolo de transporte (TCP ou UDP)
    - em UDP, mensagens limitadas a 8 kbytes
    - semântica *at-least-once:* não aceitável para algumas aplicações
    - serviço de ligação por nodo

Comunicação entre processos (RPC)

- EXERCíCIOS
  - ache exemplos para as diversas semanticas de ativação de servidores: persistentes, por sessão, por chamada.
  - Faz sentido um servidor por sessão tratar pedidos de maneira concorrente ? Explique.
  - o que é uma chamada órfã? como é o tratamento de chamadas órfãs para semântica de chamada last-of-many e at-least-once ?
  - Suponha que o serviço que você está construindo não é idempotente e que o sistema de RPC oferece semantica *at-least-once*. Que mecanismos voce deve construir para alcançar semântica *exactely-once ?*
  - implemente, usando RPC, o serviço de operações matemáticas descrito nos exercícios passados