COMUNICAÇÃO ENTRE PROCESSOS
Remote Procedure Call - RPC

Comunicação entre processos (RPC)

- IPC por passagem 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
      - erros: repete operacoes - idempotentes
  - construção de outro cliente: tem que conhecer estes detalhes
  - necessidade de um protocolo genérico para IPC para o projeto de aplicações distribuídas

Comunicação entre processos (RPC)

- RPC - Remote Procedure Call
  - caso especial de passagem de mensagem
  - proporciona a programadores mecanismos comuns para construção de aplicações distribuídas
  - suporta a necessidade da maioria das 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 na mesma ou em máquinas diferentes
  - 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)

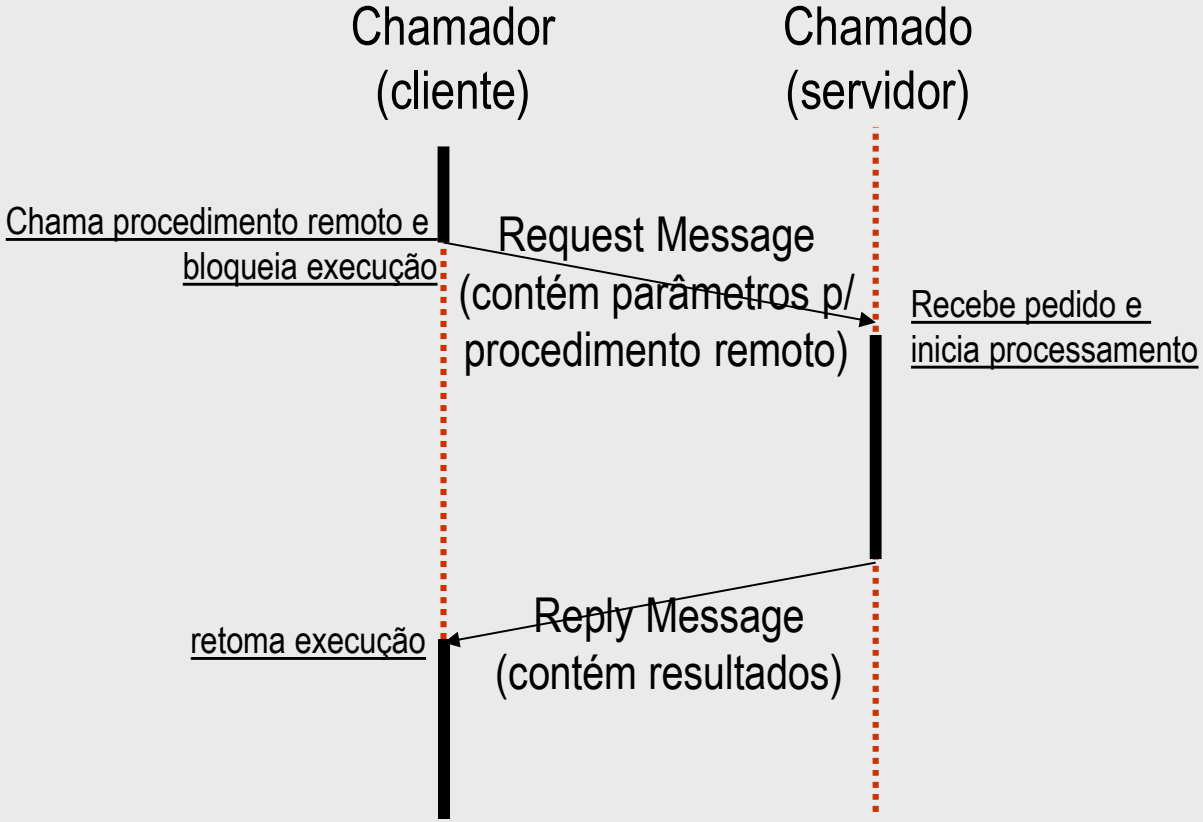
- Modelo RPC
  - similar ao modelo de chamada de procedimentos usado para transferência de controle em um programa
  - Chamador coloca argumentos para o procedimento em algum local especificado
  - Controle é passado ao procedimento chamado
  - Corpo do procedimento é executado
    - pode incluir cópia de parâmetros
  - Após o final, o controle retorna ao ponto de chamada do procedimento, podendo envolver retorno de resultados

Comunicação entre processos (RPC)

- Modelo RPC
  - 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
  - RPC usa passagem de mensagem para trocar informações entre processos chamador e chamado

Comunicação entre processos (RPC)

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



Comunicação entre processos (RPC)

- Modelo RPC - sincronismo
- 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
  - 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 endereços (ponteiros) - a menos que exista uma memória compartilhada global
    - 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
      atrasos demasiados e
      falhas
      de maneira dependente da aplicação

Comunicação entre processos (RPC)

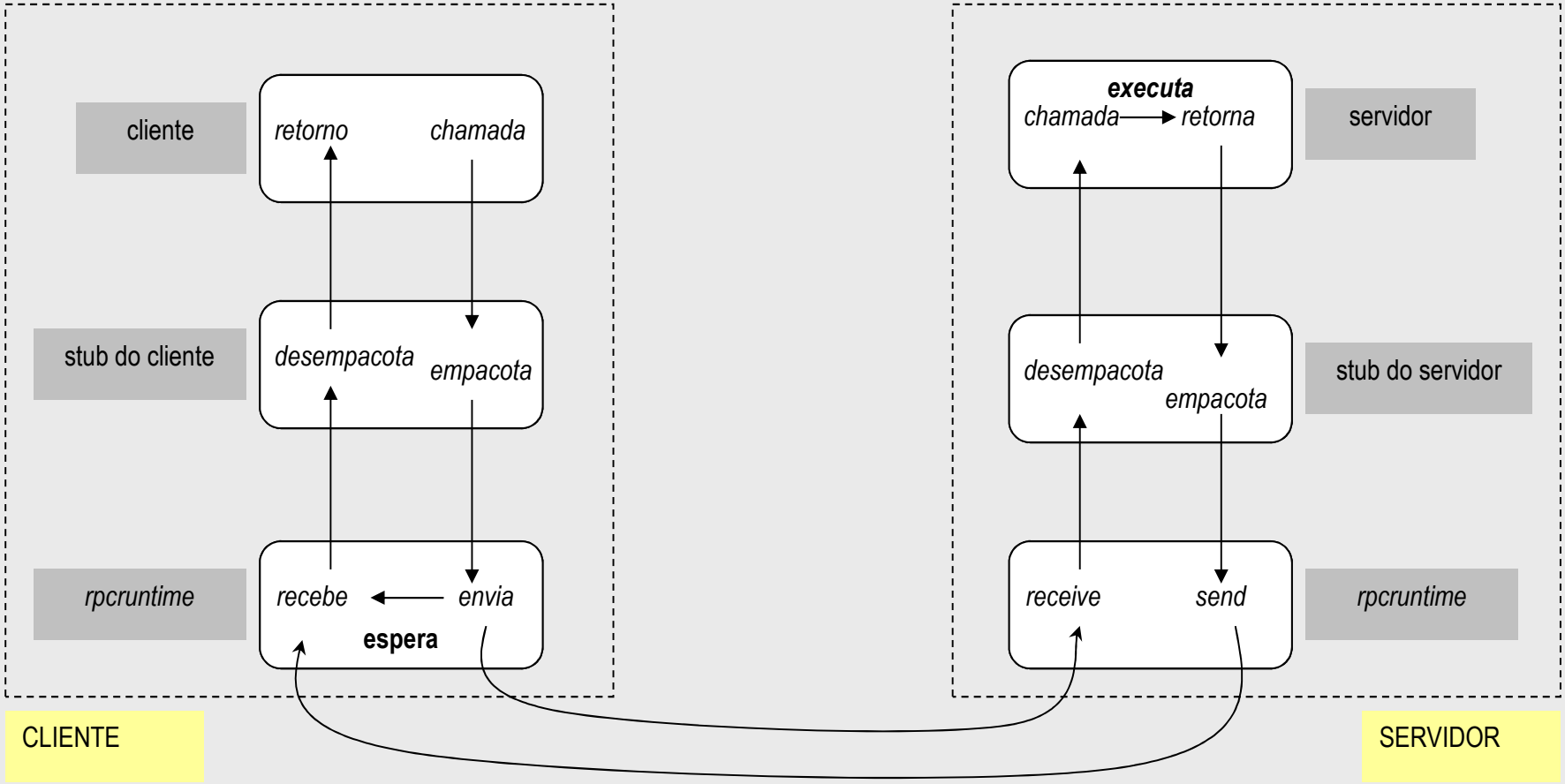
- Implementação de RPC

Comunicação entre processos (RPC)

- Composição do mecanismo:
  - **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)

- 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 infraestrutura heterogênea (SO/HW)
  - especifica características do servidor visíveis aos clientes
    - procedimentos do servidor
    - parâmetros, tipos, se são de entrada, saída ou entrada e saída

Comunicação entre processos (RPC)

- Implementação de RPC
- definição do Serviço - exemplo

```
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 de Stubs
  - manual
  - automática

    através do processo de compilação da IDL do serviço
- compilação da IDL
  - compilador para cada ambiente:
    - idl -> c
    - idl -> pascal

Comunicação entre processos (RPC)

- Implementação de RPC
- empacotamento/desempacotamento
  - operação de *marshalling*
  - também chamada *serialização*
  - realizado 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 - 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 - CLIENTE do serviço ADDIT

```
/* addit.c client code for the example application */

#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);
    }
```

Comunicação entre processos (RPC)

- Implementação de RPC - CLIENTE do serviço ADDIT (cont)

```
        ....
    rec->first_num = atoi(argv[2]);
    rec->second_num= atoi(argv[3]);
    answer = *add_args_1(rec,cl);

    if (answer <= 0) {
        printf("error: could not produce meaningful results");
        exit(1);
    }

    printf("%s + %s = %d\n", argv[2], argv[3], answer);

}
```

Comunicação entre processos (RPC)

- Implementação de RPC - SERVIDOR do serviço 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 - STUB CLIENTE do serviço ADDIT

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "addit.h"
#ifndef _KERNEL
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
#endif /* !_KERNEL */

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

int *
add_args_1(argp, clnt)
        record *argp;
        CLIENT *clnt;
{
        static int clnt_res;
```

Comunicação entre processos (RPC)

- Implementação de RPC - STUB CLIENTE do serviço ADDIT (cont)

```
int *
add_args_1(argp, clnt)
        record *argp;
        CLIENT *clnt;
{
        static int clnt_res;

        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, ADD_ARGS,
                (xdrproc_t) xdr_record, (caddr_t) argp,
                (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

Comunicação entre processos (RPC)

- Implementação de RPC - STUB SERVIDOR do serviço ADDIT

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "addit.h"
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
#include <signal.h>
#include <sys/types.h>
#include <memory.h>
#include <stropts.h>
#include <netconfig.h>
#include <sys/resource.h> /* rlimit */
#include <syslog.h>

#ifdef DEBUG
#define RPC_SVC_FG
#endif

#define _RPCSVC_CLOSEDOWN 120
static int _rpcpmstart;         /* Started by a port monitor ? */

/* States a server can be in wrt request */

#define _IDLE 0
#define _SERVED 1

static int _rpcsvcstate = _IDLE;        /* Set when a request is serviced */
static int _rpcsvccount = 0;            /* Number of requests being serviced */

static
void _msgout(msg)
        char *msg;
{
#ifdef RPC_SVC_FG
        if (_rpcpmstart)
                syslog(LOG_ERR, msg);
```

Comunicação entre processos (RPC)

- Implementação de RPC - Mensagens de chamada e retorno
- Call messages (chamada)
  - componentes básicos:
    - identificação do procedimento remoto a executar
    - parâmetros para a execução
  - componentes adicionais:
    - número de sequencia da mensagem
      - identifica mensagens perdidas e duplicadas
      - possibilita fazer “matching” de resposta referente a qual request
    - tipo: call ou reply
    - client identification:
      - 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)

- Implementação de RPC
- Servidores: stateless and statefull - impacto na interface

```
Specification of sless_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;
```

```
Specification of sfull_file_server version 2.0
    int Open(in char fname[n_size], in int mode);
    void seek(in int fd, in long position);
    long read(in int fd, out char buffer[b_size], in long bytes);
    long write(in int fd, in char buffer[b_size], in long bytes);
    void close(in int fd);
    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
- Servidores: stateless and statefull - impacto no serviço
  - statefull: normalmente tem melhor desempenho
    - ex.: stateless abre, posiciona, fecha arquivo a cada read/write
    - trafega com mais informações - mais uso de rede
  - statefull: normalmente interface mais fácil de usar
  - **entretanto: em situação de falhas é melhor ter stateless**
    - falha no servidor: se servidor falha e restarta, como fica o cliente que não percebe o reinício? o estado foi perdido ... -> cliente tem que detectar falha no servidor para então recuperar-se apropriadamente
    - Falha no cliente: estado do servidor permanece por quanto tempo ?

Comunicação entre processos (RPC)

- Implementação de RPC
- Servidores: stateless and statefull - impacto no serviço
  - servidores stateless facilitam o tratamento de falhas para o cliente
    - normalmente cliente tem só que tentar repetir o pedido de serviço até ter resposta
  - escolha de uso de stateless e statefull é bastante dependente de aplicação
  - importante opção de projeto

Comunicação entre processos (RPC)

- Implementação de RPC - semanticas de chamadas
  - erros que podem acontecer: perda da chamada, perda da resposta, nodo chamado sofre crash e é reiniciado, nodo chamador sofre request e é reiniciado
- semanticas de chamadas (considerando situações de erro)
  - may-be ou possibly call semantics
  - last-of many call semantics
  - at least once call semantics
  - exactely once call semantics

Comunicação entre processos (RPC)

- semanticas de chamadas (considerando situações de erro)
  - *may-be ou possibly call semantics*
  - semantica mais “fraca”
  - não apropriada para R**PC**
  - chamador não espera resposta ou espera resposta durante um tempo e depois desbloqueia para continuar seu processamento
  - não há retransmissões
  - útil em cenários onde chamador não quer/precisa da resposta, em redes de alta confiabilidade, ou em serviços onde a retransmissão do pedido não tem tanta utilidade

Comunicação entre processos (RPC)

- semanticas de chamadas (considerando situações de erro)
  - *last-of-many call semantics*
  - retransmite a chamada, baseado em timeouts, até receber resposta do chamado
  - chamador aceita resposta somente com identificador da última chamada gerada
  - no caso de “computações órfãs”: guardar controle das chamadas realizadas
  - computação órfã: computação cujo processo chamador não existe mais
    - ex.: chamador dispara pedido, chamado processa, chamador crash e restart, chamador repete pedido

Comunicação entre processos (RPC)

- semanticas de chamadas (considerando situações de erro)
  - *at least once call semantics*
  - semantica mais fraca que last-of-many
  - garante que a chamada é executada uma ou mais vezes e não especifica quais resultados (de que chamada) são retornados

Comunicação entre processos (RPC)

- semanticas de chamadas (considerando situações de erro)
  - *exactely once call semantics*
  - a mais desejável semântica
  - elimina possibilidade da procedure ser executada mais de uma vez, independentemente da retransmissão de mensagens
  - semanticas anteriores: forçam desenvolvedores projetarem interfaces com operações idempotentes - se mesma operação com mesmos parâmetros é chamada, então os mesmos resultados serão obtidos - sem efeitos colaterais
  - uso de time-out, retransmissões, identificadores de chamadas, cache de resposta para chamadas repetidas

Comunicação entre processos (RPC)

- Implementação de RPC - protocolos de comunicação
- Request protocol
  - procedure chamada não tem nada para responder
  - cliente não requer confirmação da execução do procedimento
  - somente uma mensagem por call
  - cliente não bloqueia
  - *semantica may-be ou possibly call*
  - RPC usando o protocolo R é chamado RPC assíncrono
  - ex.: sistema de janelas distribuído - ex.: X11
    - servidor mostra no display
    - clientes - programas de aplicação - mandam requests de display
    - clientes mandam vários requests para mostrar itens

Comunicação entre processos (RPC)

- Implementação de RPC - protocolos de comunicação
- Request protocol - continuação
  - RPCRuntime não se responsabiliza por reenviar pedido em caso de falha
  - se protocolo não confiável (UDP) for utilizado, mensagem pode ser perdida
  - aplicações optando por RPC assíncrono com protocolo não confiável tem que prever possibilidade de perda/erro;
    - pode ser usado em casos de serviços com atualização periódica
    - ex.: envio de sinal de relógio para sincronização, de tempos em tempos
  - uso de TCP resolve problema de confiabilidade

Comunicação entre processos (RPC)

- Implementação de RPC - protocolos de comunicação
- Request / Reply protocol - RR
  - para sistema envolvendo RPCs *simples*
    - todos argumentos e as respostas cabem em um “packet buffer”
  - duração da chamada (serviço) é pequena
  - intervalo entre chamadas é pequeno
  - protocolo usa confirmação implícita no reply - elimina mensagens adicionais

  - uso de técnicas de time-out e retransmissões
  - *se duplicatas não forem filtradas provê semantica at least once*

Comunicação entre processos (RPC)

- Implementação de RPC - protocolos de comunicação
- Request / Reply protocol - RR
  - *servidores podem suportar semantica exactely-once*
    mantendo registros em caches e filtrando requests e mandando replys novamente sem ter que processar o request novamente
  - 2 mensagens por call

Comunicação entre processos (RPC)

- Implementação de RPC - protocolos de comunicação
- Request / Reply / Acknowledge-Reply protocol - RRA
  - exactely-once semantics com RR - necessidade de cache
  - muitos clientes, registros saem por tempo, muita cache, limites de recursos
  - limitar este espaco com o RR pode levar a erro - registro sai antes do necessário
  - RRA - melhoria do RR
  - clientes tem que confirmar recepção
  - servidor pode retirar informação da cache depois da confirmação da recepção
  - 3 mensagens por call

Comunicação entre processos (RPC)

- Criação do servidor
  - instância por chamada
    - o processo servidor é criado pelo RPCRuntime quando de uma chamada do serviço, ao acabar, processo morre
    - stateless
    - ruim para várias chamadas: overhead
  - instância por sessão
    - existem durante várias interações de cliente e servidor
    - cliente contacta binding agent para descobrir gerente do serviço
    - cliente contata gerente do serviço e pede instância do serviço para uma sessão
    - cliente usa instância do serviço de maneira exclusiva
    - pode manter informação de estado entre chamadas

Comunicação entre processos (RPC)

- Criação do servidor
  - servidor persistente
    - existe todo o tempo
    - pode ser compartilhado por vários clientes
    - pode entrelaçar (“interleave”), executar concorrentemente, vários requests de vários clientes

Comunicação entre processos (RPC)

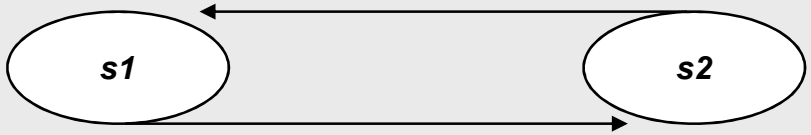
- Ligação entre Cliente e Servidor
  - localização do servidor
    - broadcast
      - mensagem para localizar servidor é enviada em broadcast
      - nodos onde existe tal servidor respondem
      - fácil de implementar
      - bom para redes pequenas
    - binding agent
      - bom para redes maiores
      - um servidor de nomes que informa localização do serviço procurado
      - operações: registrar, desregistrar, *lookup*

Comunicação entre processos (RPC)

- Broadcast RPC
  - uma chamada de um cliente é enviada em broadcast e processada por vários servidores
  - applicações paralelas
  - usando binding agent: cliente passa request ao binding agent informando ser broadcast, binding agent repassa request para os servidores cadastrados
  - usando broadcast ports: clientes ligam-se a porta broadcast; clientes mandam para a broadcast port, que manda para todos os nodos (mesma estrutura para multicast)

Comunicação entre processos (RPC)

- Situações de Deadlock
  - ocorre quando existe necessidade de um servidor se comunicar com um outro, em um RPC síncrono



Comunicação entre processos (RPC)

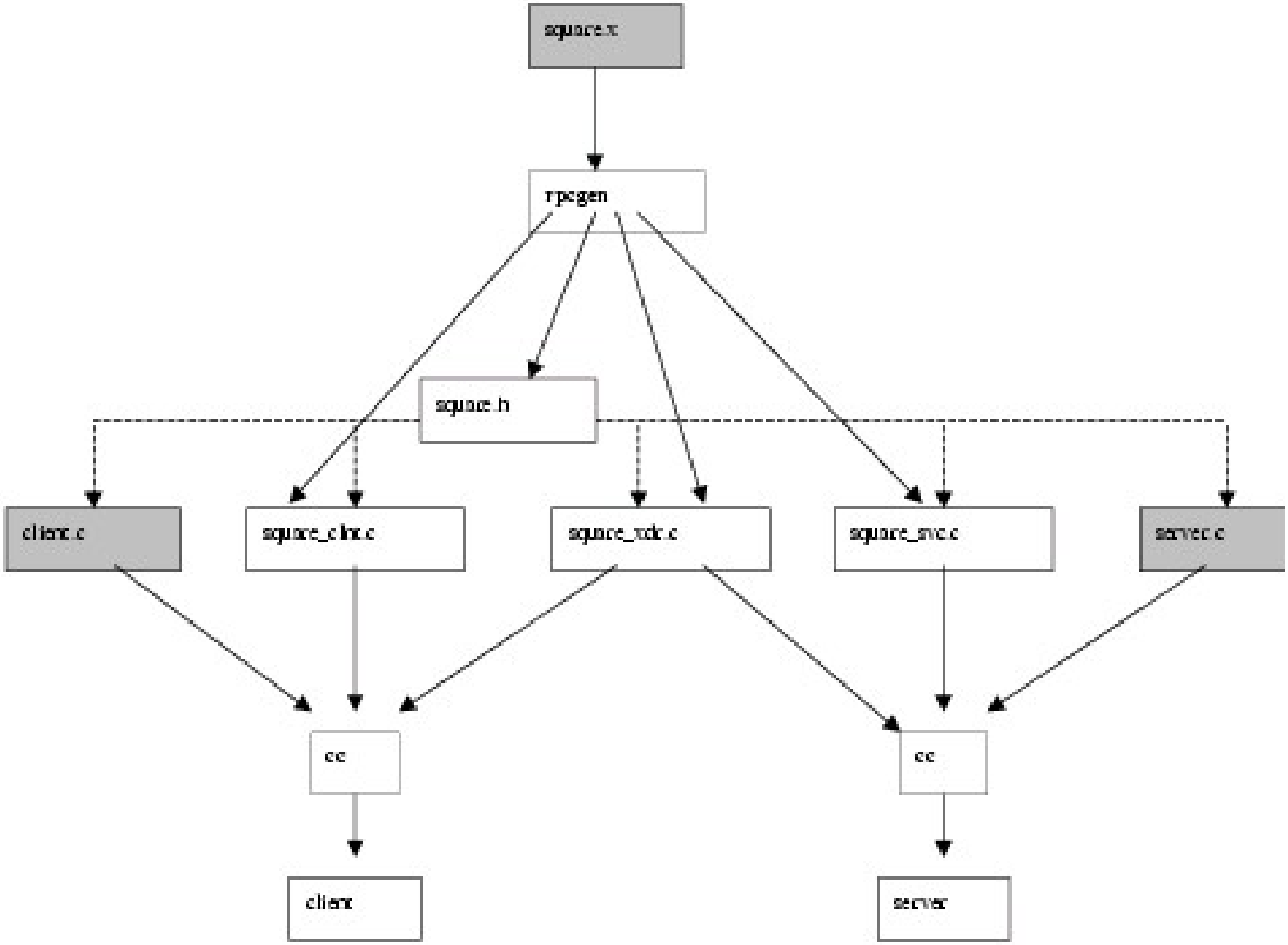
- Deadlock - Soluções:
  - a) servidores não se comunicam
  - b) servidores são classificados em camadas e a comunicação só é permitida se o servidor quer se comunicar com um outro de uma camada inferior
  - c) para cada requisição recebida, o servidor cria uma thread para executá-la

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
  - Desen-volvimento



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)
      - versao TI-RPC da sun soft - Transport Independent
    - em UDP, mensagens limitadas a 8 kbytes
    - semantica 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