1. Introdução

Dentro de computação distribuída, comunicação cliente-servidor é fundamental, tal que um problema recorrente com o desenvolvimento desses sistemas é a compatibilidade de servidores com diferentes clientes. Dessa demanda nasce o RPC, que através de uma mesma ferramenta: rpcgen, padroniza a implementação resolvendo muitos dos problemas da comunicação interprocessos.

2. Objetivo

O objetivo desse experimento é entender a arquitetura RPC(Remote Procedure Call's), considerado como um dos pilares para implementação de sistemas distribuídos.

3. Ambiente e configuração

Todos os experimentos abaixo foram executados numa distribuição linux chamada kubuntu, que difere fundamentalmente em termos de interface com o ubuntu. Portanto, deve ser possivel executar os mesmos passos em qualquer versão do Ubuntu 14.04, bem como em um Debian 7.

Já na geração automática do código, foi utilizado o rpcgen, usando a tag `-a` para geração de todos arquivos. Para mais informações sobre o rpcgen basta usar `man rpcgen`.

Para a compilação foi utilizado o Makefile também gerado pelo rpcgen.

4. Desenvolvimento

- Inicialmente foi criado um arquivo IDF(Interface Definition File) calcula.x com o seguinte conteúdo:

```
struct operandos {
    int x;
    int y;
};

program PROG {
    version VERSAO {
        int add(operandos) = 1;
        int sub(operandos) = 2;
    } = 100;
} = 11111;
```

- Após a criação do arquivo, foi executado o comando `rpcgen -a calcula.x` tal que os seguintes arquivos foram gerados:
    - calcula_xdr.c
    - calcula_client.c
    - calcula_clnt.c
    - calcula.h
    - calcula_server.c
    - calcula_svc.c
    - calcula.x
    - Makefile.calcula (Renomeado)
- Ao realizar as devidas alterações em calcula_server.c e calcula_clnt.c, o programa foi compilado utilizando o Makefile, gerando dois executáveis: calcula_client e calcula_server.
- Foi executado calcula_server utilizando o seguinte comando: `$ ./calcula_server 192.168.0.25`
    - Onde o argumento é o IP da rede local
- Enquanto o programa do passo acima executava, foi executado `rpcinfo -p` tal que foi possível obter o seguinte resultado:

```
$ rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
```

```
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
     11111  100   udp  40622
     11111  100   tcp  42204
```

- As ultimas linhas em *program* e *vers* mostram que o servidor do experimento está em execução.
- Ainda com o servidor em execução, foi executado e recebido as seguintes linhas:

```
$ ./calcula_client 192.168.0.25 10 + 20
Result is 30
...
$ ./calcula_client 192.168.0.25 50 - 25
Result is 25
...
$ ./calcula_client 192.168.0.25 10 / 2
Operador inválido
Falha de segmentação (imagem do núcleo gravada)
```

- O último comando mostra o limite da aplicação para a arquitetura auto gerada a partir do arquivo calcula.x, que só suporta soma e subtração. É possivel alterar manualmente o código para suportar mais operadores, porém seria algo custoso em vista da quantidade de código gerada. Ao invés disso basta criar um novo arquivo IDF (localizada no second_version):

```
struct operandos {
    int x;
    int y;
};

program PROG {
    version VERSAO {
        int add(operandos) = 1;
        int sub(operandos) = 2;
        int mul(operandos) = 3;
        int div(operandos) = 4;
    } = 110;
} = 22222;
```

- Seguindo o formato da implementação do primeiro programa, foi adicionado novas condições ao switch case de cliente:

```
// função prog_110 de calcula_client
case '*':
    result = mul_110(&operandos_arg, clnt);
    break;
case '/':
    result = div_110(&operandos_arg, clnt);
    break;
```

- No server apenas duas novas funções:

```
int * div_110_svc(operandos * argp, struct svc_req * rqstp)
...
int * mul_110_svc(operandos * argp, struct svc_req * rqstp)
...
```

- A execução do novo cliente/servidor agora retornou:

```
$ ./calcula_client 192.168.0.25 4 / 2

Result is 2
```

4.1 Problemas encontrados

O uso do RPC não apresentou nenhum problema.

4.2 Limitações de código

O segundo programa, presente na pasta *second_version* só aceita números maiores que 0 e realiza operações de soma, subtração, divisão e multiplicação, apesar de multiplicação ter de ser passado entre aspas, pois o terminal interpreta "*" como regex para tudo.