**RPC в LINUX**

Для работы с библиотекой RPC необходимо:

**1. Установить необходимые пакеты. Для этого в дистрибутиве Arch Linux в командной строке необходимо ввести команду *sudo pacman -S libtirpc rpcbind.* Данная команда устанавливает необходимые для работы с RPC библиотеки и самая главная из них это – tirpc (первая строка рис. 1). В других дистрибутивах имена пакетов могут отличаться, найти их можно путем поиска в репозитории. Для этого необходимо найти информацию о данных пакетах для своего дистрибутива в поисковике.**

*rpcbind (Remote Procedure Call bind) – является механизмом, при котором порты интернет-адресов могут быть назначены программе, запущенной на удаленном компьютере, чтобы действовать так, как если бы она была запущена на локальном компьютере.*



рис 1.

**2. Установить *rpcgen*. В дистрибутиве Arch Linux компилятор rpcgen находится в пакете *rpcsvc-proto,* а не пакете *rpcgen*, как написано в большинстве источников.**

**Ниже, на рис 2. показан скриншот, содержащий возможные команды установки данного пакета в различных дистрибутивах. Если установка не удалась по причине того, что в репозитории нет пакета с таким названием, необходимо выполнить поиск по репозиторию дистрибутива, установленного на компьютере.**



рис 2.

в дистрибутиве Arch Linux (версия ядра 6.0.2) rpcgen устанавливается командой (первая строка рис 3.):



рис 3.

**3. Создать файл *calculator.x* (листинг 1):**

Листинг 1.

/\*

 \* filename: calculator.x

 \* function: Define constants, non-standard data types and the calling process in remote calls

 \*/

const ADD = 0;

const SUB = 1;

const MUL = 2;

const DIV = 3;

struct CALCULATOR

{

 int op;

 float arg1;

 float arg2;

 float result;

};

program CALCULATOR\_PROG

{

 version CALCULATOR\_VER

 {

 struct CALCULATOR CALCULATOR\_PROC(struct CALCULATOR) = 1;

 } = 1; /\* Version number = 1 \*/

} = 0x20000001; /\* RPC program number \*/

**4. Сгенерировать следующие файлы, используя компилятор rpcgen:**

rpcgen -a calculator.x

Makefile.calculator: используется для компиляции кода клиента и кода сервера;

calculator.h: объявление используемых переменных и функций;

calculator\_xdr.c: “кодировка” нестандартных типов данных;

calculator\_clnt.c: прокси-сервер удаленного вызова вызывается локальной операционной системой, параметры вызова упакованы в сообщение. Далее сообщение посылается на сервер. Данный файл генерируется с помощью *rpcgen* и не нуждается в модификации;

calculator\_svc.c: преобразует запрос, вводимый по сети, в вызов локальной процедуры. Файл ответственнен за распаковку полученного сервером сообщения и вызов фактической реализации на стороне сервера и на уровне приложения. Не нуждается в модификации;

calculator\_client.c: скелетон программы клиента, требуется модифицировать;

calculator\_server.c: скелетон программы сервера, требуется модифицировать;

Исходные коды скелетонов:

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "calculator.h"

#include <stdio.h>

void

calculator\_prog\_1(char \*host)

{

 CLIENT \*clnt;

 struct CALCULATOR \*result\_1;

 struct CALCULATOR calculator\_proc\_1\_arg;

#ifndef DEBUG

 clnt = clnt\_create (host, CALCULATOR\_PROG, CALCULATOR\_VER, "udp");

 if (clnt == NULL) {

 clnt\_pcreateerror (host);

 exit (1);

 }

#endif /\* DEBUG \*/

 result\_1 = calculator\_proc\_1(&calculator\_proc\_1\_arg, clnt);

 if (result\_1 == (struct CALCULATOR \*) NULL) {

 clnt\_perror (clnt, "call failed");

 }

#ifndef DEBUG

 clnt\_destroy (clnt);

#endif /\* DEBUG \*/

} // calculator\_prog\_1

int

main (int argc, char \*argv[])

{

 char \*host;

 if (argc < 2) {

 printf ("usage: %s server\_host\n", argv[0]);

 exit (1);

 }

 host = argv[1];

 calculator\_prog\_1 (host); // вызов

exit (0);

}

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "calculator.h"

struct CALCULATOR \*

calculator\_proc\_1\_svc(struct CALCULATOR \*argp, struct svc\_req \*rqstp)

{

 static struct CALCULATOR result;

 /\*

 \* insert server code here

 \*/

 return &result;

}

**5. Модификация скелетонов клиента и сервера:**

***calculator\_client.c:***

Листинг 2.

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "calculator.h"

#include <stdio.h>

void

calculator\_prog\_1(char \*host)

{

 CLIENT \*clnt;

 struct CALCULATOR \*result\_1;

 struct CALCULATOR calculator\_proc\_1\_arg;

#ifndef DEBUG

 clnt = clnt\_create (host, CALCULATOR\_PROG, CALCULATOR\_VER, "udp");

 if (clnt == NULL) {

 clnt\_pcreateerror (host);

 exit (1);

 }

#endif /\* DEBUG \*/

 /\* -<<< Add to test \*/

 char c;

 printf("choose the operation:\n\t0---ADD\n\t1---SUB\n\t2---MUL\n\t3---DIV\n");

 c = getchar();

 if (c > '3' || c < '0')

 {

 printf("error:operate\n");

 exit(1);

 }

 calculator\_proc\_1\_arg.op = c-'0';

 printf("input the first number: ");

 scanf("%f", &calculator\_proc\_1\_arg.arg1);

 printf("input the second number:");

 scanf("%f", &calculator\_proc\_1\_arg.arg2);

 /\* -<<< Add to test \*/

 result\_1 = calculator\_proc\_1(&calculator\_proc\_1\_arg, clnt);

 if (result\_1 == (struct CALCULATOR \*) NULL) {

 clnt\_perror (clnt, "call failed");

 }

#ifndef DEBUG

 clnt\_destroy (clnt);

#endif /\* DEBUG \*/

/\* -<<< Add to test \*/

printf("The Result is %.3f\n", result\_1->result);

/\* -<<< Add to test \*/

} // calculator\_prog\_1

int

main (int argc, char \*argv[])

{

 char \*host;

 if (argc < 2) {

 printf ("usage: %s server\_host\n", argv[0]);

 exit (1);

 }

 host = argv[1];

 calculator\_prog\_1 (host); // вызов

exit (0);

}

**calculator\_server.c**

Листинг 3.

/\*

 \* This is sample code generated by rpcgen.

 \* These are only templates and you can use them

 \* as a guideline for developing your own functions.

 \*/

#include "calculator.h"

struct CALCULATOR \*

calculator\_proc\_1\_svc(struct CALCULATOR \*argp, struct svc\_req \*rqstp)

{

 static struct CALCULATOR result;

 /\*

 \* insert server code here

 \*/

 /\* -<<< Add to test \*/

 switch(argp->op)

 {

 case ADD:

 result.result = argp->arg1 + argp->arg2;

 break;

 case SUB:

 result.result = argp->arg1 - argp->arg2;

 break;

 case MUL:

 result.result = argp->arg1 \* argp->arg2;

 break;

 case DIV:

 result.result = argp->arg1 / argp->arg2;

 break;

 default:

 break;

 }

 /\* -<<< Add to test \*/

 return &result;

}

6. Выполнить сборку проекта с помощью Makefile.calculator, для этого необходимо в командной строке выполнить следующую команду: make -f Makefile.calculator. Для корректной сборки необходимо предварительно изменить файл Makefile.calculator, добавив флаг компиляции **-I/usr/include/tirpc** и флаг линковки **-ltirpc**. Листинг данного файла сборки представлен ниже.

Листинг 4.

# This is a template Makefile generated by rpcgen

# Parameters

CLIENT = calculator\_client

SERVER = calculator\_server

SOURCES\_CLNT.c =

SOURCES\_CLNT.h =

SOURCES\_SVC.c =

SOURCES\_SVC.h =

SOURCES.x = calculator.x

TARGETS\_SVC.c = calculator\_svc.c calculator\_server.c calculator\_xdr.c

TARGETS\_CLNT.c = calculator\_clnt.c calculator\_client.c calculator\_xdr.c

TARGETS = calculator.h calculator\_xdr.c calculator\_clnt.c calculator\_svc.c calculator\_client.c calculator\_server.c

OBJECTS\_CLNT = $(SOURCES\_CLNT.c:%.c=%.o) $(TARGETS\_CLNT.c:%.c=%.o)

OBJECTS\_SVC = $(SOURCES\_SVC.c:%.c=%.o) $(TARGETS\_SVC.c:%.c=%.o)

# Compiler flags

**CFLAGS += -g -I/usr/include/tirpc**

**LDLIBS += -lnsl -ltirpc**

RPCGENFLAGS =

# Targets

all : $(CLIENT) $(SERVER)

$(TARGETS) : $(SOURCES.x)

 rpcgen $(RPCGENFLAGS) $(SOURCES.x)

$(OBJECTS\_CLNT) : $(SOURCES\_CLNT.c) $(SOURCES\_CLNT.h) $(TARGETS\_CLNT.c)

$(OBJECTS\_SVC) : $(SOURCES\_SVC.c) $(SOURCES\_SVC.h) $(TARGETS\_SVC.c)

$(CLIENT) : $(OBJECTS\_CLNT)

 $(LINK.c) -o $(CLIENT) $(OBJECTS\_CLNT) $(LDLIBS)

$(SERVER) : $(OBJECTS\_SVC)

 $(LINK.c) -o $(SERVER) $(OBJECTS\_SVC) $(LDLIBS)

 clean:

 $(RM) core $(TARGETS) $(OBJECTS\_CLNT) $(OBJECTS\_SVC) $(CLIENT) $(SERVER)

7. Запустить сервер командой ./calculator\_server. Затем запустить клиент, указав в качестве параметра ip-адрес хоста (localhost в случае выполнения на ПК), выполнив команду:

./calculator\_client 127.0.0.1

Пример работы программы представлен на рис. 4:



рис. 5

**\*Замечание: в случае если при запуске ./calculator\_server терминал печатает сообщение об ошибке вида “unable to register (TESTPROG, VERSION, udp).**

**необходимо перезапустить сервис rpcbind командой** *systemctl restart rpcbind*

*Заключение:*

*RPC являются средством взаимодействия параллельных процессов (IPC - Inter-process communication). Как известно, в самом общем случае различаются два вида взаимодействия параллельных процессов (IPC):*

 *1. Вызов локальных процедур (LPC – local procedure call)*

 *2. Вызов удаленных процедур (RPC – remote procedure call)*

*Вызов удаленных процедур сделан максимально подобным вызову локальных процедур. В ядре RPC не используют протокол, в чем и состоит принципиальное различие с сокетами. Действия фактически выполняются не по протоколу. Действия выполняются в сети путем связывания портов между собой.*

Задание на лабораторную работу

Необходимо реализовать алгоритм Лампорта “Булочная”: процессы клиенты обращаются к серверу для получения номера. Сервер каждому приходящему клиенту выдает номер – максимальный из выданных + единица.