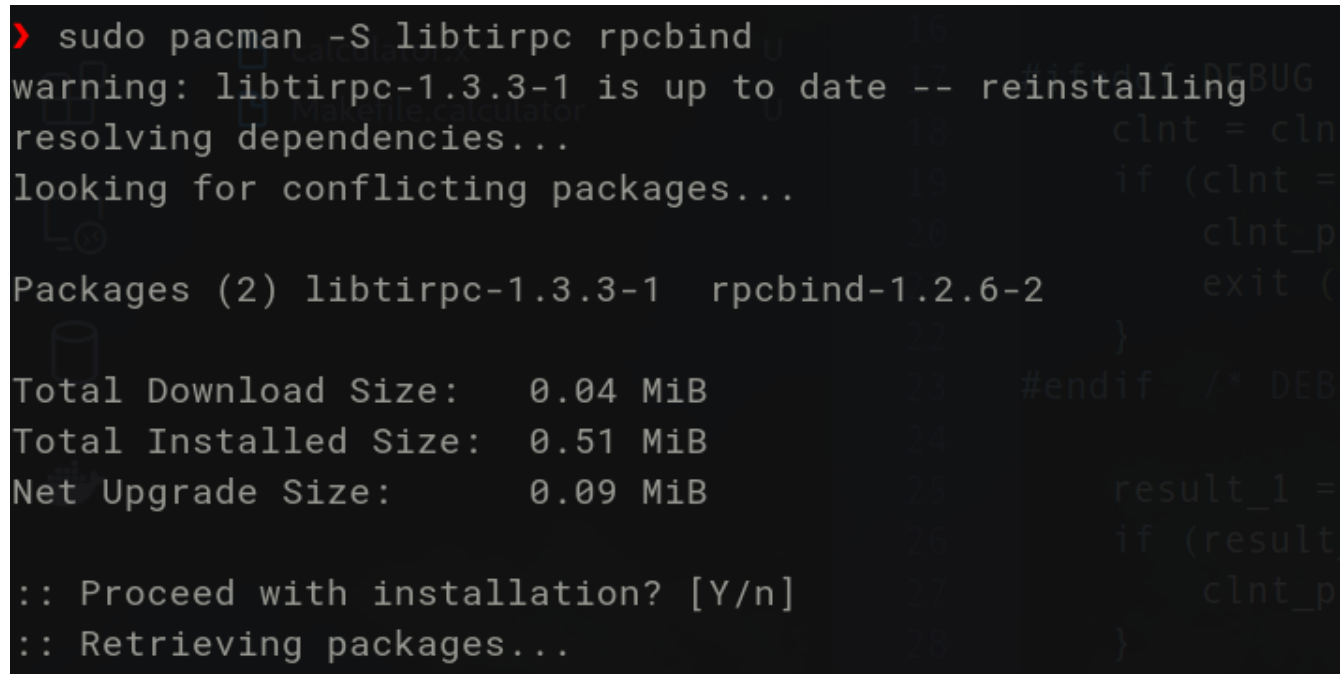


RPC в LINUX

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 – `libtirpc` (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.



```
> sudo pacman -S libtirpc rpcbind
warning: libtirpc-1.3.3-1 is up to date -- reinstalling
resolving dependencies...
looking for conflicting packages...

Packages (2) libtirpc-1.3.3-1  rpcbind-1.2.6-2

Total Download Size:    0.04 MiB
Total Installed Size:  0.51 MiB
Net Upgrade Size:       0.09 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
```

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.

Debian	<code>apt-get install libc-dev-bin</code>	libc-dev-bin
Ubuntu	<code>apt-get install libc-dev-bin</code>	GNU C Library: Development binaries This package contains utility programs related to the GNU C Library development package.
Alpine	<code>apk add rpcgen</code>	glibc-common
Arch Linux	<code>pacman -S rpcgen</code>	Common binaries and locale data for glibc
Kali Linux	<code>apt-get install libc-dev-bin</code>	rpcgen
CentOS	<code>yum install glibc-common</code>	Remote Procedure Call (RPC) protocol compiler
Fedora	<code>dnf install rpcgen</code>	glibc-arm-linux-gnu
Windows (WSL2)	<code>sudo apt-get update sudo apt-get install libc-dev-bin</code>	at arm-linux-gnu
OS X	<code>brew install rpcgen</code>	rpsecv-proto
Raspbian	<code>apt-get install libc-dev-bin</code>	rpsecv protocol definitions from glibc
Dockerfile	<code>dockerfile.run/rpcgen</code>	
Docker	<code>docker run cmd.cat/rpcgen rpcgen</code> powered by Commando	

рис 2.

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

```
> sudo pacman -S rpsecv-proto
resolving dependencies...
looking for conflicting packages...

Packages (1) rpsecv-proto-1.4.3-1

Total Download Size:    0.06 MiB
Total Installed Size:  0.21 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
```

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 ответственен 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;

#ifdef    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");
    }
#ifdef    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;

#ifdef  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");
    }
#ifdef      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:

```

1: Terminal
> ./calculator_server
SOURCES_CLNT.c =
SOURCES_CLNT.h =
SOURCES_SVC.c =
SOURCES_SVC.h =
SOURCES.x = calculator.x
TARGETS_SVC.c = calculator_svc
TARGETS_CLNT.c = calculator_cl
TARGETS = calculator.h calculator

2: Terminal
> ./calculator_client 127.0.0.1
choose the operation:
0---ADD
1---SUB
2---MUL
3---DIV
1
input the first number: 1
input the second number: 1
The Result is 0.000

```

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.