Настройка Debian Linux и выполнение примера RPC клиент-серверного приложения.[1]

Пример приложения: *вычисление на (удаленном) сервере среднего значения всех параметров камандной строки (до 200 чисел) клиента.*

План: *Результат выполнения должен быть следующий*:

1. *Запуск сервера (на локальном хосте)*

```
~$ ./server &
[1] 254
```

2. *Проверка работы сервиса (номер программы и версия задаются в XDR файле и определяют сгенерированный номер порта для UDP и TCP протоколов)*

```
~$ 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
     22855    1   udp  57312
     22855    1   tcp  60229
```

3. *Запуск клиента с RPC сервером на локальном хосте (localhost можно заменить на адрес 127.0.0.1) и 4 параметрами (случайные целые числа) для вычислений на сервере (до 200 значений)*

```
~$ ./client localhost $RANDOM $RANDOM $RANDOM $RANDOM
     value = 2.563500e+04
     value = 2.800600e+04
     value = 2.955800e+04
     value = 2.662800e+04
     average = 2.745675e+04
```

Повтор

```
~$ ./client localhost $RANDOM $RANDOM $RANDOM $RANDOM
     value = 2.687700e+04
     value = 2.476800e+04
     value = 2.983000e+04
     value = 1.345100e+04
     average = 2.373150e+04
~$
```

1 Данное упражнение выполнялось в Debian Linux под WSL Windows 10.

Решение.

Шаг 1. Проверка наличия сервиса *rpcbind.*

```
~$ which rpcbind
~$ rpcinfo
```

Шаг 2. Если шаг 1 безуспешный: Установка *portmap* (включающего сервис *rpcbind*).

```
~$ sudo apt install portmap
```

Шаг 3. Горячий (warm - без перезагрузки системы) запуск сервиса *rpcbind* (см. rpcbind(1M)) на серверном хосте (здесь — локальный *localhost*) .

```
~$ sudo /sbin/rpcbind -w
```

Шаг 4. Проверка работы сервиса.

```
~$ rpcinfo
```

Шаг 5. Создаем машинно-независимый XDR файл описания интерфейса RPC вызова.

```
~$ vi avg.x
см. файл avg.x
```

Шаг 6. Создаем клиентское приложение.

```
~$ vi client.c
см. файл client.c
```

Шаг 7. Создаем серверный RPC вызов.

```
~$ vi server.c
см. файл server.c
```

Шаг 8. Генерация skeleton (`avg_svc.c` - серверный код с функцией *main* для регистрации сервиса и RPC функции в реестре) и stub (`avg_clnt.c` - клиетнский код для реализации RPC вызова) для клиента и сервера с помощью *rpcgen* XDR компилятора

```
~$ ls
avg.x client.c server.c
~$ rpcgen avg.x
~$ ls
avg_clnt.c  avg.h  avg_svc.c  avg.x  avg_xdr.c  cient.c server.c
```

Шаг 9. Компиляция сервера и клиента.

```
~$ gcc server.c avg_svc.c avg_xdr.c -o server
~$ gcc client.c avg_clnt.c avg_xdr.c -o client
```

Шаг 10. Выполнение как планировалось в задании (см. План).

Ссылки.

1. *https://www.linuxjournal.com/article/2204*
2. *http://cobweb.cs.uga.edu/~maria/classes/4730-Fall-2016/slides/04-processes-RPC-2011.pdf*
3. http://cobweb.cs.uga.edu/~maria/classes/x730-Spring-2018/slides/04c-processes-RPC.pptx.pdf

См. Также Java RMI https://www.javatpoint.com/RMI *и CORBA* http://www.dre.vanderbilt.edu/~schmidt/corba-overview.html