Сергей Юрьевич Шилов

Системное программирование для современных платформ

Системное программирование для современных платформ

8. Remore Procedure Call (RPC)

Системное программирование для современных платформ

Набор технологий (фреймворк), позволяющий обращение к программному коду, исполняемому в чужом (и также, вероятно, удалённом) адресном пространстве. Обеспечивает представление данных в машинно-независимом формате, сереализацию и десериализацию этих данных.

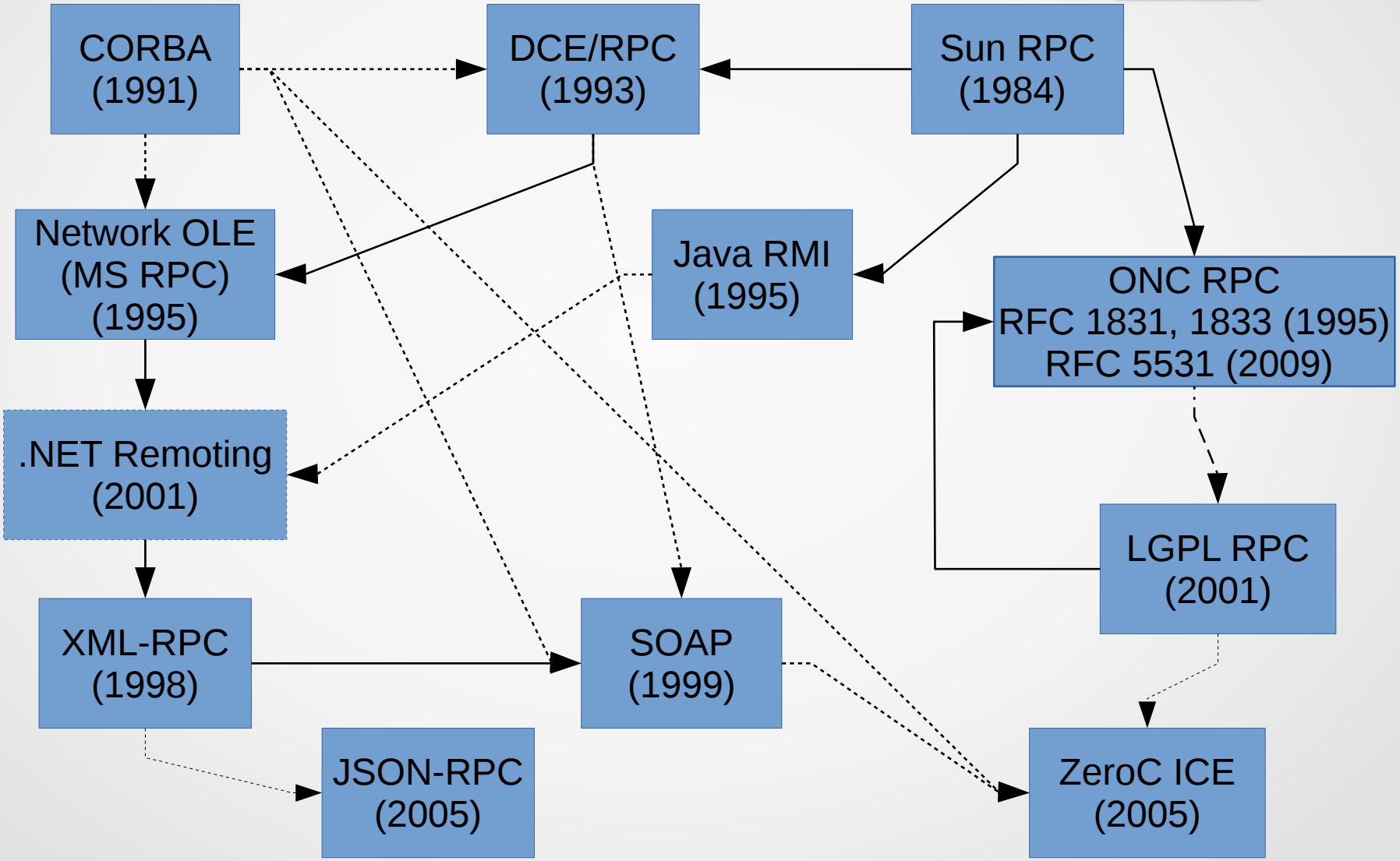
Могут быть основаны на различных соглашениях о вызовах – необъектные RPC (SUN RPC, ONC RPC) или реализовывать те или иные объектно-ориентированные модели (DCOM, CORBA, SOA)

Системное программирование для современных платформ

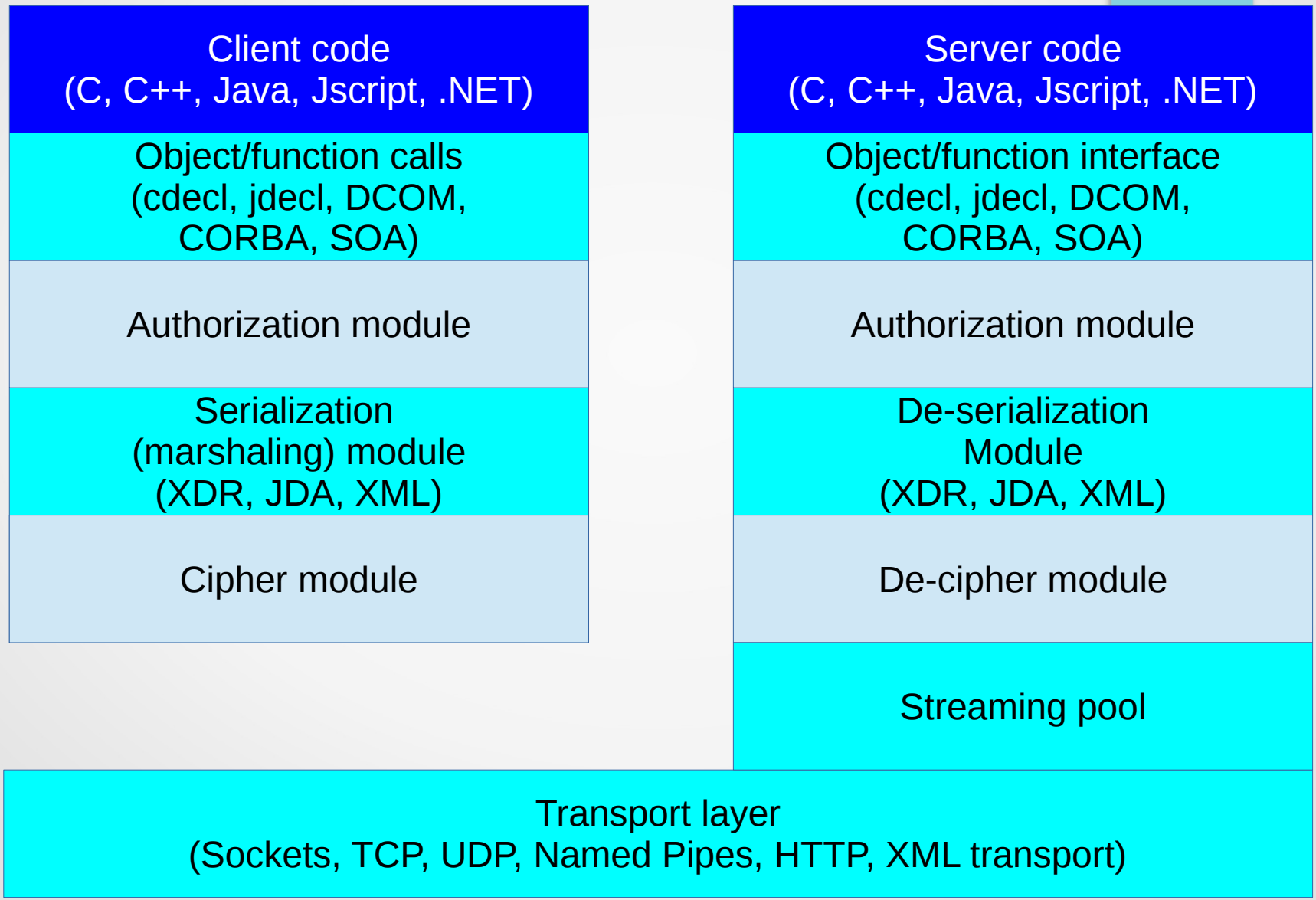
Особенности

- Архитектура клиент-сервис
- Асимметричность
- Синхронность
- Машинная независисмость
- Зависимость от языков (и даже среды) программирования

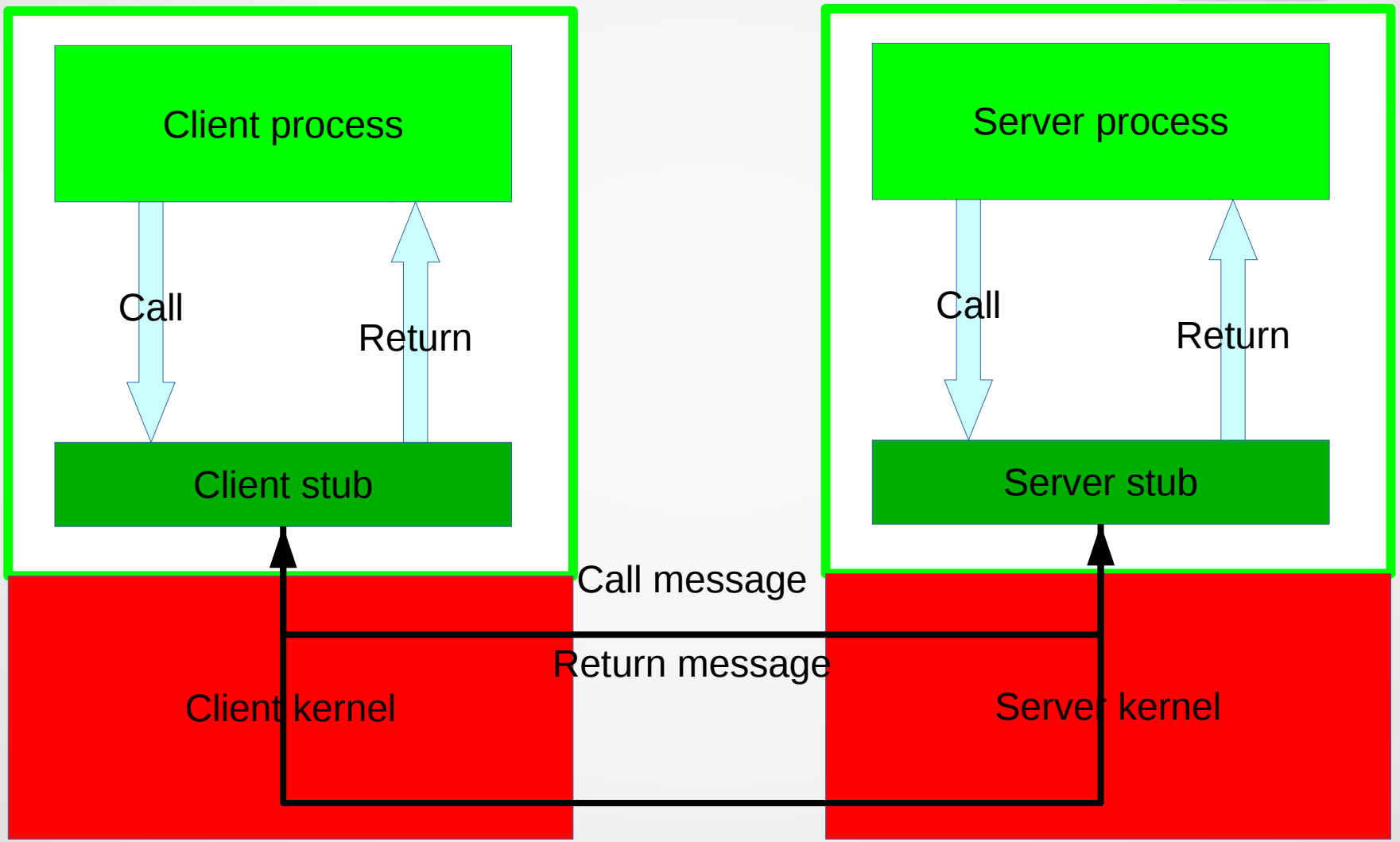
Системное программирование для современных платформ



Системное программирование для современных платформ



Системное программирование для современных платформ



Системное программирование для современных платформ

Системное программирование для современных платформ

Системное программирование для современных платформ

Level 1	Библиотечные RPC-функции (rusers, rdate, rwall, spray, rstat)
Level 2	RPC-компиллятор (препроцессор) RPCgen
Level 3	Low-level RPC programming

Системное программирование для современных платформ

XDR (External Data Representation)

- char, short, int, long (bigendian)
- uchar, usort, uint, ulong (bigendian)
- boolean (TRUE=1, FALSE=0)
- float, double (IEEE 754)
- enum
- string (pascal strings)
- union (pascal variants)
- opaque (v[n], v<n>)
- raw

Системное программирование для современных платформ

RPC preprocessor (RPCgen)

- Номер RPC-программы
- Номер RPC-версии
- Номер RPC-функции
- Описание передаваемых типов данных
- С-код удалённо-вызываемых функций (не обязательно)

Системное программирование для современных платформ

```
/* printmsg.c: вывод сообщения на консоль */
#include <stdio.h>
main(int argc, char *argv[]) {
   char *message;

   if (argc != 2) {
      fprintf(stderr, "usage: %s <message>\n",argv[0]);
      exit(1);
   }

   message = argv[1];
   if (!printmessage(message)) {
     fprintf(stderr,"%s: couldn't print your message\n",argv[0]);
     exit(1);
   }

   printf("Message Delivered!\n");
   exit(0);
}

/* Вывод сообщения на консоль
*  Возвращение стандартных кодов возврата,
*  после вывода сообщения на консоль       */

printmessage(char *msg) {
   FILE *f;
   f = fopen("/dev/console", "w");
   if (f == (FILE *)NULL) {return (0);}
   fprintf(f, "%s\n", msg);
   fclose(f);
   return(1);
}

$ gcc printmsg.c -o printmsg
$ printmsg "Hello, World!"
Hello, World!
Message delivered!
$
```

Системное программирование для современных платформ

```
/*
*  msg.x: Файл препроцессора RPC для удалённого вызова функции msg
*/

program MESSAGEPROG {
   version PRINTMESSAGEVERS {
     int PRINTMESSAGE(string) = 1;
   } = 1;
} = 0x20000001;
```

Удалённый вызов процедуры всегда описывается как часть программы-сервера. Файл препроцессора описывает программу-сервер, как содержащую единственную функцию PRINTMESSAGE, принимающую аргумент типа string и возвращающая значение типа int.

Здесь:

Функция **PRINTMESSAGE** описывается как
Процедура **1** версии **1**
Программы-сервера
MESSAGEPROG с номером **0x20000001** (**0x00000000-0x1fffffff – system applications; 0x20000000-0x3fffffff – user applications; 0x40000000-0x5fffffff – customer applications; 0x60000000-0xffffffff – reserved for future use**)

```
$ rpcgen msg.x
$ ls
msg.h msg.x msg_clnt.c msg_svc.c
```

Системное программирование для современных платформ

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

#ifndef _MSG_H_RPCGEN
#define _MSG_H_RPCGEN

#include <rpc/rpc.h>

#define MESSAGEPROG ((u_long)0x20000001)
#define PRINTMESSAGEVERSION ((u_long)1)

#ifdef __cplusplus
#define PRINTMESSAGE ((u_long)1)
extern "C" int * printmessage_1(char **, CLIENT *);
extern "C" int * printmessage_1_svc(char **, struct svc_req *);

#elif __STDC__
#define PRINTMESSAGE ((u_long)1)
extern  int * printmessage_1(char **, CLIENT *);
extern  int * printmessage_1_svc(char **, struct svc_req *);

#else /* Old Style C */
#define PRINTMESSAGE ((u_long)1)
extern  int * printmessage_1();
extern  int * printmessage_1_svc();
#endif /* Old Style C */

#endif /* !_MSG_H_RPCGEN */
```

Системное программирование для современных платформ

```
/*
 * msg_proc.c - RPC Version of printmsg.c (server side)
 *
 */

#include <stdio.h>
#include "msg.h"

/*  Print a message on the console */
int * printmessage_1_svc( char ** msg, struct svc_req * req)
{
    FILE *f;
    static int result;

    f = fopen("/dev/console","w");

    if (f == (FILE *) NULL) {
    result = 0;
    return(&result);
    }

    fprintf(f,"%s\n",*msg);
    fclose(f);
    result = 1;
    return(&result);
}
```

```c
/*
 * rprintmsg.c - RPC version of "printmsg.c" Client side
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include "msg.h"

main(int argc, char** argv){
    CLIENT* client;
    int* result;
    char* server; * message;

    if (argc != 3) {fprintf(stderr, "Usage : %s <host> <message>\n",argv[0]);exit(1);}
    server = argv[1];
    message = argv[2];

    /* Create the client handle used for calling MESSAGEPROG on the server */

    client = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERSION, "tcp");

    /* Check if connection is established to the server */
    if (client == NULL) {
      clnt_pcreateerror(server);
      exit(1);
    }

    /* Call the remote procedure */
    result = printmessage_1(&message, client);
    if (result == (int *) NULL) {
      /* Error occurred while calling the server */
      clnt_perror(client, server);
      exit(1);
    }
    /* RPC worked; check if the message was delivered */
    if (*result == 0) {
      fprintf(stderr,"%s : Coult not print your message\n",argv[0]);
      exit(1);
    };
    /* Everything (RPC, message delivery) worked */
    printf("Message delivered to %s\n",server);
    clnt_destroy(client);
    exit(0);
}
```

Системное программирование для современных платформ

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

#include <memory.h> /* for memset */
#include "msg.h"

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

int* printmessage_1(char **argp, CLIENT *clnt)
{
    static int clnt_res;

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

Системное программирование для современных платформ

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

#include "msg.h"
#include <stdio.h>
#include <stdlib.h>/* getenv, exit */
#include <rpc/pmap_clnt.h> /* for pmap_unset */
#include <string.h> /* strcmp */
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifdef __STDC__
#define SIG_PF void(*)(int)
#endif
```

Системное программирование для современных платформ

```
static void messageprog_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
      union { char *printmessage_1_arg;} argument;
      char *result;
      xdrproc_t xdr_argument, xdr_result;
      char *(*local)(char *, struct svc_req *);

      switch (rqstp->rq_proc) {
      case NULLPROC:
            (void) svc_sendreply(transp, (xdrproc_t) xdr_void, (char *)NULL);
            return;

      case PRINTMESSAGE:
            xdr_argument = (xdrproc_t) xdr_wrapstring;
            xdr_result = (xdrproc_t) xdr_int;
            local = (char *(*)(char *, struct svc_req *)) printmessage_1_svc;
            break;

      default:
            svcerr_noproc(transp);
            return;
      }

      (void) memset((char *)&argument, 0, sizeof (argument));
      if (!svc_getargs(transp, xdr_argument, (caddr_t) &argument)) {
            svcerr_decode(transp);
            return;
      }
      result = (*local)((char *)&argument, rqstp);
      if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
            svcerr_systemerr(transp);
      }
      if (!svc_freeargs(transp, xdr_argument, (caddr_t) &argument)) {
            fprintf(stderr, "unable to free arguments");
            exit(1);
      }
      return;
}
```

Системное программирование для современных платформ

```
int main(int argc, char **argv)
{
     register SVCXPRT *transp;

     (void) pmap_unset(MESSAGEPROG, PRINTMESSAGEVERSION);

     transp = svcudp_create(RPC_ANYSOCK);
     if (transp == NULL) {
          fprintf(stderr, "cannot create udp service.");
          exit(1);
     }
     if (!svc_register(transp, MESSAGEPROG, PRINTMESSAGEVERSION, messageprog_1, IPPROTO_UDP)) {
          fprintf(stderr, "unable to register (MESSAGEPROG, PRINTMESSAGEVERSION, udp).");
          exit(1);
     }

     transp = svctcp_create(RPC_ANYSOCK, 0, 0);
     if (transp == NULL) {
          fprintf(stderr, "cannot create tcp service.");
          exit(1);
     }
     if (!svc_register(transp, MESSAGEPROG, PRINTMESSAGEVERSION, messageprog_1, IPPROTO_TCP)) {
          fprintf(stderr, "unable to register (MESSAGEPROG, PRINTMESSAGEVERSION, tcp).");
          exit(1);
     }

     svc_run();
     fprintf(stderr, "svc_run returned");
     exit(1);
     /* NOTREACHED */
}
```

Системное программирование для современных платформ

Спасибо за внимание ;)