8° Esercitazione (svolta):

Remote Procedure Call

Esercizio 1:

Sviluppare un'applicazione C/S che consente di effettuare la **somma tra due interi in remoto**:

- il client viene invocato da linea di comando con due argomenti, i.e., i due interi che si vogliono sommare, esegue la chiamata alla procedura remota ***somma*** passandoli come parametri gli presi, e stampa il risultato dell'operazione ottenuto come valore di ritorno della procedura;

- il server esegue la procedura che effettua la somma tra i due parametri e restituisce il risultato dell'operazione al client.

Variante: realizzare il **client ciclico**, che chiede operandi all'utente da console fino a fine file.

File somma.x

```
struct Operandi{
    int op1;
    int op2;
};


program SOMMAPROG {
    version SOMMAVERS {
        int SOMMA(Operandi) = 1;
    } = 1;
} = 0x20000013;
```

Compilazione per generare il file header, il file per le conversioni xdr e gli stub:

rpcgen somma.x

=> produce i file:

- *somma.h*
  da includere in *somma_proc.c* e in *somma_client.c*
- *somma_xdr.c* che contiene le routine di conversione xdr
- *somma_clnt.c* stub del client
- *somma_svc.c* stub del server

File somma.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */
#ifndef _SOMMA_H_RPCGEN
#define _SOMMA_H_RPCGEN

#include <rpc/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

struct Operandi {
    int op1;
    int op2;
};

typedef struct Operandi Operandi;

#define SOMMAPROG 0x20000013
#define SOMMAVERS 1
/*ANSI C*/
#if defined(__STDC__)||defined(__cplusplus)
#define SOMMA 1
extern int * somma_1(Operandi *, CLIENT *);
extern int * somma_1_svc(Operandi *,struct
svc_req*);
extern int sommaprog_1_freeresult (SVCXPRT *,
  xdrproc_t, caddr_t);
/* K&R C */
#else
#define SOMMA 1
extern  int * somma_1();
extern  int * somma_1_svc();
extern int sommaprog_1_freeresult ();
#endif /* K&R C */
```



```
/* the xdr functions */

#if defined(__STDC__) || defined(__cplusplus)
extern  bool_t xdr_Operandi (XDR *, Operandi*);

#else /* K&R C */
extern bool_t xdr_Operandi ();

#endif /* K&R C */

#ifdef __cplusplus
}
#endif

#endif /* !_SOMMA_H_RPCGEN */
```

File somma_xdr.c: routine di conversione

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

#include "somma.h"

bool_t
xdr_Operandi (XDR *xdrs, Operandi *objp)
{
    register int32_t *buf;

     if (!xdr_int (xdrs, &objp->op1))
         return FALSE;
     if (!xdr_int (xdrs, &objp->op2))
         return FALSE;
    return TRUE;
}
```

File somma_clnt.c: stub del client

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


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


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


int *
somma_1(Operandi *argp, CLIENT *clnt)
{
    static int clnt_res;

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


File somma_svc.c: stub del server

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

#include "somma.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void
sommaprog_1(struct svc_req *rqstp, register SVCXPRT
*transp)
{
    union {
        Operandi somma_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 SOMMA:
        _xdr_argument = (xdrproc_t) xdr_Operandi;
        _xdr_result = (xdrproc_t) xdr_int;

        local = ( char *(*)
            (char *, struct svc_req *))somma_1_svc;
        break;

    default:
        svcerr_noproc (transp);
        return;
    }
    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, "%s",
                    "unable to free arguments");
        exit (1);
    }
    return;
}
```



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

  pmap_unset (SOMMAPROG, SOMMAVERS);

  transp = svcudp_create(RPC_ANYSOCK);
  if (transp == NULL) {
    fprintf (stderr, "%s",
                    "cannot create udp service.");
    exit(1);
  }
  if (!svc_register(transp, SOMMAPROG, SOMMAVERS,
                        sommaprog_1, IPPROTO_UDP)) {
    fprintf (stderr, "%s", "unable to register
                    (SOMMAPROG, SOMMAVERS, udp).");
    exit(1);
  }

  transp = svctcp_create(RPC_ANYSOCK, 0, 0);

  if (transp == NULL) {
    fprintf (stderr, "%s",
                    "cannot create tcp service.");
    exit(1);
  }
  if (!svc_register(transp, SOMMAPROG, SOMMAVERS,
                    sommaprog_1, IPPROTO_TCP)) {
    fprintf (stderr, "%s", "unable to register
                    (SOMMAPROG, SOMMAVERS, tcp).");
    exit(1);
  }

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

File somma_proc.c: implementazione della procedura

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "somma.h"

int *somma_1_svc(Operandi *op, struct svc_req *rp)
{
  static int ris;

  printf("Operandi ricevuti: %i e %i\n",
                                op->op1, op->op2);
  ris = (op->op1) + (op->op2);
  printf("Somma: %i\n", ris);
  return (&ris);
}
```

File somma_client.c: implementazione del client che chiama la procedura remota

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "somma.h"

main(int argc, char *argv[]){
/* gestore di protocollo: deve essere un puntatore,
   come nel codice scaricabile da rete */
  CLIENT *cl;
  int *ris;
  char *server;
  Operandi op;

  if (argc < 3) {
    fprintf(stderr, "uso: %s host op1 op2\n",
        argv[0]);
    exit(1);
  }
```



```
  server = argv[1];
  op.op1 = atoi(argv[2]);
  op.op2 = atoi(argv[3]);
  cl = clnt_create(
    server, SOMMAPROG, SOMMAVERS, "udp");
  if (cl == NULL) {
    clnt_pcreateerror(server);
    exit(1);
  }

  ris = somma_1(&op, cl);
  if (ris == NULL) {
    clnt_perror(cl, server);
    exit(1);
  }
  if (*ris == NULL) {
    fprintf(stderr, "%s: %s non riesce ad eseguire
        la somma\n", argv[0], server);
    exit(1);
  }
  printf("Risultato da %s: %i\n", server, *ris);
}
```

Compilazione

Compilazione per generare l'eseguibile del client:

gcc somma_client.c somma_clnt.c somma_xdr.c -o somma

=> produce il comando *somma*

Compilazione per generare l'eseguibile del server:

gcc somma_proc.c somma_svc.c somma_xdr.c -o somma_server

=> produce il comando *somma_server*

Esecuzione

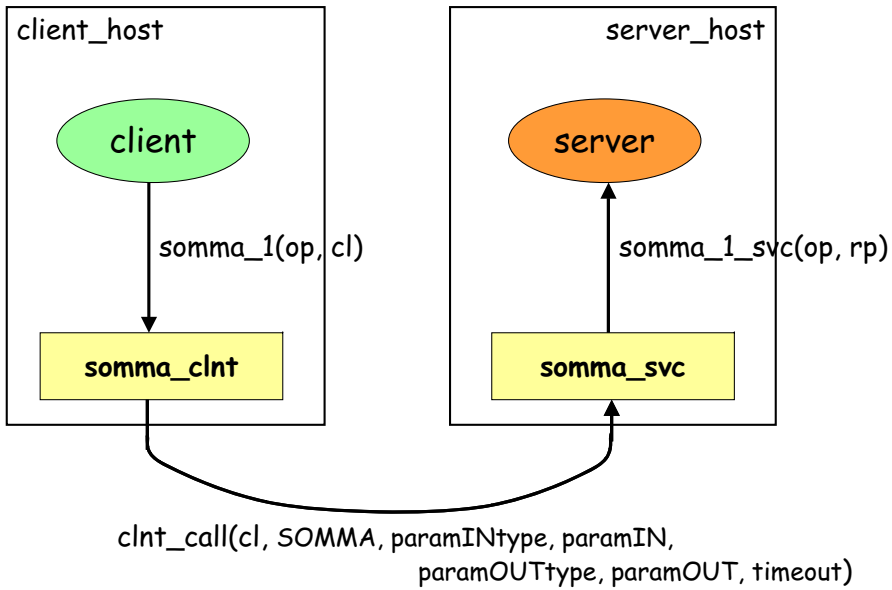
Mandare in esecuzione il server con il comando

somma_server

Mandare in esecuzione il client con il comando

somma serverhost primo_operando secondo_operando

N.B.: nella variante il client viene lanciato con un unico argomento, serverhost, gli operandi vengono recuperati interattivamente durante il ciclo d'esecuzione.



Esercizio 2:

Sviluppare un'applicazione C/S che consente di ottenere l'echo di una stringa invocando una procedura remota:

- il client realizza l'interazione con l'utente richiedendo una stringa, invoca la procedura remota ***echo*** passando come parametro la stringa letta, e stampa a video la stringa ottenuta come valore di ritorno dell'operazione invocata; si ripetono queste tre operazioni fino alla fine dell'interazione con l'utente;

- il server realizza il servizio di echo che restituisce come risultato la stringa passata come parametro di ingresso.

Variante: realizzare il **client non ciclico**, che prende la stringa come argomento da linea di comando.

File echo.x

```
program ECHOPROG {
    version ECHOVERS {
        string ECHO(string) = 1;
    } = 1;
} = 0x20000013;
```

Compilazione per generare il file header e gli stub:

rpcgen echo.x

=> produce i file:

- *echo.h*
  da includere in *echo_proc.c* e in *echo_client.c*
- *echo_clnt.c* stub del client
- *echo_svc.c* stub del server

Nota bene:
Uso di **string**, tipo predefinito di xdr. Nei file .c però si deve usare l'equivalente C, cioé **char***
Inoltre, è **necessario allocare memoria** per l'array di caratteri che conterrà la stringa (si veda sotto).

Non si definiscono nuovi tipi, quindi **NON** viene generato il file con le routine di conversione **echo_xdr.c**

File echo.h

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

#ifndef _ECHO_H_RPCGEN
#define _ECHO_H_RPCGEN

#include <rpc/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

#define ECHOPROG 0x20000013
#define ECHOVERS 1

#if defined(__STDC__) || defined(__cplusplus)
#define ECHO 1
extern  char ** echo_1(char **, CLIENT *);
extern  char ** echo_1_svc(char **,struct svc_req
*);
extern int echoprog_1_freeresult (SVCXPRT *,
  xdrproc_t, caddr_t);

#else /* K&R C */
#define ECHO 1
extern  char ** echo_1();
extern  char ** echo_1_svc();
extern int echoprog_1_freeresult ();
#endif /* K&R C */

#ifdef __cplusplus
}
#endif
#endif /* !_ECHO_H_RPCGEN */
```

File echo_clnt.c: stub del client

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

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

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

char **
echo_1(char **argp, CLIENT *clnt)
{
    static char *clnt_res;

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


File echo_svc.c: stub del server

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

#include "echo.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void
echoprog_1(struct svc_req *rqstp, register SVCXPRT
*transp)
{
    union {
        char *echo_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 ECHO:
        _xdr_argument =
                    (xdrproc_t) xdr_wrapstring;
        _xdr_result = (xdrproc_t) xdr_wrapstring;
        local = (char *(*)(char *,
                     struct svc_req *)) echo_1_svc;
        break;

    default:
        svcerr_noproc (transp);
        return;
    }
    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, "%s",
                       "unable to free arguments");
        exit (1);
    }
    return;
}
```



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

    pmap_unset (ECHOPROG, ECHOVERS);

    transp = svcudp_create(RPC_ANYSOCK);
    if (transp == NULL) {
        fprintf (stderr, "%s",
                     "cannot create udp service.");
        exit(1);
    }
    if (!svc_register(transp, ECHOPROG, ECHOVERS,
                        echoprog_1, IPPROTO_UDP)) {
        fprintf (stderr, "%s", "unable to register
                      (ECHOPROG, ECHOVERS, udp).");
        exit(1);
    }

    transp = svctcp_create(RPC_ANYSOCK, 0, 0);
    if (transp == NULL) {
        fprintf (stderr, "%s",
                     "cannot create tcp service.");
        exit(1);
    }
    if (!svc_register(transp, ECHOPROG, ECHOVERS,
                       echoprog_1, IPPROTO_TCP)) {
        fprintf (stderr, "%s", "unable to register
                      (ECHOPROG, ECHOVERS, tcp).");
        exit(1);
    }

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

File echo_proc.c: implementazione della procedura

```
/*  echo_proc.c:  implementazione  della  procedura
remota "echo" */

#include <stdio.h>
#include <rpc/rpc.h>
#include "echo.h"

char **echo_1_svc (char **msg, struct svc_req *rp)
{
    static char *echo_msg;

    free(echo_msg);
    echo_msg=(char*)malloc(strlen(*msg)+1);

    printf("Messaggio ricevuto: %s\n", *msg);
    strcpy(echo_msg, *msg);
    printf("Messaggio da rispedire: %s\n",
                echo_msg);
    return (&echo_msg);
}
```

Motivi della posizione della ***"free"***:

Prima della malloc perché serve a liberare la memoria occupata dall'invocazione precedente

=> inutile per la prima volta, ok le successive

Allocazione della memoria su lato server

E' necessario allocare esplicitamente memoria per il parametro di uscita

Es. *echo_msg* in echo_proc.c

Non serve sul parametro di ingresso, l'allocazione è fatta automaticamente dal supporto rpc

Es. *msg,* in echo_proc.c

Il **parametro di uscita** deve essere **static**.

Allocazione della memoria su lato client

E' necessario allocare esplicitamente memoria per il parametro di ingresso

Es. *msg* in echo_client.c

Non serve sul parametro di uscita, l'allocazione del valore di ritorno è fatta automaticamente dal supporto rpc

Es. *echo_msg,* in echo_client.c

File echo_client.c: implementazione del client che chiama la procedura remota

```
/* echo_client.c: client che chiama la procedura
remota */

#include <stdio.h>
#include <rpc/rpc.h>
#include "echo.h"
#define DIM 100

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

    CLIENT *cl;
    char **echo_msg;
    char *server;
    char *msg;
    char ok[5];

    if (argc < 2) {
       fprintf(stderr, "uso: %s host\n", argv[0]);
       exit(1);
    }

    server = argv[1];

    cl = clnt_create(server, ECHOPROG, ECHOVERS,
                                              "udp");
    if (cl == NULL) {
        clnt_pcreateerror(server);
        exit(1);
    }

    msg=(char*)malloc(DIM);
    printf("Qualsiasi tasto per procedere,
                          EOF per terminare: ");
```



```
    while (gets(ok))
    {
        /* lettura della stringa da inviare */
        printf("Messaggio (max 100 caratteri)? ");
        gets(msg);

        echo_msg = echo_1(&msg, cl);

        if (echo_msg == NULL) {
            fprintf(stderr, "%s: %s fallisce
                        la rpc\n", argv[0], server);
            clnt_perror(cl, server);
            exit(1);
        }

        if (*echo_msg == NULL) {
            fprintf(stderr, "%s: %s restituisce una
                stringa nulla\n", argv[0], server);
            clnt_perror(cl, server);
            exit(1);
        }

        printf("Messaggio consegnato a %s: %s\n",
                                        server, msg);
        printf("Messaggio ricevuto da %s: %s\n",
                                server, *echo_msg);

        printf("Qualsiasi tasto per procedere,
                            EOF per terminare: ");

    } // while gets(ok)

    printf("Termino...\n");
    exit(0);
}
```

Compilazione

Compilazione per generare l'eseguibile del client:

gcc echo_client.c echo_clnt.c -o remote_echo

=> produce il comando *remote_echo*

Compilazione per generare l'eseguibile del server:

gcc echo_proc.c echo_svc.c -o echo_server

=> produce il comando *echo_server*

Esecuzione

Mandare in esecuzione il server con il comando

echo_server

Mandare in esecuzione il client con il comando

remote_echo serverhost

Nota bene:
Nella variante il client viene lanciato con due argomenti: serverhost e la stringa che si vuole inviare al server.

Alcune opzioni di rpggen (vedere il man):

-a Generate all the files including sample code for client and server side.

-Sc Generate sample code to show the use of remote procedure and how to bind to the server before calling the client side stubs generated by rpcgen.

-Ss Generate skeleton code for the remote procedures on the server side. You would need to fill in the actual code for the remote procedures.