9° Esercitazione (svolta):

Remote Procedure Call: callback

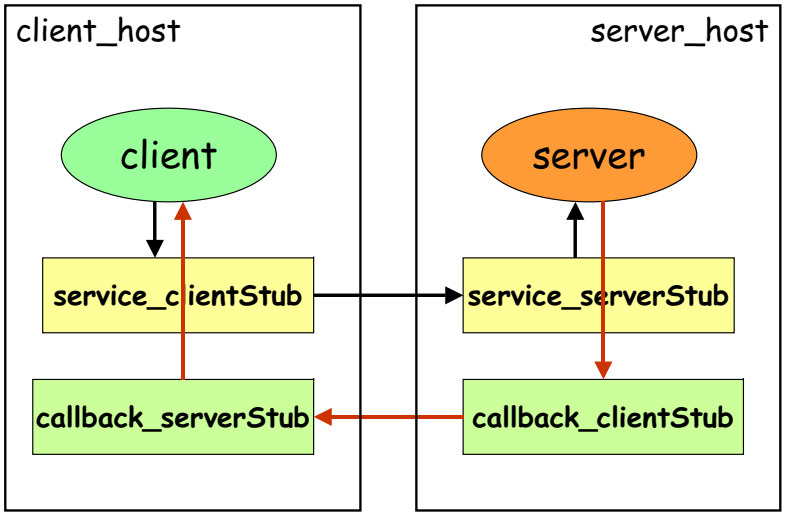
Callback con rpc

Client Callback: meccanismo che permette ad un client di essere *notificato dal server* al verificarsi dell'evento per cui si è registrato e messo in attesa.

Notifica del server = Invocazione di una procedura remota del client

Invocazione di procedure remote

Bidirezionale



Come realizzare la callback

Su lato client:

bisogna fornire la procedura remota che il server invoca per fare la callback

1. definire il file .x relativo alla procedura di callback
2. generare gli stub e le eventuali routine xdr
3. implementare la procedura di callback
4. generare l'eseguibile che lancia il processo in attesa di chiamate della procedura di callback

Su lato server:

bisogna mantenere un riferimento al client su cui fare la callback; dovendo fare una rpc, il server diventa client a sua volta, deve quindi creare un gestore di trasporto:

```
clnt_create(rpc_host, RPCPROG, RPCVERS, "udp");
```

Il server conosce **staticamente**:

- **numero di programma**, **numero di versione** e **nome procedura** della rpc che implementa la callback;

e ottiene **dinamicamente**, al momento della registrazione:

- l'host su cui è in esecuzione il processo che realizza la callback, cioè l'**host del client**.

Esercizio: Chat

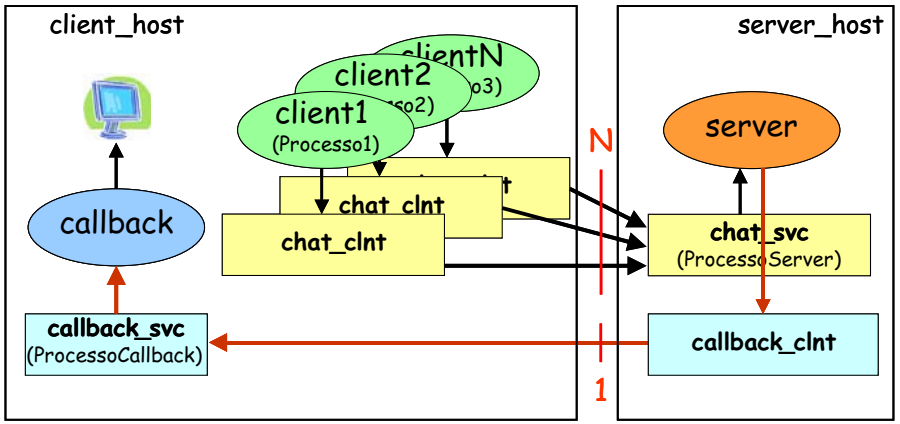
Realizzare un'applicazione che implementi una chat tra tutti i possibili client tramite un server, che manda i messaggi di tutti a tutti. In particolare:

- il **server** tiene traccia dei client registrati.
  Per ogni nuova registrazione notifica a tutti i presenti nella room il nuovo ingresso.
  Per ogni messaggio ricevuto notifica a tutti i presenti il messaggio con il mittente.
  Per ogni uscita notifica a tutti i presenti l'uscita.
  Ogni notifica viene effettuata invocando una ***callback***;

- i **client** si registrano, mandano messaggi fino all'inserimento dell'EOF da tastiera, infine si cancellano e terminano;

- il **processo di callback**:
  diversamente dal caso RMI, non è il processo del client a ricevere la callback. Viene invece attivato, per ogni macchina client, **un unico processo** che riceve le callback **di tutti i clienti** che si sono sottoscritti da quella macchina, e stampa i messaggi a video.

Architettura



Modello architetturale di servizio

Un unico processo di **callback** deve quindi essere pronto a gestire le informazioni per tutti i possibili clienti di quel nodo.

Implementazione della callback

callback.x

```
program CALLBACKPROG {
    version CALLBACKVERS {
        int CALLBACK(string) = 1;
    } = 1;
} = 0x20000014;
```

callback_proc.c

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

int *callback_1_svc(char **msg, struct svc_req *rp){

    static int ris;

    printf("%s\n", *msg);
    ris=0;
    return (&ris);
}
```


Implementazione del server

chat.x

```
struct Dati{
    char client_name[30];
    char client_host[100];
};

struct Messaggio{
    char mitt[30];
    char body[100];
};


program CHATPROG {
    version CHATVERS {
        int REGISTER(Dati) = 1;
        void SAY_SOMETHING(Messaggio) = 2;
        int UNREGISTER(Dati) = 3;
    } = 1;
} = 0x20000013;
```

chat_proc.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "chat.h"
#include "callback.h"
#define USER_HOST_MAX 10

static char user_host[USER_HOST_MAX][100];
static int user_count[USER_HOST_MAX];
int user_num=0;
/* Gestori di trasporto per la comunicazioni con tutti
 * gli host sottoscrittori. Più client che si
 * sottoscrivono da uno stesso host condividono il
 * processo di callback --> si ha al più un gestore di
 * trasporto per ogni macchina
 */
static CLIENT * clienti[USER_HOST_MAX];

void doCallback(char *msg){

    CLIENT *cl;
    int i;

    printf("Eseguo doCallback...\n");
    for (i=0; i<USER_HOST_MAX; i++){
      printf("i = %d,  user_host[%d] = %s, user_count[%d]
              = %d\n", i,i,user_host[i],i,user_count[i]);
        if (strcmp(user_host[i], "")!=0){
          /* reperimento gestore trasporto */
          cl = clienti[i];
          if (cl == NULL)
          {
            printf("Errore durante il reperimento del
                      gestore di trasporto\n");
            exit(1);
          }
          callback_1(&msg, cl);
        }  //if
    } //for
} // doCallback
```



```
int *register_1_svc(Dati *dati, struct svc_req *rp){
    static int ris;
    char callback_msg[30];
    int i, pres;

    printf("Register, dati ricevuti:\n");
    printf("dati->client_name = %s\n",
                                dati->client_name);
    printf("dati->client_host = %s\n",
                                dati->client_host);
    ris=-1; pres=0;

    /* verifico se sullo stesso host c'è già
       un utente registrato, in questo caso incremento
       il contatore corrispondente... */
    for (i=0;i<USER_HOST_MAX; i++) {
      if (strcmp(user_host[i], dati->client_host)==0){
        user_count[i]++;
        pres=1;
        ris=0;
        break;
      }
    }
    /* ... altrimenti inserisco una nuova entry */
    if (pres!=1){
      for (i=0;i<USER_HOST_MAX; i++) {
        if (strcmp(user_host[i], "")==0){
          strcpy(user_host[i], dati->client_host);
          user_count[i]=1;
          /* creazione gestore trasporto */
          clienti[i]=clnt_create(user_host[i],
                CALLBACKPROG, CALLBACKVERS, "udp");
          if (clienti[i] == NULL)
          { clnt_pcreateerror(user_host[i]);
            exit(1);
          }
         ris=0;
         break;
        }
      }
    }
```

```
/* se la registrazione è andata a buon fine… */
    if (ris==0){
        strcpy(callback_msg, "Ingresso di ");
        strcat(callback_msg, dati->client_name);
        user_num++;
        printf("Numero di utenti registrati: %d\n",
                                                user_num);
        doCallback(callback_msg);
    }
    return (&ris);
}



void *say_something_1_svc(Messaggio *msg,struct svc_req *rp){
    char callback_msg[30];

    printf("Eseguo say_something...\n");
    printf("Dati ricevuti:\n");
    printf("mitt = %s\n", msg->mitt);
    printf("body = %s\n", msg->body);
    strcpy(callback_msg, msg->mitt);
    strcat(callback_msg, ": ");
    strcat(callback_msg, msg->body);
    doCallback(callback_msg);
}
```



```
int *unregister_1_svc(Dati *dati, struct svc_req *rp){
    static int ris;
    char callback_msg[30];
    int i;

    printf("Eseguo unregister...\n");
    printf("Dati ricevuti:\n");
    printf("dati->client_name = %s\n",
                                  dati->client_name);
    printf("dati->client_host = %s\n",
                                  dati->client_host);

    ris=-1;

    for (i=0;i<USER_HOST_MAX; i++) {
      if ((strcmp(user_host[i], dati->client_host)==0)){
        user_count[i]--;
        user_num--;
        if(user_count[i]==0) strcpy(user_host[i], "");
        strcpy(callback_msg, "Uscita di ");
        strcat(callback_msg, dati->client_name);
        printf("callback_msg = %s\n", callback_msg);
        printf("Numero di utenti registrati: %d\n",
                                                user_num);
        doCallback(callback_msg);
        ris=0;
        break;
      }
    }
    return (&ris);
}
```

Implementazione del client

chat_client.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "chat.h"
#define DIM_NAME 30
#define DIM_MSG 100
#define DIM_HOST_NAME 1024

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

     CLIENT *cl;
     char *server_host, client_host[1024];
     Dati dati;
     Messaggio msg;
     int len;
     char name[20];
     int *reg_res, *unreg_res;
     void *say_res;

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

     printf("Chat client: inizio...\n");

     server_host = argv[1];


     if (gethostname(client_host, sizeof(client_host))<0){
         fprintf(stderr, "error: gethostname\n");
         exit(1);
     }
     strcpy(dati.client_host, client_host);
```



```
/* Creazione gestore di trasporto */
cl = clnt_create(server_host, CHATPROG, CHATVERS,
                                                    "udp");
if (cl == NULL) {
     clnt_pcreateerror(server_host);
     exit(1);
}


/* Registrazione utente */
printf("Registrazione\n");
printf("Nome (max 20 caratteri)? ");
gets(name);
printf("name = %s\n", name);
strcpy(dati.client_name, name);

printf("Invoco register con i dati:\n");
printf("dati.client_name = %s\n", dati.client_name);
printf("dati.client_host = %s\n", dati.client_host);


reg_res=register_1(&dati, cl);

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

if (*reg_res==0) printf("Registrazione
                                    effettuata\n");
else {
     printf("Registrazione non effettuata\n");
     printf("\nChat client: termino...\n");
     exit(1);
}

strcpy(msg.mitt, name);
```

```
    /* ciclo di interazione con l'utente ------- */
    printf("Inserire messaggi, EOF per terminare:\n");
    printf("> ");

    while (gets(msg.body)){

        say_res = say_something_1(&msg, cl);

        if (say_res == NULL) {
            fprintf(stderr, "%s: %s fallisce la
                say_something\n", argv[0], server_host);
            clnt_perror(cl, server_host);
            exit(1);
        }
        printf("> ");
    }

    unreg_res=unregister_1(&dati, cl);

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

    if (*unreg_res==0)
            printf("Cancellazione effettuata\n");
    else    printf("Cancellazione non effettuata\n");

    printf("\nChat client: termino...\n");
    exit(0);
}
```


Compilazione

- Per generare **stub** e **routine xdr**:

rpcgen chat.x
che produce: chat.h, chat_clnt.c chat_svc.c chat_xdr.c

rpcgen callback.x
che produce: callback.h, callback_clnt.c callabck_svc.c

- Per generare l'eseguibile della **callback**:

gcc –o callback callback_proc.c callback_svc.c
che produce: callback

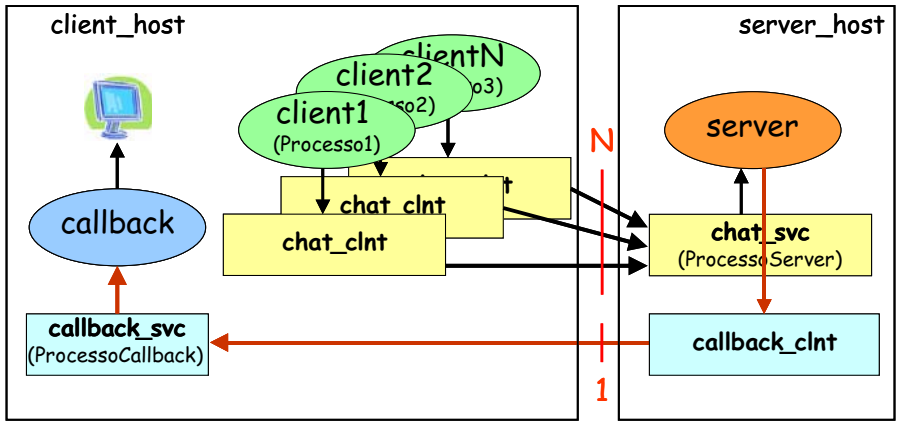
- Per generare l'eseguibile del **server**:

gcc –o server chat_proc.c chat_svc.c callback_clnt.c chat_xdr.c
che produce: server

- Per generare l'eseguibile del **client**:

gcc –o client chat_client.c chat_clnt.c chat_xdr.c
che produce: client

Architettura



Nota bene:
Esiste un problema di terminazione del **processo callback**.
Si provino a progettare e realizzare possibili soluzioni.

- Quali **politiche** realizzare?

- Quali **possibilità realizzative** ci sono?