OBIETTIVI

ESERCITAZIONE 8:

Remote Procedure Call (RPC)

- Contare i caratteri, le parole e le linee di un file di testo presente sul server remoto.
- Contare il numero di file (presenti nel direttorio remoto indicato dal client) la cui dimensione è maggiore di un intero indicato dal client.
- Richiedere le operazioni mediante una chiamata ad operazione remota (RPC)

Davide di Molfetta, Mirko Legnini, Daniele Nanni Cirulli, Natanaele Stagni, Lorenzo Venerandi

OPERAZIONI

FILE OPERATION.X

```
struct dir_scan { string dirname <4096>; int filedim;};
struct rez { int charz; int worz; int linz; };
program FILEPROG {
version FILEVERS {
rez FILE_SCAN (string) = 1;
int DIR_SCAN (dir_scan) = 2;
} = 1;
} = 0x20000013;
```

rpcgen operazioni.x

- C operazioni_clnt.c
- C operazioni_svc.c
- C operazioni_xdr.c
- h operazioni.h

gcc –o server_rpc implemetazione.c operazioni_svc.c operazioni_xdr.c

client_rpc

gcc –o client_rpc client.c operazioni_clnt.c operazioni_xdr.c

server_rpc

operation.h

```
struct dir_scan {
    char *dirname;
    int filedim;
};
typedef struct dir_scan dir_scan;

struct rez {
    int charz;
    int worz;
    int linz;
};
typedef struct rez rez;

#define FILEPROG 0x20000013
#define FILEVERS 1
```

IMPL. PROCEDURE

• FILE_SCAN: accetta come parametro d'ingresso il nome del file e restituisce tre interi che indicano: numero di caratteri, parole e linee nel file. Altrimenti un codice di errore.

```c
rez *file_scan_1_svc(char **nomefile, struct svc_req *rp)
{
    // Istanziazione risultato
    static rez res;
    res.charz=res.worz=res.linz=-1;
    int fd;
    printf("File:\t%s\n", *nomefile);

    //controllo esistenza del file
    if (!(fd = open(*nomefile, O_RDONLY)) == -1){
    res.charz = 0;
    res.worz = res.linz = 1;
    // Filtro a carattere
    char c;
    while (read(fd, &c, 1) > 0)
    {
        res.charz += 1;
        if (c == ' ')
            res.worz += 1;
        if (c == '\n')
        {
            res.worz += 1;
            res.linz += 1;
        }
    }
    close(fd);
    }
    return (&res);
}
```

• DIR_SCAN: accetta come parametro d'ingresso il nome del direttorio remoto e una soglia numerica. In caso di successo, restituisce un intero positivo con il numero di file la cui dimensione supera la soglia inserita, altrimenti -1.

```c
int *dir_scan_1_svc(dir_scan *req, struct svc_req *rp)
{
    static int res;
    int fd;
    char path[PATH_MAX];
    struct dirent *dent;
    DIR *dir;
    printf("Lunghezza minima file:\t%d\nNome directory:\t%s\n", req->filedim, req->dirname);

    //apertura directory
    if ((dir = opendir(req->dirname)) == NULL)
    {
        res = -1;
        return &res;
    }

    //main cicle
    res=0;
    while ((dent = readdir(dir)) != NULL)
    {
        strcpy(path, req->dirname);

        if (strncmp(dent->d_name, ".", 1) != 0 && dent->d_type==DT_REG
            &&(fd = open(strcat(strcat(path, "/"), dent->d_name), O_RDONLY)) != -1)
        {
            if (lseek(fd, 0, SEEK_END) >= req->filedim)
                res++;
            close(fd);
        }
    }
    return (&res);
```

CLIENT

```c
// creazione gestore di trasporto
if (( cl = clnt_create(server, FILEPROG,FILEVERS, "udp")) == NULL)
{
    clnt_pcreateerror(server);
    exit(1);
}

printf("Inserisci operazione desiderata:\n>fscan\n>dirscan\n");
while (gets(procedure))
{
```

Il Client interagisce con l'utente proponendogli ciclicamente i servizi che utilizzano le due procedure remote. Richiede gli input necessari, invoca il servizio specificato e stampa a video gli esiti delle chiamate, fino alla fine del file di input da tastiera.

```c
if (strcmp(procedure, "fscan") == 0)
{
    printf("Inserisci il nome del file: ");
    gets(filename);
    res = file_scan_1(&filename, cl);
    if (res == NULL)
    {
        clnt_perror(cl, server);
        exit(1);
    }
    if(res->charz==-1)
        printf("File selezionato vuoto!");
    if(res->charz==0 && res->worz==0 && res->linz==0)
        printf("File selezionato vuoto!");
    else printf("---------------------\nFile analizzato:
     %s\n>charz:\t%d\n>worz:\t%d\n>linz:\t%d\n---------------------\n",
     filename, res->charz, res->worz, res->linz);
}
```

```c
else if (strcmp(procedure, "dirscan") == 0)
{
    printf("Inserisci la directory da scansionare: ");
    gets(dirname);
    dirscan.dirname = dirname;
    printf("Inserisci la dimensione minima del file: ");
    while(scanf("%d%*c", &filedim)!=2){
        printf("Dimensione errata!\n\nInserisci la dimensione minima del file: ");
    }
    dirscan.filedim = filedim;

    ris = dir_scan_1(&dirscan, cl);
    if (ris == NULL)
    {
        clnt_perror(cl, server);
        exit(1);
    }
    switch (*ris)
    {
    case -1:
        printf("Errore scansione directory!\n\n");
        break;
    default:
        printf("---------------------\nTotale file grandi: %d\n---------------------\n", *ris);
        break;
    }
```

CONCLUSIONI

- Creazione e gestione delle socket UDP o TCP automaticamente generata da rpc
- Implementazione delle procedure più semplice, in quanto è necessario «dichiarare» i metodi nel file .x e implementarli quindi in un file .c
- Utilizzo delle procedure più immediato, dato che il client deve solo includere il file .h generato da rpcgen per richiamare i metodi
- Output di compilazione limitato soltanto a due eseguibili (server e client), necessaria l'esecuzione del rpcbind (port-mapper) lato server

FINE