

Soluzione Esercitazione 8

Leonardo Bambini, Patrick Di Fazio, Matteo Longhi,
Giorgio Mastrotucci, Luca Torzi

Requisiti

Si vuole realizzare una applicazione C/S che consente di effettuare operazioni remote su file in un nodo remoto

In particolare le specifiche del server prevedono che:

- si possano contare i caratteri, la parole e le linee di un file di testo
- contare il numero di file presenti nel direttorio indicato dal client, la cui dimensione è maggiore di N specificato.

Il client ha un comportamento da filtro.

XDR: Definizioni (procedure.x)

Per generare il **Client**:

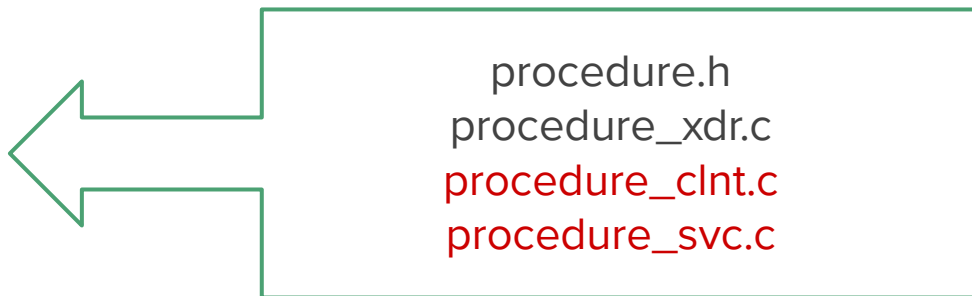
```
gcc -o procedure_client.c procedure_clnt.c  
procedure_xdr.c
```

Per generare il **Server**:

```
gcc -o procedure_server.c procedure_proc.c procedure_svc.c  
procedure_xdr.c
```

Per generare **header stub e file conversioni**:
rpcgen procedure.x

```
struct Dir_scan { string filename <256>; int dimfile;};  
struct Res { int first; int second; int third; };  
program FILEPROG {  
  version FILEEVERS {  
    Res CONTAFILE (string) = 1;  
    int CONTADIR (Dir_scan) = 2;  
  } = 1;  
} = 0x200000013;
```



Il Client

Il Client da filtro in base alla procedura attiva la prima o la seconda procedura in modo ciclico, controllando i vari parametri inseriti dall'utente.

```
while (scanf("%d", &procedure) > 0)
{
    if (procedure == 1)
    {
        printf("Inserisci il nome del file\n");
        scanf("%s", contafile);
        resconta = contafile_1(&contafile, cl);
        if (resconta == NULL)
        {
            clnt_perror(cl, server);
            exit(1);
        }
        if (&(resconta->first) == NULL || &(resconta->second) == NULL || &(resconta->third) == NULL)
        {
            clnt_perror(cl, server);
            exit(1);
        }
        if (resconta->first == -1 || resconta->second == -1 || resconta->third == -1)
        {
            printf("file %s non esistente\n", contafile);
        }
    }
    else{
        printf("Inseriti i valori: %d-%d-%d\n", resconta->first, resconta->second, resconta->third);
    }
}
```

```
printf("Inserisci il nome del direttorio\n");
scanf("%s", filename);
printf("Inserisci la dimensione minima\n");
while(scanf("%i", &((&(dirscan))->dimfile))<=4)
    gets();
    printf("Inserisci la dimensione minima\n");
}
dirscan.filename=filename;

ris = contadir_1(&dirscan, cl);
if (ris == NULL)
{
    clnt_perror(cl, server);
    exit(1);
}
if(*ris==-1){
    printf("cartella %s non esistente\n", filename);
}
else{
    printf("risultato:%d\n", *ris);
}
```

Implementazione (1)

file_scan accetta nome di file e restituisce una struct di tre interi. In caso di errore, restituisce -1 in tutti e 3 i campi della struct

lettura file char by char

```
//istanziazione risultato
static Res res;
res.first=res.second=res.third=0;
int f;
char c;
printf("nome file ricevuto: %s\n", *nomefile);

//controllo esistenza del file
if((f=open(*nomefile,O_RDONLY))== -1){

    res.first=res.second=res.third=-1;
    return (&res);
}

while(read(f,&c,1)>0){
    res.first+=1;
    if(c==' ')res.second+=1;
    if(c=='\n'){
        res.second+=1;
        res.third+=1;
    }
}
```

Implementazione (2)

dir_scan accetta nome del direttorio e soglia e in caso di successo resituisce un intero positivo.

```
while((dent=readdir(d))!=NULL){

    //preparazione path
    memset(path, '\\0', sizeof(path));
    strcpy(path, dir->filename);
    strcat(path, "/");
    strcat(path, dent->d_name);

    //controllo apertura file e cartelle speciali
    if((f=open(path, O_RDONLY))>=0 && strcmp(dent->d_name, ".")!=0 && strcmp(dent->d_name, "..")!=0){
        if(lseek(f, SEEK_END, 0)>=dir->dimfile)ris+=1;
        close(f);
    }
}
```