ESERCITAZIONE 4

Server Multiservizio: Socket C con select

OBIETTIVI

Sviluppare un'applicazione C/S che fornisca due servizi:

- eliminazione di una determinata parola (senza connessione)
- lista di nomi di file in sotto-direttori (con connessione)

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

SPECIFICHE

SPECIFICHE CLIENTE UDP:

- Il Client chiede all'utente il nome di un file e una parola e invia al Server la richiesta di eliminazione delle occorrenze della parola dal file.
- Attende quindi l'esito dell'operazione.

SPECIFICHE CLIENTE TCP:

- Chiede il nome del direttorio da esplorare e lo invia al Server.
- Riceve dal Server i nomi dei file individuati nelle directory di secondo grado

SPECIFICHE SERVER MULTISERVIZIO:

- Utilizzo della SELECT per identificare il tipo di richiesta.
- Gestione concorrente delle richieste UDP.
- Gestione concorrente e multiprocesso delle richieste TCP con chiusura della sessione alla ricezione della fine file dal client.

CLIENT UDP

Controllo argomenti
+
Creazione ed apertura socket
+
Assegnazione IP e porta (bind)

Invio parola al server →

Attesa risposta server →

Risposta →

```c
printf("Inserire nome file, ^D per terminare\n");

while (gets(req.nomeFile) != NULL)
{

    printf("Inserire parola da eliminare\n");
    gets(req.parola);

    /* richiesta operazione */
    len = sizeof(servaddr);
    if (sendto(sd, &req, sizeof(request), 0, (struct sockaddr *)&servaddr, len) < 0)
    {
        perror("sendto");
        continue;
    }

    /* ricezione del risultato */
    printf("Attesa del risultato...\n");
    if (recvfrom(sd, &ris, sizeof(ris), 0, (struct sockaddr *)&servaddr, &len) < 0)
    {
        perror("recvfrom");
        continue;
    }

    if ((int)ntohl(ris) < 0)
        printf("Il file %s non esiste!\n\n", req.nomeFile);
    else
        printf("Numero di eliminazioni: %i\n", (int)ntohl(ris));
    printf("Inserire nome file, ^D per terminare\n");

} // while gets
```

CLIENT TCP

Controllo argomenti
+
Creazione ed apertura socket
+
Assegnazione IP e porta (bind)

Ciclo accettazione richieste utente

Ricezione lista dei nomi di file

```c
printf("Nome della directory da eplorare, EOF per terminare: ");

while (gets(nome_sorg))
{
    printf("Directory da esplorare: __%s__\n", nome_sorg);
    write(sd,nome_sorg,strlen(nome_sorg));

    /*RICEZIONE FILES*/
    printf("Client: ricevo e stampo file individuati della directory\n");
    while ((nread = read(sd, buff, DIM_BUFF)) > 0 && buff[0]!=(char)4 )
    {
        write(1, buff, nread);
        //printf("%s",buff);
    }

    printf("Operazione terminata\n");

    printf("Nome della directory da eplorare, EOF per terminare: ");
} //while
```

SERVER TCP

Controllo argomenti
+
Creazione ed apertura socket
+
Assegnazione IP e porta (bind)

```
FD_SET(listenfd, &rset);
FD_SET(udpfd, &rset);

if ((nready = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0)
{
    if (errno == EINTR)
        continue;
    else
    {
        perror("select");
        exit(8);
    }
}
```

← Ciclo accettazione select

Gestione richiesta TCP con accept →

```
/* GESTIONE RICHIESTE DI GET DI UN FILE ---------------------------------
if (FD_ISSET(listenfd, &rset)) // Richiesta TCP
{
    printf("Ricevuta richiesta di get di un file\n");
    len = sizeof(struct sockaddr_in);
    if ((connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &len)) < 0)
    {
        if (errno == EINTR)
            continue;
        else
        {
            perror("accept");
            exit(9);
        }
    }
```

Creazione del figlio

Lettura directory mandata dal client

Controllo e validità directory ed eventuale comunicazione al client

Lettura nome directory

Ciclo di controllo esistenza sotto-directory

Scrittura su socket sotto-directory trovate

Dice al client quando fermarsi

```c
if (fork() == 0)
{
    close(listenfd);
    printf("Dentro il figlio, pid=%i\n", getpid());
    while ((num = read(connfd, nome_dir, sizeof(nome_dir))) > 0)
    {
        printf("%d\t", num);
        nome_dir[num] = '\0';
        printf("Richiesta directory %s\n", nome_dir);
        tcpStartMillis=clock();

        mainDir = opendir(nome_dir);
        if (mainDir == NULL)
        {
            printf("Directory non valida\n");
            write(connfd, "La directory è errata \n", strlen("La directory è errata\n"));
        }
        else
        {
            while ((cur = readdir(mainDir)) != NULL)
            {

                if (cur->d_type == DT_DIR && cur->d_name[0] != '.')
                {
                    strcpy(directory, nome_dir);
                    strcat(strcat(directory, "/"), cur->d_name);

                    printf("Directory:\t%s\n", directory);

                    currentDir = opendir(directory);
                    if (currentDir == NULL)
                    {
                        printf("Directory %s non valida\n", directory);
                        continue;
                    }

                    while ((cur = readdir(currentDir)) != NULL)
                    {
                        if (cur->d_name[0] != '.')
                        {
                            printf("%s\n", cur->d_name);
                            write(connfd, cur->d_name, strlen(cur->d_name));
                            write(connfd,"\n",sizeof(char));
                        }
                    }
                }
            }
        }

        write(connfd, end, sizeof(end)); // Print carattere terminatore
        printf("Tempo impiegato: %.2f ms\n",(((double)(clock()-tcpStartMillis)/CLOCKS_PER_SEC)*1000));
```

Apertura directory

SERVER UDP

```c
if (FD_ISSET(udpfd, &rset))
{
    printf("Ricevuta richiesta di eliminazione parola dal file\n");

    len = sizeof(struct sockaddr_in);
    if (recvfrom(udpfd, &req, sizeof(request), 0, (struct sockaddr *)&cliaddr, &len) < 0)
    {
        perror("recvfrom");
        continue;
    }

    printf("Richiesta eliminazione parola %s dal file %s\n", req.parola, req.nomeFile);
    num = 0;
    lenght = 0;
    udpStartMillis=clock();

    if ((fd_file = open(req.nomeFile, O_RDONLY,mode)) < 0)
    {
        perror("Errore");
        num = -1;
    }

    if ((fd_tmp = open("tmp", O_WRONLY | O_CREAT | O_TRUNC, mode)) < 0)
    {
        perror("Errore file temporaneo");
        continue;
    }
```

Accettazione richieste ed apertura file

```c
lParola = strlen(req.parola);
printf("Inizio lettura file\n");
while (read(fd_file,&c, sizeof(c)) > 0)
{
    parola[lenght] = c;
    lenght++;
    if (c == ' ' || c == '\n')
    {
        if (strncmp(req.parola,parola,lenght-1) != 0){
            write(fd_tmp, parola, lenght);
        }
        else{
            num++;
        }
        lenght = 0;
    }
}
printf("Fine lettura file\n");
printf("Tempo impiegato: %.2f ms\n",(((double)(clock()-udpStartMillis)/CLOCKS_PER_SEC)*1000));

rename ("tmp",req.nomeFile);

close(fd_tmp);
close(fd_file);
```

Ciclo per cercare le parole da eliminare

Rinominazione file temporaneo

CONCLUSIONI

Analisi tempo di esecuzione del Server concorrente

- Il test misura il tempo necessario all'eliminazione di 1 parola all'interno di un file.
- Il test è stato eseguito in modo concorrente, cioè eseguendo contemporaneamente un client TCP ed uno UDP.