9 Novembre 2021

Esercitazione 4

Socket C con Select

Bernardini Claudio
Corsetti Luca
Giardina Gianluca
Straccali Leonardo

Obiettivi esercitazione

Hard skills

1. Gestione socket con Select
2. Gestione direttori
3. Eliminazione di occorrenze di una parola richiesta

Soft skills

1. Collaborazione per raggiungere un fine comune
2. Organizzazione del proprio lavoro in team

Introduzione datagram

Client

1. Richiesta informazioni al cliente
2. Invia al server la richiesta di eliminazione
3. Attende la risposta del server contenente il numero di occorrenze cancellate

Server

1. Attendo l’arrivo di un messaggio
2. Riceve il nome del file, e se esiste, cancella ogni occorrenza della parola richiesta
3. Invia il numero di occorrenze eliminate

Introduzione stream

Client

1. Stabilisce una connessione con il servitore
2. Invia il nome del direttorio padre
3. Attende la risposta del server contenente la lista di nomi dei file contenuti nei sotto-direttori

Server

1. Attende richiesta di una connessione
2. Gestione concorrente delle richieste
3. Restituisce la lista dei nomi dei sotto-direttori del padre

Implementazione client

```c
/* CORPO DEL CLIENT: */
printf("Inserisci nome del file seguito da parola da rimuovere [Formato: nome_file.ext;parola] :");

while (gets(msg)){
    /* invio richiesta */
    len=sizeof(servaddr);
    if (sendto(sd, msg, (strlen(msg)+1), 0, (struct sockaddr *)&servaddr, len)<0){
        perror("scrittura socket");
        printf("Inserisci nome del file seguito da parola da rimuovere [Formato: nome_file.ext;parola] :");
        continue; // se questo invio fallisce il client torna all'inzio del ciclo
    }

    /* ricezione del risultato */
    printf("Attesa del risultato...\n");
    if (recvfrom(sd, &num_file, sizeof(num_file), 0, (struct sockaddr *)&servaddr, &len)<0){
        perror("recvfrom");
        printf("Inserisci nome del file seguito da parola da rimuovere [Formato: nome_file.ext;parola] :");
        continue; // se questa ricezione fallisce il client torna all'inzio del ciclo
    }

    if (num_file<0) printf("Il messaggio %s è scorretto o non esiste il file\n", msg);

    else printf("Nel file del messaggio %s ci sono %d occorrenze.\n", msg, ntohl(num_file));

    printf("Inserisci nome del file seguito da parola da rimuovere [Formato: nome_file.ext;parola] :");

} // while
```

Implementazione server

```c
/*Funzione conteggio parole in un file*/
/*******************************************************/
int conta_parole_cancellate(char *msg)
{
    int numOcc = 0, fdread, fdwrite, lenParola, curDimBuff = 0, dimStringFileName = 0;
    char filename[LENGTH_FILE_NAME];
    char buff[DIM_BUFF], temp;

    while (*(msg) != ';' && *(msg) != '\0')
    {
        filename[dimStringFileName] = *(msg);
        msg++;
        dimStringFileName++;
    }
    if (strlen(msg) > 1 && *(msg) == ';') msg++; else return  -1;//msg++ per consumare ;
    filename[dimStringFileName] = '\0';
    printf("Nome file estrapolato %s\n", filename);
    printf("Parola da rimuovere %s\n", msg);

    lenParola = strlen(msg);
    if ((fdread = open(filename, O_RDONLY)) == -1)
        return -1;
    if ((fdwrite = open("temp.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644)) == -1)
        return -1;
    while ((read(fdread, &temp, 1)) > 0)
    {
        printf("Carattere letto: %c\n", temp);
        if (curDimBuff == lenParola)
        {
            if ((temp >= 'A' && temp <= 'Z') || (temp >= 'a' && temp <= 'z')) //Parola composta
            {
                write(fdwrite, buff, curDimBuff);
                write(fdwrite, &temp, 1);
                curDimBuff = 0;
            } else { //Parola cancellata
                numOcc++;
                curDimBuff = 0;
                printf("Carattere scritto dopo parola rifiutata: %c\n", temp);
                write(fdwrite, &temp, 1);
            }
        }
        else //Non ho ancora niente di significativo buffer
        {
            if (curDimBuff < lenParola && temp == msg[curDimBuff]) //Ho un carattere interessante
            {
                printf("Carattere letto interessante: %c\n", temp);
                buff[curDimBuff] = temp;
                curDimBuff++;
            }
            else
            {
                printf("Carattere letto poco utile: %c\n", temp);
                buff[curDimBuff] = temp;
                write(fdwrite, buff, curDimBuff+1);
                curDimBuff = 0;
            }
        }
    }
    if (curDimBuff > 0) {
        buff[curDimBuff] = '\0';
        printf("Buff restante: %s\n", buff);
        if (strcmp(buff, msg) == 0) numOcc++;
    }
    close(fdwrite); close(fdread);
    rename("temp.txt",filename);
    printf("Numero totale di parole %d\n", numOcc);
    return numOcc;
}
```

Implementazione client

```c
/* CORPO DEL CLIENT: */
/* ciclo di accettazione di richieste di file ------- */
printf("Nome del direttorio da richiedere: ");

while (gets(nome_file)){

    if (write(sd, nome_file, (strlen(nome_file)+1))<0){
        perror("write");
        break;
    }
    printf("Richiesta del file %s inviata... \n", nome_file);

    if (read(sd, &ok, 1)<0){
        perror("read");
        break;
    }

    if (ok=='S'){
        printf("Ricevo i file:\n");
        while((nread = read(sd,buff,sizeof(buff))) > 0) {
            if(strcmp(buff,"\0") != 0){
                write(0,buff,nread);
                printf("\n");
            }
            else break;
        }
    }
    else if (ok=='N') printf("File inesistente\n");
    else printf("Errore di protocollo\n"); //controllare sempre che il protocollo sia rispettato

    printf("Nome del direttorio da richiedere: ");
}//while
```

Implementazione server

```
if (fork() == 0) {
    close(listenfd);
    printf("Dentro il figlio, pid=%i\n", getpid());

    struct dirent *dp;
    struct stat sb;
    char dirPath[255];
    char fileName[255];
    char tempD_name[255];

    while(read(connfd, &nomedir, sizeof(nomedir)) > 0){

        getcwd(dirPath, sizeof(dirPath));
        strcat(dirPath,"/");
        strcat(dirPath,nomedir);

        printf("Richiesto Dir %s\n", dirPath);

        DIR *dir = opendir(dirPath);

        if (!dir) {
            printf("Dir inesistente\n");
            write(connfd, "N", 1);
        }
        else {
            write(connfd, "S", 1);
            while ((dp = readdir(dir)) != NULL) {
                if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0){

                    strcpy(tempD_name,nomedir);
                    strcat(tempD_name,"/");
                    strcat(tempD_name,dp->d_name);

                    if(stat(tempD_name, &sb) == 0 && S_ISDIR(sb.st_mode)) {
                        DIR *dir2 = opendir(tempD_name);

                        while ((dp = readdir(dir2)) != NULL) {;
                            if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0) {
                                strncpy(fileName, dp->d_name, 254);
                                fileName[254] = '\0';
                                write(0, fileName, sizeof(fileName));
                                if (write(connfd, fileName, sizeof(fileName)) < 0) {
                                    perror("write");
                                    break;
```

Gestione progetto

Coding

Team

Bernardini Claudio
Corsetti Luca
Giardina Gianluca
Straccali Leonardo

Grazie!