Soluzione Esercitazione 4

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

Requisiti

Si vuole realizzare un applicativo multiservizio in cui il Server **seleziona** i clienti e fornisce due servizi:

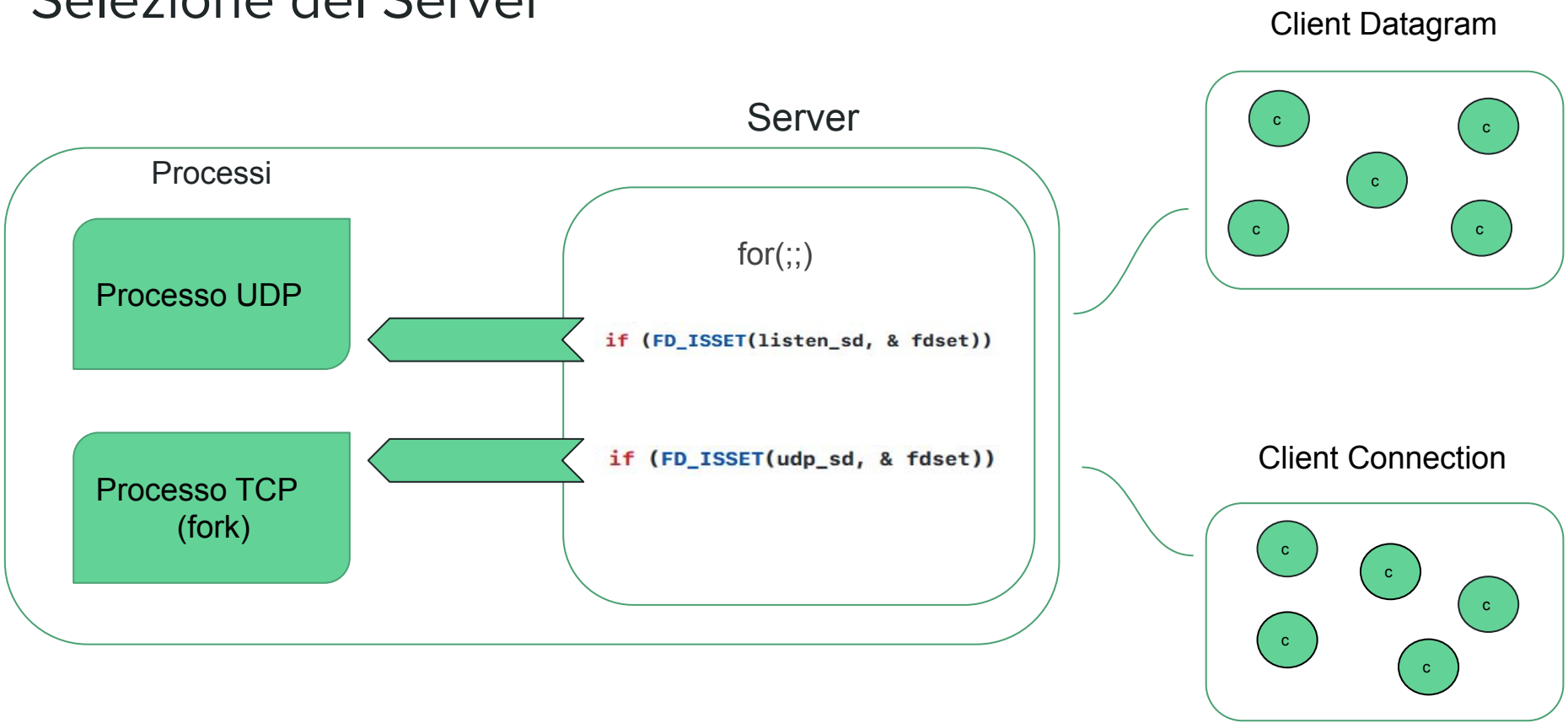
In particolare le specifiche del server prevedono che:

- Il primo servizio **UDP** elimina tutte le occorrenze di una parola in un file presente nel filesystem del server remoto, agisce in modo sequenziale.

- Il secondo servizio **TCP** restituisce tutti i nomi dei file presenti nei direttori di secondo livello, sul filesystem del server remoto, agisce in modo concorrente.

I due client interrogano ciclicamente l'utente in base al tipo di servizio che svolgono.

Selezione del Server



Il Client UDP

Il Client UDP richiede all'utente il nome del file e la parola e attende il risultato dell'operazione (**-1** se il file non esiste).

```
while (scanf("%s", fileName) > 0) {
    printf("Inserire la parola da eliminare\n");
    if (scanf("%s", parola) <= 0) {
        perror("Errore di input\n");
        printf("Inserisci il nome di un file o EOF per terminare\n");
        continue;
    }
    printf("Invio fileName: %s parola: %s\n", fileName, parola);
    //invio il risultato
    if (sendto(sd, fileName, sizeof(fileName), 0, (struct sockaddr * ) & servaddr, lenServAddr) < 0) {
        perror("sendto filename\n");
        continue;
    }
    if (sendto(sd, parola, sizeof(parola), 0, (struct sockaddr * ) & servaddr, lenServAddr) < 0) {
        perror("sendto parola\n");
        continue;
    }
    // ricezione del risultato
    if (recvfrom(sd, & result, sizeof(result), 0, (struct sockaddr * ) & servaddr, & lenServAddr) < 0) {
        perror("recvfrom\n");
        continue;
    }
    if (result == -1) {
        printf("Il file %s non esiste\n", fileName);
    } else {
        printf("Il file %s esiste, il risultato e' %d\n", fileName, result);
    }
    printf("Inserisci il nome di un file o EOF per terminare\n");
}
```

Il Client TCP

Il Client TCP richiede all'utente il nome del direttorio e viene notificato appena è pronto un nome di file, quindi lo riceve e lo stampa a video.

Se non è presente il file, il server restituisce **-1**

```
while (scanf("%s", nomeFile) != EOF) {

    //invio il nome del file
    if (sendto(sd, nomeFile, sizeof(nomeFile), 0, (struct sockaddr * ) & serveraddr, len) < 0) {
        perror("Errore di trasmissione\n");
        exit(4);
    }

    printf("Inviato nome file %s\n", nomeFile);

    if (recv(sd, & res, sizeof(res), 0) < 0) {
        perror("recv\n");
        continue;
    }
    printf("Esito: %d\n", res);
    int counter = 0;
    char buff[256];
    if (res >= 0) {
        //metto la maschera in ascolto sulla socket
        FD_SET(sd, & read_mask);
        select(nfds, & read_mask, NULL, NULL, & time);
        while (FD_ISSET(sd, & read_mask)) {

            //fai la recv
            if (recv(sd, buff, sizeof(buff), 0) < 0) {
                perror("recv\n");
                continue;
            }
            printf("Nome file: %s\n", buff);

            //risetto la mascehra in ascolto sulla socket
            FD_SET(sd, & read_mask);

            select(nfds, & read_mask, NULL, NULL, & time);

        }
        memset((char * ) & buff, 0, sizeof(buff));

    } else {
        printf("Il file non è stato trovato\n");
    }
    printf("\nInserisci nome Directory\n");
```

Il Server

Il Server gestisce i clienti TCP leggendo le directory e ricostruendo le sottodirectory, saltando cartella corrente “.” e cartella precedente “..” e alla fine invia il risultato al client.

Max: (2^16)-1 files per directory **FAT32**

Max: (2^32)-1 files per directory **NTFS**

```
while ((dir = readdir(d)) != NULL) {
        //printf("%s\n", dir -> d_name);
        if (strcmp(dir -> d_name, ".") == 0 || strcmp(dir -> d_name, "..") == 0) {
                //printf("cartella . o .. da saltare\n");
                continue;
        }

        //costruzione della sotto-directory
        strcpy(dirPath, nomeDir);
        strcat(dirPath, "/");
        strcat(dirPath, dir -> d_name);
        printf("sotto-directory: %s\n", dirPath);
        stat(dirPath, & stats);
        if (S_ISDIR(stats.st_mode)) {
                //apro la sotto-directory
                innerD = opendir(dirPath);
                //ciclo finche' leggo file
                while ((innerDir = readdir(innerD)) != NULL) {
                        //creo il path del file
                        strcpy(dirPath, nomeDir);
                        strcat(dirPath, "/");
                        strcat(dirPath, dir -> d_name);
                        strcat(dirPath, "/");
                        strcat(dirPath, innerDir -> d_name);
                        stat(dirPath, & stats);

                        if (strcmp(innerDir -> d_name, ".") && strcmp(innerDir -> d_name, "..") && !S_ISDIR(stats.st_mode)) {
                                //printf("found: %s\n", innerDir -> d_name);
                                write(conn_sd, innerDir -> d_name, 256);
                        }
                        //else printf("Cartella da saltare\n");

                }

        }

}
```

Il Server

Il Server gestisce i clienti UDP leggendo il file specificato e riscrivendo su un file temporaneo.

```
//leggo il file carattere per carattere
while (read(fd, & c, 1) > 0) {
    if (c == ' ' || c == '\n') {

        buf[i] = '\0';
        if (strcmp(parola, buf)) {
            //parola diversa strcmp=1
            //inserisco il terminatore nel buf e lo scrivo
            buf[i] = c;
            write(fd2, buf, i + 1);
            i = 0;
        } else {
            //caso in cui la parola sia stata trovata strcmp=0
            result++;
            i = 0;
            //scrivo solo il terminatore
            write(fd2, & c, 1);
        }
    } else {
        //se non incontro il terminatore salvo il carattere letto nel buffer
        buf[i] = c;
        i++;
    }

}

//controllo cio' che e' rimasto nel buffer
buf[i] = '\0';
if (strcmp(parola, buf)) {
    //parola diversa strcmp=1
    write(fd2, buf, i);
    i = 0;
} else {
    result++;
    i = 0;
}
```

Conclusioni

L’utilizzo della **select** permette di distinguere le richieste (UDP/TCP).

Per evitare di inviare al client TCP il numero di files in una directory, facciamo un ciclo con la select.

Il primo fd che viene controllato è quello del TCP, perché fa una **fork()**.