Università degli Studi di Bologna
Scuola di Ingegneria

Corso di
Reti di Calcolatori T

Esercitazione 4 (proposta)
Server Multiservizio: Socket C con select

Antonio Corradi, Luca Foschini
Michele Solimando, Giuseppe Martuscelli,
Marco Torello
Anno accademico 2019/2020

PROGETTO DI UN SERVER MULTISERVIZIO

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

Il primo servizio **elimina tutte le occorrenze di una parola in un file** presente sul server remoto,

il secondo servizio, **restituisce tutti i nomi dei file** (sul direttorio remoto) **presenti nei direttori di secondo livello**

Intendiamo il contenuto di tutti i direttori di primo livello contenuti nel direttorio

Il primo servizio viene realizzato utilizzando **socket senza connessione (datagram)**, mentre il secondo utilizzando **socket con connessione (stream)**

In particolare, si dovranno realizzare due client, e un server unico multiservizio (uso di select)

DETTAGLI CLIENT

Il **primo Client**, chiede ciclicamente all'utente il nome del **file e la parola**, invia al server la richiesta di eliminazione, e attende il pacchetto con l'esito dell'operazione, cioè il numero di **eliminazioni della parola** che stampa a video

Il **secondo Client**, chiede ciclicamente all'utente **il nome del direttorio,** invia al server la richiesta, **riceve la lista di nomi di file remoti che stanno nei direttori contenuti nel direttorio specificato**, e la stampa a video

SERVER MULTISERVIZIO: PARTE DATAGRAM

Il **Server** discrimina i due tipi di richiesta utilizzando la primitiva `select`

Le richieste di **eliminare le occorrenze di una parola** vengono gestite in maniera **sequenziale** diretta dal server senza introdurre una connessione e usando una **socket datagram**

Il server riceve il datagramma con il nome del file e la parola: se il file esiste, elimina le occorrenze della parola all'interno del file e invia al client l'intero positivo corrispondente alle eliminazioni fatte; altrimenti, in caso di errore, invia un intero negativo: ad esempio se il file non esiste

DETTAGLI SERVER: PARTE STREAM

Le richieste per ottenere **i nomi di file presenti nei sotto-direttori remoti di secondo livello**, vengono gestite in maniera **concorrente multiprocesso** con un **processo per ogni client** (usando **un'unica connessione per gestire l'intera sessione col client**)

Il server riceve il nome del direttorio; se il direttorio esiste restituisce il nome dei file contenuti in tutti i direttori di secondo livello

In caso di errore, il server deve notificare tale situazione al cliente, ad esempio se il direttorio remoto richiesto non esiste. In tal caso si richiede che **il server non chiuda la sessione**, ma, dopo la notifica di errore, si predisponga alla ricezione di una nuova richiesta

Il processo server chiude la comunicazione **solo alla ricezione della fine file dal client**

PROPOSTA DI ESTENSIONE: REALIZZAZIONE SHELL REMOTO

Si vuole sviluppare un semplice **shell remoto** per l'esecuzioni di comandi sul nodo server

In particolare, si vuole realizzare un'applicazione C/S che fornisca **due servizi**,

il **primo** realizzato utilizzando **socket senza connessione** (**datagram**), mentre

il **secondo** utilizzando **socket con connessione** (**stream**)

Si dovranno quindi realizzare **due client**, e un **server unico multiservizio** (uso di select)

PRIMO SERVIZIO: SOCKET DATAGRAM

Il primo servizio realizza l'**esecuzione remota di un comando** e il **recupero del valore di ritorno**. Si prevede un ciclo di:

- Invio da parte del client del **nome del comando** da eseguire e della **stringa degli argomenti** del comando;
- **Esecuzione remota del comando** sul server;
- stampa del **valore di ritorno** del comando sul client

Il secondo servizio realizza l'**esecuzione di un comando remoto** e il **recupero dell'output del comando** e prevede:

- Invio del client del comando da eseguire e della stringa con **gli argomenti del comando**;
- **Esecuzione remota del comando** sul server, con **ridirezione dello standard output** del comando sulla socket aperta con il client;
- **Stampa** a video dell'**output del comando** sul client

NOTE REALIZZATIVE: GESTIONE FIGLI

Suggerimento: Il **primo servizio** può essere gestito da **un processo figlio** con **attesa sincrona di terminazione**

Per ogni richiesta ricevuta, il server (padre) genera un **processo figlio** per l'esecuzione del comando richiesto (**`fork`**), recupera il process id (pid) del figlio generato e si mette in attesa del risultato di tale figlio effettuando una **`waitpid`**. Alla **terminazione del figlio**, il padre recupera il **valore di ritorno del comando** e lo spedisce al client; e poi si mette in attesa di una nuova richiesta da servire in ordine

Consiglio: si consulti il manuale (**`waitpid`**) per verificare a quale valore viene impostato il parametro di output (**`int *status`**) a fronte della terminazione di un figlio (in questo caso alla terminazione del comando eseguito dal figlio tramite l'invocazione dell'**`exec`**)

In particolare, esistono alcune macro per verificare le condizioni di terminazione del processo, vedi **`WIFEXITED`** e **`WEXITSTATUS`**

NOTE REALIZZATIVE: GESTIONE FIGLI (ANCORA)

Il **secondo servizio** viene gestito da un **processo figlio senza attesa di terminazione del padre** (**gestione multiprocesso**):
per ogni richiesta ricevuta il server genera un processo figlio.
Il padre, dopo aver lanciato il figlio, si mette in attesa di nuove richieste

Il figlio si occupa della ridirezione del **canale di output** sulla socket aperta con il client e dell'**esecuzione del comando richiesto** (exec)