Università degli Studi di Bologna
Scuola di Ingegneria

Corso di
Reti di Calcolatori T

Esercitazione 4
Focalizzazione conoscenze acquisite

Luca Foschini
Michele Solimando

Anno accademico 2016/2017

A - Trasferimento di una struttura dati

Organizzare e sviluppare **il trasferimento delle due strutture dati** seguenti in linguaggio C. Il cliente deve trasferire le strutture prestando attenzione ai campi che contengono.

```
typedef struct
{ char stringa[20];
  int intero;
  char carattere;
} struttura;
```

```
typedef struct
{ char* stringa;
  int intero;
  char carattere;
} struttura;
```

Si ragione sulle modalità di trasferimento delle strutture che contengono campi *puntatore*. Per la seconda struttura, ad esempio, il campo stringa va inviato usando un *separatore* '\0' oppure *inviando preventivamente la lunghezza del dato trasmesso*). **Si giustifichino di conseguenza le scelte progettuali fatte**.

A - Funzione Sleep()

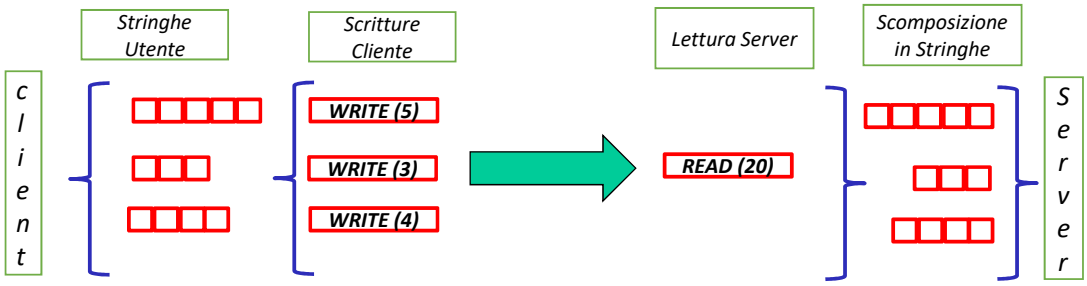
Per rallentare il programma, al fine di poter usare il terminale per esaminare lo stato delle risorse di rete, si può utilizzare la funzione **sleep()**, dichiarata nella libreria **<unistd.h>**.

int sleep (int seconds)

La funzione blocca l'esecuzione fino allo scadere dei secondi indicati oppure fino a quando arriva un segnale. Se la funzione termina dopo l'intervallo inserito, restituisce zero, se invece termina a causa dell'arrivo di un segnale, l'intero di ritorno rappresenta il tempo rimanente dell'intervallo di sleep.

A - Trasferimento stringhe

Organizzare e sviluppare **il trasferimento di stringhe da un pari ad un altro** in linguaggio C. Il client chiede all'utente di inserire stringhe di dimensioni massime prefissate. Ogni stringa viene inviata al server. Il server legge le stringhe tramite una variabile dalle dimensioni adatte a contenerne molte di quelle inviate dal client. Il server utilizza **un figlio per ogni cliente**.



Esempio con **DIM_MAX_STRINGA_CLIENT = 5** e **STRINGA_RICEZIONE_SERVER = 20.**

Prestare attenzione alla **ricezione**: è opportuno che il server *attenda,* prima di eseguire la read, per il tempo necessario all'utente per inserire tutte le stringhe. Inoltre, come mostrato nel disegno, non possiamo trattare la stringa letta come una unica stringa perché potrebbe contenerne diverse al suo interno. Occorre *separarle* opportunamente.

B - Chiusura Socket

Si verifichino gli effetti delle primitive **shutdown()** e **close()** sulla socket.

In particolare, si verifichi cosa avvenga a livello di rete dopo la chiamata a queste primitive, attraverso il comando **netstat** e l'utilizzo di **grep** per filtrare l'output relativo al processo che si vuole monitorare:

netstat –np | grep $pid

Monitorare, inoltre, l'avvenuta liberazione del **file descriptor** associato alla socket con il comando **lsof**:

lsof –p $pid | grep –w 'IPv6\|sock'

Quando viene liberato il file descriptor? Al momento della close() o dopo aver fatto lo shutdown dei canali di I/O?

C - Letture bufferizzate e tempi di trasferimento

Si imposti una funzionalità per il trasferimento di dati binari di dimensione variabile: ad esempio frame di una sequenza stream di dati. Ad esempio, con:

- **uso di un buffer** (di grosse dimensioni) per il trasferimento del singolo frame;
- **controllo sul numero di byte effettivamente letti** a fronte di scritture molteplici da parte del pari;
- **controllo del tempo di trasferimento** del file lato client e server.

C - Controllo del tempo di trasferimento

Per valutare il tempo di trasferimento (**Ttrasf**) bisogna prendere il tempo di sistema **prima** (ad es. **T1**) e **dopo** (ad es. **T2**) le parti di codice che realizzano il trasferimento file. Il tempo di trasferimento viene valutato approssimativamente come:

Ttrasf = T2 – T1

Come ottenere il tempo di sistema?

Varie possibilità (con risoluzione diversa) a seconda del sistema operativo e del linguaggio utilizzato.

Ad esempio, in Java, la funzione statica:

```
public static long System.currentTimeMillis();
```

restituisce “the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.”

C - Letture bufferizzate e tempi di trasferimento

In particolare, si realizzino i programmi client e server con le seguenti interfacce:

client serverHost serverPort bufferDim
server localPort bufferDim

Per ottenere il tempo di sistema in C si può utilizzare la funzione **gettimeofday** (definita in sys/time.h), che sfrutta le strutture:

```
// timeval salva il tempo, sempre dalle 00:00, dell'1/1/1970
// ha due campi, uno per i secondi (tv_sec) e uno per i microsecondi (tv_usec)
  struct timeval tv;
  struct timezone tz;
  gettimeofday(&tv, &tz);
tv.tv_sec; tv.tv_usec; // secondi e microsecondi
tv.tv_usec/1000;  // millisecondi
```

Si veda anche il **man di sistema.**

C - Letture bufferizzate e tempi di trasferimento

I programmi **client** e **server** devono stampare a video il tempo totale di trasferimento del file.
In particolare il server visualizza un Log di sistema: per ogni trasferimento vengono stampate le dimensioni, sia del buffer che del file trasferito, e il tempo impiegato.

Si effettuino un po' di prove, **cambiando le dimensioni dei buffer** dai due lati. Rispettivamente provare, per ogni file fornito, con buffer da 128 – 512 – 1024 – 2048 Byte e si esamini il Log fornito dal server.

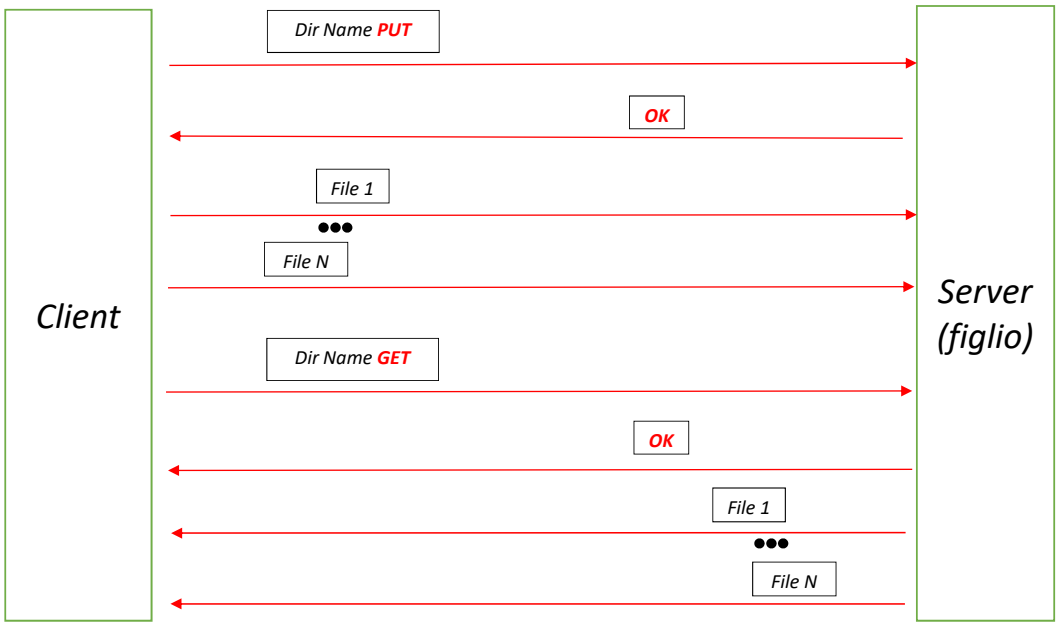
Quali considerazioni si possono fare?
Quale **dipendenza dalle dimensioni dei buffer** dai due lati?

D - Multiple PUT & Multiple GET

Si vuole sviluppare **un'applicazione C/S basata su socket con connessione** per il trasferimento di direttori. Un **cliente** deve eseguire due operazioni consecutive e obbligatorie (a meno di fallimenti). La prima consiste nell'inviare un intero direttorio al server (**mput**). Nella seconda deve chiedere un direttorio al server e scaricarlo in locale (**mget**).
Il **server** gestisce ogni cliente con **un solo figlio**, a cui delegare tutta l'interazione con quel cliente. Se il server riceve una richiesta di put per un direttorio esistente, nega il trasferimento, **senza sovrascrivere** nessun file.
Gestire il caso in cui il server riceve una get di un direttorio inesistente.

D - Protocollo di trasferimento

Si vuole realizzare il seguente protocollo.



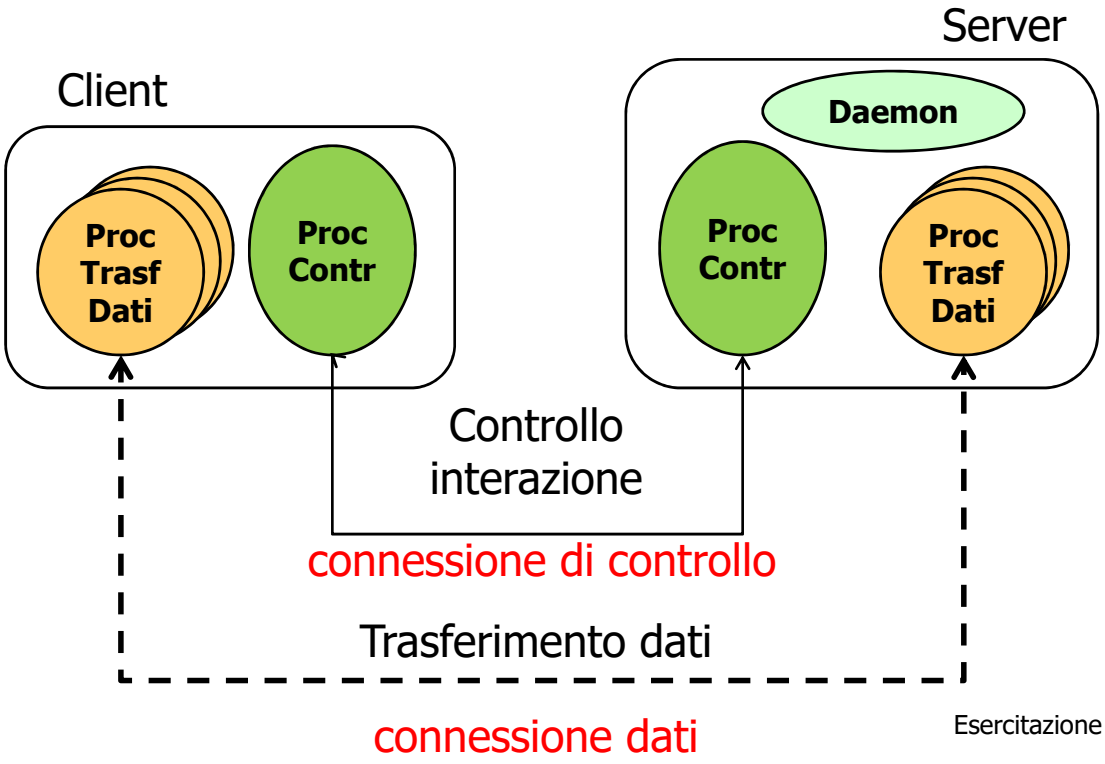
Proposta di estensione: Realizzazione servizio mget FTP

Si vuole sviluppare **un'applicazione C/S basata su socket con connessione** per il trasferimento di tutti i file di un direttorio remoto dal server al client (**multiple get**). Si dovranno realizzare i programmi **client** e **server**.

Si preveda **un'interazione sincrona (realizzata con una richiesta al processo di controllo lato server)** per trasferire il **nome del direttorio** da cui si devono prelevare i file da inviare; quindi, una seconda fase di **trasferimento dei file** dal server al client (realizzata da **processi figli** ulteriori con socket connesse).

Si vogliono realizzare due modalità di trasferimento, la prima con **client attivo** (**il client effettua la connect**), la seconda con **server attivo** (**il server effettua la connect**).

Proposta di estensione: Architettura



Trasferimento di file in direttori

Per ogni **Cliente**, il server prevede di creare un figlio, **processo di sessione** (chiamiamolo **processo di controllo del server**), a cui delegare tutta la interazione con quel cliente (**processo di controllo client**).

La **connessione di controllo** viene quindi usata tra il **processo di controllo server** e il **processo controllo cliente**. Questa deve servire per tutta la sessione di lavoro per preparare i trasferimenti dei file: ad esempio su questa connessione, diciamo **connessione di controllo**, passa il **nome del direttorio** da cui si devono trasferire i file; assumiamo il cliente ciclico che lavora fino alla fine del file di input.

Per ogni richiesta di direttorio del **cliente**, si vogliono **trasferire i file** attraverso **una nuova connessione dati** gestita da processi diversi (figli dei precedenti processi): il tutto va coordinato in modo corretto.

Trasferimento dei file

Per ogni **Direttorio**, il **processo di controllo del server** e il **processo di controllo client devono coordinarsi per il trasferimento dati creando una coppia di figli ad hoc.**

Si vogliono realizzare **due modalità** di trasferimento dei dati e una politica opportuna di gestione del trasferimento stesso.

- la prima con **client attivo** (**il client effettua la connect**),
- la seconda con **server attivo** (**il server effettua la connect**).

  Nel primo caso, il cliente deve **richiedere la connessione dati** (attraverso un connect) e il server deve **disporsi ad accettarla** (su una porta specificata).

  Nel secondo caso, il cliente deve **aspettare la richiesta di connessione** (attraverso una accept) e il server deve disporsi a **richiedere la connessione dati** (connect su una porta specificata dal client).

Trasferimento più direttori: Client Attivo

Per ogni richiesta di direttorio, il Cliente passa il **nome del direttorio** e aspetta la **porta di ascolto del server** (il numero di porta su cui connettersi) prima di attivare il **figlio per il trasferimento dati**.

Il **Client** richiede ciclicamente all'utente il **nome del direttorio** da cui trasferire i file e **comanda la operazione, ricevendo la porta di ascolto**, su cui dovrà attivare un **processo figlio** che **stabilisce una connessione** con il server remoto e riceve i file salvandoli nel direttorio locale in modo autonomo.

Il **Server** lavora come **processo di controllo (per quel client) che gestisce l'intera sessione**. Per ogni interazione, il processo di controllo, dopo **aver accettato la richiesta**, **crea una socket di ascolto** (su porta scelta) e su questa attiva un **processo figlio (processo trasferimento dati)**, dedicato a ricevere i file del direttorio; quindi il **processo di controllo server** restituisce il **numero di porta al processo di controllo del cliente corrispondente, in attesa per attuare la connessione**.

Trasferimento più direttori: Server Attivo

Per ogni richiesta di direttorio, il Cliente passa il **nome del direttorio** e la **porta di ascolto del processo trasferimento dati (figlio) in attesa** (il numero di porta su cui il processo di trasferimento dati server deve connettersi), dopo avere attivato il figlio con una certa porta per il trasferimento.

Il **Client** richiede ciclicamente all'utente **il nome del direttorio** da trasferire, **crea la socket di ascolto**, (delegandone la gestione ad un processo figlio), **poi effettua la chiamata per il trasferimento**.

Nel **Server** lavora il **processo di controllo (per quel client) che gestisce l'intera sessione**. Per ogni interazione, il processo di controllo, dopo aver **accettato una richiesta iniziale**, attiva un processo figlio a cui affida la **creazione della socket**, l'esecuzione della **connect** e il completamento del servizio richiesto.

Consegna

Chi vuole può inviare lo svolgimento ai docenti, con lo strumento specificato sul sito del corso.