RELAZIONE PROGETTO RETI 1

SERVER:

NB: Il server utilizza la funzione *pow(double, double)* della libreria *math.h* , di conseguenza quando lo si compila bisogna utilizzare -lm → gcc -o server server.c -lm

Il server dispone di alcune variabili globali che vengono utilizzate per mantenere lo stato della singola connessione; queste variabili vengono resettate alla chiusura della comunicazione per permettere al server di esser pronto per ogni nuova connessione.

Nello specifico le variabili sono:

- *auxToInt[]* → array ausiliario per permettere poi la memorizzazione dei dati in charToIntAry[] (e quindi la conversione di ogni carattere char in int). Qui vengono salvati, uno alla volta, i valori del buffer, in modo che poi possano essere convertiti in interi nell'array charToIntAry sfruttando la funzione atoi
- *charToIntAry[]* → array di elementi interi (il primo elemento è numEL, i restanti sono i valori di
  cui bisogna calcolare media e varianza)
- *counterNumEl* → conta il numero di valori di cui si deve fare la media (i valori TOTALI ricevuti fino ad ora)
- *numDatiLetti* → conta il numero di dati all'interno del buffer
- *numEl* → memorizza il primo valore del buffer (il numero di dati del messaggio appena ricevuto)
- *sum* → somma di tutti i valori per il calcolo della media
- *sumVariance* → somma di tutti i valori per il calcolo della varianza
- *varianza*
- *media*
- *firstMessage* → se il client ha già mandato un messaggio, il server non deve considerare il primo
  elemento del prossimo messaggio come uno degli elementi di cui fare la media (0 = il messaggio è il primo che il server riceve, 1 = il server ha già ricevuto altri messaggi dal client)
- *actualCounterNumEl* → conta gli elementi del messaggio appena ricevuto
- *indexCharInt* → indice dell'array di interi charToIntAry

NB:

- counterNumEl != numEl, perchè uno memorizza il primo elemento del buffer (numEL), l'altro (counterNumEl) conta i valori per cui bisogna fare la media. Inoltre, inizia da -1 perché nel conteggio fatto nel *if(isdigit(msg[i]) == 0)* della checkSyntax devo saltare il primo elemento del buffer (cioè numEL) che salvo in posizione 0 di charToIntAry.
- Allo stesso modo ActualCounterNumEl != counterNumEl, perché counterNumEl conta il numero di valori TOTALI, quindi considerando anche gli eventuali messaggi precedenti. Inoltre, serve per verificare che, per ogni messaggio, NumEl corrisponda ad ActualCounterNumEl.

Il server crea una socket (simpleSocket) e una porta (simplePort), mette in piedi la struttura dell'indirizzo e successivamente lega l'indirizzo e la porta con la socket; infine rimane in attesa sulla socket per eventuali connessioni

Il server utilizza una funzione, la **checkSyntax(char msg[])**, per valutare se il messaggio che ha appena ricevuto dal client sia corretto sintatticamente; la funzione ritorna 1 se il messaggio è corretto, 0 altrimenti.

La checkSyntax prende in input il messaggio (nello specifico, quando viene richiamata riceve in input il buffer) e lo elabora.

All'inizio verifica, tramite una serie di IF-ELSE se il massaggio è stato scritto nel modo corretto; in ordine verifica:

- che l'ultimo carattere del messaggio non sia uno spazio (**isspace(msg[strlen(msg)-1]) > 0**)
- che non ci siano segni di punteggiatura (**ispunct(msg[i]) > 0**)
- che il primo elemento non sia uno spazio (**isspace(msg[0]) > 0**)
- che non ci siano due spazi consecutivi (**isspace(msg[i - 1]) >0) && (isspace(msg[i]) >0**)
- che nel messaggio non ci siano caratteri speciali (**isalpha(msg[i]**).

Successivamente la funzione inizia a valutare la presenza di cifre; se ci sono devono essere memorizzate, tenendo conto del fatto che i valori >= 10 staranno in celle adiacenti del buffer e non tutti in una stessa cella.
Inoltre se il client ha già mandato un messaggio, dal secondo in poi non bisogna considerare il primo elemento come un elemento di cui bisogna fare la media dato che specifica quanti sono i valori, di quel messaggio, di cui si vuole calcolare media e varianza (**firstMessage == 1 && i != 0**).
I valori vengono inseriti nell'array di caratteri auxToInt, dove ad ogni "cella" dell'array corrisponderà una cifra del numero da salvare (Quindi ad ogni passo in auxToInt ci sarà solo un valore del buffer, che poi verrà inserito in charToIntAry).
Questa operazione continua finché non si incontra uno **SPAZIO** o, in caso di terminazione della stringa, **\0**, in tal caso il numero da memorizzare è terminato e quindi deve essere salvato nell'array di interi charToIntAry (sempre saltando il primo elemento nel caso di più messaggi successivi) facendo una atoi di auxToInt; quindi ora, in ogni cella di charToIntAry ci sarà un valore del messaggio.
La funzione incrementa, a seconda del caso, le tre variabili per il conteggio degli elementi actualCounterNumEl, counterNumEl e numDatiLetti.

Se la checkSyntax verifica che il messaggio è sintatticamente corretto, il server valuterà la corrispondenza tra i valori del buffer tramite una serie di IF-ELSE.

In ordine:

- se il numero di dati letti è corretto e maggiore di zero, memorizza i dati e risponde OK DATA <numero_dati_letti>
- se il numero di dati letti non è corretto, ovvero il valore dichiarato non è coerente con il contenuto del messaggio, il server risponde ERR DATA <messaggio>
- Se media e varianza dei campioni possono essere calcolati correttamente, il server calcola media e varianza dei dati memorizzati fin ora e trasmette OK STATS <numero_totale_di_campioni> <media> <varianza>
- Se media o varianza dei campioni non possono essere calcolati correttamente, il server trasmette ERR STATS <messaggio>

La connessione con il client si chiude (ma il server continua a rimanere attivo in attesa di nuove connessioni) se: 1) il numero di dati non è coerente con il valore dichiarato, 2) media e varianza vengono calcolate o 3) se media e varianza non possono essere calcolate correttamente.

Prima di chiudere la connessione con il client le variabili globali vengono resettate in attesa di nuove connessioni

CLIENT:

Il client crea una socket (simpleSocket), recupera il numero di porta per la connessione dai valori passati in input dall'utente creando la porta (simplePort), imposta la struttura degli indirizzi e poi connette l'indirizzo e la porta con la socket.

Se la connessione con il server va a buon fine, stampa il messaggio di benvenuto del server e mostra all'utente le regole.

Successivamente prende i dati che l'utente passerà in input da tastiera e li manda al server (se l'utente non inserisce dati, la richiesta viene ripetuta).

Il client quindi leggerà il buffer e, a seconda della risposta che avrà ricevuto dal server, mostrerà qualcosa di diverso all'utente, informandolo di ciò che è successo.
Per capire che cosa è successo, e quindi mostrare le informazioni corrette all'utente, il client elabora ciò che c'è nel buffer; nello specifico valuterà l'intestazione (le parole chiave del protocollo) del messaggio ricevuto dal server per valutare cosa mostrare all'utente.
Questo avviene tramite la funzione strncmp(const char *, const char *, size_t) che valuta la prima stringa passata in input con i primi n caratteri della seconda stringa passata in input
Quindi utilizzando questa funzione e una serie di IF-ELSE valuterà:

- if(strncmp("S: OK DATA", buffer, 10)==0)
  I dati inviati erano corretti e il server li ha salvati correttamente
- if(strncmp("S: OK STATS", buffer, 11) == 0)
  Il client ha inviato 0 e il server è riuscito a calcolare media e varianza, che verranno mostrate all'utente, estrapolandole dal buffer come fa il server nella checkSyntax quando deve estrapolare i valori passati dal client
- if(strncmp("S: ERR DATA", buffer, 11)==0)
  Il numero di dati inviati non era coerente con quello dichiarato come primo elemento del messaggio
- if(strncmp("S: ERR STATS", buffer, 12) == 0)
  Il client ha inviato 0 e il server non è riuscito a calcolare la media e la varianza correttamente
- if(strncmp("S: ERR SYNTAX", buffer, 11) == 0)
  la sintassi del messaggio inviato non era corretta

Questa procedura viene ripetuta finché o il server chiude la connessione o finché il client non invia il valore 0, al che il client chiuderà la simpleSocket.