Università del Piemonte Orientale - Vercelli - Reti

RELAZIONE PROGETTO

DESCRIZIONE

Il software è una semplice applicazione di rete sviluppata in linguaggio C, ed è composta in due parti: un “client” e un “server”. I programmi sono stati realizzati e testati sul sistema operativo Ubuntu 23.04, e sono stati compilati da terminale attraverso il comando “gcc file.c -Wall -Wextra -std=c11 -pedantic -g -o file”.
Il progetto ha lo scopo di calcolare la media e la varianza campionaria, in base al numero di dati che l’utente fornisce. Il client invia tramite uno o più messaggi dei numeri interi positivi su cui eseguire le operazioni, e quando il server li riceve esso calcola e memorizza la media e la varianza dei campioni. Una volta che il client inserisce il carattere “0”, il server restituisce la media e la varianza calcolate e termina la connessione.
Per poter comunicare il client ed il server utilizzano i protocolli IP e TCP, e per garantire l’interoperabilità tra applicazioni analoghe i programmi utilizzano dei messaggi specifici che rispettano un determinato formato. Per essere precisi, entrambi i programmi hanno un limite massimo di 512 caratteri per messaggio e terminano sempre con un carattere d’invio (\n); inoltre, i messaggi inviati dal client devono essere composti da un primo numero (che rappresenta il numero di dati nel messaggio) seguito dalla sequenza di dati (separati fra di loro con un singolo spazio), mentre il server deve inviare dei messaggi formati esclusivamente da caratteri maiuscoli, ed essi devono essere composti da un esito (OK o ERR), da un tipo di risposta (START, DATA, STATS, o SYNTAX) e da un contenuto, separati opportunamente da singoli caratteri di spazio. Quando il server invia un messaggio di errore al client la connessione viene interrotta ed il programma client termina, mentre il programma server (dopo aver stampato a terminale l’errore riscontrato) rimane sempre attivo per poter gestire le eventuali connessioni future.

SPECIFICHE DEL CLIENT

Per avviare il client bisogna conoscere l’indirizzo IP e la porta del server a cui ci si vuole connettere; perciò, per eseguire il programma, sono necessari tre argomenti: “./<nome programma> <indirizzo IP server> <porta del server>”.
Una volta avviato il programma inizializza alcune variabili e strutture, tra cui “simpleServer” (per memorizzare i dettagli dell'indirizzo del server) e “simpleSocket” (per il file descriptor del socket). Inoltre, viene chiamata la funzione “check_args” per assicurarsi che gli argomenti forniti al programma siano corretti. A seguire viene creato il socket tramite la relativa funzione e, in caso di insuccesso, il programma stampa un messaggio di errore e termina con il codice “EXIT_FAILURE” (1). L'indirizzo IP e la porta del server vengono poi impostati nella struttura “simpleServer” e un tentativo di connessione al server viene effettuato tramite la funzione “connect”.
Se la connessione avviene con successo la funzione “write_read” viene chiamata per gestire la comunicazione con il server, dopodiché il socket viene chiuso e il programma termina con successo con codice “EXIT_SUCCESS” (0). Nel caso la connessione fallisse, come per il controllo precedente viene stampato un messaggio di errore, il socket viene chiuso e il programma termina con uno stato di fallimento.
Come definito precedentemente la funzione “main” chiama due funzioni, “check_args” e “write_read”, e queste funzioni hanno il seguente comportamento:

- **Check_args**: è una funzione che prende come parametri il conteggio degli argomenti passati al programma (argc) e un array di stringhe che contiene gli argomenti stessi (argv). La funzione inizia con la dichiarazione di due variabili, “checkarg” (utilizzato come puntatore per controllare i caratteri dell’argomento) e “arg”, che è un puntatore all'argomento specifico da controllare, ovvero la porta del server. Lo scopo della funzione, infatti, è molto semplice: controlla se il numero degli argomenti forniti è esattamente tre, controlla se la porta fornita contiene solo caratteri numerici, e controlla se quel valore numerico è compreso tra 1024 e 65535.
  Se i controlli passano con successo la funzione non fa nulla, altrimenti viene stampato un messaggio di errore e l'esecuzione del programma viene terminata. In questo modo, grazie ai messaggi di errore dettagliati, l'utente ha una chiara indicazione su come avviare il client in maniera corretta;

- **Write_read**: è la funzione principale del client. Implementa un ciclo continuo di scrittura e lettura con il server, gestisce gli errori di input dell'utente e di comunicazione con il server, e permette all'utente di terminare l'interazione quando desidera.
  Write_read prende come argomenti “*returnStatus” e “simpleSocket” per gestire l’invio e la ricezione dei messaggi con il server, e poi comincia con la dichiarazione di diverse variabili, tra cui “buffer”, che è un array di caratteri di dimensione “MAX_LENGTH_BUFFER” (costante predefinita con il valore 513) che permette di memorizzare 512 caratteri più il terminatore di stringa “\0”. Variabili numeriche come “check”, “repeat”, e “first_time” sono inizializzate per mantenere lo stato del processo, mentre “num_data” e “total_data_count” sono utilizzate come contatori interni per il numero di dati inviati, al fine di eseguire controlli di validità sulle risposte ricevute dal server.
  Il corpo principale della funzione si svolge all'interno di un ciclo “do-while”, che continua fino a quando un valore diverso da “1” viene assegnato a “*returnStatus” (ovvero 0), andando a indicare un errore o una richiesta dell'utente di terminare la connessione. Quando il ciclo “do-while” viene eseguito per la prima volta, il client si aspetta di ricevere un messaggio di benvenuto dal server; perciò, nel primo ciclo viene saltata la prima parte e si esegue solamente la funzione “read”. Viceversa, se non è la prima volta che il ciclo viene eseguito, si procede con l’invio del messaggio, e quindi l'utente viene sollecitato a inserire un messaggio da inviare al server. Una volta che l’utente ha inserito i dati, si verifica la lunghezza dell'input e ci si assicura che non superi la lunghezza massima consentita. Se l'input è troppo lungo, viene stampato un messaggio di errore e si invita l’utente a inserire nuovamente il messaggio tramite un ulteriore ciclo “do-while”. Una volta che nel buffer è stato inserito un messaggio di dimensioni accettabili, viene richiamata la funzione “valid_input” per testare ulteriormente il formato dell'input. Se il messaggio passa le verifiche di “valid_input” allora si procede ad inviare il messaggio al server con la funzione “write”, altrimenti si continua a richiedere all’utente di inserire un input valido.
  Successivamente, quando il messaggio è stato inviato correttamente al server, ci si prepara a leggere la risposta del server. Il buffer viene pulito tramite la funzione “memset” e si procede a leggere il messaggio con la funzione “read”, che verrà salvato nel buffer. Se la lettura ha successo, la funzione “parse_and_print_message” viene utilizzata per analizzare la risposta del server e stamparla correttamente a schermo. Questa funzione gestisce anche l'aggiornamento di “*returnStatus” in base alla risposta ricevuta. Nel caso in cui la lettura fallisca, la funzione segnala un errore, imposta “*returnStatus” a “1” si prepara per terminare il programma. La stessa cosa avveniva anche con il fallimento della scrittura, solo che in quel caso “*returnStatus” veniva impostato a “-1” in modo tale da evitare l’operazione di lettura e passare direttamente all’interruzione del programma.

- **Valid_input**: viene richiamata all'interno di "write_read", e ha il compito controllare l'input dell'utente prima che venga inviato al server. La funzione prende come parametri il messaggio dell'utente, "*num_data" per il controllo del numero dei dati, e "*returnStatus" nel caso fosse prevista la terminazione del programma.
  Inizialmente la funzione calcola la lunghezza dell'input, dopodiché esegue una serie di controlli per controllare il formato. Prima di tutto si controlla se il primo carattere dell'input è "0", che viene interpretato come un comando per terminare l'applicazione. Se questo è il caso, controlla che l'input "0" sia seguito dall'invio (carattere di nuova linea "\n"). Se l'input è corretto "*returnStatus" viene impostato a "1" e la funzione ritorna il valore "0", altrimenti viene stampato un messaggio di errore e il valore di ritorno viene impostato a "1".
  Se il messaggio dell'utente non è destinato a ricevere le statistiche del server e a terminare la connessione, si effettuano una serie di controlli per verificare che i dati inseriti rispettino il formato prestabilito, e in caso contrario viene stampato, come in precedenza, un messaggio di errore specifico, e la funzione ritorna il valore "1" per far ripetere l'input all'utente.
  Il primo controllo si assicura che la dimensione minima dell'input sia di almeno quattro caratteri e che termini con l'invio, (perché il messaggio più corto che l'utente può inserire è, per esempio, "1 7\n"). Dopodiché, i controlli successivi verificano che il primo e il penultimo carattere siano numerici, andando a stampare un errore specifico nel caso ci fosse uno spazio o un altro carattere non ammesso. Successivamente la funzione esamina, tramite un ciclo "for", ogni carattere dell'input (eccetto l'invio), andando a verificare che dopo ogni spazio sia presente un numero, e che l'input contenga solamente numeri e spazi. Nel ciclo "for", inoltre, vengono contati gli spazi al fine di eseguire gli ultimi controlli, che permettono di verificare la presenza degli spazi e la correttezza del numero dei dati.
  Nell'ultima verifica "*num_data" viene impostato al primo numero che l'utente ha inserito, in modo da utilizzarlo come variabile di controllo nella funzione "parse_and_print_message". Dopo aver verificato che il numero di dati, inserito dall'utente, corrisponde effettivamente a quello inserito, la funzione termina e restituisce il valore "0".

- **Parse_and_print_message**: ha lo scopo di analizzare e stampare i messaggi ricevuti dal server. Questa funzione prende in ingresso quattro puntatori: uno per il messaggio del server, un puntatore alla variabile “*returnStatus” (che viene impostato a 1 in caso di errore), un puntatore “*num_data” (che rappresenta il numero di dati inviati con il messaggio precedente) e un puntatore che conta il numero totale dei dati inviati.
  La funzione inizia verificando la struttura del messaggio con una chiamata alla funzione “check_message_structure”. Se la struttura del messaggio non è valida, il flag “returnStatus” viene impostato a 1 e la funzione ritorna immediatamente. Dopo il primo controllo vengono dichiarate alcune variabili, come “*message_type” e “*message_subtype”, che verranno utilizzate per salvare l’esito e il tipo di risposta del messaggio ricevuto. Il procedimento viene applicato con la funzione “sscanf”, e all’inizio del messaggio devono essere presenti solo quei due messaggi, altrimenti viene stampato un messaggio di errore e la funzione termina con “returnStatus” a 1. A questo punto la funzione inizia a identificare il messaggio in base all’esito. Se l’esito del messaggio equivale a "OK" si genera una stringa di ricerca con funzione “sprintf”, composta da “OK” seguito da uno spazio più il tipo di risposta seguito da un ulteriore spazio. La stringa di ricerca viene utilizzata nel caso in cui “*message_subtype” equivalga a "START", ma ha anche lo scopo di verificare la presenza dei singoli spazi separatori che devono separare le varie parti del messaggio. Se si conferma la presenza della stringa "OK START ", il puntatore del messaggio viene incrementato con la lunghezza della stringa di ricerca, in modo che punti all’inizio del contenuto del messaggio, e una volta stampato il messaggio di benvenuto vengono stampate anche le istruzioni per l’utilizzo del programma. Con "DATA", invece, il numero di controlli è inferiore rispetto a “START”. Infatti “OK DATA <numero>” è solo un messaggio di conferma, perciò basterebbe controllare che il <numero> del messaggio sia uguale a “*num_data” e stampare a schermo un messaggio di successo per l’invio del dato/dei dati. Nel caso di "STATS", invece, bisogna eseguire più controlli: bisogna innanzitutto verificare che gli argomenti di STATS siano tre (numero totale di dati inviati, media e varianza), poi bisogna verificare che il numero totale di dati che il server ha ricevuto sia pari a quello inviato dal client, ed infine bisogna verificare che la media e la varianza non siano negative, e che la varianza sia maggiore di zero quando la media è pari a zero; in assenza di errori vengono stampati i dati del messaggio in un formato opportuno. Il messaggio, al posto dell’esito “OK”, potrebbe cominciare con i caratteri "ERR", e in quel caso bisognerebbe verificare che il tipo di risposta sia, esclusivamente, "DATA", "STATS" o "SYNTAX". Superato il controllo precedente, viene generata una stringa di ricerca contenente l’esito e la risposta del messaggio, separati da uno spazio e con un ulteriore spazio finale dopo la risposta. Come per l’esito “OK”, si utilizza funzione “ strstr “ per verificare la presenza della stringa di ricerca nel messaggio, e per controllare che gli spazi separatori siano presenti. Se la stringa viene trovata il puntatore del messaggio viene incrementato per la lunghezza della stringa di ricerca, in modo che esso punti all’inizio del suo contenuto (ovvero all’inizio del messaggio), che verrà stampato nell’istruzione successiva.

- **Check_message_structure**: è una funzione che ha il compito di verificare alcune regole di base che la struttura del messaggio deve rispettare; come parametro riceve la costante “const char * input “, ovvero il messaggio in input non modificabile.
  La prima operazione eseguita dalla funzione è il controllo della lunghezza del messaggio. La lunghezza viene determinata usando la funzione “strlen”, e si controlla che essa sia maggiore di 10 e che, nella posizione “length - 1” della stringa, sia presente il carattere “\n”. Successivamente la funzione controlla, tramite “strncmp”, se il messaggio inizia esclusivamente con "OK" o "ERR". Se il messaggio passa anche questo controllo, si utilizza un ciclo “for” per contare il numero di spazi, e si verifica che nel messaggio ce ne siano almeno due (ovvero il numero minimo di spazi per separare le varie componenti del messaggio). Se il messaggio passa tutti i controlli, alla fine della funzione viene ritornato il valore 1, altrimenti viene stampato a schermo un messaggio di errore specifico e la funzione ritorna il valore 0.

SPECIFICHE DEL SERVER

Il programma server, per essere avviato da terminale, deve avere i seguente argomenti: “./<nome programma> <numero porta>”. Una volta inseriti, nella funzione “main” vengono inizializzate diverse strutture e variabili, tra cui “sockaddr_in” (per il server e il client), il socket, la lunghezza del nome del client, la porta e altre variabili per verificare lo stato di ritorno delle chiamate di sistema. Prima di tutto, viene chiamata la funzione “check_args” per controllare gli argomenti passati al programma; poi viene creato un nuovo socket utilizzando la funzione “socket”, e se questa operazione fallisce, viene stampato un messaggio di errore e il programma termina con il codice “EXIT_FAILURE” (1). Se il programma non termina, la porta in input presa dall’argomento viene convertita in un intero con la funzione “atoi”, e successivamente la struttura del server viene configurata nel seguente maniera: la famiglia di protocolli viene impostata su “AF_INET”, l'indirizzo viene impostato su “INADDR_ANY”, e la porta viene convertita in formato di rete e impostata nella struttura del server. Dopodiché viene eseguita la funzione “bind” per collegare il socket all'indirizzo e alla porta specificati, e anche in questo caso, se l’operazione non va a buon fine, viene stampato un messaggio di errore e il socket, insieme al programma, vengono terminati. Una volta completato il binding, il server inizia ad ascoltare le connessioni con la funzione “listen”, e in caso di errore si eseguono le stesse operazioni del “bind”. Viene poi stampata l'ora locale tramite la funzione “print_localtime”, ed infine il programma entra in un loop infinito in modo da rispondere a tutte le richieste di connessioni da parte dei client. Infatti, in questo ciclo il server utilizza la funzione “accept” per stabilire la connessione, e dopo aver eseguito gli stessi controlli del “listen” e del “bind” viene eseguita la funzione “read_write” per analizzare i messaggi del client, elaborarli ed inviare una risposta attraverso il socket. Al termine della funzione “check_args” anche la connessione con il client viene terminata, e il server torna ad attendere nuove connessioni.
Anche in questo caso la funzione main richiama diverse funzioni, ovvero “check_args”, “print_localtime” e “read_ write”, e “read_ write” a sua volta richiama altre funzioni per eseguire le varie funzioni descritte precedentemente. Nello specifico queste funzioni eseguono le seguenti operazioni:

- **Check_args**: come per il client, “check_args” è una funzione che prende come parametri il conteggio degli argomenti passati al programma (argc) e un array di stringhe che contiene gli argomenti stessi (argv). Anche in questo caso vengono eseguiti gli stessi controlli del client, perciò si controlla che la porta contenga solamente caratteri numerici e che il suo valore sia compreso tra 1024 e 65535; l’unica differenza rispetto al client è che si controlla solamente la presenza di due argomenti, perché il server non ha bisogno di avere l’indirizzo IP come argomento.

- **Print_localtime**: è una funzione che stampa la data e l'ora, e viene utilizzata anche per generare un messaggio di benvenuto. Essa prende come parametri il puntatore del buffer e la sua lunghezza, che vengono utilizzati solamente quando la funzione viene richiamata in "read_ write". La funzione utilizza le chiamate al sistema "time" e "localtime" per ottenere il tempo corrente in secondi dal 1970 (Epoch Unix) e per convertirlo in una struttura "tm" locale. Successivamente viene impostata la localizzazione del tempo in italiano usando "setlocale" e viene inizializzato un buffer locale 'date' con tutti i caratteri nulli attraverso la funzione "memset", dopodiché si verifica se il buffer e la sua lunghezza abbiano, rispettivamente, i valori "NULL" e "0". Se i parametri sono nulli la data e l'ora vengono salvati in "date" con la funzione "strftime", in modo da stampare a schermo la data e l'ora in cui è stato avviato il server; altrimenti, in caso contrario, si utilizza sempre la funzione "strftime" per generare il messaggio di benvenuto nel formato "OK START <messaggio>" e la data e l'ora vengono inseriti all'interno del <messaggio>.

- **Read_write**: come per "write_read" nel client, "read_write" è la funzione principale del programma server. Essa riceve i messaggi dal socket, elabora i dati ricevuti e invia le risposte al client. L'esecuzione della funzione continua fino a quando non viene ricevuto il messaggio di terminazione "0\n", o fino a quando non si verifica qualche tipo di errore. Riceve due argomenti: un puntatore a un intero "returnStatus" e un identificatore di socket "simpleChildSocket".
  Dopo aver inizializzato le variabili e impostato il valore puntato da "returnStatus" a 0, la funzione entra in un ciclo "do-while" che termina solo quando "returnStatus" assume il valore 1. All'inizio del ciclo si "pulisce" il buffer con la funzione "memset", e nella prima iterazione viene scritto un messaggio di benvenuto al suo interno, tramite la funzione "print_localtime". Il programma, invece, si mette in attesa di un messaggio da parte del client, se non è la prima volta che il ciclo viene eseguito. Una volta ricevuto il messaggio si controlla se esso possa contenere i caratteri "0\n", e in quel caso si procede all'invio delle statistiche calcolate; mentre se non si riesce a leggere sul socket "returnStatus" viene impostato a "-1" in modo da evitare la scrittura sul socket. Quando si riceve il carattere di terminazione, oltre a verificare la presenza dell'invio (\n) si controlla anche che siano stati inseriti come minimo due dati, e si verifica che la media e la varianza siano abbastanza piccoli da poter essere salvati all'interno di un messaggio di 512 caratteri. Comunque, se al posto del carattere di terminazione il messaggio ricevuto contiene i dati, allora deve essere richiamata la funzione "process_input" per controllare i numeri e aggiornare le statistiche, e "returnStatus" assumerà il valore opportuno a seconda del successo delle operazioni. Infine, dopo avere eseguito la lettura del messaggio, la funzione "write_read" esegue il blocco di codice per inviare la risposta al client, risposta che è memorizzata nel buffer. Se la scrittura sul socket fallisce viene stampato, anche in questo caso, un messaggio di errore, e "returnStatus" viene impostato a "1" andando così a terminare il ciclo "do-while" e la connessione.

- **Process_input**: serve per controllare la semantica del messaggio, e permette di aggiornare i valori della media e della varianza. Come parametri prende quattro puntatori in input: “char *input” (il messaggio ricevuto dal client), “int *total_data_count” (il conteggio totale dei numeri ricevuti fino a quel momento), “double *mean” (la media dei numeri ricevuti fino a quel momento), e “double *m2” (la somma della differenza di quadrati tra i dati e la media). La funzione inizia controllando se l'input ricevuto è sintatticamente valido attraverso la funzione ausiliaria “valid_input”. Se l'input non è valido la funzione restituisce immediatamente “1”, mentre in caso contrario viene rimosso il carattere di ritorno a capo (\n) dal messaggio. Dopo aver estratto il primo numero dall'input (che rappresenta il numero totale di numeri contenuti nel messaggio) la funzione inizia a suddividere il messaggio in vari pezzi, detti "token", dividendoli attraverso gli spazi. Ogni token viene poi controllato per verificare se rappresenta un numero più grande rispetto al valore massimo rappresentabile con una variabile di tipo “double”, e come sempre in caso di errore si invia al client il messaggio specifico, terminando la funzione con il valore “1”. Una volta che il numero in caratteri è stato convertito, viene implementato l'algoritmo di Knuth (basato sulla formula di Welford) per calcolare la media e la varianza dei numeri. Durante l’esecuzione dell’algoritmo vengono implementati vari controlli per verificare se le somme o le sottrazioni possano generare un “overflow” o un “underflow”. Infine, dopo aver anche controllato il numero totale dei numeri processati con quello previsto, la funzione genera un messaggio di conferma, lo invia al client, aggiorna il conteggio totale dei numeri, calcola la media e la varianza campionaria e restituisce il valore 0, indicando che le operazioni sono andate a buon fine.

- **Valid_input**: questa funzione è progettata per validare la sintassi dell'input ricevuto. Il parametro passato alla funzione è un puntatore a stringa “char *input”, che punta al messaggio inviato dal client. Il codice della funzione esegue una serie di controlli per assicurarsi che l'input ricevuto rispetti una serie di regole ben definite.
  I primi controlli si effettuato sulla lunghezza dell'input (che deve avere un valore minimo pari a quattro) e sulla presenza del carattere di invio (\n) nell’ultima posizione del messaggio; in seguito, la funzione verifica se il primo e il penultimo carattere del messaggio siano esclusivamente numerici, e successivamente si controlla il resto dei caratteri.
  Nel messaggio devono essere presenti soltanto numeri e spazi, e il carattere successivo allo spazio deve essere esclusivamente numerico; se queste condizioni non venissero rispettate verrebbe generato un messaggio di errore.
  Infine, ci si assicura la presenza di almeno uno spazio nell'input per distinguere il numero di dati e i dati stessi, attraverso una variabile inizializzata precedentemente per contare gli spazi.
  In caso di qualsiasi errore esso verrebbe stampato a terminale, e il buffer che contiene il messaggio verrebbe modificato con l’errore riscontrato, con il tipo di errore che potrebbe assumere un formato specifico in base alla presenza del carattere spazio o alla presenza di caratteri differenti da quelli consentiti; in ogni caso, dopo il messaggio di errore, la funzione restituisce il valore “0”, mentre se passa i controlli e si arriva alla fine essa ritorna il valore “1”.