

UNIVERSITA’ DEGLI STUDI DI NAPOLI

“FEDERICO II”

LABORATORIO SISTEMI OPERATIVI

Prof. CLEMENTE GALDI

PROGETTO: cORRIERE ESPRESSO





MARCO CARROZZO N86/

MAURIZIO DEL PRETE N86/783

INDICE

1. **Guida d’uso**

Modalità di compilazione........................................................................................Pag.

* + - Lato Client
    - Lato Server

Utilizzo Server..........................................................................................................Pag.

Utilizzo Client...........................................................................................................Pag.

1. **Comunicazione tra Client-Server**

Socket “Server”………...............................................................................................Pag.

Socket “Client” ...……………………………………………………………………………………………….Pag.

Protocollo di Comunicazione tra Client-Server .………………………………………………….Pag.

1. **Dettagli implementativi**

Scelte implementative ...........................................................................................Pag.

1. **Appendice**

Codice Client ..........................................................................................................Pag.

Codice Server …………………………………………………………………………………………………….Pag.

**1 - GUIDA D’USO**

**Modalità di compilazione** *(da rivedere)*

Il progetto si compone di quattro file: tre file con estensione .c (server.c, client.c e library.c) e un file con estensione .h (library.h)

Compilazione Client

La compilazione lato client dovrà essere effettuata nel seguente modo:

“………………………………………”

Se il processo di compilazione è andato a buon fine il compilatore non invierà alcun messaggio

all'utente.

Compilazione Server

Il server è un’applicazione multithread e in quanto tale il processo di compilazione differisce, bisogna aggiungere il flag –pthread

“………………………………………”

Se il processo di compilazione è andato a buon fine il compilatore non invierà alcun messaggio

all'utente.

**Utilizzo Server**

Una volta compilato correttamente il file server.c è possibile eseguire il file di output ottenuto utilizzando

la sintassi:

“./…...... <N> < K> < S>”

Dove N definisce il numero massimo di operatori attivi, K definisce il numero di oggetti assegnati inizialmente ad ogni operatore ed infine S che identifica il nome del file contenente la lista degli oggetti da consegnare

(vanno aggiunti tutti i screenshot che descrivono l’utilizzo)

Screenshot:

**Utilizzo Client**

Una volta lanciato il server è possibile collegarsi con uno o più client. Questi per potersi connettere deveno

conoscere l’indirizzo e la porta, utilizzando la sintassi:

“./…...... <iP> <porta>”

(vanno aggiunti tutti i screenshot che descrivono l’utilizzo)

Screenshot:

**2 - COMUNICAZIONE CLIENT-SERVER**

Per effettuare la comunicazione tra client e server, abbiamo utilizzato i socket.

Il socket è un canale di comunicazione asimmetrico tra due processi in esecuzione sullo stesso calcolatore o su due computer diversi interconnessi da una rete di comunicazione.

**Socket “ Server ”**

La creazione del socket avviene tramite la system call ***“int socket(int famiglia, int tipo, int protocollo)”***, nel nostro progetto la famiglia alla quale appartiene il protocollo è la “AF\_INET”(protocolli per internet), il tipo di connessione specifico è “SOCK\_STREAM”(socket con connessione locale o TCP) ed infine il protocollo che tipicamente viene settato a 0.

Se la system call Socket ha esito positivo, restituisce il sockfd (SocketDescriptor) altrimenti -1 in caso di errore.

Creato il socket associamo un indirizzo tramite la system call ***“int bind(int SocketDescriptor, struct sockaddr \*my\_add, socklen\_t addrlen)”***. Il primo parametro è sockfd precedentemente creato, come secondo parametro viene passato un puntatore alla struttura sockaddr che rappresenta l’indirizzo da associare a sockfd ed infine come terzo parametro viene passato la dimensione del secondo parametro.

La system call Bind restituisce il valore di ritorno che viene memorizzato nella variabile bindErr, se il valore è 0 l’associazione è avvenuta con successo, invece -1 in caso di errore.

Il successivo passo è quello di mettere in ascolto il canale di comunicazione attraverso la system call ***“int listen(int SocketDescriptor, int lunghezza\_coda)”***. Il primo parametro è sockfd, il secondo parametro specifica quante connessioni possono essere in attesa di essere accettate, questo parametro “operatorsNumber” viene inserito all’avvio del server

Il tutto è contenuto dalla function ***“int InitSocket (struct sockaddr\_in \*server, int port, int operatorsNumber)”***

int InitSocket (struct sockaddr\_in \*server, int port, int operatorsNumber) {

//creo il socket

int sockfd = socket(AF\_INET , SOCK\_STREAM , 0);

if (sockfd == -1)

{

char mess[] = "Non è possibile creare il socket";

write(STDOUT\_FILENO, mess, strlen (mess));

}

puts("Socket creato");

//ma il server che indirizzo ip deve avere?!

//preaparo la struttura sockaddr\_in che contenga le informazioni di connessione

server = (struct sockaddr\_in \*) malloc (sizeof (struct sockaddr\_in));

server->sin\_family = AF\_INET;

server->sin\_addr.s\_addr = INADDR\_ANY;

// server->sin\_port = htons( 8888 );

if (port == -1)

server->sin\_port = htons( 8888 );

else

server->sin\_port = htons ( port );

//effettuo la Bind e gestisco l'eventuale errore

//questo ciclo tenta la bind nel caso in cui non la porta non sia libera

int bindErr;

do {

bindErr = bind(sockfd,(struct sockaddr \*)server , sizeof(struct sockaddr\_in));

int errsv = errno;

if (bindErr < 0) {

if ( /\* errsv == EINVAL || \*/ errsv == EADDRINUSE )

perror ("Bind > Error3: Nuovo tentativo di assegnazione tra 5 secondi"), sleep (5);

else

perror ("Bind > Error4: exiting...."), exit (-1);

} else if (bindErr == 0)

write (STDOUT\_FILENO, "Bind eseguita.\n", strlen ("Bind eseguita.\n"));

} while (bindErr < 0);

//avvio la Listen su un numero N di operatori

listen(sockfd , operatorsNumber);

char portH[6];

sprintf (portH, "%d\n", port);

char \*message = "In attesa di connessioni...su porta: ";

write (STDOUT\_FILENO, message, strlen (message));

write (STDOUT\_FILENO, portH, strlen (portH));

return sockfd;

}

# L’accettazione di una connessione viene effettuata attraverso la system call ***“int accept(int SocketDescriptor, struct sockaddr \*IndirizzoClient, socklen\_t \*DimensioneIndirizzo)”***, con il primo parametro specifichiamo il socket in ascolto “sockfd”, il secondo e terzo parametro vengono ritornati dall’accept e contegono le informazioni sul client connesso (indirizzo iP e Porta), possono essere settati a NULL.

Il server attraverso un ciclo while accetta richieste da parte dei client

while( (client\_sock = accept(sockfd, (struct sockaddr \*)&client, (socklen\_t\*)&clientsize)) )

{

if( pthread\_create( &thread\_id , NULL , connection\_handler , (void\*) &client\_sock) < 0)

{

perror("could not create thread");

return 1;

}

//join dei thread, per evitare che il master thread chiuda prima del termine dei figli.

pthread\_join( thread\_id , NULL);

puts("Handler assigned");

}

**Socket “ Client ”**

# **Protocollo di comunicazione tra Client-Server**

Creato un canale di comunicazione tra Client-Server, passiamo a descrivere il **protocollo di comunicazione**

**Caso 1**: Il Client si connette al Server e riceve una lista iniziale di K stringhe, così definite:

* Id#descrizione#indirizzo
  + Il server all’atto dell’invio modifica lo stato dell’oggetto nel suo database in “in consegna”.
  + Il client imposta lo stato nella lista locale dell’oggetto in “in consegna”.

**Caso 2**: Il Client invia il comando *elencaserver*, ed il server invia tutte le stringhe in memoria:

* Elencaserver#id#descrizione#indirizzo#stato ????

(non dovrebbe inviare le stringhe del tipo id#descrizione#indirizzo#stato)

**Caso 3**: Il Client invia il comando *ritirato#codice#descrizione#indirizzo*

* + Il server aggiunge l’oggetto nel suo database, ponendo lo stato “ritirato”.
  + Il client aggiunge l’oggetto nella sua lista locale, ponendo lo stato “ritirato”.

**Caso 4**: il Client invia il comando *smista#codice*

* + Il server modifica lo stato dell’oggetto nel suo database in “in magazzino”.
  + Il client elimina l’oggetto nella sua lista locale.

**3 - DETTAGLI IMPLEMENTATIVI**

**Scelte implementative**

( prendere dal file progetto “strutture e funzioni ”)

**APPENDICE**

**CODICE SORGENTE**