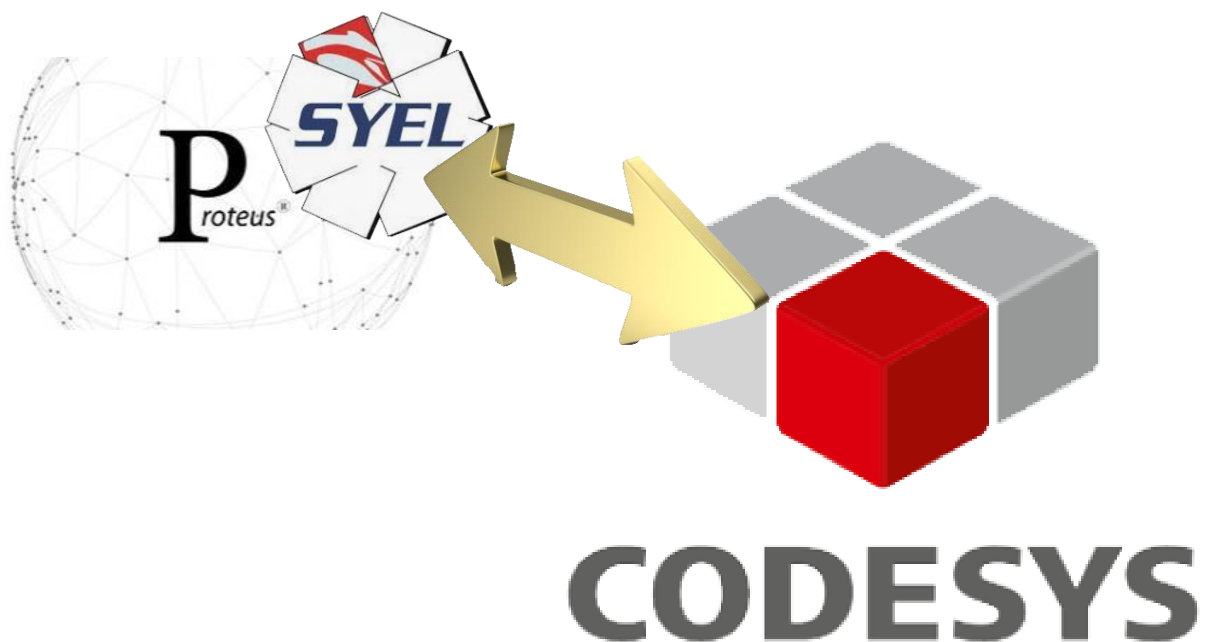
Breve nota sulla comunicazione fra programma realizzato in ambiente Codesys e programma realizzato in ambiente Proteus sui dispositivi Syel basati su Raspberry PI

09-02-2023

Draft, version 1.0



Preambolo

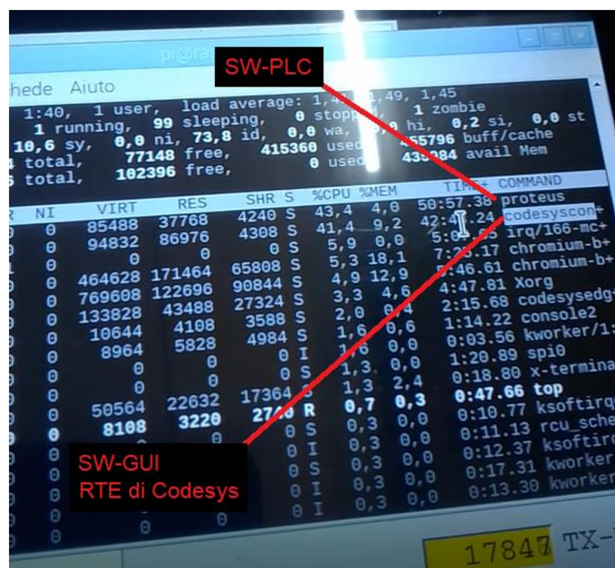
A partire dai nostri dispositivi HMI basati su Raspberry Pi CM (Compute Module) 3 e 4, LCD dai 7 pollici in su (7, 10, 15), abbiamo la possibilità di programmare le applicazioni di lavoro, che girano su tali HMI, anche con il famoso ambiente di sviluppo Codesys.
Naturalmente è rimasta la possibilità di continuare a sviluppare i programmi di lavoro con l'ambiente di sviluppo Proteus, IDE sviluppato da Syel.
Ciascuno dei 2 ambienti di sviluppo consente di implementare un certo set di funzionalità, e chiaramente ciascuno di questi 2 IDE ha i propri punti di forza.
Tuttavia questi non sono mutuamente esclusivi: è pertanto possibile sviluppare un'applicazione in ambiente Proteus, un'altra in ambiente Codesys, traendo il meglio dall'uno e dall'altro IDE (a seconda di quello che il programmatore deve fare/deve ottenere, compatibilmente con le librerie disponibili per Proteus e per Codesys), e consentendo a questi due programmi, a questi due processi, di comunicare fra di loro. Entrambi i processi, quello sviluppato in Proteus e quello sviluppato in Codesys, chiaramente girano sullo stesso HMI-Linux by Syel, ad esempio il P10L.
In tal modo quello che si sa fare meglio in Proteus lo si fa fare al processo sviluppato con Proteus, e quello che si sa fare meglio in Codesys lo si fa fare al processo sviluppato con Codesys.

Generalmente, un’applicazione di lavoro necessita di un programma che implementi l’interfaccia grafica utente, ovvero una GUI (Graphic User Interface), ed un programma che implementi la funzionalità di lavoro PLC.
Ovviamente le cose nella realtà possono essere molto più complesse: ad esempio la GUI, oltre alle pagine grafiche sul touch-screen (con bottoni, sliders, campi di edit, icone varie, ecc …), può tranquillamente implementare anche altre funzionalità, come ad esempio un TCP-client, piuttosto che un colloquio seriale in CANbus o Rs232, ecc …
Viceversa, anche il programma PLC può implementare tutta una serie di funzionalità aggiuntive, come ad esempio un colloquio Modbus-RTU su layer fisico Rs485, piuttosto che il pilotaggio di motori controllati da PID evoluti, ecc …
GUI e PLC sono due processi concorrenti ed indipendenti (se da shell di comandi Rasbian-Linux si batte il comando “top”, nella lista dei processi in esecuzione che ne deriva si deve vedere sia il programma sviluppato con Proteus, sia il supporto a run-time RTE “codesyscontrol” di Codesys, che in quel momento sta eseguendo l’applicazione di lavoro sviluppata con Codesys-IDE): GUI occupa lo schermo dell’HMI di turno, ad esempio il P10L, mentre il PLC non fruisce delle risorse grafiche messe a disposizione dall’OS (Rasbian, nel nostro caso), non stampa nulla a video, è di fatto un processo “nascosto”, “cieco”, che fa il suo lavoro senza impegnare neppure in pixel dello schermo. GUI e PLC devono, tuttavia, sempre essere in contatto fra di loro, per lo scambio di informazioni.

Abbiamo 2 macro-possibilità:

- *Sviluppare il programma GUI con Codesys ed il programma PLC con Proteus;*
- *Sviluppare il programma GUI con Proteus ed il programma PLC con Codesys;*

In questa nota tecnica spieghiamo, a grandi linee, un esempio SW sviluppato da Syel, che aderisce alla prima situazione, la quale è peraltro la più frequente, poiché molti clienti che migrano verso Codesys, non vogliono rinunciare a tutto il codice, scritto in c, compilato in Proteus, il cui sviluppo magari ha richiesto mesi o anni, e che ovviamente non hanno la possibilità di portare dal c al linguaggio ST di Codesys in breve tempo, pertanto spesso serve proprio questo, ovvero mantenere il programma Proteus, ed iniziare a sviluppare un nuovo SW con Codesys, il quale faccia anche da GUI, dato che il comparto grafico messo a disposizione da Codesys è accattivante e versatile (trasparenze con canale alfa, effetti di evanescenza temporizzabile, grafica vettoriale SVG, ecc …), migliore di quello messo a disposizione da Proteus.



SW PLC cieco sviluppato con Proteus

Questo è il SW PLC:

C:\Users\EnricoMolinari\Desktop\P10L_localhost_Proteus_tcp_server\page1.v10

SW fatto con Proteus

messaggio da Codesys

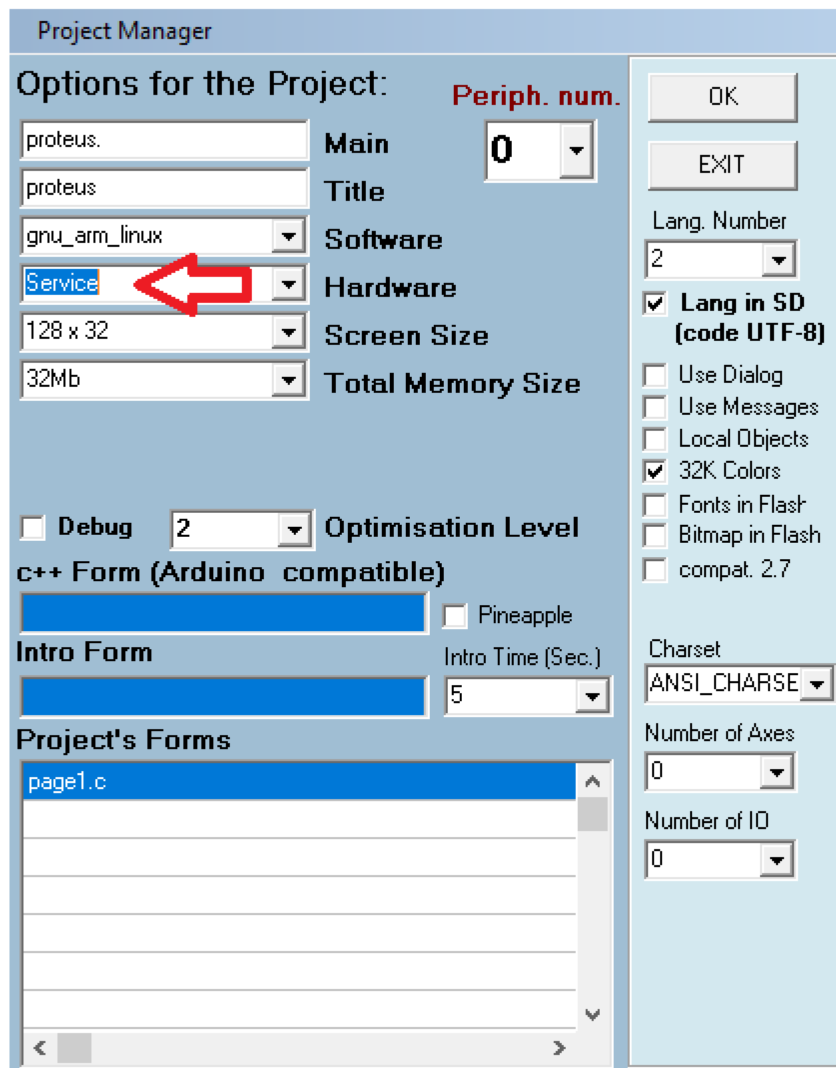
abc............................ RX

risposta verso Codesys

abc............................ TX

9999999 TX-RX loop

Per renderlo cieco, quindi per far si che il SW PLC non impegni la grafica, bisogna fare così, nel project manager di Proteus:



Inoltre bisogna preparare un task main, nel common.h, in questo modo:

```
int main(int argc, char** argv)
{
   // printf("start del main\n");
   fopen("TCP-SERVER:28800",(int)tcp_server1);
   fopen("TCP-SERVER:28700",(int)tcp_server2);
   while(1)
   {
      dummy1++;
   }
}
```

Lanciamo 2 server TCP, uno che ascolta sulla porta 28800, l’altro che ascolta sulla porta 28700. Quest’ultimo ascolta ed INVIA anche sulla porta 28700, pertanto sul SW GUI (Codesys) ci dovrà essere un server TCP che ascolta sulla porta 28700, oltre ad un client TCP che invia sulla porta 28800.
L’host a cui questi clients e servers TCP inviano i messaggi/ascoltano i messaggi (sia lato Proteus che lato Codesys), è il “local-host”, ovvero: IP = 127.0.0.1.

Dopo aver lanciato questi 2 server, il task main si mette in un loop infinito, senza far nulla.

```
void tcp_server1(FILE *f)
{
   U8   *bufferr=0;
   static int clients=0;
   connessione1++;
   if (f==0) return 0;
   if (++clients>20) goto sorry;
   bufferr=calloc(278,1);
   clienti1=clients;
// ------------------------------------------------------------
// aspetto messaggio da Codesys
// ------------------------------------------------------------
   while( !fread(bufferr,1,278,f) );
   msg_arrived++;
   strcpy(rxx_buff,bufferr);
   flag_reply=1;
// ------------------------------------------------------------
// fine sessione
// ------------------------------------------------------------
   free(bufferr);
sorry:  clients--;
   if (f)
   {end_session1++;fclose(f);} // close connection and return
}
```

Il server TCP “tcp_server1” funziona come callback di ricezione, lato SW PLC-Proteus.

Il server TCP “tcp_server1” aspetta all’infinito (freccia rossa) l’arrivo di un messaggio da parte del client TCP implementato sul programma GUI fatto con Codesys. Non appena il client TCP della GUI invia un messaggio, sulla porta 28800, ovviamente a 127.0.0.1, il server “tcp_server1” esce dal loop ricettivo indicato dalla freccia rossa e va verso la freccia blu, dove si vede il settaggio di un semaforo:

```
flag_reply=1;
```

Adesso vediamo il server TCP “tcp_server2”:

```
void tcp_server2(FILE *f)
{
   static int clients=0;
   connessione2++;
   if (f==0) return 0;
   if (++clients>20) goto sorry;
   clienti2=clients;
// -------------------------------------------------------------
// aspetto messaggio da Codesys
// -------------------------------------------------------------
   while(flag_reply==0);
   flag_reply=0;
   msg_replied++;
   fwrite(string_reply,1,strlen(string_reply),f);
// -------------------------------------------------------------
// fine sessione
// -------------------------------------------------------------
sorry:  clients--;
   if (f)
   {end_session2++;fclose(f);} // close connection and return
}
```

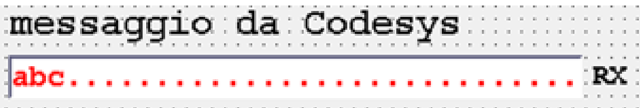
Il server TCP “tcp_server2” funziona come callback di trasmissione, lato SW PLC-Proteus.

Il server TCP “tcp_server2” aspetta all’infinito (freccia blu) che quel semaforo venga settato dalla callback di ricezione. Quando viene settato, vuol dire che è arrivato un messaggio TCP/IP dal SW GUI-Codesys. Quindi non appena il client TCP della GUI invia un messaggio, sulla porta 28800, ovviamente a 127.0.0.1, il server “tcp_server2” esce dal loop indicato dalla freccia blu e va verso la freccia viola, dove si vede l’invio (fwrite) di una stringa, che nel nostro esempio è:

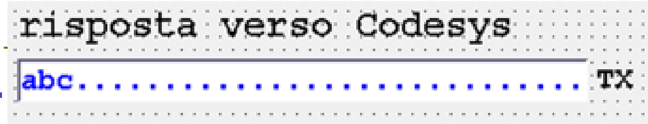
```
U8 string_reply[128]="risposta di prova";
```

In altri termini, ecco cosa accade:

```
void tcp_server1(FILE *f)
{
   U8   *bufferr=0;
   static int clients=0;
   connessione1++;
   if (f==0) return 0;
   if (++clients>20) goto sorry;
   bufferr=calloc(278,1);
   clienti1=clients;
// ----------------------------------------------------------------
// aspetto messaggio da Codesys
// ----------------------------------------------------------------
   while( !fread(bufferr,1,278,f) );
   msg_arrived++;
   strcpy(rxx_buff,bufferr);
   flag_reply=1;
// ----------------------------------------------------------------
// fine sessione
// ----------------------------------------------------------------
   free(bufferr);
sorry:  clients--;
   if (f)
   {end_session1++;fclose(f);} // close connection and return
}
```

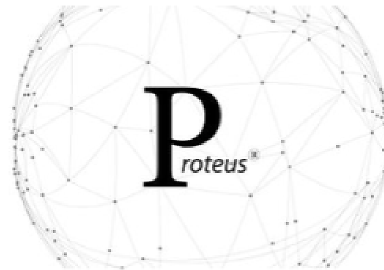



```
void tcp_server2(FILE *f)
{
   static int clients=0;
   connessione2++;
   if (f==0) return 0;
   if (++clients>20) goto sorry;
   clienti2=clients;
// ----------------------------------------------------------------
// aspetto messaggio da Codesys
// ----------------------------------------------------------------
   while(flag_reply==0);
   flag_reply=0;
   msg_replied++;
   fwrite(string_reply,1,strlen(string_reply),f);
// ----------------------------------------------------------------
// fine sessione
// ----------------------------------------------------------------
sorry:  clients--;
   if (f)
   {end_session2++;fclose(f);} // close connection and return
}
```



SW GUI sviluppato con Codesys

Questo è il SW GUI:



Comunicazione fra processo Codesys
e processo Proteus

messaggio inviato verso Proteus

%S

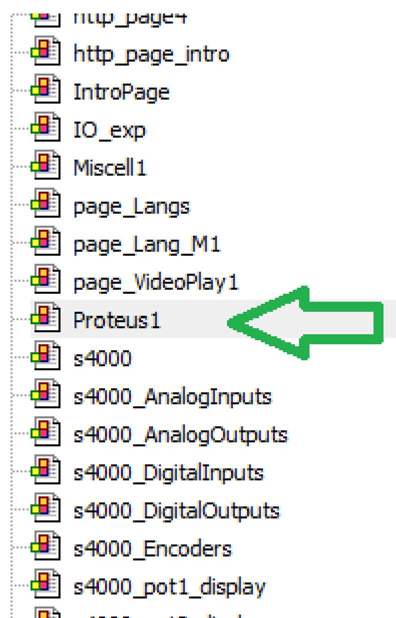
messaggio arrivato da Proteus

%S

conta messaggi arrivati finora

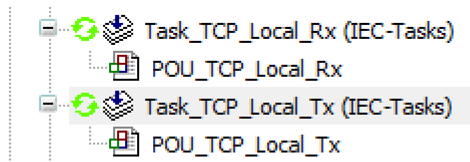
%i

Questa è solo la pagina grafica:



Per poter editare il messaggio da inviare al SW PLC-Proteus, e per poter vedere il messaggio di risposta inviato dal SW PLC-Proteus.

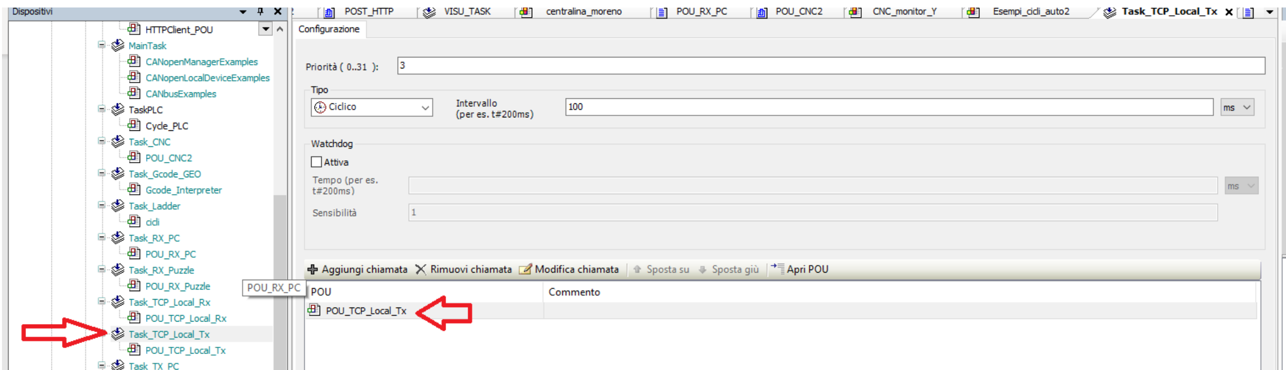
Questi sono i due task ciclici inseriti nell’albero di progetto:



Ciascuno ha associata una POU scritta in ST: un task serve solo per la trasmissione, l’altro solo per la ricezione.

Diamo loro un’occhiata:

Questo è il task di trasmissione, sulla TX-socket 127.0.0.1 : 28800



```
1   PROGRAM POU_TCP_Local_Tx
2   VAR
3       sa:sockaddress;
4       hSocket:UDINT;
5       SetIPAdress:STRING[14]:='127.0.0.1';
6       port:WORD:=28800;
7       Result:UDINT;
8       NrOfBytes:DINT;
9       socket_created:BOOL:=FALSE;
10      connection:BOOL:=FALSE;
11      //---------------------------
12      contatore1 : INT :=0;
13      contatore2 : INT :=0;
14      contatore3 : INT :=0;
15      contatore4 : INT :=0;
16      contatore5 : INT :=0;
17      contatore6 : INT :=0;
18      contatore7 : INT :=0;
19  END_VAR
20
```

```
IF socket_created = FALSE THEN
    // Socket NON creata: tento di creare la Socket "localHost:port"
    contatore1 := contatore1+1;
    sa.sin_family:=SOCKET_AF_INET;
    sa.sin_port:=SysSockHtons(port);
    Result:=SysSockInetAddr(SetIPAdress, ADR(sa.sin_addr.ulAddr));
    hSocket:=SysSockCreate(SOCKET_AF_INET, SOCKET_STREAM, SOCKET_IPPROTO_TCP, ADR(Result));
    IF hSocket>0 THEN
        // Socket creata con successo (hSocket = 14, 15 ... )
        socket_created := TRUE;
        contatore2 := contatore2+1;
    END_IF;
    // finchè non è creata, entro qui
ELSE
    // la Socket "localHost:port" è stata creata con successo ... adesso valutiamo la connessione
    IF connection = FALSE THEN
        // connessione NON stabilita: tento di connettermi a localHost
        Result:=SysSockConnect(hSocket, ADR(sa), SIZEOF(sa));
        contatore3 := contatore3+1;
        IF Result=0 THEN
            // connessione con device hSocket stabilita con successo
            connection :=TRUE;
            contatore4 := contatore4+1;
        END_IF;
    ELSE
        // connessione stabilita con successo
        contatore5 := contatore5+1;
        //--------------------------------------------------------------------
        //--------------------------------------------------------------------
        IF
        (
            GVLA.trigger_tcp_tx
        )
        THEN
            // invio automatico/periodico
            contatore6 := contatore6+1;
            NrOfBytes:=SysSocksend(hSocket, ADR(GVLA.str3), SIZEOF(GVLA.str3)+1,0, ADR (Result));
            //--------------------------------
            SysSockClose(hSocket);
            socket_created := FALSE;
            connection := FALSE;
            //--------------------------------
        ELSE
            // NON è ancora il momento di inviare il telegramma
            contatore7 := contatore7+1;
        END_IF;
        //--------------------------------------------------------------------
        //--------------------------------------------------------------------
    END_IF;
```

messaggio inviato verso Proteus

%S

127.0.0.1 : 28800

La funzione:

```
SysSocksend(hSocket, ADR(GVLA.str3), SIZEOF(GVLA.str3)+1,0, ADR (Result));
```

Serve ad inviare fisicamente il messaggio alla controparte PLC-Proteus.

Adesso vediamo il task di ricezione, sulla RX-socket 127.0.0.1 : 28700

```
PROGRAM POU_TCP_Local_Rx
VAR
    sa:sockaddress;
    hSocket:UDINT;
    SetIPAdress:STRING[14]:='127.0.0.1';
    port:WORD:=28700;
    Result:UDINT;
    NBR: DINT;
    socket_created:BOOL:=FALSE;
    connection:BOOL:=FALSE;
    //--------------------------
    contatore1 : INT :=0;
    contatore2 : INT :=0;
    contatore3 : INT :=0;
    contatore4 : INT :=0;
    contatore5 : INT :=0;
    contatore6 : INT :=0;
    j          : INT :=0;
END_VAR
```

```
IF socket_created = FALSE THEN
    // Socket NON creata: tento di creare la Socket "localHost:port"
    // finchè non è creata, entro qui
    contatore1 := contatore1+1;
    sa.sin_family:=SOCKET_AF_INET;
    sa.sin_port:=SysSockHtons(port);
    Result:=SysSockInetAddr(SetIPAdress, ADR(sa.sin_addr.ulAddr));
    hSocket:=SysSockCreate(SOCKET_AF_INET, SOCKET_STREAM, SOCKET_IPPROTO_TCP, ADR(Result));
    IF hSocket > 0 THEN
        // Socket creata con successo (hSocket = 14, 15 ... )
        socket_created := TRUE;
        contatore2 := contatore2+1;
    END_IF;
ELSE
    // la Socket "localHost:port" è stata creata con successo ... adesso valutiamo la connessione
    IF connection = FALSE THEN
        // connessione NON stabilita: tento di connettermi a localHost
        contatore3 := contatore3+1;
        Result:=SysSockConnect(hSocket, ADR(sa), SIZEOF(sa));
        IF Result = 0 THEN
            connection := TRUE;
            contatore4 := contatore4+1;
            // connessione stabilita con successo
        END_IF;
    ELSE
        // connessione stabilita con successo
        contatore5 := contatore5+1;
        // GVLA.String_RX_tcp := WSTRING_TO_STRING("                                        ");
        FOR j := 0 TO 60 BY 1 DO // 63
            GVLA.String_RX_tcp[j] := 32; // ' '
        END_FOR;
        NBR:=SysSockRecv(hSocket, ADR(GVLA.String_RX_tcp), 40, 0, ADR (Result));
        IF NBR <>0 THEN
            GVLA.Rx_Cnt_tcp := GVLA.Rx_Cnt_tcp+1;
            // ------------------------------------------------ //
            // ------------------ parsing: ------------------- //
            // scanner_msg_rx_CANbus();
            GVLA.String_RX_tcp_visual := GVLA.String_RX_tcp;
            // ------------------------------------------------ //
            // ------------------------------------------------ //
            SysSockClose(hSocket);
            socket_created := FALSE;
            connection := FALSE;
            //--------------------------------
        END_IF
    END_IF
END_IF
```

messaggio arrivato da Proteus

%S

conta messaggi arrivati finora

%i

La funzione:

```
SysSockRecv(hSocket, ADR(GVLA.String_RX_tcp), 40, 0, ADR (Result));
```

Serve a ricevere fisicamente il messaggio dalla controparte PLC-Proteus.

Le porte 28800 e 28700 sono state prese casualmente, quindi andrebbero bene qualsiasi altra coppia di porte.