rxsocketita

COLLABORATORS			
	TITLE : rxsocketita		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 28, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rxsocketita

1.1 main

```
------------------------------------------------------------------------------------------

           RXSOCKET.LIBRARY versione 2.1

------------------------------------------------------------------------------------------

                    AVVERTENZA
 IL SOFTWARE E LE INFORMAZIONI CONTENUTE IN QUESTO ARCHIVIO
            SONO FORNITE "COSI' COME E'".
L'USO E' A TUO PROPRIO RISCHIO E NESSUNA RESPONSABILITA' E' ASSUNTA.
       NESSUNA GARANZIA E' DATA, ECCETTO CHE
  "QUESTO SOFTWARE E' DEL TUTTO PRIVO DI BACKDOORS".
------------------------------------------------------------------------------------------


 Introduzione
 Installazione
 Requisiti
 Autore
 Distribuzione
 Termini
 Bugs
 Strutture
 Funzioni
 Lista Funzioni
 Ringraziamenti
 Bibliografia
 Da Fare
 Note
 Cambiamenti
 Sopporto inedt

------------------------------------------------------------------------------ ↩
```

1.2 introduzione

Lo scopo di questa libreria è offrire un insieme di funzioni socket alle macro ARexx, in modo da rendere l'accesso ad Internet da ARexx semplice ed immediato, senza perdere la complessità delle funzioni della bsdsocket.library.

Le funzioni di questa libreria chiamano le corrispondenti, che d'ora in poi dirò "originali", funzioni della bsdsocket.library, conservando per quanto è possibile la loro sintassi, effetti e side-effects.

Proprio per questo motivo, questo documento non vuole essere affatto una introduzione alle funzioni della bsdsocket.library ed alla programmazione TCP/IP, ma soltanto una guida all'uso delle funzioni della rxsocket.library.

L'ambiente in cui le funzioni operano è "privato della macro". La prima chiamata ad una funzione crea un'entrata in una lista nella base della libreria e "segna" la macro ARexx chiamante in modo tale che alla sua uscita, normale o prematura, ogni risorsa allocata, tra le quali ovviamente la bsdsocket.library, verrà automaticamente rilasciata.

E' necessario spendere subito qualche parola sul metodo usato per il passaggio degli argomenti ed il ritorno dei risultati:

- quando la funzione originale richiede un argomento semplice, non una struttura, lo stesso argomento è passato alla funzione ARexx;

- quando la funzione originale richiede una struttura come argomento, alla funzione ARexx viene passato un nome di variabile ARexx, detto qui stem, con vari campi settati;

- quando la funzione originale ritorna un intero, lo stesso intero è ritornato dalla funzione ARexx;

- quando la funzione originale ritorna una struttura, alla funzione ARexx viene passato uno stem, alcuni campi del quale verranno settati dalla funzione stessa.

Questo metodologia è generale nelle mie librerie ARexx e tenta in qualche modo di emulare il passaggio degli argomenti per Tags tipico di AmigaOS.

La libreria offre quasi tutte le funzioni della bsdsocket.library più alcune di livello superiore (hehe giusto un paio per ora). La libreria non supporta direttamente pacchetti RAW o diversi da TCP ed UDP; in ogni caso, allocando un po' di memoria e settado a mano i vari campi del pacchetto, anche l'uso di altri protocolli e di pacchetti RAW diviene possibile.

1.3 installazione

Non è fornito, data la semplicità, alcuno script di installazione.

Per installare la libreria:
- copia rxsocket.library in LIBS:
- apri una shell e scrivi: rx "call addlib('rxsocket.libary',0,-30)"

1.4 requisiti

La libreria gira su AmigaOS, versione >=2, ed ovviamente uno stack TCP/IP.
In particolare è stata creata su Miami e funziona su AmiTCP.
Per quanto riguarda TermiteTCP, la libreria è stata testata ed è risultata funzionante per la maggior parte delle funzioni; però, data la particolarità di TermiteTCP, non tutte le funzioni "vanno" e molte sono state appositamente disabilitate.

Se uno stack non è in funzione, la libreria può essere comunque nella lista degli host ARexx, poichè ogni chiamata ad una sua funzione torna errore ARexx 1, ovvero come se la funzione non esistesse.

1.5 autore

L'autore di questo gioiello è:

Alfonso Ranieri

la sua e-mail è:

alfier@iol.it

Lo puoi trovare in irc su:

- #amichat dalnet
- #amigaita ircnet
- #amyita ircnet

L'ultima versione di questa libreria è reperibile su web a:

http://users.iol.it/alfier/

1.6 distribuzione

rxsocket.library è FREEWARE

Chiunque può distribuirla fintanto che l'archivio originale rimane intatto.
L'uso commerciale o l'inclusione in altro software sono proibiti previo consenso da parte dell'autore.

1.7 termini

- stem o stemName: è una valido nome di variabile ARexx come ad esempio var var.0 var.ciao e così via;

- socket: è l'ambiente creato ad esempio dalla funzione socket(), su cui operano la maggior parte delle funzioni della bsdsocket.library;

- socketfd: è l'interno, tornato ad esempio da socket(), che rappresenta il
  descrittore per un socket; lo si paragoni ad un descrittore di file in c;

- addr o indirizzo: è un indirizzo Internet, nella forma dotted.
  Un indirizzo Internet è un intero a 32 bits, che viene rappresentato nella
  forma dotted "a.b.c.d" "a.b.c" "a.b" "a" oppure come un nome simbolico.
  In questa libreria si è scelto di usare come forma biù bassa solo indirizzi
  dotted, ad esempio resolve() ritorna la forma dotted oppure -1.
  Il mio scopo è mantenere la API orginale della bsdsocket.function, ma gli
  interi in ARexx non permettono un agevole l'uso di indirizzi a numero intero;

- tipi degli argomenti: sono i tipi degli argomenti delle funzioni ARexx,
  rappresentati da una lettera maiuscola se obbligatori o minuscola altrimenti:
  - D   qualsiasi tipo   --
  - N   numerico   /N
  - S   simbolo ARexx   /S
  - V   nome variabile, come S ma di lunghezza inferiore a 20 caratteri /V

1.8 bugs

Hehe, ci sono bugs nel nostro sistema operativo.
Ci sono bugs in software commerciali costosissimi.
Vuoi che non ce ne siano qui?

Attualmente, con Miami, che prendo come riferimento,
una bug è "ben conosciuta":
se un socket non connesso viene rilasciato, ottenuto e poi connesso,
ovvero se su di esso si esegue la sequenza:
- ReleaseSocket()
- ObtainSocket()
- connect()
un hit di Enforcer viene fuori.
Questo è un problema di Miami, non mio.

Ti prego di segnalarmi al più presto ogni bugs che potessi scoprire.

1.9 strutture

Attualmente 4 strutture sono passate e tornate a/da le funzioni della libreria:
- struct hostent
- struct servent
- struct protoent
- struct sockaddr_in

Come detto sopra, si cerca di emulare il passaggio od il ritorno di strutture
per mezzo di uno stem.

Un esempio aiuterà:

vogliamo leggere la entry del servizio echo TCP, quindi invochiamo la funzione
GetServByName():

```
  if ~GetServByName("SE","echo","TCP") then do
    say "service echo non trovato"
    exit
  end

  say "Name: " se.servName
  say "Port: " se.servPort
  say "Proto:" se.servProto

  if se.servAliases.num=0 then say "echo non ha alias"
  else do
    say "alias di echo"
    do i = 0 to se.servAliases.num-1
      say se.servAliases.i
    end
  end
```

La funzione GetServByName(), se ha successo, setta alcuni campi dello
stem che gli viene passato come primo argomento:
- SERVNAME
- SERVPORT
- SERVPROTO
- SERVALIASES.NUM
- SERVALIASES.0 , ... , SERVALIASES.last (last = SERVALIASES.NUM-1)

In generale lo stesso meccanismo viene applicato sia nel passaggio degli
argomenti che nel ritorno dei risultati.

Inoltre se una lista deve essere ritornata, il numero dei membri viene scritto
in STEM.NOME.NUM ed i membri della lista si trovano,
se STEM.NOME.NUM>0, in
STEM.NOME.0, ..., STEM.NOME.last (last = STEM.NOME.NUM-1).

Per ciascuna struttura, i campi letti o scritti dalle funzioni sono:

- hostent (settata da GetHostByName() GetHostByAddr())
  - HOSTNAME
  - HOSTADDRTYPE
  - HOSTLENGTH
  - HOSTALIASES.NUM
  - HOSTALIASES.INDEX (INDEX=0,...,HOSTALIASES.NUM-1)
  - HOSTADDRLIST.NUM
  - HOSTADDRLIST.INDEX (INDEX=0,...,HOSTADDRLIST.NUM-1)

- servent (settata da GetServByName() GetServByPort())
  - SERVNAME
  - SERVPORT
  - SERVPROTO
  - SERVALIASES.NUM
  - SERVALIASES.INDEX (INDEX=0,...,SERVALIASES.NUM-1)

- protoent (settata da GetProtoByName() GetProtoByNumber())
  - PROTONAME
  - PROTOPROTO
  - PROTOALIASES.NUM
  - PROTOALIASES.INDEX (INDEX=0,...,PROTOALIASES.NUM-1)

- struct sockaddr_in (passata e tornata a/da varie funzioni)
  come argomento di una funzione:
  - ADDRFAMILY   /D ("INET")
  - ADDRPORT   /N
  - ADDRADDR   /N
  come risultato di una funzione:
  - ADDRLEN
  - ADDRFAMILY
  - ADDRPORT
  - ADDRADDR

Per rendere la vita più semplice, molti degli argomenti da passare alle funzioni direttamente o come campi di uno stem, oltre al valore numerico, hanno una "forma umana". Specialmente:

- FAMILY come in socket(family,...) o in ADDRFAMILY ha la forma "INET"
  e solamente "INET" in questa versione della libreria;

- TYPE come in socket(family,type,...) può essere un intero
  oppure una tra le stringhe:
  - "STREAM"
  - "DGRAM"
  - "RAW"
  - "RDM"
  - "SEQPACKET"

- PROTOCOLL come in socket(family,type,protocoll) può essere un intero
  oppure una tra le stringhe:
  - "IP"
  - "HOPOPTS"
  - "ICMP"
  - "IGMP"
  - "GGP"
  - "IPIP"
  - "TCP"
  - "EGP"
  - "PUP"
  - "UDP"
  - "IDP"
  - "TP"
  - "IPV6"
  - "ROUTING"
  - "FRAGMENT"
  - "RSVP"
  - "ESP"
  - "AH"
  - "ICMPV6"
  - "NONE"
  - "DSTOPTS"
  - "EON"
  - "ENCAP"
  - "DIVERT"
  - "RAW"

Comunque....Solo alcuni protocolli funzioneranno qui, ma...
Se sei molto bravo in TCP/IP ed ARexx, puoi allocare un po’ di memoria nella macro, creare il tuo pacchetto a mano inserendo i giusti valori direttamente

```
in memoria e spedire pacchetti ICMP o RAW a tuo piacere. Forse, una futura
versione della libreria supporterà la creazione di pacchetti RAW.

Quindi, tirando il fiato, quando nella descrizione di una funzione, scrivo:
"lo stem va settato come sockaddr_in", intendo semplicemente che prima di
chiamare la funzione, bisogna settare i campi
 - ADDRFAMILY     "INET"        D
 - ADDRPORT       porta         N
 - ADDRADDR       ndirizzo      N
dello stem che si passerà alla funzione.

Ugualmente, quando scrivo:
"setta lo stem alla struttura hostest", intendo che la funzione setta i campi
 - HOSTNAME
 - HOSTADDRTYPE
 - HOSTLENGTH
 - HOSTALIASES.NUM
 - HOSTALIASES.INDEX
 - HOSTADDRLIST.NUM
dello stem passatogli come argomento.
```

1.10 funzioni

```
QUESTA NON E' UNA GUIDA ALLE FUNZIONI DELLA BSDSOCKET.LIBRARY, AL LORO USO ED
AI LORO EFFETTI. SI LEGGA UN BUON LIBRO SUL TCP/IP, LE RFC OPPURE UN AUTODOC
SULLA SOCKET.LIBRARY PER AMIGAOS.

I nome delle funzioni sono NON case-sensitive; in ogni modo, io uso, per ragioni
stilistiche, le lettere maiuscole se il nome è composto, per ogni parola che
costituisce il nome della funzione, compresa la iniziale; nel caso degli argomenti
lo stesso stile si applica, eccetto che la prima lettera della prima parola è
sempre minuscola.

La descrizione delle funzioni ha il formato:
"NOME"                 nome della funzione tutti in maiuscole
"Uso"                  chiamata tipica della funzione, con tutti gli argomenti
"maschera argomenti"   maschera argomenti, secondo lo stile AmigaDOS

"Descrizione"          descrizione della funzione.
"Risultati"            risultati, se non già contenuti in "Descrizione"
"ESEMPIO"              codice ARexx d'esempio

La maschera degli argomenti è del tipo AmigaDOS;
ad esempio:
<stem/V>  significa argomento obbligatorio di tipo V;
[flags]       significa argomento facoltativo di tipo D;
{value/N}     significa nessuno o più argomenti di tipo N.

Se un argomento è facoltativo, può essere omesso (hummm tautologia rulez :-).

Gli argomenti facoltativi si trovano DOPO gli argomenti obbligatori.
Ad esempio:
    res=SendTo(socketfd,"Ciao",4)             omessi [flag] e [stem]
    res=SendTo(socketfd,"Ciao",4,,"REMOTE")   omesso [flag]
sono usi validi della funzione SendTo().
```

```
A volte un argomento facoltativo può divenire obbligatorio in una particolare
forma della funzione (ciò viene sempre specificato nella descrizione della
funzione).

L'omissione di un argomento ha un effetto dipendente dalla funzione.

Ovviamente ogni funzione può causare errori ARexx; ad esempio:
    res=SendTo("ciao","A",1)
causa un errore ARexx 17, perché il primo argomento di SendTo() deve essere
di tipo N.

A volte ci si imbatte in errori ARexx di cui non si riesce subito a trovare
la causa; si ricordi che:

- anche gli argomenti settati negli stem sono sottoposti al controllo del tipo;

- se si passa come stem un nome minuscolo, lo stesso nome minuscolo deve
  essere settato:
    res=bind(socketf,"locale")
  legge nello stem "locale" e non "LOCALE" i campi di sockaddr_in;

- conviene sempre mettere tra "" i nomi di stem e degli argomenti in "forma
  umana":
    inet=129
    sockfd=socket(INET,"STREAM","IP")
  causa errore 18 (invalid argument to a function), perché INET vale 129 e 129 è  ↩
     un
  valore non valido come primo argomento della funzione socket().

Questa è la ragione per nomi così "buffi" per le strutture, tipo ADDRADDR o
SERVPORT, al posto di più naturali ADDR o PORT; infatti facilmente si
assegnano addr e port all'interno di una macro ed il risultato è, ad esempio:
    port=4000
    locale.port=port /* diviene locale.4000=4000 */
```

1.11 listafunzioni

```
 @{ "ACCEPT" LINK "ACCEPT"}
 @{ "ADDR2C" LINK "ADDR2C"}
 @{ "BIND" LINK "BIND"}
 @{ "CLOSESOCKET" LINK "CLOSESOCKET"}
 @{ "CONNECT" LINK "CONNECT"}
 @{ "DUP2SOCKET" LINK "DUP2SOCKET"}
 @{ "ERRNO" LINK "ERRNO"}
 @{ "ERRORSTRING" LINK "ERRORSTRING"}
 @{ "GETHOST" LINK "GETHOST"}
 @{ "GETHOSTBYADDR" LINK "GETHOSTBYADDR"}
 @{ "GETHOSTBYNAME" LINK "GETHOSTBYNAME"}
 @{ "GETHOSTID" LINK "GETHOSTID"}
 @{ "GETHOSTNAME" LINK "GETHOSTNAME"}
 @{ "GETPEERNAME" LINK "GETPEERNAME"}
 @{ "GETPROTOBYNAME" LINK "GETPROTOBYNAME"}
 @{ "GETPROTOBYNUMBER" LINK "GETPROTOBYNUMBER"}
 @{ "GETSERVBYNAME" LINK "GETSERVBYNAME"}
```

```
 @{ "GETSERVBYPORT" LINK "GETSERVBYPORT"}
 @{ "GETSOCKETBASE" LINK "GETSOCKETBASE"}
 @{ "GETSOCKETEVENTS" LINK "GETSOCKETEVENTS"}
 @{ "GETSOCKNAME" LINK "GETSOCKNAME"}
 @{ "GETSOCKOPT" LINK "GETSOCKOPT"}
 @{ "HELP" LINK "HELP"}
 @{ "HOSTERRORNO" LINK "HOSTERRORNO"}
 @{ "HOSTERRORSTRING" LINK "HOSTERRORSTRING"}
 @{ "INETCKSUM" LINK "INETCKSUM"}
 @{ "IOCTLSOCKET" LINK "IOCTLSOCKET"}
 @{ "ISDOTADDR" LINK "ISDOTADDR"}
 @{ "ISLIBON" LINK "ISLIBON"}
 @{ "ISSOCKET" LINK "ISSOCKET" }
 @{ "LISTEN" LINK "LISTEN"}
 @{ "MIAMIDISALLOWDNS" LINK "MIAMIDISALLOWDNS" }
 @{ "MIAMIGETPID" LINK "MIAMIGETPID" }
 @{ "MIAMIISONLINE" LINK "MIAMIISONLINE" }
 @{ "MIAMIONOFFLINE" LINK "MIAMIONOFFLINE"}
 @{ "MIAMISUPPORTSIPV6" LINK "MIAMISUPPORTSIPV6" }
 @{ "OBTAINSOCKET" LINK "OBTAINSOCKET"}
 @{ "OPENCONNECTION" LINK "OPENCONNECTION"}
 @{ "RECV" LINK "RECV"}
 @{ "RECVFROM" LINK "RECVFROM"}
 @{ "RECVLINE" LINK "RECVLINE"}
 @{ "RELEASECOPYOFSOCKET" LINK "RELEASECOPYOFSOCKET"}
 @{ "RELEASESOCKET" LINK "RELEASESOCKET"}
 @{ "RESOLVE" LINK "RESOLVE"}
 @{ "SEND" LINK "SEND"}
 @{ "SENDTO" LINK "SENDTO"}
 @{ "SETSOCKETBASE" LINK "SETSOCKETBASE"}
 @{ "SETSOCKETSIGNALS" LINK "SETSOCKETSIGNALS"}
 @{ "SETSOCKOPT" LINK "SETSOCKOPT"}
 @{ "SHUTDOWN" LINK "SHUTDOWN"}
 @{ "SOCKET" LINK "SOCKET"}
 @{ "SOCKATMARK" LINK "SOCKATMARK" }
 @{ "SYSLOG" LINK "SYSLOG"}
 @{ "WAITSELECT" LINK "WAITSELECT"}
```

1.12 accept

```
ACCEPT
Uso: sockfd=accept(socketfd,remote)
<socketfd/N>,<remote/V>

Accetta una connessione su un socket dopo un bind() ed un listen().
Crea un nuovo socket e torna il suo id, settando remote al sockaddr_in
del peer connessosi al socket.
La funzione è di solito usata all'interno di un service.

Ritorna -1 in caso di fallimento od un intero >=0 in caso di successo.
```

1.13 addr2c

```
ADDR2C
Uso: packetAddr=Addr2C(addr)
<addr/N>

Converte da addr a packed-char.
Utile quando si deve settare una zona di memoria con un indirizzo Internet.
```

1.14 bind

```
BIND
Uso: res=bind(socketfd,locale)
<socketfd/N>,<locale/V>

Prova a "collegare" il socket ad una porta.
locale deve essere settato come un socketaddr_in, di solito con il campo
ADDRADDR posto a 0.

Ritorna -1 in caso d'errore
```

1.15 closesocket

```
CLOSESOCKET
Uso: res=CloseSocket(socketfd)
<socketfd/N>

Chiude un socket e rilascia tutte le risorse ad esso connesse.
Come i dati da ricevere o da sendare vengono trattati, in caso di
socket di tipo orientato alla connessione (TCP), dipende dal
settaggio di LINGER.

Ritorna -1 in caso d'errore
```

1.16 connect

```
CONNECT
Uso: res=connect(socketfd,remote)
<sockeft/N>,<remote/V>

Connette il socket ad un socketaddr_in definito in remote, che contiene un
indirizzo ed una porta, ovvero il quarto e quinto membro della cinquina che
rappresenta una connessione Internet.

Ritorna -1 in caso d'errore

ESEMPIO

    /* "risolviamo" l'indirizzo www.unina.it */
    addr=resolve("www.unina.it")
    if addr==-1 then do
```

```
        say "indirizzo www.unina.it non trovato:" HostErrorno()
    exit
    end

    /* allochiamo un socket */
    socketfd=socket("INET","STREAM","IP")
    if socketfd==-1 then do
        say "socket non creato:" errno()
        exit
    end

    /* settiamo DOVE connetterci */
    remote.ADDRFAMILY="INET" /* sempre e solo "INET" */
    remote.ADDRPORT=80       /* service http */
    remote.ADDRADDR=addr

    /* connettiamoci */
    if ~connect(socketfd,"REMOTE") then do
        say "connessione fallita:" errno()
        exit
    end
    /* connessione stabilita ... */
```

1.17 dup2socket

```
DUP2SOCKET
Uso: sockfd=Dup2Socket(socketfd)
<socketfd/N>

Duplica un socket e torna il descrittore del nuovo socket o -1 in caso d'errore.
Chiama la dup2socket() della bsdsocket.library con secondo argomento -1.
```

1.18 errno

```
ERRNO
Uso: error=errno()
-

Ritorna il codice errore corrente.
Utile dopo quasi ogni funzione, salvo se diversamente specificato, per scoprire
la causa del fallimento.
```

1.19 errorstring

```
ERRORSTRING
Uso: string=ErrorString(code)
<code/N>

Ritorna la stringa associata all'errore indicato da code.
La stringa è non localizzata. Dovrebbe essere lo stack a localizzarla, non io.
```

1.20 gethost

```
Uso: res=getHost(host,host)
<host/V>,<host/N>

Setta host come una struttura hostent di un host passato per indirizzo o nome.
Questa funzione, di solito, se non diversamente settato nello stack o se addr è
presente in un database di host, causa l'interrogazione del server DNS da parte
dello stack.
In pratica questa funzione è l'unione di GetHostByName() e GetHostByAddr().

Ritorna un booleano indicante l'esito dell'operazione.
In questo caso HostErrerno() può essere usata in caso di fallimento per  scoprirne
la causa.
```

1.21 gethostbyaddr

```
GETHOSTBYADDR
Uso: res=getHostByAddr(host,addr)
<host/V>,<addr,N>

Setta host come una struttura hostent di un host passato per indirizzo. Questa
funzione, di solito, se non diversamente settato nello stack o se addr è presente
in un database di host, causa l'interrogazione del server DNS da parte dello stack ↩
    .
In pratica questa funzione è l'inversa di GetHostByName().

Ritorna un booleano indicante l'esito dell'operazione.
In questo caso HostErrerno() può essere usata in caso di fallimento per  scoprirne
la causa.
```

1.22 gethostbyname

```
GETHOSTBYNAME
Uso: res=GetHostByName(host,hostName)
<host/V>,<hostName>

Setta host come una struttura hostest di un host passato per nome.
Questa funzione, di solito, se non diversamente settato nello stack o se hostName
è presente in un database di host, causa l'interrogazione del  server DNS da parte
dello stack.
In pratica questa funzione è l'inversa di GetHostByAddr().

Ritorna un booleano indicante l'esito dell'operazione.
In questo caso HostErrno() può essere usata in caso di fallimento per scoprirne la
causa.
```

1.23 gethostid

```
GETHOSTID
Uso: id=GetHostID()
-

Ritorna l'id dello host corrente.
```

1.24 gethostname

```
GETHOSTNAME
Uso: res=GetHostName(name)
<name/S>

Scrive in name il nome dello host corrente.

Ritorna un booleano ad indicare l'esito dell'operazione.
```

1.25 getpeername

```
GETPEERNAME
Uso: res=GetPeerName(socketfd,remote)
<socketfd/N>,<remote/V>

Setta remote come una struttura sockaddr_in relativo al peer connesso con il
socket.

Ritorna un booleano indicante l'esito dell'operazione.

Disabilitata su TermiteTCP.
```

1.26 getprotobyname

```
GETPROTOBYNAME
Uso: res=GetProtoByName(stem,protoName)
<stem/V>,<protoName>

Setta stem come una protoent relativa a proto passato per nome.
La funzione è usata per sapere se un protocollo è disponibile nello stack in uso
e per conoscere il suo id.

Ritorna un booleano indicante l'esito dell'operazione.
```

1.27 getprotobynumber

```
GETPROTOBYNUMBER
Uso: res=GetProtoByNumber(stem,protoID)
<stem/V>,<protoID/N>
```

```
Setta stem come una protoent relativa al protocollo passato per numero.
La funzione è usata per conoscere il nome di un protocollo.

Ritorna un booleano indicante l'esito dell'operazione.
```

1.28 getservbyname

```
GETSERVBYNAME
Uso: res=GetServByName(stem,serviceName,protoName)
<stem/V>,<serviceName>,<protoName>

Setta stem come una servent relative a service passato per nome e protocollo.
La servent viene letta localmente, dal database dei services dello stack.

Ritorna un booleano indicante l'esito dell'operazione.

Disabilitata su TermiteTCP (non avendo TermiteTCP un database di services).
```

1.29 getservbyport

```
GETSERVBYPORT
Uso: res=GetServByPort(stem,portNumber,protoName)
<stem/V>,<potNumber/N>,<protoName>

Setta stem come una servent relativa al service passato per numero di porta
e protocollo.
La servent viene letta localmente, dal database dei services dello stack.

Ritorna un booleano indicante l'esito dell'operazione.

Disabilitata su TermiteTCP (non avendo TermiteTCP un database di services).
```

1.30 getsocketbase

```
GETSOCKETBASE
Uso: res=GetSocketBase(stem)
<stem/V>

Legge alcuni parametri globali della base della libreria bsdsocket.library.
La funzione originale SocketBaseTagList(), che legge e setta, è qui divisa  in 2
funzioni GetSocketBase(), che legge, e SetSocketBase(), che setta.

Bisogna settare in stem i campi che si vogliono leggere, quindi chiamare la
funzione, che per ogni campo che trova settato, leggerà il parametro
corrispondente.

I campi sono (si vedano gli autodoc di socket.library per il loro significato):
- "BREAKMASK"
- "DTABLESIZE"
- "ERRNO"
```

```
- "ERRNOSTRPTR"
- "HERRNOSTRPTR"
- "HERRNO"
- "SIGIOMASK
- "SIGURGMASK
- "LOGFACILITY"
- "LOGMASK"
- "LOGSTAT"

Ritorna -1 in caso di fallimento.

ESEMPIO

    drop a.  /* per essere sicuri di leggere solo i parametri desiderati */

    a.ERRNOSTRPTR=40        /* errore di codice 40 */
    a.BREAKMASK=1           /*  maschera di segnali exec */
    a.HERRNOSTRPTR=2        /* errore di host-lookup di codice 2 */

    call GetSocketBase("A")

    say a.ERRNOSTRPTR       ----->"Message too long"
    say a.BREAKMASK         ----->4096
    say a.HERRNOSTRPTR      ----->"Host name lookup failure"
```

1.31 getsocketevents

```
GETSOCKETEVENTS
Uso: res=GetSocketEvents(stem)
<stem/V>

Ritorna gli eventi asincroni accaduti ai sockets, settando i campi dello
stem passato come argomento con un booleano.

I campi settati sono:
- ACCEPT
- CLOSE
- CONNECT
- ERROR
- OOB
- READ
- WRITE

Ritorna il socketfd su cui sono accaduti gli eventi asincroni o
-1 se nessuno sockets è interessato ad eventi asincroni.

In questo caso Errno() NON PUO' essere usato in caso di errore.
```

1.32 getsockname

```
GETSOCKETNAME
Uso: res=GetSocketName(socketfd,stem)
```

```
<socketfd/N>,<stem/V>

Setta stem come una sockaddr_in relativa al socket.

Ritorna -1 in caso di fallimento.
```

1.33 getsockopt

```
GETSOCKOPT
Uso: res=GetSockOpt(socketfd,level,parm,stem)
<socketfd/N>,<level>,<parm>,<stem/V>

Scrive in stem il valore della parametro parm del socket al livello level.

Attualmente il livello è uno tra:
- "SOCKET"
- "IP"

Parametri per "SOCKET" sono:
- "DEBUG"
- "REUSEADDR"
- "REUSEPORT"
- "KEEPALIVE"
- "DONTROUTE"
- "LINGER"
- "BROADCAST"
- "OOBINLINE"
- "TYPE"
- "ERROR"

Il valore è scritto direttamente in stem, salvo che per "LINGER",  nel qual caso
sono settati i campi di stem:
- "ONOFF"
- "LINGER"

Parametri per "IP" sono:
- "HDRINCL"
- "IPOPTIONS"
- "TTL"
- "TOS"

Il valore è scritto direttamente in stem;  nel caso di "IPOPTIONS" il valore ↩
    scritto
in stem è un booleano non una stringa contenente le opzioni dello header IP.

Ritorna -1 in caso d'errore.
```

1.34 help

```
HELP
Uso: argsMask=help(funName)
<funName>
```

Ritorna la maschera dei tipi degli argomenti relativa della funzione funName.

1.35 hosterrorno

HOSTERRORNO
Uso: error=HostErrorno()
-

Ritorna il codice d'errore corrente di host-lookup.

1.36 hosterrorstring

HOSTERRORSTRING
Uso: errorString=HostErrorString(code)
<code/N>

Ritorna la stringa di errore di host-lookup associata al codice code.
La stringa è non localizzata. Dovrebbe essere lo stack a localizzarla, non io.

1.37 inetcksum

INETCKSUM
Uso: cksum=InetCksum(data,len)
<data>,[len/N]

Calcola la checksum Internet su data per len bytes.
Se len è omesso, la ckecksum viene calcolata su tutto data.
La checksum calcolata è quella che, ad esempio, è nello header IP, ovvero:
"il complemento ad uno della parola a 16 bits somma del complemento ad uno di ciascuna parola di 16 bits di 'data', per una lunghezza di 'len' bytes";
se 'len' è dispari, un byte di pad posto a 0 è aggiunto in coda a 'data' nel computo della checksum.

Facilita la creazione di pacchetti RAW.

1.38 ioctlsocket

IOCTLSOCKET
Uso: res=IOCtlSocket(socketfd,parm,value)
<socketfd/N>,<parm>,value/N>

Controlla i parametri di un socket.

Attualmente parm può essere uno tra:
- "FIOASYNC"
- "FIONBIO"

```
- "FIONREAD"
- "SIOCATMARK"

Ritorna -1 in caso d'errore.

ESEMPIO

    res=IOCtlSocket(socketfd,"FIONBIO",1)
    if res==-1 then say "errore nel settare il socket come non bloccante"
    else say "socket settato come non bloccante"
```

1.39 islibon

```
ISLIBON
Uso: res=IsLibON(name)
<name>

Testa su che stack si è o se una libreria è presente.

Attualmente name può essere uno tra:
- "MIAMI"           Miami è lo stack
- "TTCP"            TermiteTCP è lo stack
- "USERGROUP"       la usergroup.library è presente (inutile per ora)

Ritorna un booleano.
```

1.40 isdotaddr

```
Uso: res=IsDotAddr(addr)
<addr>

Testa se un addr è un indirizzo Internet dotted e ben formato oppure no.

Ritorna un booleano.

ESEMPIO

  say IsDotAddr('127.0.0.1')   ----> 1
  say IsDotAddr('.0.2')        ----> 0
  say IsDotAddr('www.iol.it')  ----> 0
```

1.41 issocket

```
Uso: res=IsSocket(socketfd)
<socketfd/N>

Testa su che il socket esiste.

Ritorna un booleano.
```

1.42 listen

```
LISTEN
Uso: res=listen(socketfd,backlog)
<socketfd/N>,<backlog/N>

Avverte il sistema che il socket vuole ricevere connessioni remote.

Backlog è il numero di connessioni massime accettate.
Un valore di backlog pari a 5 significa che si accettano connessioni in numero
massimo che, di solito, lo stack permette di definire in numero maggiore di 5.

Ritorna -1 in caso d'errore.
```

1.43 miamidisallowdns

```
Usage: MiamiDisallowDNS(status)
[status/N]

FUNZIONE SOLO PER MIAMI

Attiva/disattiva il lookup DNS. Il valore di default per status è 0.

Returns always 1.
```

1.44 miamigetpid

```
Usage pid = MiamiGetPid()
-

FUNZIONE SOLO PER MIAMI

Ritorna il pid del processo come packet chars.
```

1.45 miamionoffline

```
Usage: MiamiOnOffline(interface,status)
<interface>,[statu/N]

FUNZIONE SOLO PER MIAMI

Cambia lo stato dell'interfaccia specificata. Il valore di default
per status è 0.

iterface è una tra:
- "mi0"
- "lo0"

La funzione non attende il cambia e torna subito 1.
```

1.46 miamiisonline

```
Usage: res = MiamiIsOnline(interface)
<interface>

FUNZIONE SOLO PER MIAMI

testa se l'interfaccia è online.

iterface è una tra:
- "mi0"
- "lo0"

Ritorna un booleano ARexx.
```

1.47 miamisupportsipv6

```
Usage: res = MiamiSupportsIPV6()
-

FUNZIONE SOLO PER MIAMI

Testa la versione di Miami supporta il protocollo IPV6.

Ritorna un booleano ARexx.
```

1.48 obtainsocket

```
OBTAINSOCKET
Uso: sockfd=ObtainSocket(key,family,type,protocol)
<key><family>,<type>,<protocol>

La funzione serve per il passaggio dei socket tra una macro e l'altra.
Ottiene un socket rilasciato altrove con ReleaseSocket() o
ReleaseCopyOfSocket().
Key deve essere la stringa ritornata proprio da una di quelle due funzioni.

Il meccanismo usato per rilasciare i socket è il seguente:
- alla chiamata di ReleaseSocket() o ReleaseCopyOfSocket() il socket fa ancora
  parte dell'ambiente della macro in cui è stato creato;
- se il socket non viene ottenuto per mezzo di ObtainSocket(), all'uscita della
  macro in cui il socket è stato creato, esso viene deallocato;
- se invece il socket viene ottenuto per mezzo di ObtainSocket(), tutte le risorse
  ad esso connesse vengono passate alla macro chiamante ObtainSocket() ed è
  come se il socket fosse stato creato all'interno della macro chiamante
  ObtainSocket();
- al momento della chiamata a ObtainSocket() originale, se il socket non può
  essere ottenuto o c'è poca memoria, esso viene lasciato nell'ambiente della ↩
     macro
  in cui è stato creato;
- una volta rilasciato, il socket non può essere più usato nella macro ove è stato
  creato, salvo che non venga prima ottenuto;
```

```
- la key è un pacchetto di caratteri di lunghezza 8 in caso di successo o  ↩
    lunghezza
  4, paragonabile a null(), come per i messaggi che arrivano ad una porta ARexx;
  diversi controlli sono effettuati per verificare la coerenza di una key e  ↩
     dovrebbe
  essere difficile "imbrogliare" la ObtainSocket(), passandogli una key falsa;
  in ogni caso, si consiglia di non "giochicchiare" con le key ritornate da
  ReleaseSocket() o ReleaseCopyOfSocket() e passate ad ObtainSocket().

L'uso di questa funzione è comune all'interno di un service "concorrente", che
rilascia il socket ritornato da accept() con ReleaseSocket() e lancia una macro
che si occupa della gestione della connessione, passandogli come argomento
proprio la key ottenuta da ReleaseSocket().
La prima operazione che la macro lanciata dal service dovrebbe effettuare
dovrebbe essere proprio ottenere il socket con ObtainSocket() ed avvisare la
macro "padre" del risultato dell'operazione (ad esempio inviandole un messaggio
su una porta ARexx).

Ritorna un intero >=0 indicante il socketfd in caso di successo o
-1 in caso di fallimento.
```

1.49 openconnection

```
OPENCONNECTION
Uso: sockfd=OpenConnection(proto,localPort,host,remotePort,stem)
<proto>,<localPort>,[host],[remotePort],[stem/V]

Crea socket, fa un bind(), tenta una connessione.

Esempi sulle sue diverse forme aiuteranno a capire il suo uso
(proto è una fra "TCP" o "UDP"):

    res=OpenConnection(proto,4050)
        crea un socket
        fa un Bind() del socket sulla porta 4050

    res=OpenConnection(proto,"funnyService")
        cerca un service di nome "funnyService"
        crea un socket
        fa un Bind() del socket sulla porta relativa a "funnyService"

    res=OpenConnection("TCP","echo","www.nasa.org")
        cerca un service di nome "echo"
        risolve l'indirizizzo www.nasa.org
        crea un socket
        connette il socket a www.nasa.org:echo

    res=OpenConnection("UDP","echo","www.nasa.org")
        "Errore Arexx 18"

    res=OpenConnection("TCP",4000,"www.nasa.org")
        "Errore Arexx 18"

    res=OpenConnection("UDP",4000,"www.nasa.org","echo")
        cerca un service di nome "echo"
```

```
        risolve www.nasa.org
        crea un socket
        fa un bind() del socket sulla porta 4000
        connette il socket a www.nasa.org:echo

Il socket viene creato di type e protocol dipendenti dal proto specificato.
In pratica la funzione crea un socket e lo "binda" o "connette" a seconda del
protocollo specificato.
Il quinto argomento è uno stem, che se specificato, ed in caso di connessione,
viene settato a sockaddr_in.

A parte la confusione che ora avrai, l'uso della funzione è semplicissimo e
velocizza le procedure di connessione, ad esempio con

    res=OpenConnection("TCP","http","ftp.unina.it/pub/aminet")

ci si connette al service http aminet presso unina, saltando completamente, la
risoluzione dell'indirizzo, la creazione esplicita del socket e la connessione.

In caso di errore il socket viene deallocato.

In pratica, la funzione accetta da 2 a 5 argomenti:

- la forma con 2 argomenti:
  - protocollo       "TCP" oppure "UDP"
  - porta            nome del service o numero porta
  crea un socket e lo "binda" alla porta specificata.

- la forma con 3 argomenti
  - protocollo       "TCP"
  - porta            nome del service
  - indirizzo        indirizzo a cui connettersi
  crea un socket e lo connette al service dell'indirizzo specificato.

- la forma con 4 argomenti
  - protocollo       "UDP"
  - portaLocale      nome del service o numero porta
  - indirizzo        indirizzo a cui connettersi
  - portaRemota      nome del service o numero porta
  crea un socket, lo "binda" a portaLocale lo connette al service portaRemota
  dell'indirizzo specificato.

Ogni altra forma genera errore ARexx 18.

Ritorna un intero:
-  >=0  socketfd in caso di successo;
-  -3   in caso di service non trovato;
-  -2   in caso di errore host-lookup;
-  -1   in caso di errore generico di bsdsocket.library.

Disabilitata su TermiteTCP.
```

1.50 recv

```
RECV
Uso: res=recv(socketf,buff,len,flags)
<socketfd/N>,<buff/S>,<len/N>,[flags]

Riceve massimo len bytes da un socket connesso e li scrive in buff.

Flags è una giustapposizione di:
- "OOB"
- "PEEK"
- "WAITALL"
ovvero una stringa formata da una o più delle precedenti "parole" separate da
spazi, come ad esempio "OOB PEEK".

In caso di 0 byte ricevuti, buff viene posto a "".

Ritorna il numero di bytes ricevuti o -1 in caso d'errore.
```

1.51 recvfrom

```
RECVFROM
Uso: res=RecvFrom(socketfd,buff,len,flags,stem)
<sockefd/N>,<bufferStem/S>,<len/N>,[flags],[stem/V]

Riceve len bytes da un socket connesso o meno e li scrive in buff.

Flags è una giustapposizione di:
- "OOB"
- "PEEK"
- "WAITALL"
ovvero una stringa formata da una o più delle precedenti "parole" separate da
spazi, come ad esempio "OOB PEEK".

Se il socket è connesso, stem può essere omesso.
Altrimenti stem deve essere presente e settato a sockaddr_in (ma il controllo non
viene effettuato dalla funzione, perchè questa versione della libreria non tiene
traccia dei socket connessi).

In caso di 0 byte ricevuti, buff viene posto a "".

Ritorna il numero di bytes ricevuti o -1 in caso d'errore.
```

1.52 recvline

```
RECVLINE
Uso: res=RecvLine(socketfd,buff,len,flags,stem)
<socketfd/N>,<buffer/S>,len/N>,[flags],[stem]

Riceve una linea di max len bytes da un socket e la scrive in buff.

Questa funzione è una ReadLine() abbastanza "stupida" e non bufferizzata".

Si veda RecvFrom().
```

1.53 releasecopyofsocket

```
RELEASECOPYOFSOCKET
Uso: key=ReleaseCopySocket(socketfd)
<socketfd/N>

Rilascia una copia di un socket.
Il socket è ancora attivo all'interno della macro in cui è stato creato.

Ritorna una key stringa di 8 caratteri in caso di successo o 4, confrontabile
con null(), in caso di fallimento.

Si veda ObtainSocket().

ESEMPIO

    key=ReleaseCopyOfSocket(socketfd)
    if key==null() then ... /* errore */
    else ... /* successo */
```

1.54 releasesocket

```
RELEASESOCKET
Uso: key=ReleaseSocket(socketfd)
<socketfd/N>

Rilascia un socket.
Il socket non è più attivo all'interno della macro in cui è stato creato.

Ritorna una key stringa di 8 caratteri in caso di successo o 4, confrontabile
con null(), in caso di fallimento.

Si veda ObtainSocket().

ESEMPIO

    key=ReleaseSocket(socketfd)
    if key==null() then ... /* errore */
    else ... /* successo */
```

1.55 resolve

```
RESOLVE
Uso: addrValue=resolve(host)
<host>

Converte da nome host ad indirizzo Internet.
Chiama in sequenza inet_addr() e se questa fallisce, gethostbyname().

Ritorna l'indirizzo Internet di host oppure -1.

ESEMPIO
```

```
  addr=resolve("www.AlfonsoRanieri.edu")
  say addr        -----> -1 Hehe
```

1.56 send

```
SEND
Uso: res=send(socketfd,data,flags)
<socketfd>,<data>,[flags]

Invia data ad un socket connesso.

Flags è una giustapposizione di:
- "OOB"
- "DONTROUTE"
ovvero una stringa formata da una o più delle precedenti "parole" separate da
spazi, come ad esempio "OOB DONTROUTE".

Ritorna il numero dei bytes inviati o -1 in caso d'errore.
```

1.57 sendto

```
SENDTO
Uso: res=SendTo(socketfd,data,flags,stem)
<socketfd/N>,<data>,[flags],[stem/V]

Invia data ad un socket.

Flags è una giustapposizione di:
- "OOB"
- "DONTROUTE"
Ovvero una stringa formata da una o più delle precedenti "parole" separate da
spazi, come ad esempio "OOB DONTROUTE".

Se il socket è connesso, stem può essere omesso.
Altrimenti deve essere settato a sockaddr_in.

Ritorna il numero dei bytes inviati o -1 in caso d'errore.
```

1.58 setsocketbase

```
SETSOCKETBASE
Uso: res=SetSocketBase(stem)
<stem/V>

Setta alcuni parametri globali della base della libreria bsdsocket.library.

La funzione originale della bsdsocket.library è SocketBaseTagList(),  che è divisa
in GetSocketBase() e SetSocketBase() in questa libreria.
```

```
Di stem si devono impostare i campi corrispondenti ai parametri che si vogliono
settare.

I parametri attualmente ammessi sono:
- "BREAKMASK"
- "DTABLESIZE"
- "ERRNO"
- "HERRNO"
- "SIGEVENTMASK"
- "SIGIOMASK"
- "SIGURGMASK"
- "LOGFACILITY"
- "LOGMASK"
- "LOGSTAT"

Ritorna -1 in caso di errore.
```

1.59 setsocketsignals

```
Uso: SetSocketSignals(intrMask,ioMask,urgMask)
[intrMask/N],[ioMask/N],[urgMask/N]

Idica allo stack quale segnali usare in corrispondenza di
SIGINT, SIGIO and SIGURG.
la stessa cosa può essere fatto con SetSocketBase().
L'uso di questa funzione è deprecato negli autodoc di Miami.

Ritorna sempre 1.
```

1.60 setsockopt

```
SETSOCKOPT
Uso: res=SetSockOpt(socketfd,level,parm,value,value2)
<socketfd/N>,<level>,<parm>,<value>,[value2/N]

Setta il parametro parm del socket al livello level al valore value (e value2, se
ammesso).

Attualmente il livello può essere uno tra:
- "SOCKET"
- "IP"
- "TCP"

Parametri per "SOCKET" (con accanto il tipo) sono:
- "DEBUG"           N
- "REUSEADDR"       N
- "REUSEPORT"       N
- "KEEPALIVE"       N
- "DONTROUTE"       N
- "LINGER"          N
- "BROADCAST"       N
- "OOBINLINE"       N
```

```
- "SNDBUF"          N
- "RCVBUF"          N
- "SNDLOWAT"        N
- "RCVLOWAT"        N
- "SNDTIMEO"        N
- "RCVTIMEO"        N
- "TYPE"            N
- "ERROR"           N
- "EVENTMASK"       D

Nel caso di "EVENTMASK", value è una giustapposizione di:
- "ACCEPT"
- "CLOSE"
- "CONNECT"
- "ERROR"
- "OOB"
- "READ"
- "WRITE"
ovvero una stringa formata da una o più delle precedenti "parole" separate da
spazi, come ad esempio "CONNECT ERROR".

Nel caso di "LINGER", "SNDTMEO" e "RCVTIMEO" anche value2 può essere
passato.

Parametri per "IP" sono:
- "HDRINCL"         N
- "TTL"             N
- "TOS"             N

Parametri per TCP sono:
- "NODELAY"         N
- "MAXSEG"          N
- "NOPUSH"          N
- "NOOPT"           N

Ritorna -1 in caso d'errore.

ESEMPIO

    /* settiamo il socket per informarlo che i dati scritti su di
       esso contengono già lo header ip */
    res=SetSockOpt(socketfd,"IP","HDRINCL",1)
    if res==-1 then do
        say "non posso settare il socket come HDRINCL"
        exit
    end
    /* fatto. Ora sono guai vostri :-) */
```

1.61 shutdown

```
SHUTDOWN
Uso: res=ShutDown(socketfd,how)
<socketfd/N>,<how/N>

Causa la chiusura di una parte o di tutta una connessione full-duplex.
```

```
Se how è 0 la ricezione è chiusa.
Se how è 1 la trasmissione è chiusa.
Se how è 2 sia la ricezione che la trasmissione sono chiuse.

Dealloca il socket.

Ritorna -1 in caso d'errore.
```

1.62 sockatmark

```
Usage: res = SockatMark(socketfd)
<socketfd/N>

FUNZIONE SOLO PER MIAMI

Testa se il socket è out of bans.

Ritorna un booleano ARexx.
```

1.63 socket

```
SOCKET
Uso: sockfd=socket(family,type,protocol)
<family>,<type>,<protocol>

Crea un socket e tutte le risorse ad esso associate.

Si ricordi che solo le funzioni
- accept()
- Dup2Socket()
- ObtainSocket()
- OpenConnection()
- socket()
creano un socket, che è la base di ogni operazione di trasmissione e  ricezione.

Family per ora è solo "INET".

Non tutte le combinazioni di type e protocol hanno senso e/o funzionano su gli
stack AmigaOS.

Ritorna il socketfd come un intero >=0 o -1 in caso d'errore.
```

1.64 syslog

```
SYSLOG
Uso: res=SysLog(text)
<text>

Scrive text in syslog.
```

1.65 waitselect

```
WAITSELECT
Uso: res=WaitSelect(stem,secs,micros,signals)
<stem/V>,[secs/N],[micros/N],[signals/N]

Attende eventi su sockets o timeout o segnali exec.

Un esempio aiuterà a capire il suo funzionamento:
supponiamo di avere due socket i cui socketfd sono sfd1 e sfd2;
vogliamo attendere che qualcosa capiti ai sockets, oppure un timeout di 10
secondi,
oppure un segnale exec specificato nella maschera signals (ad esempio un segnale
di una porta oppure un segnale allocato precedentemente):

    WAIT.READ.0 = sfd1  /* per attendere l'evento "pronto a ricevere" */
    WAIT.READ.1 = sfd2

    WAIT.WRITE.0 = sfd1 /* per attendere l'evento "pronto a sendare" */
    WAIT.WRITE.1 = sfd1

    WAIT.EX.0 = sfd1    /* per attendere una eccezione */
    WAIT.EX.1 = sfd2

    res = WaitSelect("WAIT",10,0,signals)

    /* a questo punto res è un intero:
       <0 errore
       =0 nessuno evento sui sockets
       >0 numero sei sockets su cui sono occorsi eventi

      A parte res, un qualcosa è occorso e possiamo testare ogni socket
      per vedere se qualcosa è accaduto su di esso
   */
   if wait.0.read then ...  /* il socket sfd1 è pronto per ricevere */
   if wait.1.write then ... /* il socket ssfd2 è pronto a sendare */
   if wait.0.ex then ...    /* una eccezione è occorsa sul socket sfd1 */

Ritorna -1 in caso d'errore.
```

1.66 ringraziamenti

- grazie a shido per il suo regalo "Grazie shido. Tanti ovetti per te :-)";

- grazie ad [X_MaN] che mi ha introdotto nel mondo delle irc ed in generale in
  Internet;

- grazie a poing per il suo aiuto;

- grazie ad Amiga "Possa vivere per sempre!";

- grazie a Kruse per il suo meraviglioso Miami "Si, mi registro al più presto.
  Tu però togli le backdoors, OK? Hehe"

1.67 bibliografia

- Quasi tutte le rfc (<nic.nordu.net:21> - ftp anonimo);

- "Unix Network Programming" - W. Richard Stevens PTR Prentice Hall;

- socket.library autodoc di MiamiSDK, AmiTCPSDK, TermiteTCPSDK,
  prelevabili liberamente su aminet o sui siti delle rispettive case
  produttrici dei tre stacks.

1.68 dafare

- un serio debugging, con l'aiuto di qualche betatester volenteroso;

- una serie di funzioni di basso livello, per spedire pacchetti RAW;

- una serie di funzioni di livello superiore (tipo OpenConnection(), anche se il
  mio scopo rimane quello di mantenere la API originale di bsdsocket.library);

- una serie di funzioni della miami.library e della usergroup.library;

1.69 note

Ho detto più volte che questo documento non vuole in nessun modo essere una guida all'uso delle funzioni della bsdsocket.library, ma voglio richiamare alcuni semplici elementi (e prego gli esperti in TCP/IP di scusare la mia superficialità) ↩
.

Per realizzare una connessione o in generale una trasmissione in Internet si deve specificare una cinquina:

{protocollo, hostMittente, portaMittente, hostDesinatario, portaDestinatario}.

La creazione di un socket con Socket() specifica solo il primo elemento di questa cinquina.

Il secondo ed il terzo sono specificati in maniera diversa a seconda del ↩
protocollo.
Ad esempio, in TCP lo stack li assegna automaticamente, in UDP bisogna fare un bind() ad una porta.

Gli ultimi due devono essere forniti e per farlo è necessario passare la struttura sockaddr_in, che non è altro che un indirizzo Internet ed una porta, al momento della connessione a Connect() od al momento della trasmissione con SendTO().

In pratica, la sequenza minima di operazioni da eseguire è:

Per un service TCP:
socket()
bind()
listen()

```
Accept()

Per un client TCP:
socket()
connect()

Per un service UDP:
socket()
bind()
recv()

Per un client UDP:
socket()
bind()
( connect() )
SendTo()
```

E' tutto abbastanza semplice salvo quando si deve cominciare a pensare alla gestione di diversi sockets ed alla contemporanea ricezioni di segnali exec, cose per cui rimando alla bibliografia citata ed in piccola parte ai miei esempi acclusi nell'archivio.
Questa parte della API di bsdsocket.library è "veramente brutta" (IMHO).

Si ricordi che ogni service a cui ci si collega è regolato da un ben determinato protocollo che bisogna rispettare.
Ad esempio un service pop3 (quello da cui si preleva la posta) è regolato dal protocollo POP3, un service irc è regolato dal protocollo IRC e così via.
Le fonti sono le rfc (in particolare la rfc1920 per avere una lista delle ultime versioni dei vari protocolli Internet, anche so credo che la 1920 sia ormai obsoleta, comunque...).

Nessun esempio di creazione di pacchetti "a mano" è accluso. Per la maggior parte, essi non sono esempi "buoni".

In qualsiasi momento si può uscire da una macro senza chiudere un bel niente.
In ogni modo, è meglio rilasciare al più presto sockets non usati.
Non ci si preoccupi se alcuni sockets restano in stato di attesa, dopo la loro chiusura, (cosa leggibile ad esempio con miaminetstat su Miami), perché è "normale".

La deallocazione automatica delle risorse è realizzata sfruttando i campi pr_ExitCode e pr_ExitData della struttura Process connessa con la macro ARexx.
All'uscita una catena di pr_EXitCode(pr_EXitData) viene chiamata.
Questo può essere un problema se si usano altre librerie che sfruttano questi campi della struttura Process (non è un problema con altre mie librerie, perché come detto una catena di queste funzioni viene chiamata e per lo stesso motivo neanche con la init_inetd() di AmiTCP sorgono particolari problemi).

Si è scelto di rinunciare ad una maggiore velocità, aprendo e chiudendo la bsdsocket.library quando necessario, cosicché in ogni momento si può chiudere lo stack senza problemi; alla chiusura dello stack, se qualche macro ancora usa la bsdsocket.library, essa viene segnalata dallo stack stesso con un CTRL-C; stai quindi ben attento alla ricezione di questo segnale, che oltre ad indicare che l'utente vuole uscire dalla macro, può anche indicare che lo stack stesso sta per chiudere; in ogni caso la macro può rifiutarsi di ubbidire al CTRL-C da parte dello stack senza alcun problema.

1.70 cambiamenti

```
Cambiamenti rispetto alle versioni < 1.9:

- gli indirizzi Internet erano ritornati e passati come interi ARexx.
  Ciò causava non pochi problemi e si è scelto di usare indirizzi Internet in ↩
     forma
  dotted, ovvero gli indirizzi tornati da resolve() o settati dalle funzioni di
  get host sono nella forma XXX.XXX.XXX.XXX (vedere ad esempio socket.doc/ ↩
     inet_addr).
  Sono state eliminate le funzioni InetAddr() ed InetNtoA().
  E' stata aggiunta la funzione IsDotAddr() che torna un booleano indicante se
  l'indirizzo passato è nella forma dotted o no.
  In ogni caso si può risalire all'intero rappresentante l'indirizzo Internet
  attraverso la funzione Addr2C().
  Non dovrebbe cambiare un granchè, poichè resolve() torna la stringa "-1" se
  l'indirizzo è errato o non esiste, ma diverse modifiche sono necessarie.

Cambiamenti rispetto alle versioni < 2.0:
- le funzioni rmh sono ora nella rmh.library, parte dell'archivio rxsocket.lha.
```

1.71 supportoinetd

```
Il supporto inet di AmiTCP e Miami (TermiteTCP non ha inetd) è realizzato in ↩
    questa maniera:

- come nome servizio in inetd va inserito il nome di un programma (vedi dopo) che ↩
    lancia
  la macro;

- poiche in entrambi gli stack, inetd passa i dati necessari per ottenere il ↩
    socket
  accettato da inetd nei campi pr_ExitCode e pr_ExitData, non è possibile lanciare
  la macro un il programma "rx", ma si deve usare un programma che provi ad ↩
     ottenere
  il socket, crei una struttura socket privata nella base della libreria e chiami ↩
     la
  macro; il programma è fornito nell'archivo e si chiama rxs, va usato solo in ↩
     inetd;
  riceve un solo argomento, il nome della macro da lanciare;

- se la macro non esiste, il socket viene chiuso; se la macro esiste, il socket ↩
    relativo
  alla connessione che deve gestire è quello di id 0;
  per verificare all'interno della macro, se il socket 0 esiste si può usare " ↩
     IsSocket(0)";

Un esempio è fornito: una macro che attende una stringa di max 256 caratteri e la ↩
    riinvia
rovesciata. Per attivare il service,
- copiare in c: rxs ed in ram: pserv.rexx
- aggiungere in service:
  prova 4000 tcp
- aggiungere in inetd:
```

```
  prova stream tcp nowait root c:rxs c:rxs ram:pserv.rexx
```

A questo punto si può usare pclient per verificare il service.