IBM TRAINING

Session V28

Socket Programming

Will J. Roden, Jr.

IBM
SYSTEM z9 AND zSERIES EXPO
October 9 - 13, 2006

Orlando, FL

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

CICS*
DB2*
DB2 Connect
DB2 Universal Database
DFSMS/MVS*
DFSMS/VM*
e-business logo*
Enterprise Storage Server
ESCON
FICON
GDDM
HiperSockets
IBM*
IBM logo
Language Environment*
MQSeries*
Multiprise*
MVS
NetRexx
OpenEdition*
OpenExtensions
OS/390*
Parallel Sysplex*
PR/SM
QMF
RACF*
RAMAC*
RISC
S/370
S/370
S/390*
S/390 Parallel Enterprise Server
Virtual Image Facility
VisualAge*
VisualGen*
VM/ESA*
VTAM*
VSE/ESA
WebSphere
z/Architecture
z/OS
zSeries
z/VM*

* Registered trademarks of the IBM Corporation

The following are trademarks or registered trademarks of other companies.

Lotus, Notes, and Domino are trademarks or registered trademarks of Lotus Development Corporation.
Tivoli is a trademark of Tivoli Systems Inc.
Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries
UNIX is a registered trademark of The Open Group in the United States and other countries.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

IBM considers a product "Year 2000 ready" if the product, when used in accordance with its associated documentation, is capable of correctly processing, providing and/or receiving date data within and between the 20th and 21st centuries, provided that all products (for example, hardware, software and firmware) used with the product properly exchange accurate date data with it. Any statements concerning the Year 2000 readiness of any IBM products contained in this presentation are Year 2000 Readiness Disclosures, subject to the Year 2000 Information and Readiness Disclosure Act of 1998.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Disclaimer

The information contained in this document is not intended to be an assertion of future action by IBM. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adopt these techniques to their own environment do so at their own risk.

In this presentation, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this presentation was determined in a controlled environment and, therefore, the results which may be obtained in other operating environments may vary significantly. Users of this presentation should verify the applicable data for their specific environment.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

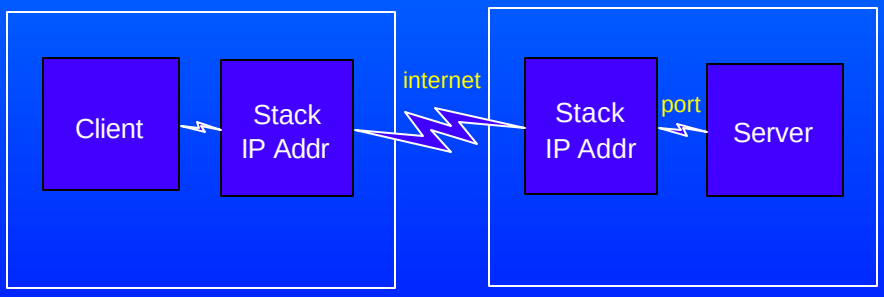
Any feedback that you give IBM regarding this presentation will be treated as non-confidential information. IBM reserves the right to use this information in any form.

Agenda

- What are sockets?
- C Socket Calls
  - Basics of Sockets
- REXX Socket Calls
- Pipeline Socket Stages
- Sniffer

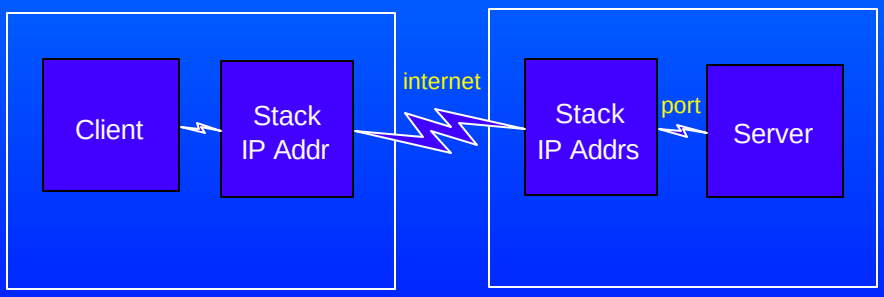
What are Sockets?

- Goal
  - Provide a way for two computers to communicate
- Socket
  - One way for that to happen



What are Sockets?

- What is needed?
  - Get communication with the stack
  - Describe yourself to the stack
  - Tell the stack if you are asking (client) or waiting to be asked (server)
  - Ask or wait to be asked, then wait for data
  - Send / Receive communications
  - End communication



What are sockets?

- What is needed?
  - Get communication with the stack
    - socket()
  - Describe yourself to the stack
    - bind()
  - Tell the stack if you are asking (client) or waiting to be asked (server)
    - connect() or listen()
  - Wait to be asked (server) or wait for data (both)
    - accept() or select()
  - Send / Receive communications
    - send() or recv()
  - End communication
    - close()

What are Sockets?

- Just like a new Dr's office
  - socket() - Register your business
  - bind() - Put up the sign with your skills
  - listen() - Put an advertisement in the paper
    - Mention the number of chairs in your waiting room
  - accept() or select() - Wait for patients
    - Put "Open" sign on your door
    - Assign patients to their examination room
  - send() or recv() - Communicate with patient
  - close() - discharge patient
    - Put "Closed" sign on your door

C Socket Calls

- Socket - establishes connection with stack
  - Used by both client and server
  - socket(domain, type, 0);
    - domain
      - AF_INET - use the internet
        - AF_IUCV - use VM's IUCV
        - AF_UNIX - Local use
    - type:
      - SOCK_STREAM - TCP
        - SOCK_DGRAM - UDP
        - SOCK_RAW - IP, ICMP
    - protocol:
      - 0, IPPROTP_UDP, IPPROTO_TCP
  - Returns the socket number (int)

C Socket Calls

- Bind - describe who you are to the stack
  - Used by server and optionally client
  - bind(socket, struct sockaddr *, sizeof());
    - if AF_INET specified in socket() call
    - use 'inet_addr("1.2.3.4");' to specify IP @ in sin_addr

```
struct sockaddr_in {
   unsigned char sin_len;      /* len of sockaddr or 0 */
   unsigned char sin_family;   /* AF_INET              */
   unsigned short sin_port;    /* htons(5001)          */
   struct in_addr sin_addr;    /* INADDR_ANY           */
   unsigned char sin_zero[8]; /* 0's                   */
}
```

C Socket Calls

- Listen - tells stack that this port will only listen.
  - Used by server
  - listen(socket , backlog);
    - socket: rec'd from socket()
    - backlog: 0 to SOMAXCONN
      - Number of chairs in waiting room

C Socket Calls

- Accept - waits for and accepts a new connection
  - Used by server
    - adlen = sizeof adminIn;
  - Blocked, use fcntl() to unblock
  - accept(SO, (struct sockaddr *) &adminIn, &adlen);
  - Returns the new socket number that was obtained for this connection. This is usually passsed to a different thread allowing this thread to get ready for another connection.

C Socket Calls

- Connect - attempt to establish a connection
  - Only used by client
  - Will also do bind's function
  - connect(socket, struct sockaddr *, sizeof());
    - if AF_INET specified in socket() call

```
struct sockaddr_in {
   unsigned char sin_len;       /* len of sockaddr or 0 */
   unsigned char sin_family;  /* AF_INET                */
   unsigned short sin_port;    /* htons(5001)            */
   struct in_addr sin_addr;     /* INADDR_ANY          */
   unsigned char sin_zero[8]; /* 0's                     */
}
```

C Socket Calls

- Send
  - Used by both client and server
  - Blocked, use fcntl() to unblock
  - bytes = send(socket, &buffer, len, flags)
    - &buffer is ptr to the buffer
    - len is the length of the buffer
    - flags are usually '0'
- consider converting buffer to ascii or ebcdic
- others: sendmsg, sendto, write, and writev

C Socket Calls

- Recv
  - Used by both client and server
  - Blocked, use fcntl() to unblock
  - bytes = recv(socket, &buffer, len, flags)
    - &buffer is ptr to the buffer
    - len is the length of the buffer
    - flags are usually '0'
- consider converting result to ascii or ebcdic
- others: recvmsg, recvfrom, read, and readv

C Socket Calls

- Select - waits for multiple events
  - Used by both client and server
  - Can wait for accept, send, recv, or time out
  - rc2 = select(num_fds, read, write, except, timeout);
    - rc2 contains the number of ready file descriptors
    - Test for ... with FD_ISSET()
    - zero and reset FDs before every call to select().

```
fd_set reading;

FD_ZERO(&reading);                  /* reset to zero      */
FD_SET(0, &reading);                /* std in             */
FD_SET(socket, &reading);           /* other socket       */
num_fds = 1+socket;                 /* max socket num */
rc2 = select(num_fds, &reading, NULL, NULL, NULL);
FD_ISSET(0, &reading)               /* test               */
```

C Socket Calls

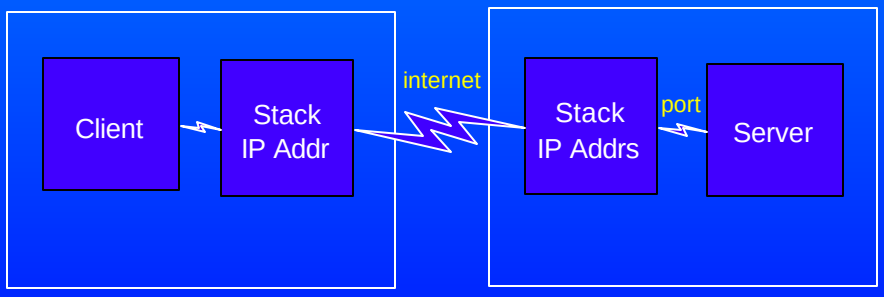
- Close - shuts down the socket
  - Used by both client and server
  - rc = close(socket);
    - socket - rec'd from socket()

C Socket Calls

- Error Checking
  - rc is different on different calls
    - socket, accept: <= socket number
    - send, recv: <= number of bytes
    - select: <= number of ready file descriptors
    - all others: rc=0 <= success
    - all: rc=-1 <= error
      - Check variable 'errno' for specifics

C Socket Calls

- **Client**
  - socket()
  - connect()
  - send() recv()
  - close()
- **Server**
  - socket()
  - bind()
  - listen()
  - accept()
  - recv() send()
  - close()



C Socket Calls

- **Example - socket preparation**

```
int SO, NS, rc, adlen;
struct  sockaddr_in sinet;
struct  sockaddr_in adminIn;

SO = socket(AF_INET, SOCK_STREAM, 0);
rc = bind(SO, (struct sockaddr *) &sinet, sizeof(sinet));
rc = listen(SO, 5);
adlen = sizeof adminIn;
NS = accept(SO, (struct sockaddr *) &adminIn, &adlen);
```

C Socket Calls

- **Example - receive data**

```
char buffer[999]
char *pBuf;
int read_len = 0, nb = 0, len = 998;

while (read_len < len) {
  nb = recv(NS, pBuf + read_len, len - read_len, 0);
  if (nb > 0)
    read_len += nb;
  else
    break;
}
    ........ or ........
rc = recv(NS, buffer, len, 0);
buffer[rc] = '\0';
```

C Socket Calls

- **Example - send data**

```
char out[999];  /* data to be sent */
int rc=0, len;

len = strlen(out);
rc = send(NS, out, len, 0);
```

- **Example - Close**

```
rc = close(NS);
```

REXX Socket Calls

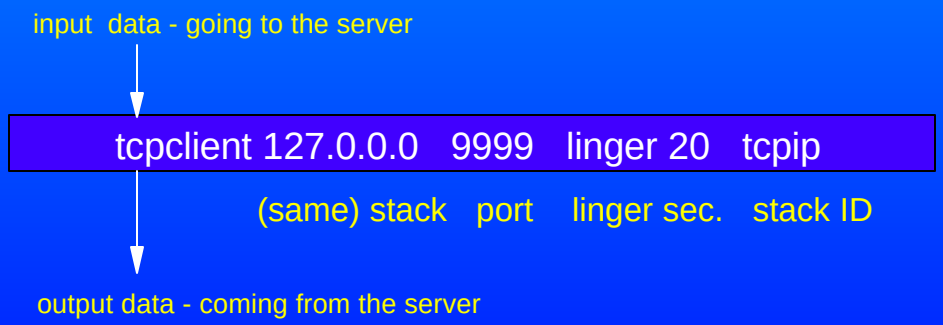
- socket('SOCKET', 'AF_INET', 'SOCK_STREAM')
- socket('BIND', socket, 'AF_INET port ip@')
- socket('LISTEN', socket, backlog)
- socket('ACCEPT', socket)
- socket('CONNECT', socket, AF_INET port ip@')
- socket('SEND', socket, data, '')
- socket('RECV', socket)
- socket('SELECT', mask, timeout)
- socket('CLOSE', socket)

Pipeline Socket Stages

- tcpclient - all client calls
  - socket, connect, send, recv, close
- tcplisten - server setup
  - socket, bind, listen, accept, close
- tcpdata - server data movement
  - send, recv, close

Pipeline Socket Stages

- Tcpclient
  - primary input stream -> tcpip
  - tcpip -> primary output stream
  - stops when output stream terminated or 20 seconds have passed

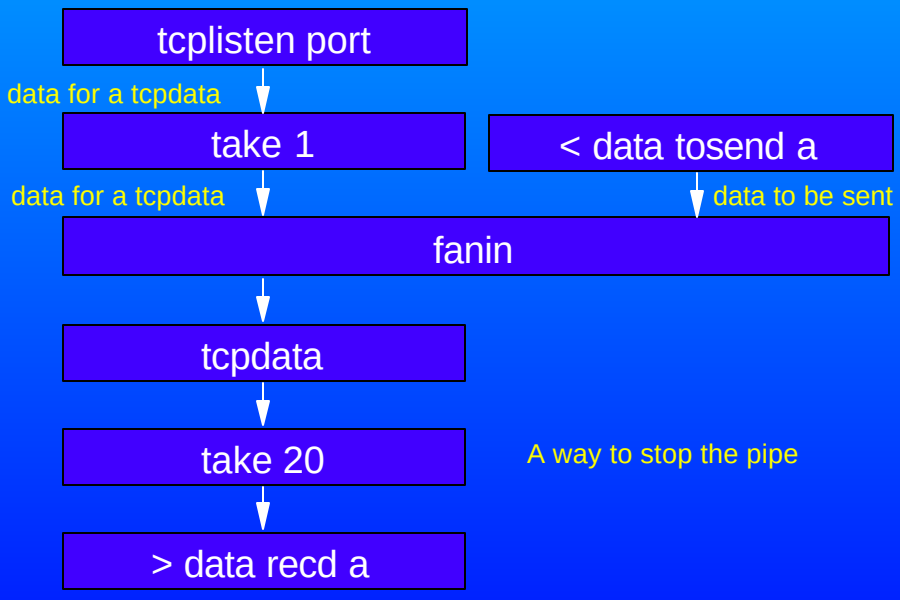


Pipeline Socket Stages

- Tcplisten
  - must be the first stage
  - passes information to tcpdata
  - stops when output stream terminated
- Tcpdata
  - input stream:
    - first record must be from tcplisten
    - following records are treated as data
  - output stream is data from the network
    - can do blocking
  - stops when output stream terminated
- Considerations
  - how to handle multiple connections
  - how to stop the pipeline

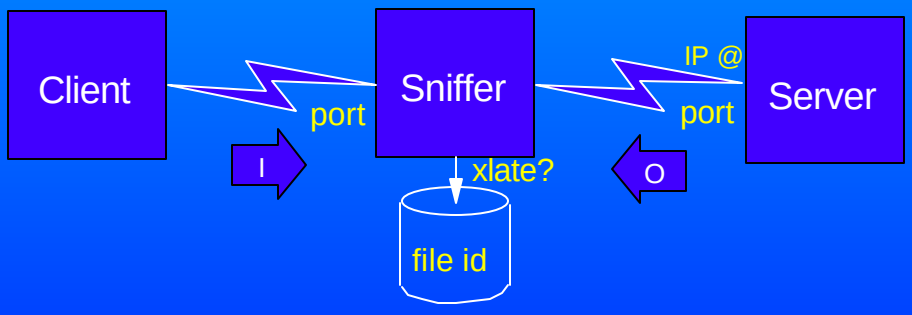
Pipeline Socket Stages

- Flow diagram for only one connection



Pipeline Sniffer

- What is a sniffer?
  - ►Debug tool that records packets sent between a client and a server



- To run TCPSNIFF

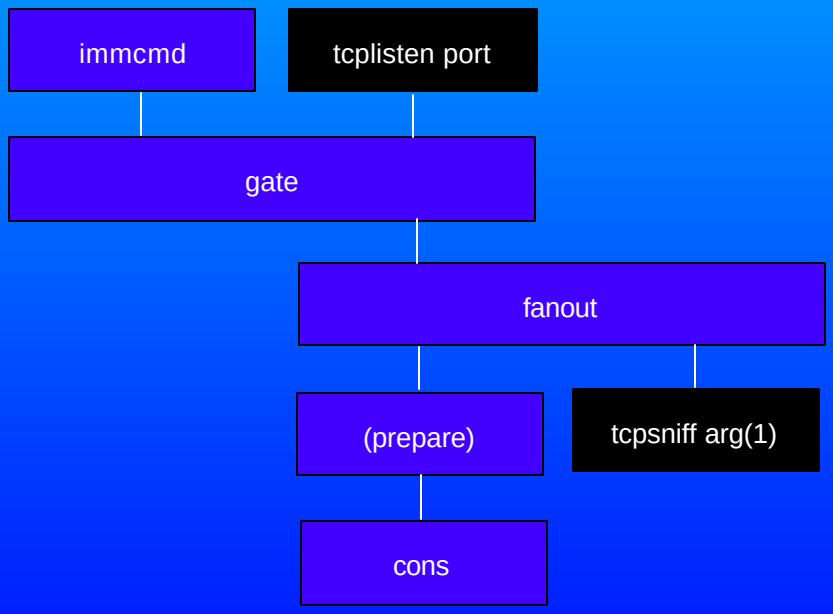
tcpsniff 10001 — input port

1.2.3.4 3000 — IP@out and port

tcpsn output a — output fileid

Pipeline Sniffer Code

- TCPSNIFF EXEC - diagram



Pipeline Sniffer Code

- TCPSNIFF EXEC - page 1

```
/* TCP port sniffer.                                    */
/*                        John Hartmann  7 Feb 1996 12:38:57 */
Signal on novalue

parse arg port remote_sys remote_port fn ft fm '(' options

/*********************************************************************/
/* Send TCP socket  data to a remote host and  pass the result back. */
/* To sniff  a telnet session, this  program would be the  server as */
/* seen by the  telnet client.  The real telnet  server is specified */
/* as the second and third operand.                                  */
/*                                                                   */
/* Specify (ascii  if the data stream  is in ASCII and  you want the */
/* log file in EBCDIC (from 850 to 1047).                            */
/*********************************************************************/
say 'Type:  stop   to stop this sniffer.'
```

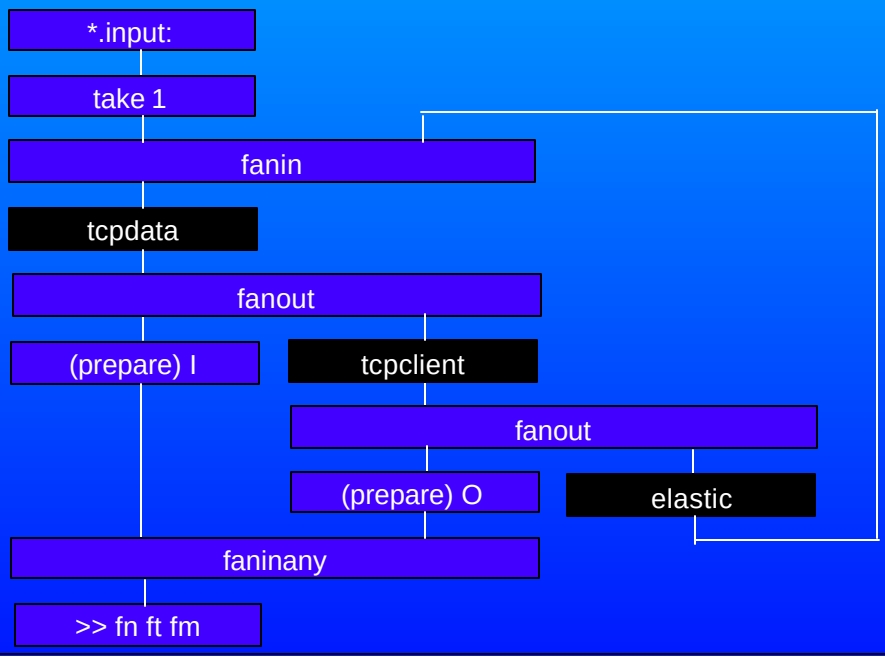
Pipeline Sniffer Code

TCPSNIFF EXEC - page 2

```
address command
'PIPE (end \ name TCPSNIFF.EXEC:18)',
  '\immcmd stop',
  '|g:gate',
  '\tcplisten' port,
  '|g:',
  '|go:fanout',
  '|change 72 //x00',          /* Make sure positive...      */
  '|change 71 //x00',          /* ...numbers for...          */
  '|change 70 //x00',          /* ......formatting.          */
  '|change 69 //x00',
  '|spec 67.2 c2d 1',
     '69.2 c2d nw  71.2 c2d nw',
     '73.2 c2d nw  75.2 c2d nw',
  '|spec /Request from port/ 1 w1 nw /on/ nw',
     'w2 nw /./ n w3 n /./ n',
     'w4 n /./ n w5 n',
  '|cons',
  '\go:',
  '|tcpsniff' arg(1)
Exit RC
```

Pipeline Sniffer Code

TCPSNIFF REXX - diagram



Pipeline Sniffer Code

- TCPSNIFF REXX - page 1

```
/* Sniffer to process one connection request                    */
/*                          John Hartmann  7 Feb 1996 12:49:10 */
Signal on novalue
signal on error

parse arg . remote_sys remote_port fn ft fm '(' options
upper options
xlate=''
If wordpos('ASCII', options)>0
  Then xlate='|xlate from 850 to 1047'
```

Pipeline Sniffer Code

- TCPSNIFF REXX - page 2

```
do forever
  'peekto'
  'callpipe (end \ name TCPECHOD.REXX:7)',
    '\*.input:',
    '|take 1',
    '|i:fanin',
    '|tcpdata',
    '|in:fanout',
      xlate,
    '|change ,,I ,',
    '|wi:faninany',
    '|>>' word(fn 'TCP', 1) word(ft 'TRACE', 1) word(fm 'A', 1),
    '\in:',
    '|tcpclient' remote_sys remote_port,
    '|o:fanout',
      xlate,
    '|change ,,O ,',
    '|wi:',
    '\o:',
    '|elastic',
    '|i:'
  say 'Connection closed.'
end
error: exit RC*(RC<>12)
```

What did we discuss?

- What are sockets?
- C Socket Calls
  - Basics of Sockets
- REXX Socket Calls
- Pipeline Socket Stages
- Sniffer

References

- John Hartmann (IBM Denmark)
  - JOHN@dk.ibm.com
    - Pipelines author and Sniffer author
- Melinda Varian's pitch on Marist
  - "Plumbing the Internet: CMS/TSO Pipelines Support for TCP/IP", revised 06/09/97
  - http://vm.marist.edu/%7Epipeline/#MWV

References

- VM Library
  - "OE for VM/ESA Sockets Reference" SC24-5741
  - "REXX/VM Reference" SC24-5770 Ch 16
  - "CMS Pipelines Users Guide" SC24-6077 Ch 11
  - "CMS Pipelines Reference" SC24-6076
  - "CMS Pipelines Author's Edition" SL26-0018

Development Contacts

- Will J. Roden, Jr.
  - Phone: (607) 429-3278
  - Internet: RODEN@US.IBM.COM
  - Web: http://www.vm.ibm.com/devpages/roden
  - Postal mail
    - IBM Department G79G
    - 1701 North Street
    - Endicott, NY 13760 U.S.A.