Session NM059

TCP/IP Programming on VMS

Geoff Bryant
Process Software

Course Roadmap

- NM055 (11:00-12:00) Important Terms and Concepts
  - TCP/IP and Client/Server Model
  - Sockets and TLI
  - Client/Server in TCP/IP
- NM056 (1:00-2:00) Socket Routines
- NM057 (2:00-3:00) Library Routines
- NM058 (3:00-4:00) Sample Client/Server
- NM059 (4:00-5:00) VMS specifics (QIOs)
- NM067 (6:00-7:00) Clinic - Q&A

TCP/IP Programming

Slides and Source Code available via anonymous FTP:

Host:: ftp.process.com
Directory: [pub.decus]
Slides: DECUS_F96_PROG.PS
Examples: DECUS_F96_PROG_EXAMPLES.TXT

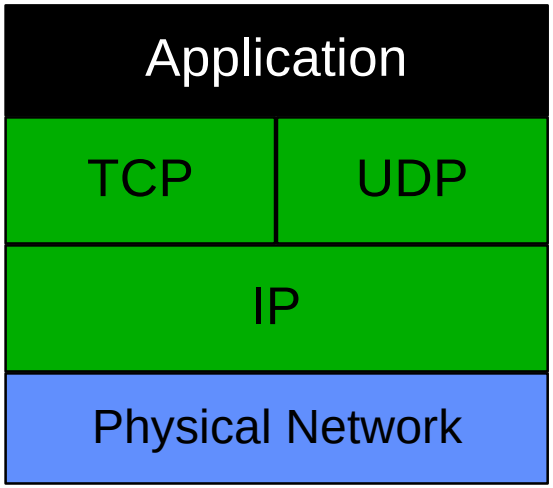
Host: ftp.opus1.com
Slides: DECUS_F96_PROG.PS
Examples: DECUS_F96_PROG_EXAMPLES.TXT

Overview

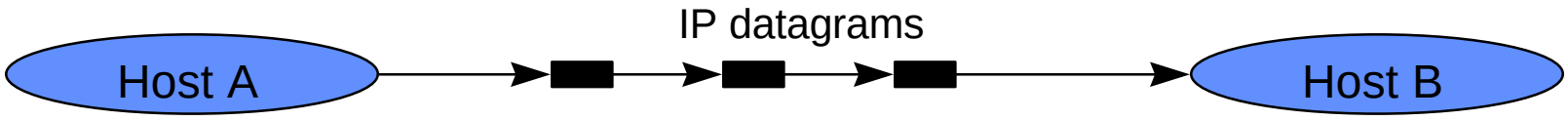
- TCP/IP Concepts
  - TCP byte stream service
  - UDP datagram service
  - TCP or UDP? (how to choose)
- Sockets vs. QIOs
- Stream (TCP) Client calls
- Stream (TCP) Server calls
- Datagram (UDP) Client calls
- Datagram (UDP) Server calls
- Multithreaded servers

TCP/IP Concepts

- Networking services
  - TCP - reliable byte stream service
  - UDP - connectionless datagram service
  - IP - connectionless datagram service
- Naming communication endpoints
  - Internet Addresses
  - Port Numbers
  - Servers contacted at well-known ports
    - Ports 1 - 1023 are privileged port numbers
    - Ports 1024 - 65535 are unrestricted
- Sockets
  - BSD Unix abstraction used to access the network
  - Available in DECC Runtime Library

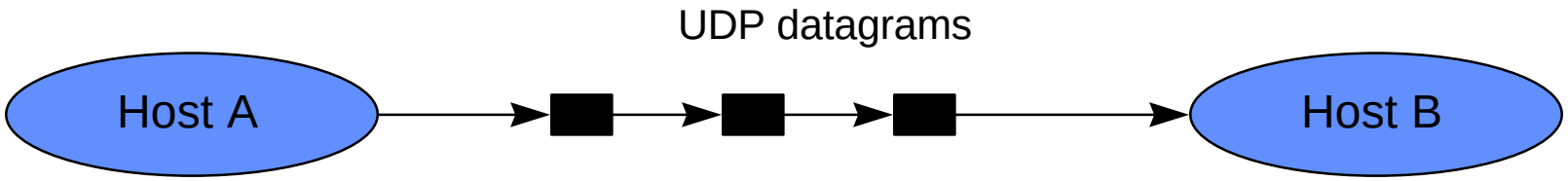


IP - Internet Protocol



- Lowest Layer of TCP/IP suite
- Typically not used directly by application programs
- Connectionless
- “Unreliable” datagram service
  - Low overhead
  - Delivery of a datagram is not guaranteed
  - Datagrams may be delivered out of order
  - Duplicate datagrams may be received
- Upper layer protocols implement reliability
- Used by TCP and UDP

UDP - User Datagram Protocol



- "Thin" Layer on top of IP (adds port concept)
- Used directly by application programs
- Like IP (low overhead, connectionless, unreliable)
- Not reliable
  - Client must retransmit request if no reply received
  - Server may receive duplicate datagrams
- Best for request/reply applications
- Used by DNS, NFS, SNMP, RIP, etc....

TCP - Transmission Control Protocol

- Layered on top of IP
- Adds port and byte stream concepts
- Used directly by application programs
- Connection oriented
- Reliable byte stream service
  - Delivery of bytes is guaranteed
  - Bytes delivered in order
  - No duplication of bytes
- Used by FTP, TELNET, SMTP, etc....

TCP - Transmission Control Protocol

TCP does NOT do messages

- TCP is a reliable BYTE STREAM protocol
  - No one-to-one correspondence between send operations and receive operations
    - Number of send operations may NOT be equal to the number of receive operations
    - A receive operation may receive fewer bytes than was sent in a send operation
    - A receive operation may receive data from several sends (more than one message may be received)

TCP - Transmission Control Protocol

TCP does NOT do messages

- Your application must implement messages
  - Client/Server need to agree on and implement a message protocol:
    - Messages separated by <CR><LF>
    - Messages preceded by byte count length
    - Messages of fixed length
  - Choose TCP for applications which require reliable transmission

TCP or UDP - Which to use?

- TCP
  - Connection oriented, reliable byte stream
  - More overhead than UDP
- UDP
  - Connectionless, unreliable, datagram service
  - Less overhead than TCP

- Choose TCP for applications which require reliable transmission
  - File transfer (FTP)
  - Virtual terminals (TELNET)
- Choose UDP for:
  - Simple request/reply applications (DNS)
  - Applications which can tolerate unreliable transmission (Routing protocols such as RIP)

Sockets vs. QIOs

- Sockets
  - Portable to other platforms
  - Available via DEC C RTL
    - 3rd Party Socket Libraries
    - Recommend DEC C RTL, supported by 3rd parties
- $QIOs
  - VMS only
  - Can easily write asynchronous code
  - Can easily write multithreaded servers
  - BGDRIVER (Digital TCP/IP Services/UCX interface)
    - 3rd Party interfaces: INETDRIVER, TCPDRIVER, UDPDRIVER
    - Recommend BGDRIVER, supported by 3rd parties

$QIO Programming

- SYS$ASSIGN to assign/create channel to device
- SYS$QIO
  - Queue an I/O request to device
  - Asynchronous
  - AST routine invoked on completion (Optional)
  - Event Flag set on completion (Optional)
- SYS$QIOW
  - Synchronous - waits for completion
- SYS$DASSGN to deassign/release channel

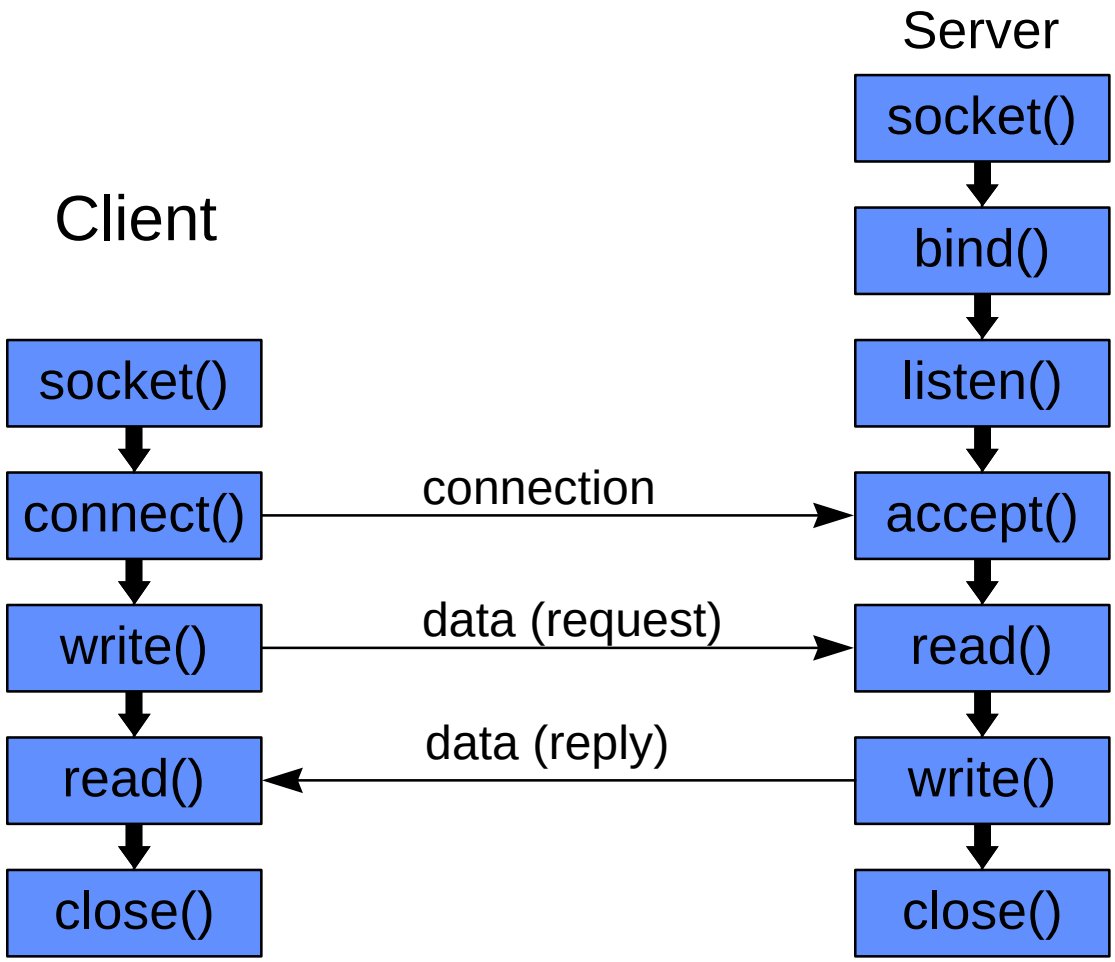
$QIO Programming

- SYS$QIO(EF,CHN,FUNC,IOSB,ASTADR,ASTARG,P1,...,P6)
  - EF Optional Event Flag
  - CHN Required Channel (SYS$ASSIGN)
  - FUNC Required Function to perform
  - IOSB Recommended I/O Status Block
  - ASTADR Optional AST routine address
  - ASTARG Optional AST parameter
  - P1-P6 Opt/Req Function specific args

BGDRIVER $QIO - Socket Equivalents

- SOCKET()
  - SYS$ASSIGN
    SYS$QIO IO$_SETMODE
- BIND()
  - SYS$QIO IO$_SETMODE
- LISTEN()
  - SYS$QIO IO$_SETMODE
- ACCEPT()
  - SYS$QIO IO$_ACCESS ! IO$M_ACCEPT
- CONNECT()
  - SYS$QIO IO$_ACCESS
- SEND()
  - SYS$QIO IO$_WRITEVBLK
- RECV()
  - SYS$QIO IO$_READVBLK
- CLOSE()
  - SYS$QIO IO$_DEACCES
    SYS$DASSGN
- GETxxBYyy
  - SYS$QIO IO$_ACPCONTROL

Stream (TCP) Client and Server



Stream (TCP) Client

1. Create a network device ($ASSIGN BG0:)
2. Create a socket ($QIO IO$_SETMODE)
3. Open connection to the server ($QIO IO$_ACCESS)
   - Specify internet address of server obtained from gethostbyname() or inet_addr
   - Specify port number of server's well-known port - obtained from getservbyname()
4. Send and receive data ($QIO IO$_WRITEVBLK and $QIO IO$_READVBLK)
5. Close the connection ($QIO IO$_DEACCESS)
6. Deallocate network device ($DASSGN)

Stream (TCP) Client

- Example client
  - Program prompts user for server's host name
  - Looks up server's internet address from host name
  - Connects to server at port 65000
  - Sends message to server
  - Closes connection

```
#define SERVER_PORT 65000
/*
**   UCX-compatible Stream (TCP) Client using BGDRIVER $QIOs
**
**  To build against TCPware on VAX:
**     $ CC/DEFINE=TCPWARE UCX_CLIENT
**     $ LINK UCX_CLIENT,TCPWARE:UCX$IPC/LIB,SYS$INPUT/OPTIONS
**     SYS$SHARE:VAXCRTL/SHARE
**
**  To build against TCPware on AXP:
**     $ CC/DEFINE=TCPWARE/STANDARD=VAXC UCX_CLIENT
**     $ LINK UCX_CLIENT
**
```

Stream (TCP) Client

```
**  To build against UCX on VAX:
**     $ CC UCX_CLIENT
**     $ LINK UCX_CLIENT,SYS$LIBRARY:UCX$IPC/LIB,SYS$INPUT/OPTIONS
**     SYS$SHARE:VAXCRTL/SHARE
**
**  To build against UCX on AXP:
**     $ CC/STANDARD=VAXC UCX_CLIENT
**     $ LINK UCX_CLIENT
**
*/
#include <in.h>
#include <netdb.h>
#include <string.h>
#include <starlet.h>
#include <descrip.h>
#include <iodef.h>

#ifdef TCPWARE
#include "TCPWARE_INCLUDE:ucx$inetdef.h"    /* Provided by TCPware install */
#else
#include <ucx$inetdef.h>                    /* Provided during UCX installation */
#endif
```

Stream (TCP) Client

```
/*
**  Macros to check VMS success/failure status
*/
#define SUCCESS(status)             (status&1)
#define FAILURE(status)             (!(status&1))
/*
** Values for boolean variables
*/
#define FALSE 0
#define TRUE  1
/*******************************/
/*      Client Program         */
/*******************************/
main() {
            int         status;                      /* For return status */
            short       channel;                     /* BG channel number */
            short       sck_parm[2];                 /* Socket creation parameter */
            short       iosb[4];                     /* I/O status block */
            char        buf[512];                    /* buffer to transmit */
            struct      SOCKADDRIN remote_host;      /* remote host's address & port */
            struct      hostent *he;                 /* returned from gethostbyname */
            char        server_name[256];            /* name of remote server */
            int         got_host;                    /* boolean */
```

Stream (TCP) Client

```
/*
**  UCX item list 2 descriptor used during $QIO IO$_ACCESS
*/
struct      IL2 {
            unsigned int il2_length;
            char *il2_address;
} rhst_adrs = {sizeof remote_host, &remote_host};

/*
**  String descriptor used during $ASSIGN
*/
struct      dsc$descriptor inet_dev =
            {10, DSC$K_CLASS_S, DSC$K_DTYPE_T, "UCX$DEVICE"};

/*
** Assign a channel to the UCX device.
*/
status = sys$assign( &inet_dev, &channel , 0, 0);
if (FAILURE(status)) {
            printf("Failed to assign channel to UCX device.\n");
            exit(status);
}
```

Stream (TCP) Client

```
/*
**  Prompt user for server name.
**  Lookup the corresponding internet address .
*/

got_host = FALSE;
while( got_host == FALSE ) {
  printf("Enter name of remote host:");
  gets( server_name );
  if ((he = gethostbyname( server_name )) == NULL )
    printf("Error, gethostbyname failed\n");
  else
    got_host = TRUE;
}
memcpy((char *)&remote_host.SIN$L_ADDR, he->h_addr, he->h_length );
```

Stream (TCP) Client

```
/*
** Create the socket
*/

sck_parm[0] = UCX$C_TCP;                        /* TCP/IP protocol */
sck_parm[1] = INET_PROTYP$C_STREAM ;   /* stream type of socket */

status = sys$qiow ( 3,                                  /* Event flag */
                        channel,                        /* Channel number */
                        IO$_SETMODE,            /* I/O function */
                        iosb,                           /* I/O status block */
                        0, 0,
                        &sck_parm , 0,          /* P1 Socket creation parameter  */
                        0, 0,
                        0, 0);

if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status)) {
                printf("Failed to create and bind the device socket.\n");
                exit(status);
}
```

Stream (TCP) Client

```
/*
** Connect to specified host and port number
*/

remote_host.SIN$W_FAMILY = UCX$C_AF_INET;          /* INET family */
remote_host.SIN$W_PORT = htons( SERVER_PORT );    /* port number */

status = sys$qiow ( 3,                              /* Event flag */
                    channel,                        /* Channel number */
                    IO$_ACCESS,                     /* I/O function */
                    iosb,                          /* I/O status block */
                    0, 0,
                    0, 0,
                    &rhst_adrs,                     /* P3 Remote IP address */
                    0, 0, 0);

if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status)) {
        printf("Failed to connect to remote host.\n");
        exit(status);
}
```

Stream (TCP) Client

```
/*
** Send message to the server
**
** Note that this client/server combination implements
** a very simple message protocol - the client simply
** sends a text string of any size and then closes
** the connection.   The server simply keeps reading
** (and printing the received text to SYS$OUTPUT)
** until the connection is closed by the client.
*/
strcpy( buf, "Hello there Mr. Server.  Are you listening?");
status = sys$qiow ( 3,                          /* Event flag */
                    channel,                    /* Channel number */
                    IO$_WRITEVBLK,              /* I/O function */
                    iosb,                       /* I/O status block */
                    0, 0,
                    buf,                        /* P1 buffer */
                    strlen(buf),                /* P2 buffer length */
                    0, 0, 0, 0);
if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status))
        printf("Failed to write to socket.\n");
```

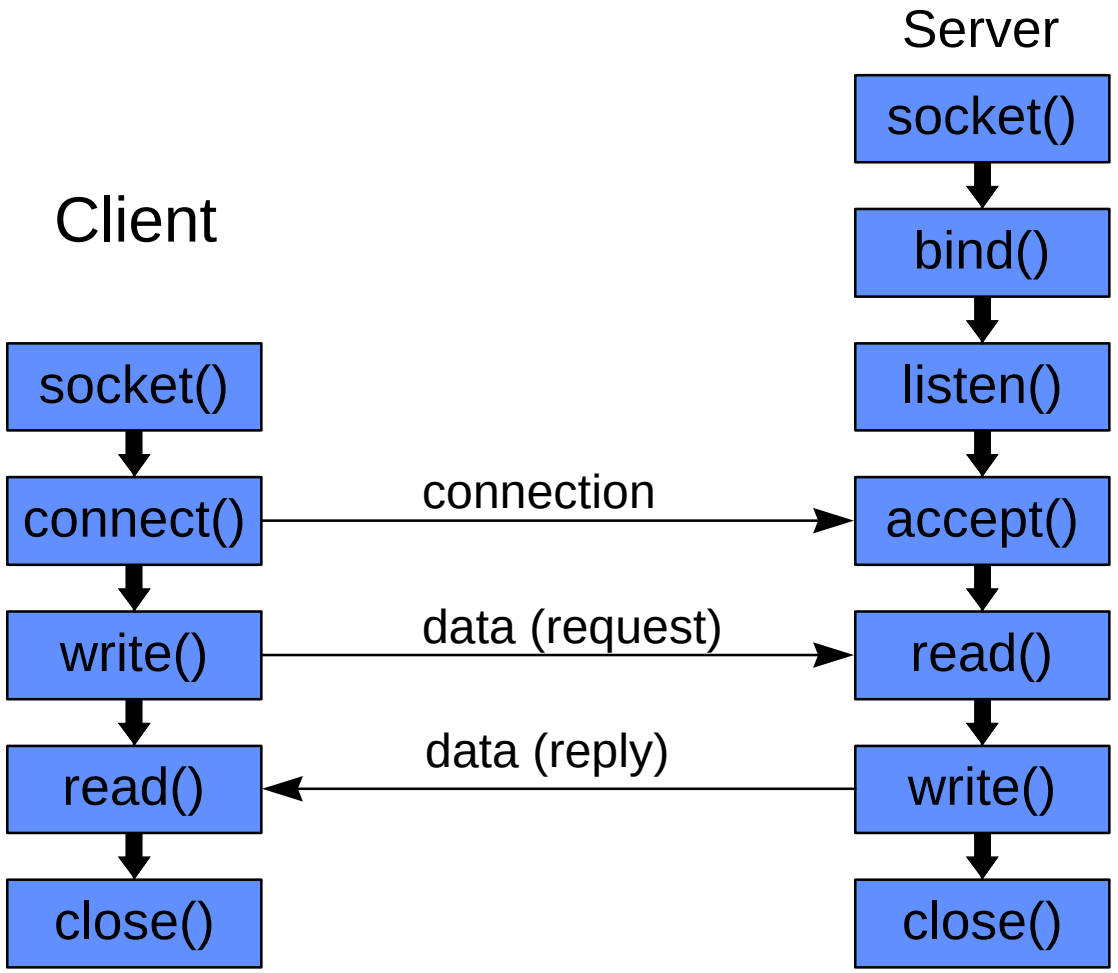
Stream (TCP) Client

```
        /*
        ** Close the socket  (optional - could just let $DASSGN
        **                  perform the close)
        */
        status = sys$qiow ( 3,
                                channel,
                                IO$_DEACCESS,
                                iosb,
                                0, 0,
                                0, 0, 0, 0, 0, 0);
        if (SUCCESS(status)) status = iosb[0];
        if (FAILURE(status))
                printf("Failed to close the socket.\n");

        /*
        ** Deassign the UCX device channel.
        */

        status = sys$dassgn( channel );
        if (FAILURE(status))
                printf("Failed to deassign the channel.\n");
}
```

Stream (TCP) Client and Server



Stream (TCP) Server

1. Create network device ($ASSIGN BG0:)
2. Create socket and bind to well-known port for the service ($QIO IO$_SETMODE)
   - "Official" servers use getservbyname() to obtain assigned port number for the service
   - Examples here use an unreserved port number
3. Put socket in listen mode ($QIO IO$_SETMODE)
4. Wait for and accept incoming connection
   - Create new BG device for connection ($ASSIGN BG0:)
   - Accept connection ($QIO IO$_ACCESS ! IO$M_ACCEPT)
   - Original BG device continues listening for new connections

Stream (TCP) Server

5. Receive and send data as needed for the service ($QIO IO$_READVBLK and IO$_WRITEVBLK)
6. Close the connection ($QIO IO$_DEACCESS)
7. Go to step 4
8. Deallocate the BG device ($DASSGN)

Stream (TCP) Server

- Example server
  - Bind and listen at port 65000
  - Accept incoming connection
  - Read message sent by client
  - Write message to SYS$OUTPUT
  - Close connection
  - Go back and accept next incoming connection

Stream (TCP) Server

```
#define SERVER_PORT  65000   /* server's port number to use */
/*
**
**   UCX-compatible Stream (TCP) Server using BGDRIVER $QIOs
**
**  To build against TCPware on VAX:
**     $ CC/DEFINE=TCPWARE UCX_SERVER
**     $ LINK UCX_SERVER,TCPWARE:UCX$IPC/LIB,SYS$INPUT/OPTIONS
**     SYS$SHARE:VAXCRTL/SHARE
**
**  To build against TCPware on AXP:
**     $ CC/DEFINE=TCPWARE/STANDARD=VAXC UCX_SERVER
**     $ LINK UCX_SERVER
**
**  To build against UCX on VAX:
**     $ CC UCX_SERVER
**     $ LINK UCX_SERVER,SYS$LIBRARY:UCX$IPC/LIB,SYS$INPUT/OPTIONS
**     SYS$SHARE:VAXCRTL/SHARE
**
**  To build against UCX on AXP:
**     $ CC/STANDARD=VAXC UCX_SERVER
**     $ LINK UCX_SERVER
*/
```

Stream (TCP) Server

```
#include <in.h>
#include <string.h>
#include <starlet.h>
#include <descrip.h>
#include <iodef.h>
#include <ssdef.h>
#ifdef TCPWARE
#include "TCPWARE_INCLUDE:ucx$inetdef.h" /* Provided by TCPware install  */
#else                                    /* assume UCX */
#include <ucx$inetdef.h>                 /* Provided during UCX installation */
#endif
/*
**  Macros to check VMS success/failure status
*/
#define SUCCESS(status)             (status&1)
#define FAILURE(status)             (!(status&1))
/*
** Values for boolean variables
*/
#define FALSE 0
#define TRUE  1
```

Stream (TCP) Server

```
main() {
        int             status;                 /* For return status */
        short           listen_chan;            /* Listening device's channel */
        short           con_chan;               /* Accepted connection's channel */
        short           sck_parm[2];            /* Socket creation parameter */
        short           iosb[4];                /* I/O status block */
        char            buf[512];               /* buffer for received data */
        int             buflen = 512;           /* buffer length */
        unsigned char *ria;                     /* remote host's internet address */
        int             r_retlen;               /* length of remote host name structure */
        int     connected;                      /* boolean */
        /*
        **  Socket name structures for local and remote endpoints
        **  (filled in during IO$_ACCESS|IO$M_ACCEPT)
        */
        struct          SOCKADDRIN local_host, remote_host;
        /*
        **  UCX item list 2 descriptor used for local host name
        **  (filled in during IO$_ACCESS|IO$M_ACCEPT)
        */
        struct          IL2 {
                        unsigned int il2_length;
                        char *il2_address;
        } lhst_adrs = {sizeof local_host, &local_host};
```

Stream (TCP) Server

```
/*
** UCX item list 3 descriptor used for remote host name
** (filled in during IO$_ACCESS|IO$M_ACCEPT)
*/
struct        IL3 {
              unsigned int il3_length;
              char *il3_address;
              unsigned int il3_retlen;
} rhst_adrs = {sizeof remote_host, &remote_host, &r_retlen};
/*
**  String descriptor used during $ASSIGN BG0:
*/
struct        dsc$descriptor inet_dev =
              {10, DSC$K_CLASS_S, DSC$K_DTYPE_T, "UCX$DEVICE"};
/*
**  Socket options item list (used during IO$_SETMODE that
**  creates the listen socket to set the REUSEADDR option)
*/
int optval = 1;   /* option value = 1 means turn it on */
struct        { short len, param; int *ptr; }
              item_list[] = {{sizeof(optval), UCX$C_REUSEADDR, &optval}},
              options = {sizeof(item_list), UCX$C_SOCKOPT, item_list};
```

Stream (TCP) Server

```
/*
** Assign a channel for the listen socket.
*/
status = sys$assign(&inet_dev, &listen_chan , 0, 0);
if (FAILURE(status)) {
        printf("Failed to assign channel to UCX device.\n");
        exit(status);  }
/*
** Create the listen socket and set the REUSEADDR option.
*/
sck_parm[0] = UCX$C_TCP;                    /* TCP protocol */
sck_parm[1] = INET_PROTYP$C_STREAM ;        /* stream type of socket */
status = sys$qiow ( 3,                      /* Event flag */
                    listen_chan,            /* Channel number */
                    IO$_SETMODE,            /* I/O function */
                    iosb,                   /* I/O status block */
                    0, 0,
                    &sck_parm , 0,          /* P1 Socket creation parameter */
                    0, 0,
                    &options, 0);           /* P5 Socket option descriptor */
if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status)) {
        printf("Failed to create the device socket.\n");
        exit(status);  }
```

Stream (TCP) Server

```
/*
** Bind and listen to specified port number  (after REUSEADDR is set above).
*/
local_host.SIN$W_FAMILY = UCX$C_AF_INET;            /* INET (TCP/IP) family */
local_host.SIN$L_ADDR = UCX$C_INADDR_ANY;           /* any address */
local_host.SIN$W_PORT = htons( SERVER_PORT );       /* server's port # */

status = sys$qiow ( 3,                  /* Event flag */
                    listen_chan,        /* Channel number */
                    IO$_SETMODE,        /* I/O function */
                    iosb,       /* I/O status block */
                    0, 0,
                    0, 0,
                    &lhst_adrs,         /* P3 local socket name */
                    5,                  /* P4 Connection backlog */
                    0, 0);

if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status)) {
            printf("Failed to bind the device socket.\n");
            exit(status);  }
   else
            printf("Port is bound and listening\n");
```

Stream (TCP) Server

```
/*******************************************************/
/*   Main server loop                                  */
/*      1. accept new connection                       */
/*      2. read message from client                    */
/*      3. print message to SYS$OUTPUT                 */
/*      4. close connection                            */
/*      5. go to step 1                                */
/*******************************************************/

while ( 1 )   {   /* sit in this loop forever (break with CTRL-Y) */

 /*
 ** Assign channel for the new connection
 */

 status = sys$assign(&inet_dev, &con_chan,   0, 0);
 if (FAILURE(status)) {
   printf("Failed to assign channel for accept.\n");
   exit(status);
 }
```

Stream (TCP) Server

```
/*
** Accept a connection from a client.
*/
status = sys$qiow ( 3,                                  /* Event flag */
                    listen_chan,                        /* listen channel number */
                    IO$_ACCESS|IO$M_ACCEPT,             /* I/O function */
                    iosb,                               /* I/O status block */
                    0, 0,
                    0, 0,
                    &rhst_adrs,                         /* P3 Remote IP address*/
                    &con_chan,                          /* P4 Channel for new socket */
                    0, 0);
if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status)) {
  printf("Failed to accept a connection from a client.\n");
  exit(status);    }
/*
** Print out the connecting client's internet address & port
*/
ria = &remote_host.SIN$L_ADDR;
printf("Connection from host: %d.%d.%d.%d, port: %d\n",
            ria[0], ria[1], ria[2], ria[3],
            htons(remote_host.SIN$W_PORT));
```

Stream (TCP) Server

```
/*
**  Read data from client (and write it to SYS$OUTPUT)
**  until client closes it's end of the connection
**
**  Note that this client/server combination implements
**  a very simple message protocol - the client simply
**  sends a text string of any size and then closes
**  the connection.   The server simply keeps reading
**  (and printing the received text to SYS$OUTPUT)
**  until the connection is closed by the client.
**
**  Recall that the TCP byte-stream does NOT guarantee
**  a one-to-one correspondence of client writes to server
**  reads, so you may see several reads for the client's
**  single write.    (With the short message used here,
**  you probably won't, but you might.)
*/

connected = TRUE;
while ( connected ) {
```

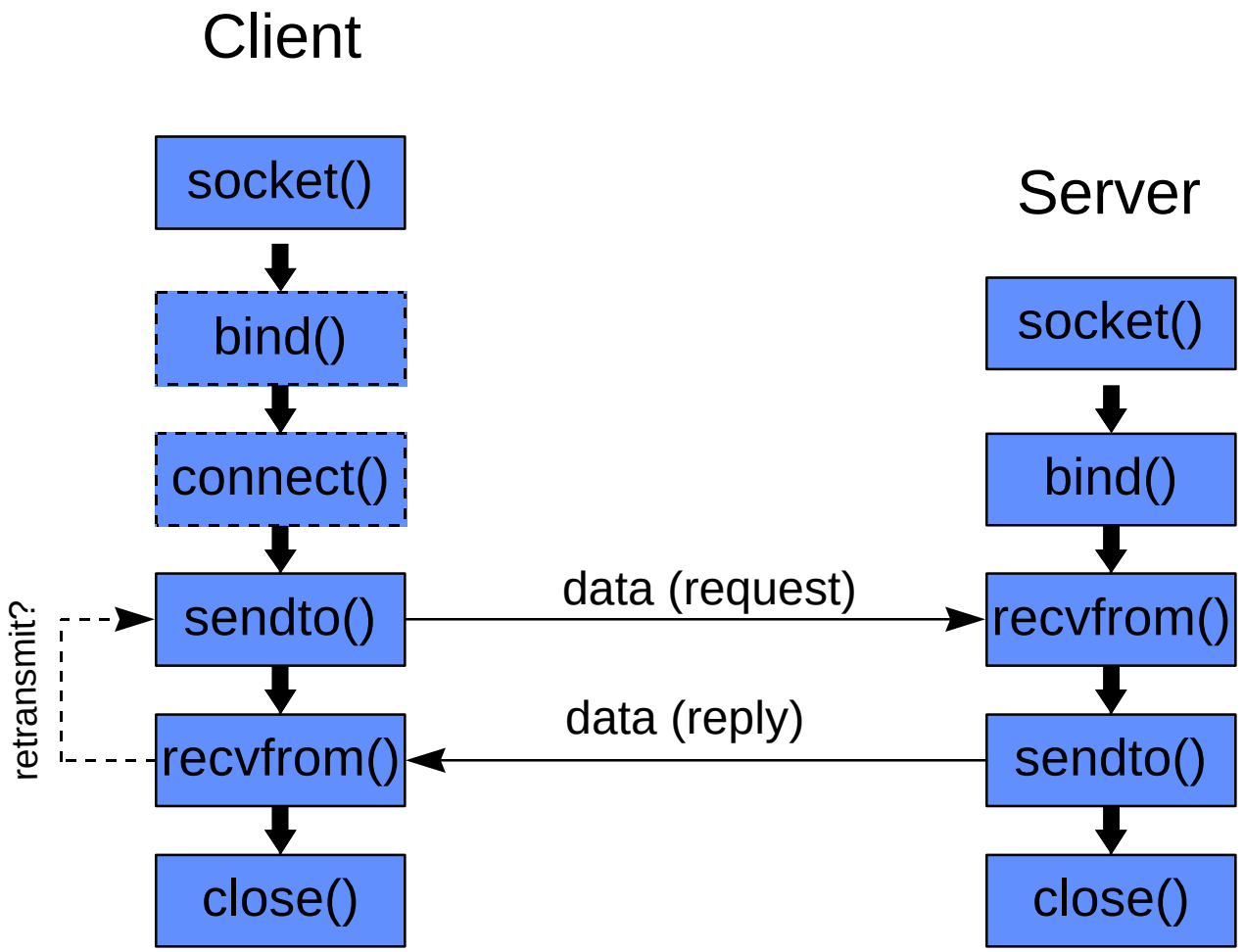
Stream (TCP) Server

```
/*
** Read I/O buffer
*/
status = sys$qiow( 3,                          /* Event flag */
                   con_chan,                   /* Channel number */
                   IO$_READVBLK,               /* I/O function */
                   iosb,                       /* I/O status block */
                   0, 0,                       /* no AST, no AST parm */
                   buf,                        /* P1 buffer */
                   buflen,                     /* P2 buffer length */
                   0, 0, 0, 0);
if (SUCCESS(status)) status = iosb[0];
if (FAILURE(status))  {
  if (status != SS$_LINKDISCON  ) {   /* SS$_LINKDISCON indicates normal termination */
            printf("Failed to read from socket.\n");
            printf("iosb[0] = 0x%x\n", iosb[0] );
            printf("iosb[1] = 0x%x\n", iosb[1] );        }
  connected = FALSE;        /* terminate while loop */          }
else {
  /* All is well, print message */
  buf[iosb[1]] = '\0';  /* make sure message is null-terminated */
  printf ("Received text: %s\n", buf);                }
}  /* end of connected while loop */
```

Stream (TCP) Server

```
/*              ** Shut down the socket  that was connected to the client
                ** and deassign the channel used with that socket.          */
                status = sys$qiow ( 3,
                                    con_chan,
                                    IO$_DEACCESS|IO$M_SHUTDOWN,
                                    iosb,
                                    0, 0,
                                    0, 0, 0,
                                    UCX$C_DSC_ALL,  /* P4 Discard all packets */
                                    0, 0 );
                 if (SUCCESS(status)) status = iosb[0];
                 if (FAILURE(status))
                   printf("Failed to shut down the client socket.\n");
                 status = sys$dassgn(con_chan);
                 if (FAILURE(status))
                   printf("Failed to deassign the client socket.\n");
                 printf("FYI: Port is still bound and listening...\n");
               }   /* end of main while loop */
               /*
               ** Note:  We don't explicitly shutdown the listen socket
               ** and deassign the listen channel.   This will be done
               ** during image run-down whey you CTRL-Y EXIT the program
               */
}
```

Datagram (UDP) Client and Server



Datagram (UDP) Client

1. Create network device ($ASSIGN BG0:)
2. [Optional] “Connect” to server ($QIO IO$_ACCESS)
   - UDP is connectionless
   - Simply specifies communication endpoint for send
3. Send a request datagram ($QIO IO$_WRITEVBLK)
4. Start a timer
5. Receive reply datagram ($QIO IO$_READVBLK)
   - If timer expires during receive, repeat at step 3
6. If more to request, go to step 3
7. Close and deallocate network device ($QIO IO$_DEACCESS followed by $DASSGN)

Datagram (UDP) Server

1. Create network device ($ASSIGN BG0:)
2. Bind network device to well-known port ($QIO IO$_SETMODE)
3. Wait for request ($QIO IO$_READVBLK)
4. Send reply ($QIO IO$_WRITEVBLK)
5. Go to step 3

More Sophisticated Servers

- Multithreaded Servers
  - Use ASTs (or Event Flags)
  - One thread for each connection
  - Must maintain “context block” for each connection

More Sophisticated Servers

- Server started by INET Master Server (inetd)
  - On UCX this is a “UCX auxiliary server”
  - One server process per connection
    - Easily supports multiple connections
    - Costs a VMS process
  - Add server information to inetd database
  - Server automatically started when connection request comes in to host
    - UCX master server invokes .COM or .EXE
    - Server’s input, output, error files assigned to BG device
    - Maser server does “socket”,”bind”,”listen”,”accept”

References

- Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications (BSD Socket Version), Douglas E. Comer and Richard L. Stevens, Prentice-Hall, Inc. 1993. ISBN #0-13-474222-2
- Internetworking with TCP/IP, Volume I, Principles, Protocols, and Architecture, @3rd ed., Douglas E. Comer, Prentice-Hall, Inc. 1995. ISBN #0-13-468505-9
- DEC TCP/IP Services for OpenVMS system Services and C Socket Programming manual
  - Documentation for BG and socket routines
- DEC OpenVMS System Services Reference manual
  - SYS$ASSIGN, SYS$QIO(W), SYS$DASSGN
- Programming manuals for 3rd party TCP/IP stacks
  - When using UCX emulation, usually only need compile and link information; otherwise use DEC TCP/IP Services manuals

TCP/IP Programming

Slides and Source Code available via anonymous FTP:

Host:: ftp.process.com
Directory: [pub.decus]
Slides: DECUS_F96_PROG.PS
Examples: DECUS_F96_PROG_EXAMPLES.TXT

Host: ftp.opus1.com
Slides: DECUS_F96_PROG.PS
Examples: DECUS_F96_PROG_EXAMPLES.TXT