System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
Header file:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define         SEMKEY  (key_t)5763
#define         MEMKEY  (key_t)5763
#define         NUMFLAVORS       4
#define         NUMSLOTS        50
#define         NUMSEMIDS        3
#define         PROD             0
#define         CONSUMER         1
#define         OUTPTR           2

struct  donut_ring{
        int     flavor[NUMFLAVORS][NUMSLOTS];
        int     outptr[NUMFLAVORS];
};

void    handler();
void    p();
void    v();
void    semsetall();
```

System V IPC (Cont'd)

```
EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The utility file:

#include "donuts.h"

void    p(int semidgroup, int donut_type)
{
        struct sembuf semopbuf;

        semopbuf.sem_num=donut_type;
        semopbuf.sem_op=(-1);
        semopbuf.sem_flg=0;

        if(semop(semidgroup, &semopbuf,1) == -1){
                perror("p operation failed: ");
                exit(2);
        }
}

void    v(int semidgroup, int donut_type)
{
        struct sembuf semopbuf;

        semopbuf.sem_num=donut_type;
        semopbuf.sem_op=(+1);
        semopbuf.sem_flg=0;

        if(semop(semidgroup, &semopbuf,1) == -1){
                perror("p operation failed: ");
                exit(2);
        }
}
```

System V IPC (Cont'd)

```
EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The utility file    cont'd:

void    semsetall(int semgroup, int number_in_group,
                  int set_all_value){
        int     i,j,k;

        for(i=0; i<number_in_group; i++){
          if(semctl(semgroup, i, SETVAL, set_all_value) == -1){
                perror("semset failed");
                handler(-3);
                exit(3);
          }
        }
}

void    handler(int sig){
        int     i,j,k;

        printf("In signal handler with signal # %d\n",sig);
        if(shmctl(shmid, IPC_RMID) == -1){
                perror("handler failed shm RMID: ");
                exit(4);
        }
        for(i=0; i<NUMSEMIDS; i++){
          if(semctl(semid[i], 0, IPC_RMID) == -1){
                perror("handler failed sem RMID: ");
                exit(4);
          }
        }
}
```

System V IPC (Cont'd)

```
EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The producer:

#include "donuts.h"

int main(int argc, char * argv[]){
   int    cntr, *addr1, *addr2;
   struct sigaction   new;
   sigset_t   mask_sigs;
   int sigs[] = {SIGHUP,SIGINT,SIGQUIT, SIGPIPE
                 SIGTERM, SIGBUS, SIGSEGV, SIGFPE};
   int    in_ptr[NUMFLAVORS];
   int    serial[NUMFLAVORS];
   int    i,j,k, nsigs;
   struct donut_ring *shared_ring;


   nsigs = sizeof(sigs)/sizeof(int)
   sigemptyset(&mask_sigs);
   for(i=0; i< nsigs; i++)
       sigaddset(&mask_sigs, sigs[i]);        .
   for(i=0; i< nsigs; i++){
      new.sa_handler = sig_handler;
      new.sa_mask = mask_sigs;
      new.sa_flags = SA_RESTART;
      if(sigaction(sigs[i], &new, NULL) == -1){
         perror("can't set signals: ");
         exit(1);
      }
   }
```

System V IPC (Cont'd)

```
EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The producer    cont'd:

        for(i=0; i<NUMFLAVORS; i++){
                in_ptr[i]=0;
                serial[i]=0;
        }

        if((shmid=shmget(MEMKEY, sizeof(struct donut_ring),
                        IPC_CREAT | 0666)) == -1){
                perror("shared get failed: ");
                exit(1);
        }
        if((shared_ring=(struct donut_ring *)shmat(shmid,0,0))== -1){
                perror("shared attach failed: ");
                handler(-1);
        }

        for(i=0; i<NUMSEMIDS; i++)
                if((semid[i]=semget(SEMKEY+i, NUMFLAVORS,
                        IPC_CREAT | 0666)) == -1){
                perror("semaphore allocation failed: ");
                handler(-1);
        }

        srandom((time((time_t *)0) + getpid());
        semsetall(semid[PROD], NUMFLAVORS, NUMSLOTS);
        semsetall(semid[CONSUMER], NUMFLAVORS, 0);
        semsetall(semid[OUTPTR], NUMFLAVORS, 1);
```

System V IPC (Cont'd)

```
EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The producer    cont'd:


        while(1){
          j=random() & 3;
          p(semid[PROD], j);
          shared_ring->flavor[j][in_ptr[j]]=serial[j];
          in_ptr[j] = (in_ptr[j]+1) % NUMSLOTS;
          serial[j]++;
          printf("prod type %d serial %d\n",j,serial[j]-1);
          v(semid[CONSUMER], j);
        }
}
```

System V IPC (Cont'd)

```
EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The consumer:

#include "donuts.h"

int main(int argc, char * argv[]){
        int     i,j,k,donut;
        struct donut_ring *shared_ring;




        if((shmid=shmget(MEMKEY, sizeof(struct donut_ring),
                                0)) == -1){
                perror("shared get failed: ");
                exit(1);
        }

        if((shared_ring=(struct donut_ring *)shmat(shmid,0,0))== -1){
                perror("shared attach failed: ");
                exit(1);
        }

        for(i=0; i<NUMSEMIDS; i++){
                if((semid[i]=semget(SEMKEY+i, NUMFLAVORS,
                        0)) == -1){
                perror("semaphore allocation failed: ");
                exit(1);
                }
        }
```

System V IPC (Cont'd)

EXAMPLE:
Single producer, multiple consumer, multiple flavor problem
The consumer   cont'd:

```
        srandom(time((time_t)0) + getpid());

        for(i=0; i<10; i++){
           for(k=0; k<12; k++){
                j=random() & 3;
                printf("in consumer with rand %d\n",j);
                p(semid[CONSUMER], j);
                p(semid[OUTPTR], j);
                donut=shared_ring->flavor[j][shared_ring->outptr[j]];
                shared_ring->outptr[j] =
                        (shared_ring->outptr[j]+1) % NUMSLOTS;
                v(semid[PROD], j);
                v(semid[OUTPTR], j);
           }
        printf("dozen number %d completed \n\n",i);
        }
}
```

Interhost IPC With Sockets

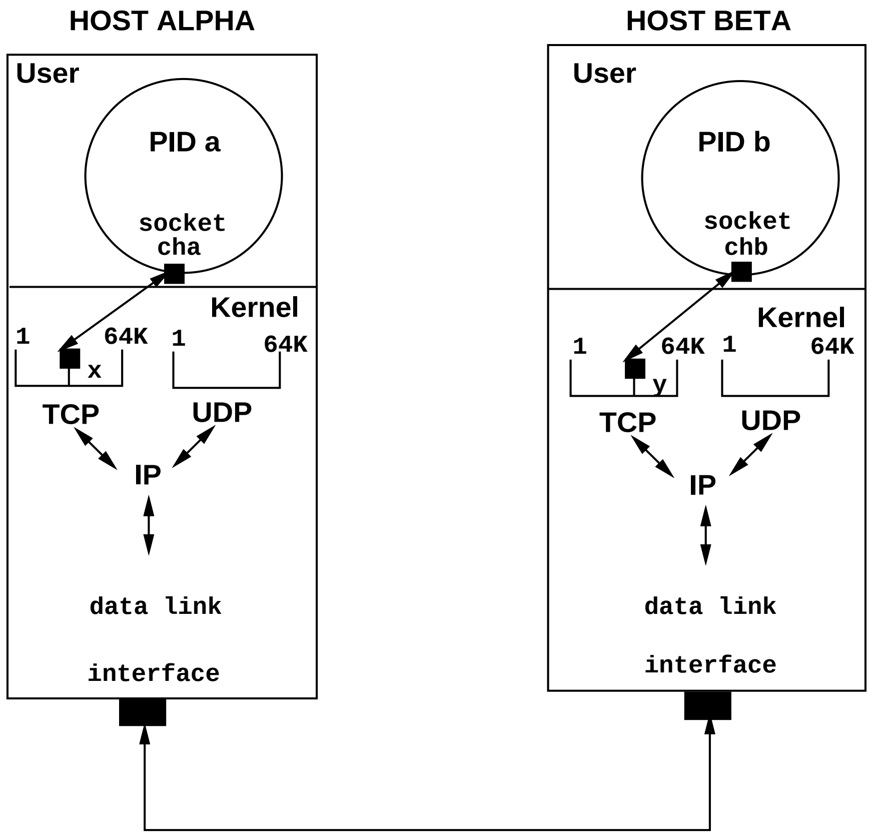
The UNIX socket facility provides a full duplex communication abstraction for process to process communication within and across hosts:

- Sockets support both virtual circuit connection based service and unconnected datagram service
- While sockets provide an API to a network environment and have been implemented over several different lower layer protocols, in the UNIX world they are most commonly found on top of the transport protocols TCP and UDP and the network protocol of the internet, IP
- The most common applications built over sockets include packages such as ftp, telnet, smtp, and the remote commands from BSD. These all use a client- server model with a connection based structure

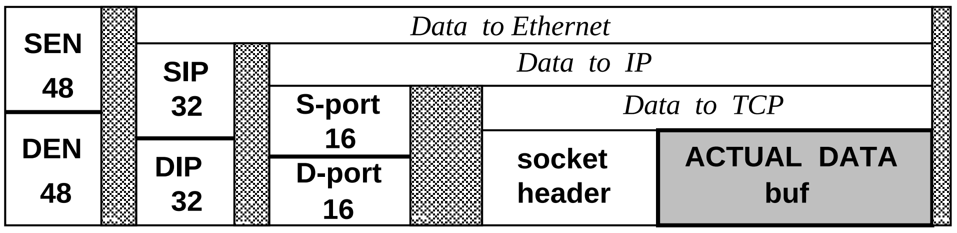
The Network Landscape

Application	ftp mail rcp rlogin telnet	NFS	NIS	tftp talk named time rwho
Presentation		XDR		
Session		RPC		
Transport	TCP		UDP	
Network	IP			
Data Link	Ethernet	X.25		....
Physical	Ethernet	Sync		....

Socket Interhost IPC with TCP/IP



write(cha, buf, sizeof(buf));



PID a writes to PID b over a TCP connection using PORT **x** on HOST ALPHA and PORT **y** on HOST BETA. After the connection is established, simple *read()* and *write()* calls can be used with channels **cha** and **chb**

Socket Interhost IPC (Cont'd)

```
SYNOPSIS
Create an endpoint for communication
       #include <sys/socket.h>
       #include <netinet/in.h>


       int  socket (af, type, protocol)
       int  af;
       int  type;
       int  protocol;

   where:
       af        Address family (domain)
       type      Type of service desired
       protocol  Optional protocol id (usually 0)

                 possible family values
       AF_UNIX   local host domain
       AF_INET   internet domain

                 possible service type values
       SOCK_STREAM   virtual circuit socket
       SOCK_DGRAM    datagram socket
       SOCK_RAW      access to internal network

   returns:      channel number on success, or -1
```

Socket Interhost IPC (Cont'd)

SYNOPSIS

Create a pair of connected UNIX doamin sockets

```
       #include <sys/socket.h>

       int  socketpair (af, type, protocol, sv)
       int  d;
       int  type;
       int  protocol;
       int  sv[2];
```

```
   where:
       af        Address family (domain)
                 of the socket, AF_UNIX
       type      Type of service, SOCK_STREAM/SOCK_DGRAM
       protocol  Protocol of interest, 0 for default
       sv        Buffer in which to return descriptors
```

DESCRIPTION

The socketpair call is used in the UNIX domain only to create a connected pair of sockets in the style of the pipe() call. A creator would typically fork() a child (or children) and use the channel inheritance of the offspring for full duplex IPC. Unlike a pipe, both channel descriptors are open for reading and writing with no possible crosstalk.

```
   returns:      0 on success, or -1
```

Socket Interhost IPC (Cont'd)

```
SYNOPSIS
Initiate a connection on a socket
       #include <sys/socket.h>
       #include <netinet/in.h>
       #include <sys/un.h>

       int    connect (s, name, namelen)
       int    s;
       const struct sockaddr * name;
       int    namelen;
   where:
     s        The file descriptor of a socket to connect
     name     Name of peer or listening socket through
              which the connection will be made
     namelen  Length of name (bytes)

                 possible name address formats
struct sockaddr_un{
  short sun_family;       /* AF_UNIX   */
  char sun_path[109];    /* UNIX file path */
};
struct sockaddr_in{
          short   sin_family;         /* AF_INET     */
          u_short sin_port;       /* port number */
          struct  in_addr sin_addr;  /* inet addr   */
          char    sin_zero[8];
}
   returns:   0 on success, or -1
```

Sockaddr Format Types

struct sockaddr_un

byte size : 2	**AF_UNIX**
byte size : up to 109 **the size is the number of characters in the path name**	**/usr/users/bill/sock**

lenght argument:
sizeof (short) + strlen (pathname)

struct sockaddr_in

byte size : 2	**AF_INET**
byte size : 2	**PORT # (1 - 64k)**
byte size : 4	**IP ADDRESS**
byte size : 8	**PADDING**

lenght argument:
sizeof (struct sockaddr_in)

Socket Interhost IPC (Cont'd)

```
SYNOPSIS
Bind a name to a socket
       #include <sys/socket.h>
       #include <netinet/in.h>
       #include <sys/un.h>

       int    bind (s, name, namelen)
       int    s;
       const struct sockaddr * name;
       int    namelen;
   where:
       s          Socket to bind
       name       Name to bind to socket
       namelen    Length of name (bytes)
DESCRIPTION
       This call allows a process to establish a
       link to an underlying TCP or UDP port (in
       the case of an internet socket) or a link
       to a path name in the file system (in the
       case of a UNIX domain socket).  In general,
       a server must use this call to give herself
       an address, whereas a client is not required
       to use this call (but may if she so chooses),
       but is automatically allocated a source port
       at the time a connect() call is made.  It can
       be seen that the call arguments are the same
       as those for the connect call.
   returns:      0 on success, or -1
```

Socket Interhost IPC (Cont'd)

```
SYNOPSIS
Listen for connections on a socket

       int  listen (s, backlog)
       int  s;
       int  backlog;

   where:
       s         File descriptor of socket to listen on.
       backlog   Maximum number of waiting connections.

   returns:      0 on success, or -1

SYNOPSIS
       #include <sys/socket.h>
       #include <netinet/in.h>
       #include <sys/un.h>

       int    accept (s, addr, addrlen)
       int    s;
       struct sockaddr * addr;
       int *  addrlen;

   where:
     s        Socket channel listening for connection requests
     addr     Structure to receive the address of connected peer
     addrlen  On input contains the number of bytes available
              for the peer address; updated to indicate the
              number of bytes returned

   returns:   Connected new channel number, or -1
```

Unix Domain Socket IPC

EXAMPLE:
Client/Server example using UDS
The client code

```
/*   client process for UNIX domain socket example   */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

int main(int argc, char *argv[])
{
  int  sock, nread;
  char buf[256];
  struct sockaddr_un nameformat;

  sock = socket(AF_UNIX,SOCK_STREAM,0);
  nameformat.sun_family = AF_UNIX;
  strcpy(nameformat.sun_path, "/usr/users/bill/sock");

  /*   connect to name in file system    */
  /*   notice size of address value      */

  if ( connect(sock, &nameformat,
          strlen(nameformat.sun_path)
          + sizeof(nameformat.sun_family) ) == -1)
     exit(1);
  write(sock, "hello from client", 17);
  nread = read(sock, buf, sizeof(buf));
  *(buf + nread) = '\0';
  printf("server's message is:  %s\n", buf);
  close(sock);
  return(1);
}
```

Unix Domain Socket IPC Cont'd

```
EXAMPLE:
Client/Server example using UDS
The server code

/*   server process for UNIX domain socket example   */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

int main(int argc, char *argv[])
{
  int  sock, newsock, nread;
  char buf[256];
  struct sockaddr_un nameformat;

  sock = socket(AF_UNIX, SOCK_STREAM, 0);
  nameformat.sun_family = AF_UNIX;
  strcpy(nameformat.sun_path, "/usr/users/bill/sock");
  unlink(nameformat.sun_path);

/*   bind to name in file system       */
/*   notice size of address value      */

  if (bind(sock, &nameformat,
           strlen(nameformat.sun_path)
           + sizeof(nameformat.sun_family)) == -1)
     exit(1);
```

Unix Domain Socket IPC Cont'd

```
EXAMPLE:
Client/Server example using UDS    cont'd
The server code

  listen(sock,3);

  for(;;)                      /* forever */
  {
     if( (newsock = accept(sock, NULL, NULL)) == -1){
         perror("accept failed: ");
         exit(2);
     }

     if (fork() == 0){        /* child */
        close(sock);
        nread = read(newsock, buf, sizeof(buf));
        *(buf + nread) = '\0';
        printf("client's  message is:  %s\n", buf);
write(newsock, "Server is closing connection", 28);
        close(newsock);
        exit(0);
      }

  close(newsock);
  }
}
```

Example With Internet Sockets

```
EXAMPLE:
Client/Server example using internet sockets
A "one-time" FTP client

/*          client.c  a minimal FTP client          */
/*   this version works on all BSD based  systems   */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <strings.h>

#define MSG_BODY  248
#define MSG_SIZE  (MSG_BODY+8)
#define PORT      27439
#define SEND      1
#define RECV      2
#define DATA      100
#define END_DATA  101

typedef  struct{
        int  mtype;
        int  msize;
        char mbody[MSG_BODY];
} MSG;

typedef union{
        MSG m;
        char buf[MSG_SIZE];
} MBUF;
```

Example With Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
A "one-time" FTP client

main(int argc, char *argv[])
{
        MSG     msg;
        MBUF    raw;
        int     inet_sock, local_file;
        int     type_val, size_val, read_val, local_size;
        int     i,j,k;
        char    *buffer_ptr, *token_ptr, *last_token_ptr;
        union type_size;
        struct sockaddr_in inet_telnum;
        struct hostent *heptr, *gethostbyname();

        if((inet_sock=socket(AF_INET, SOCK_STREAM, 0)) == -1){
          perror("inet_sock allocation failed: ");
          exit(1);
        }

        if((heptr = gethostbyname( argv[1] )) == NULL){
          perror("gethostbyname failed: ");
          exit(1);
        }

        bcopy(heptr->h_addr, &inet_telnum.sin_addr, heptr->h_length);
        inet_telnum.sin_family = AF_INET;
        inet_telnum.sin_port = htons( (u_short)PORT );

        if(connect(inet_sock, &inet_telnum, sizeof(struct sockaddr_in)) == -1){
          perror("inet_sock connect failed: ");
          exit(2);
        }
```

Example With Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
A "one-time" FTP client

        msg.mtype = htonl(RECV);
        strcpy(msg.mbody, argv[2]);
        local_size = strlen(argv[2]) + 1;
        msg.msize = htonl(local_size);

        token_ptr = strtok(argv[2], "/");

        do{
          if((last_token_ptr = token_ptr) == NULL){
              printf("\n    *** Illegal path name, terminating\n\n");
              exit(2);
          }
        } while(token_ptr = strtok(NULL, "/"));

        if((local_file=open(last_token_ptr, O_RDWR | O_CREAT | O_TRUNC, 0666))
                                                                    == -1){
          perror("local_file open failed: ");
          exit(3);
        }

        if(write(inet_sock, &msg, local_size + (2*sizeof(int))) == -1){
          perror("inet_sock write failed: ");

        if(write(inet_sock, &msg, local_size + (2*sizeof(int))) == -1){
          perror("inet_sock write failed: ");
          exit(3);
        }
```

Example With Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
A "one-time" FTP client

        while(1){
          for(i=0; i<(2*sizeof(int)); i++){
             if(read(inet_sock, raw.buf+i, 1) != 1){
                perror("read type_size failed: ");
                exit(3);
             }
          }
          type_val = ntohl(raw.m.mtype);
          size_val = ntohl(raw.m.msize);
          read_val = size_val;
          buffer_ptr   = raw.buf;
          switch(type_val){
            case DATA:
                while ((j = read(inet_sock, buffer_ptr, read_val)) != read_val){
                      switch(j){
                        default:  read_val -= j;
                                  buffer_ptr += j;
                                  break;
                        case -1:  perror("inet_sock read failed: ");
                                  exit(3);
                        case  0:  printf("unexpected EOF on inet_sock\n");
                                  exit(4);
                      }
                }
                if(write(local_file, raw.buf, size_val) == -1){
                  perror("local write failed: ");
                  exit(3);
                }
                break;
            case END_DATA:
                printf("transfer of %s completed successfully, goodbye\n",argv[2]);
                exit(0);
            default:
                printf("unknown message type %d size of %d\n",type_val,size_val);
                printf("this is an unrecoverable error, goodbye\n");
                exit(5);
          }
        }
}
```

Example WithInterhost IPC with Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
The simple FTP demon server

/*    server.c  a minimal ftp      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>

#define MSG_BODY  248
#define MSG_SIZE  (MSG_BODY+8)
#define PORT      27439
#define SEND      1
#define RECV      2
#define DATA      100
#define END_DATA  101

typedef  struct{
        int  mtype;
        int  msize;
        char mbody[MSG_BODY];
} MSG;

typedef union{
        MSG m;
        char buf[MSG_SIZE];
} MBUF;

extern int errno;

void child_handler(sig){
  wait(NULL);
}
```

Example With Interhost IPC with Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
The simple FTP demon server

main(int argc, char *argv[])
{
        MSG     msg;
        MBUF    raw;
        int     inet_sock, new_sock, local_file;
        int     type_val, size_val, read_val;
        int     i,j,k,fromlen;
        char    *buffer_ptr;
        union type_size;
        struct sockaddr_in inet_telnum;
        struct hostent *heptr, *gethostbyname();
        int     wild_card = INADDR_ANY;
        struct sigaction sigstrc;

        sigstrc.sa_handler = child_handler;
        sigstrc.sa_mask = 0;
        sigstrc.sa_flags = SA_NOCLDSTOP;
        sigaction(SIGCHLD, &sigstrc, NULL);

        if((inet_sock=socket(AF_INET, SOCK_STREAM, 0)) == -1){
          perror("inet_sock allocation failed: ");
          exit(1);
        }
        bcopy(&wild_card, &inet_telnum.sin_addr, sizeof(int));
        inet_telnum.sin_family = AF_INET;
        inet_telnum.sin_port = htons( (u_short)PORT );
        if(bind(inet_sock, &inet_telnum, sizeof(struct sockaddr_in)) == -1){
          perror("inet_sock bind failed: ");
          exit(2);
        }
        listen(inet_sock, 5);
```

Example WithInterhost IPC with Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
The simple FTP demon server
```

```
        for(;;){                      /* forever */

         while((new_sock = accept(inet_sock, &inet_telnum, &fromlen)) == -1 &&
                                                           errno == EINTR);
         if(new_sock == -1){
                perror("accept failed: ");
                exit(2);
         }
         switch(fork()){
           default:  close(new_sock);
             break;

           case -1:  perror("fork failed: ");
             exit(1);

           case  0:  close(inet_sock);
             while(1){
               for(i=0; i<(2*sizeof(int)); i++){
                 if(read(new_sock, raw.buf+i, 1) != 1){
                    perror("read type_size failed: ");
                    exit(3);
                 }
               }

               type_val = ntohl(raw.m.mtype);
               size_val = ntohl(raw.m.msize);
               read_val = size_val;
               buffer_ptr   = raw.buf;
               read_val = size_val;
               buffer_ptr   = raw.buf;
```

Example WithInterhost IPC with Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
The simple FTP demon server

               switch(type_val){
                case RECV:
                  while ((j=read(new_sock, buffer_ptr, read_val)) != read_val){
                      switch(j){
                        default:  read_val -= j;
                                  buffer_ptr += j;
                                  break;
                        case -1:  perror("new_sock read failed: ");
                                  exit(3);

                        case  0:  printf("unexpected EOF on new_sock\n");
                                  exit(4);
                      }
                   }
                   if((local_file = open(raw.buf, O_RDONLY, 0)) == -1){
                      perror("local open failed: ");
                      exit(3);
                   }
                   while(1){
                    switch(j=read(local_file, msg.mbody, MSG_BODY)){
                        default:  msg.msize = htonl(j);
                                  msg.mtype = htonl(DATA);
                                  if(write(new_sock, &msg, (2*sizeof(int)+j))
                                                                      == -1){
                                    perror("new_sock write failed: ");
                                    exit(3);
                                  }
                                  break;
                        case  0:  msg.msize = htonl(0);
                                  msg.mtype = htonl(END_DATA);
                                  if(write(new_sock, &msg, (2*sizeof(int)))
                                                                      == -1){
                                    perror("new_sock write failed: ");
                                    exit(3);
                                  }
                                  printf("server done sending file %s\n", raw.buf);
                                  exit(0);
```

Example WithInterhost IPC with Internet Sockets Cont'd

```
EXAMPLE:
Client/Server example using internet sockets    cont'd
The simple FTP demon server

                        case -1:  perror("local_file read failed: ");
                                  exit(3);
                     }
                   }
               case TEST:
                while ((j=read(new_sock, buffer_ptr, read_val)) != read_val){
                      switch(j){
                        default:  read_val -= j;
                                  buffer_ptr += j;
                                  break;
                        case -1:  perror("new_sock read failed: ");
                                  exit(3);
                        case  0:  printf("unexpected EOF on new_sock\n");
                                  exit(4);
                      }
                   }

                printf("server child received test packet number %d as #%d\n",
                                        *((int *)raw.buf),      test_id++);
                sleep(sleep_interval);
                break;

               case END_TEST:
                printf("server child finished test run, goodbye\n\n");
                exit(0);

               default:
                printf("unknown message type %d claims size of %d\n",
                                        type_val,         size_val);
                printf("this is an unrecoverable error, goodbye\n");
                exit(5);
              }
        }
      }
   }
}
```