CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

Sockets

Today's Topics

- New Way of Communicating Between Processes
- Sockets

Sockets and the Internet (IPv4)

- `AF_INET` communication domain
- SOCK_DGRAM -Provides *datagram* semantics, only promises best-effort delivery! **UDP/IP**
- SOCK_STREAM - Provides a FIFO type point-to-point communication system! **TCP/IP**
- We need a way to associate names with sockets to be able to do network I/O through a socket file descriptor

Sockets and the Internet (IPv4)

- The header file <netinet/in.h> defines a 32-bit for an Internet host.
- This actually identifies a specific network interface on a specific system on the Internet.
- It's represented by a 32 bit unsigned number

```
struct in_addr {
    __u32 s_addr;
}
```

- The addresses are usually represented by dotted decimal notation.

Representing the Address in C

- In header file <netinet/in.h>

```
#define __SOCK_SIZE__ 16 /* sizeof(struct sockaddr) */

struct sockaddr_in {
  short int sin_family; /* Address family */
  unsigned short int sin_port; /* Port number */
  struct in_addr sin_addr; /* Internet address */
  /* Pad to size of 'struct sockaddr'. */
  unsigned char __pad[__SOCK_SIZE__ - sizeof(short int) -
  sizeof(unsigned short int) - sizeof(struct in_addr)];
};
```

- Declare and/or allocate instance of struct sockaddr_in whenever you need to specify a full address on the Internet
- A port is an Internet communication endpoint associated with an application. *(host,port)* defines an Internet address.
- Ports in range [0,1023] reserved for root; others available to ordinary users. (See RFC 1700)

Usual Ports for Services

- FTP uses 20 and 21
- SSH uses 22
- Telnet uses 23
- HTTP uses 80, commonly
- HTTPS uses 443
- Check /etc/services to see what "well-known" ports are

Translating Host Names into IP Address(es)

- Library function to map symbolic host name into IP address(es):

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);

void herror(const char *s);
```

- The hostent data structure:

```
struct hostent {
    char  *h_name;              /* official name of host */
    char **h_aliases;           /* alias list */
    int    h_addrtype;          /* host address type */
    int    h_length;            /* length of address */
    char **h_addr_list;         /* list of addresses */
}
#define h_addr h_addr_list[0] /* for backward compatibility */
```

Translating Host Names into IP Address(es)

- If we have a dotted decimal string and we want to convert it into an address we can use, the above function is useful.
- Also see man inet for more functions!

getaddrs.c Example

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
main(argc,argv)
int argc; char **argv;
{
   struct hostent *entry; char **next;
   struct in_addr address, **addrptr;
   entry = gethostbyname(argv[1]);
   if (!entry) { herror("lookup error"); exit(1); }
   printf("Official name -> %s\n", entry->h_name);
   if (entry->h_aliases[0]) {
      printf("Aliases ->\n");
      for (next = entry->h_aliases; *next; next++)
         printf(" %s\n", *next);
   }
   printf("IP Addresses:\n");
   for (addrptr=(struct in_addr **) entry->h_addr_list;
     *addrptr; addrptr++)
      printf(" %s\n", inet_ntoa(**addrptr));
}
```

Can Also Get Symbolic Name from IP Address

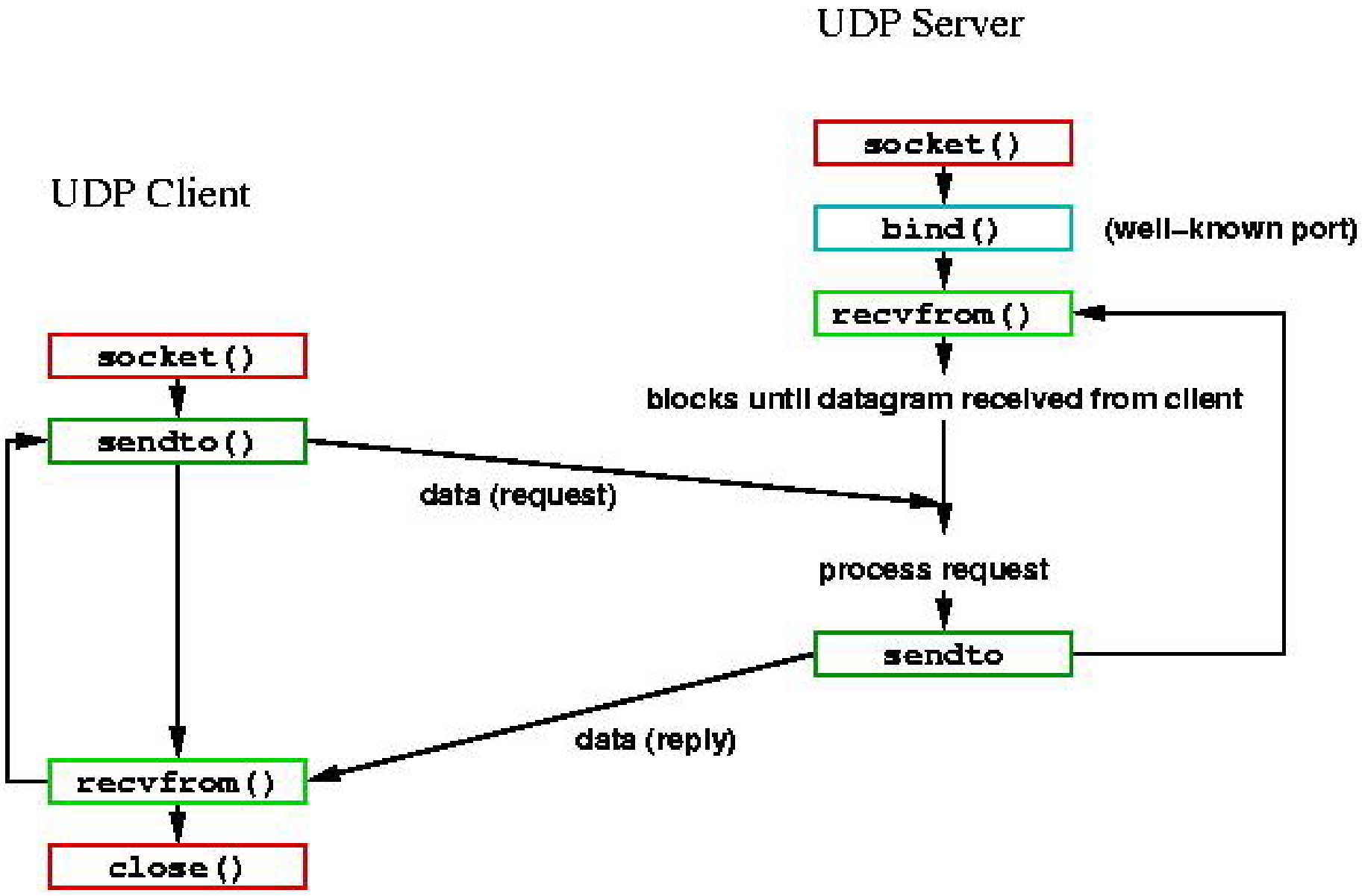
- There's also an inverse function (we know IP address, want symbolic name)

```
#include <netdb.h>

struct hostent *gethostbyaddr(const char *addr,
                              int len, int type);
```

gethost.c Example

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
main(argc,argv)
int argc; char **argv;
{
   struct hostent *entry; char **next;
   struct in_addr address, **addrptr;
   inet_aton(argv[1], &address);
   entry = gethostbyaddr((char *)&address, sizeof(address),
     AF_INET);
   if (!entry) { herror("lookup error"); exit(1); }
   printf("Official name -> %s\n", entry->h_name);
   if (entry->h_aliases[0]) {
      printf("Aliases ->\n");
      for (next = entry->h_aliases; *next; next++)
         printf(" %s\n", *next);
   }
   printf("IP Addresses:\n");
   for (addrptr=(struct in_addr **) entry->h_addr_list;
     *addrptr; addrptr++)
      printf(" %s\n", inet_ntoa(**addrptr));
}
```


UDP Server
UDP Client
socket()
bind()
(well-known port)
recvfrom()
socket()
blocks until datagram received from client
sendto()
data (request)
process request
sendto
data (reply)
recvfrom()
close()

recv_upd.c: UDP/IP Server I

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
void printsin(s_in, s1, s2)
struct sockaddr_in *s_in; char *s1, *s2;
{
   printf ("Program: %s\n%s ", s1, s2);
   printf ("(%d,%d)\n", s_in->sin_addr.s_addr, s_in->sin_port);
}

main()
{
   short p_len;
   int socket_fd, cc, h_len, fsize, namelen;
   struct sockaddr_in s_in, from;
   struct { char head; u_long body; char tail;} msg;

   socket_fd = socket (AF_INET, SOCK_DGRAM, 0);
   /* You must do this just in case */
```

recv_upd.c: UDP/IP Server II

```
    bzero((char *) &s_in, sizeof(s_in));

    s_in.sin_family = (short)AF_INET;
    s_in.sin_addr.s_addr = htonl(INADDR_ANY); /* WILDCARD */
    s_in.sin_port = htons((u_short)0x3333);
    printsin( &s_in, "RECV_UDP", "Local socket is:");
    fflush(stdout);
    bind(socket_fd, (struct sockaddr *)&s_in, sizeof(s_in));
    for(;;) {
      fsize = sizeof(from);
      cc = recvfrom(socket_fd, &msg, sizeof(msg), 0,
              (struct sockaddr *)&from,&fsize);
      printsin( &from, "recv_udp: ", "Packet from:");
      printf("Got data ::%c%ld%c\n",msg.head,
        ntohl(msg.body),msg.tail);
      fflush(stdout);
    }
}
```

send_upd.c: UDP/IP Client I

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
main(argc,argv)
int argc; char **argv;
{
   int socket_fd;
   struct sockaddr_in dest;
   struct hostent *hostptr;
   struct { char head; u_long body; char tail; } msgbuf;
   socket_fd = socket (AF_INET, SOCK_DGRAM, 0);
   /* You must do this just in case */
   bzero((char *) &dest, sizeof(dest));

   hostptr = gethostbyname(argv[1]);
   dest.sin_family = (short) AF_INET;
   bcopy(hostptr->h_addr, (char *)&dest.sin_addr,
      hostptr->h_length);
   dest.sin_port = htons((u_short)0x3333);
```

send_upd.c: UDP/IP Client II

```
    msgbuf.head = '<';
    msgbuf.body = htonl(getpid()); /* IMPORTANT! */
    msgbuf.tail = '>';
    sendto(socket_fd,&msgbuf,sizeof(msgbuf),0,
      (struct sockaddr *)&dest, sizeof(dest));
}
```

Similarities and Differences

- Note that there are striking similarities between Unix datagram programs and Internet datagram programs
- We do need to do extra work for Internet programs
- Socket creation parameters are trivially different
- Naming conventions are significantly
- The underlying implementation is completely different! But hidden from the programmers
- Practical note: You can always test and develop network programs on "localhost" (127.0.0.1). The implementation should be smart enough to *NOT* send the packets over the network (instead just pass it from output buffer to input buffer)