

Digital Communication in the Modern World

Transport Layer: Berkeley Sockets

<http://www.cs.huji.ac.il/~com1>
com1@cs.huji.ac.il

Some of the slides have been borrowed from:
Computer Networking: A Top Down Approach Featuring the Internet,
2nd edition,
Jim Kurose, Keith Ross
Addison-Wesley, July 2002.

Computer Communication 2004-5

1

Berkeley Sockets

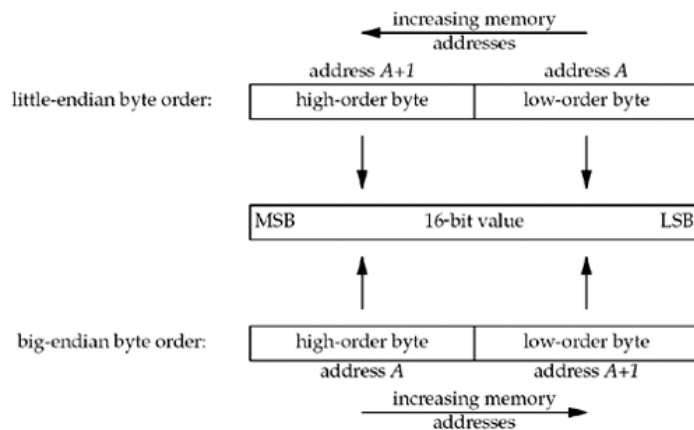
Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

The socket primitives for TCP.

2

Endian Byte Order

Little-endian byte order and big-endian byte order for a 16-bit integer.



3

Program to determine host byte order

```
1 #include      "unp.h"

2 int
3 main(int argc, char **argv)
4 {
5     union {
6         short    s;
7         char     c[sizeof(short)];
8     } un;

9     un.s = 0x0102;
10    printf("%s: ", CPU_VENDOR_OS);
11    if (sizeof(short) == 2) {
12        if (un.c[0] == 1 && un.c[1] == 2)
13            printf("big-endian\n");
14        else if (un.c[0] == 2 && un.c[1] == 1)
15            printf("little-endian\n");
16        else
17            printf("unknown\n");
18    } else
19        printf("sizeof(short) = %d\n", sizeof(short));

20    exit(0);
21 }
```

4

Socket Programming Example: Internet File Server

Client code using
sockets.

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

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

#define SERVER_PORT 12345 /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096 /* block transfer size */

int main(int argc, char **argv)
{
    int c, s, bytes;
    char buf[BUF_SIZE]; /* buffer for incoming file */
    struct hostent *h; /* info about server */
    struct sockaddr_in channel; /* holds IP address */

    if (argc != 3) fatal("Usage: client server-name file-name");
    h = gethostbyname(argv[1]); /* look up host's IP address */
    if (!h) fatal("gethostbyname failed");

    s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s < 0) fatal("socket");
    memset(&channel, 0, sizeof(channel));
    channel.sin_family = AF_INET;
    memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
    channel.sin_port = htons(SERVER_PORT);

    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");

    /* Connection is now established. Send file name including 0 byte at end. */
    write(s, argv[2], strlen(argv[2])+1);

    /* Go get the file and write it to standard output. */
    while (1) {
        bytes = read(s, buf, BUF_SIZE); /* read from socket */
        if (bytes <= 0) exit(0); /* check for end of file */
        write(1, buf, bytes); /* write to standard output */
    }

    fatal(char *string)
    {
        printf("%s\n", string);
        exit(1);
    }
}
```

Socket Programming Example: Internet File Server (2)

Client code using
sockets.

```
#include <sys/types.h> /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345 /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096 /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
    int s, b, l, fd, sa, bytes, on = 1;
    char buf[BUF_SIZE]; /* buffer for outgoing file */
    struct sockaddr_in channel; /* hold's IP address */

    /* Build address structure to bind to socket. */
    memset(&channel, 0, sizeof(channel)); /* zero channel */
    channel.sin_family = AF_INET;
    channel.sin_addr.s_addr = htonl(INADDR_ANY);
    channel.sin_port = htons(SERVER_PORT);

    /* Passive open. Wait for connection. */
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* create socket */
    if (s < 0) fatal("socket failed");
    setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

    b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
    if (b < 0) fatal("bind failed");

    l = listen(s, QUEUE_SIZE); /* specify queue size */
    if (l < 0) fatal("listen failed");

    /* Socket is now set up and bound. Wait for connection and process it. */
    while (1) {
        sa = accept(s, 0, 0); /* block for connection request */
        if (sa < 0) fatal("accept failed");
        read(sa, buf, BUF_SIZE); /* read file name from socket */

        /* Get and return the file. */
        fd = open(buf, O_RDONLY); /* open the file to be sent back */
        if (fd < 0) fatal("open failed");

        while (1) {
            bytes = read(fd, buf, BUF_SIZE); /* read from file */
            if (bytes <= 0) break; /* check for end of file */
            write(sa, buf, bytes); /* write bytes to socket */
        }
        close(fd); /* close file */
        close(sa); /* close connection */
    }
}
```