Client-Server Model

Message passing systems are popular because they support *client-server* interactions, where:

- *clients* — send messages to servers requesting a server.
- *servers* — provide services requested by clients. A client sends a message requesting service to the server, and the server returns its response in a response message.

Note: a process can be a client and a server at the same time.

Ports

Need to locate a process. Use a *port* (like a mailbox).

There are two types of ports:

- *well known ports* — those that rarely change over time. For instance, servers that provide mail, file transfer, remote login, etc.
- *dynamic ports* — typically used only for the life of a process. For instance, pipes can be implemented using message passing.

Examples of Servers

- mail server — allows client mail program to connect to mail server on remote machine.
- login server — allows clients to establish login sessions on remote machine.
- file servers — client requests to read or write part of a file. The server might support several operations, including *read*, *write*, *open*, *close* and *seek*.
- sort server — takes a list of items from the client, sorts them, and returns the sorted list
- authentication server — grants permissions to access files, log into the system, etc.

Sockets

How do we provide interprocess communication between processes that do not share a common ancestor?

Sockets are a generalization of file input/output that supports interprocess communication.

Solution: applications use *sockets* and *bind* port names to them. Processes then send messages to the bound (port) name.

Finger Server Example

1. Create socket (using *socket()* (don't look at details)
2. Bind port number to socket. Look in `/etc/services` using *getservbyname()*. Well-known port number.
3. Wait for request. On request gather information and send response to requesting port.

Finger Client:

1. Create socket and bind arbitrary port number to socket.
2. Get port number and host address of finger port.
3. Send request to finger port (containing client port address).
4. Wait for response, read it, and cleanup.

Describe for a server that forks off a process to handle the connection. Use an alternate port for the server.

Look at picture!

Code Example

Client sends user name and receives back a message.

```
< ccc1 /cs/cs513/public/example 1 >make sockclient
gcc  -g sockclient.c  -o sockclient
< ccc1 /cs/cs513/public/example 2 >make sockserver
gcc  -g sockserver.c  -o sockserver
< ccc1 /cs/cs513/public/example 3 >./sockserver 4242&
[1] 28721
< ccc1 /cs/cs513/public/example 4 >./sockclient ccc1 4242
Hello cew, you are visitor number 1 to this server
< ccc1 /cs/cs513/public/example 5 >./sockclient ccc1 4242
Hello cew, you are visitor number 2 to this server
< ccc1 /cs/cs513/public/example 6 >kill %1
```

Client

```
/* adapted from Sample 25_1 from "Computer Networks and Internets, 2nd ed by Comer */
/* sockclient.c - code for example client program that uses TCP */

#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>
#include <pwd.h>

#define PROTOPORT       5193            /* default protocol port number */

extern  int             errno;
char    localhost[] =   "localhost";    /* default host name            */
/*----------------------------------------------------------------------
 * Program:   client
 *
 * Purpose:   allocate a socket, connect to a server, and print all output
 *
 * Syntax:    client [ host [port] ]
 *
 *               host  - name of a computer on which server is executing
 *               port  - protocol port number server is using
 *
 * Note:      Both arguments are optional.  If no host name is specified,
 *            the client uses "localhost"; if no protocol port is
 *            specified, the client uses the default given by PROTOPORT.
 *
 *----------------------------------------------------------------------
 */
main(argc, argv)
    int     argc;
```

```
    char    *argv[];
{
    struct  hostent  *ptrh;  /* pointer to a host table entry       */
    struct  protoent *ptrp;  /* pointer to a protocol table entry   */
    struct  sockaddr_in sad; /* structure to hold an IP address     */
    int     sd;              /* socket descriptor                   */
    int     port;            /* protocol port number                */
    char    *host;           /* pointer to host name                */
    int     n;               /* number of characters read           */
    char    buf[1000];       /* buffer for data from the server     */
#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    struct passwd *ppwd;
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;         /* set family to Internet     */

    /* Check command-line argument for protocol port and extract    */
    /* port number if one is specified.  Otherwise, use the default */
    /* port value given by constant PROTOPORT                       */

    if (argc > 2) {                 /* if protocol port specified   */
        port = atoi(argv[2]);   /* convert to binary            */
    } else {
        port = PROTOPORT;       /* use default port number      */
    }
    if (port > 0)                   /* test for legal value         */
        sad.sin_port = htons((u_short)port);
    else {                          /* print error message and exit */
        fprintf(stderr,"bad port number %s\n",argv[2]);
        exit(1);
    }

    /* Check host argument and assign host name. */

    if (argc > 1) {
        host = argv[1];         /* if host argument specified   */
    } else {
        host = localhost;
    }

    /* Convert host name to equivalent IP address and copy to sad. */

    ptrh = gethostbyname(host);
```

```
if ( ((char *)ptrh) == NULL ) {
    fprintf(stderr,"invalid host: %s\n", host);
    exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

/* Map TCP transport protocol name to protocol number. */

if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
    fprintf(stderr, "cannot map \"tcp\" to protocol number");
    exit(1);
}

/* Create a socket. */

sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
    fprintf(stderr, "socket creation failed\n");
    exit(1);
}

/* Connect the socket to the specified server. */

if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
    fprintf(stderr,"connect failed\n");
    exit(1);
}

/* Repeatedly read data from socket and write to user's screen. */

if ((ppwd = getpwuid(getuid())) == NULL) {
    fprintf(stderr,"could not get user name\n");
    exit(1);
}
/* send our user name */
send(sd, ppwd->pw_name, strlen(ppwd->pw_name)+1, 0);
n = recv(sd, buf, sizeof(buf), 0);
while (n > 0) {
    write(1,buf,n);
    n = recv(sd, buf, sizeof(buf), 0);
}

/* Close the socket. */

closesocket(sd);
```

```
    /* Terminate the client program gracefully. */

    exit(0);
}
```

Client

```
/* adapted from Sample 25_2 from "Computer Networks and Internets, 2nd ed by Comer */
/* sockserver.c - code for example server program that uses TCP */

#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT       5193            /* default protocol port number */
#define QLEN            6               /* size of request queue        */

int     visits      =   0;              /* counts client connections    */
/*------------------------------------------------------------------------
 * Program:   server
 *
 * Purpose:   allocate a socket and then repeatedly execute the following:
 *              (1) wait for the next connection from a client
 *              (2) send a short message to the client
 *              (3) close the connection
 *              (4) go back to step (1)
 *
 * Syntax:    server [ port ]
 *
 *               port  - protocol port number to use
 *
 * Note:      The port argument is optional.  If no port is specified,
 *            the server uses the default given by PROTOPORT.
 *
 *------------------------------------------------------------------------
 */
main(argc, argv)
    int     argc;
```

```
    char    *argv[];
{
    struct  hostent  *ptrh;  /* pointer to a host table entry       */
    struct  protoent *ptrp;  /* pointer to a protocol table entry   */
    struct  sockaddr_in sad; /* structure to hold server's address  */
    struct  sockaddr_in cad; /* structure to hold client's address  */
    int     sd, sd2;         /* socket descriptors                  */
    int     port;            /* protocol port number                */
    int     alen;            /* length of address                   */
    char    buf[1000];       /* buffer for string the server sends  */
    char    ubuf[100];       /* buffer for user string              */

#ifdef WIN32
    WSADATA wsaData;
    WSAStartup(0x0101, &wsaData);
#endif
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;         /* set family to Internet     */
    sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address   */

    /* Check command-line argument for protocol port and extract    */
    /* port number if one is specified.  Otherwise, use the default */
    /* port value given by constant PROTOPORT                       */

    if (argc > 1) {                 /* if argument specified        */
        port = atoi(argv[1]);   /* convert argument to binary   */
    } else {
        port = PROTOPORT;       /* use default port number      */
    }
    if (port > 0)                   /* test for illegal value       */
        sad.sin_port = htons((u_short)port);
    else {                          /* print error message and exit */
        fprintf(stderr,"bad port number %s\n",argv[1]);
        exit(1);
    }

    /* Map TCP transport protocol name to protocol number */

    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
    }

    /* Create a socket */
```

```
    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
    }

    /* Bind a local address to the socket */

    if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
    }

    /* Specify size of request queue */

    if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
    }

    /* Main server loop - accept and handle requests */

    while (1) {
        alen = sizeof(cad);
        if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
            fprintf(stderr, "accept failed\n");
            exit(1);
        }
        if (recv(sd2, ubuf, sizeof(ubuf), 0) > 0) {
            visits++;
            sprintf(buf, "Hello %s, you are visitor number %d to this server\n",
                    ubuf, visits);
            send(sd2,buf,strlen(buf),0);
        }
        closesocket(sd2);
    }
}
```