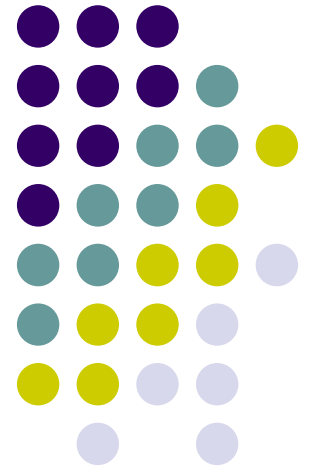


CS241

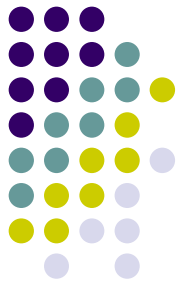
System Programming

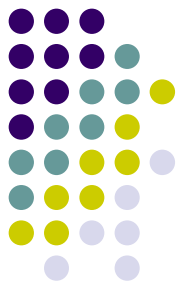
Discussion Section 11
April 17 – April 20



Outline

- Socket Programming
 - Library Functions
 - UICI Implementation
- Hypertext Transfer Protocol

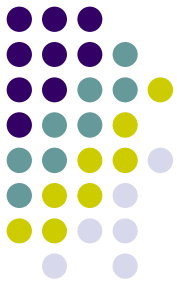




Socket

- Standard APIs for sending and receiving data across computer networks
- Introduced by BSD operating systems in 1983
- POSIX incorporated 4.3BSD sockets and XTI in 2001
- `#include <sys/socket.h>`

Socket Functions



- `socket`
- `bind`
- `connect`
- `listen`
- `accept`
- `send`, `sendto`
- `recv`, `recvfrom`
- `close`, `shutdown`

socket



```
int socket(int domain, int type, int protocol);
```

- Creates a communication endpoint
- Parameters
 - domain: `AF_INET` (IPv4)
 - type: `SOCK_STREAM` (TCP) or `SOCK_DGRAM` (UDP)
 - protocol: 0 (socket chooses the correct protocol based on type)
- Returns a nonnegative integer corresponding to a socket file descriptor if successful, -1 with `errno` set if unsuccessful

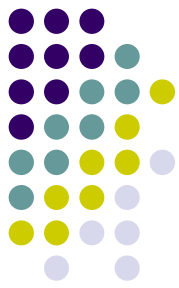


bind

```
int bind(int socket, const struct sockaddr
        *address, socklen_t address_len);
```

- Associates the socket with a port on your local machine
- `struct sockaddr_in` used for `struct sockaddr`

```
sa_family_t    sin_family; /* AF_INET */
in_port_t      sin_port;    /* port number */
struct in_addr sin_addr;    /* IP address */
```
- Returns 0 if successful, -1 with `errno` set if unsuccessful

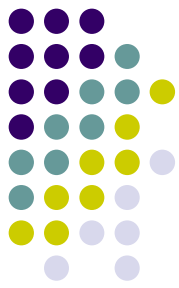


connect

```
int connect(int socket, const struct sockaddr  
            *address, socklen_t address_len);
```

- Establishes a link to the well-known port of the remote server
- Initiates the TCP 3-way handshake
 - Cannot be restarted even if interrupted
- Returns 0 if successful, -1 with `errno` set if unsuccessful

listen



```
int listen(int socket, int backlog);
```

- Puts the socket into the passive state to accept incoming requests
- Internally, it causes the network infrastructure to allocate queues to hold pending requests
 - `backlog`: number of connections allowed on the incoming queue
- `bind` should be called beforehand
- Returns 0 if successful, -1 with `errno` set if unsuccessful

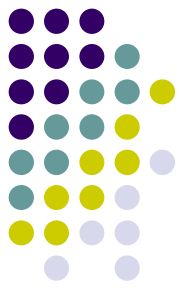
accept



```
int accept(int socket, struct sockaddr *restrict  
    address, socklen_t *restrict address_len);
```

- Accepts the pending requests in the incoming queue
- *address is used to return the information about the client making the connection.
 - `sin_addr.s_addr` holds the Internet address
- `listen` should be called beforehand
- Returns nonnegative file descriptor corresponding to the accepted socket if successful, -1 with `errno` set if unsuccessful

send and sendto



```
int send(int socket, const void *msg, int len, int flags);
```

```
int sendto(int socket, const void *msg, int len, int flags,  
           const struct sockaddr *to, socklen_t tolen);
```

- sends data pointed by `msg`
- `sendto` is used for unconnected datagram sockets. If used in connection-mode, last two parameters are ignored.
- Returns the number of bytes actually sent out if successful, -1 with `errno` set if unsuccessful



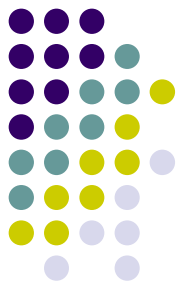
recv and recvfrom

```
int recv(int socket, void *buf, int len, int flags);
```

```
int recvfrom(int socket, void *buf, int len, int flags,  
             const struct sockaddr *from, socklen_t fromlen);
```

- receives data into the buffer `buf`
- `recvfrom` is used for unconnected datagram sockets. If used in connection-mode, last two parameters are ignored.
- Returns the number of bytes actually read if successful, -1 with `errno` set if unsuccessful

close and shutdown



```
int close(int socket);
```

```
int shutdown(int socket, int how);
```

- `close`
 - Prevents any more reads and writes
 - same function covered in file systems
- `shutdown`
 - provides a little more control
 - `how`
 - 0 – Further receives are disallowed
 - 1 – Further sends are disallowed
 - 2 – same as `close`
- Returns 0 if successful, -1 with `errno` set if unsuccessful

Implementation of u_open



```
int u_open(u_port_t port) {
    int error;
    struct sockaddr_in server;
    int sock;
    int true = 1;

    if ((u_ignore_sigpipe() == -1) ||
        ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1))
        return -1;

    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char *)&true,
                  sizeof(true)) == -1) {
        error = errno;
        while ((close(sock) == -1) && (errno == EINTR));
        errno = error;
        return -1;
    }
    /* continued on the next page */
}
```

- `setsockopt`
 - Sets the options on sockets
 - `SO_REUSEADDR` permits the server to be restarted immediately
- `u_ignore_sigpipe`
 - Sets `SIGPIPE` to be ignored. It terminates the process by default



Implementation of u_open

```
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons((short)port);

if ((bind(sock, (struct sockaddr *)&server, sizeof(server)) == -1) ||
    (listen(sock, MAXBACKLOG) == -1)) {
    error = errno;
    while ((close(sock) == -1) && (errno == EINTR));
    errno = error;
    return -1;
}
return sock;
}
```

- `htoln`, `htons`
 - Converts the address and port number fields to network byte order

Implementation of u_accept

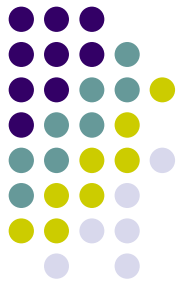


```
int u_accept(int fd, char *hostn, int hostnsize) {
    int len = sizeof(struct sockaddr);
    struct sockaddr_in netclient;
    int retval;

    while (((retval =
            accept(fd, (struct sockaddr *)&netclient, &len)) == -1) &&
            (errno == EINTR))
        ;
    if ((retval == -1) || (hostn == NULL) || (hostnsize <= 0))
        return retval;
    addr2name(netclient.sin_addr, hostn, hostnsize);
    return retval;
}
```

- addr2name
 - Converts the address to an ASCII host name

Implementation of u_connect



```
int u_connect(u_port_t port, char *hostn) {
    int error;
    int retval;
    struct sockaddr_in server;
    int sock;
    fd_set sockset;

    if (name2addr(hostn, &(server.sin_addr.s_addr)) == -1) {
        errno = EINVAL;
        return -1;
    }
    server.sin_port = htons((short)port);
    server.sin_family = AF_INET;

    if ((u_ignore_sigpipe() == -1) ||
        ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1))
        return -1;
}
```

- name2addr
 - Converts the host name into a binary address and stores it to its second parameter

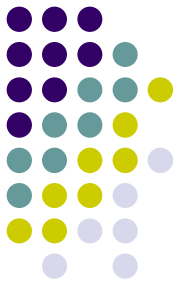
Implementation of u_connect



```
if (((retval =
    connect(sock, (struct sockaddr *)&server, sizeof(server))) == -1) &&
    ((errno == EINTR) || (errno == EALREADY))) {          /* asynchronous */
    FD_ZERO(&sockset);
    FD_SET(sock, &sockset);
    while (((retval = select(sock+1, NULL, &sockset, NULL, NULL)) == -1)
        && (errno == EINTR)) {
        FD_ZERO(&sockset);
        FD_SET(sock, &sockset);
    }
}
if (retval == -1) {
    error = errno;
    while ((close(sock) == -1) && (errno == EINTR));
    errno = error;
    return -1;
}
return sock;
}
```

Reference

- Beej's Guide to Network Programming
 - <http://beej.us/guide/bgnet/>



HTTP



- Hypertext Transfer Protocol
 - Delivers virtually all files and resources on the World Wide Web
 - Uses Client-Server Model
- HTTP transaction
 - HTTP client opens a connection and sends a request to HTTP server
 - HTTP server returns a response message

HTTP (continued)



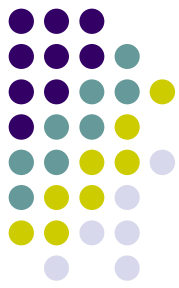
- Request

- GET /path/to/file/index.html HTTP/1.0
- Other methods (POST, HEAD) possible for request

- Response

- HTTP/1.0 200 OK
- Common Status Codes
 - 200 OK
 - 404 Not Found
 - 500 Server Error

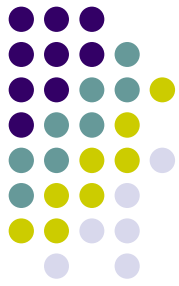
Sample HTTP exchange



- Scenario
 - Client wants to retrieve the file at the following URL (<http://www.somehost.com/path/file.html>)
- What a client does
 - Client opens a socket to the host www.somehost.com, port 80
 - Client sends the following message through the socket

```
GET /path/file.html HTTP/1.0
From: someuser@uiuc.edu
User-Agent: HTTPTool/1.0
[blank line here]
```

Sample HTTP exchange



- What a server does
 - Server responds through the same socket

```
HTTP/1.0 200 OK
```

```
Date: Mon, 17 Apr 2006 23:59:59 GMT
```

```
Content-Type: text/html
```

```
Content-Length: 1354
```

```
<html>
```

```
<body>
```

```
(more file contents)
```

```
.
```

```
.
```

```
.
```

```
</body>
```

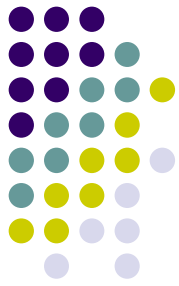
```
</html>
```



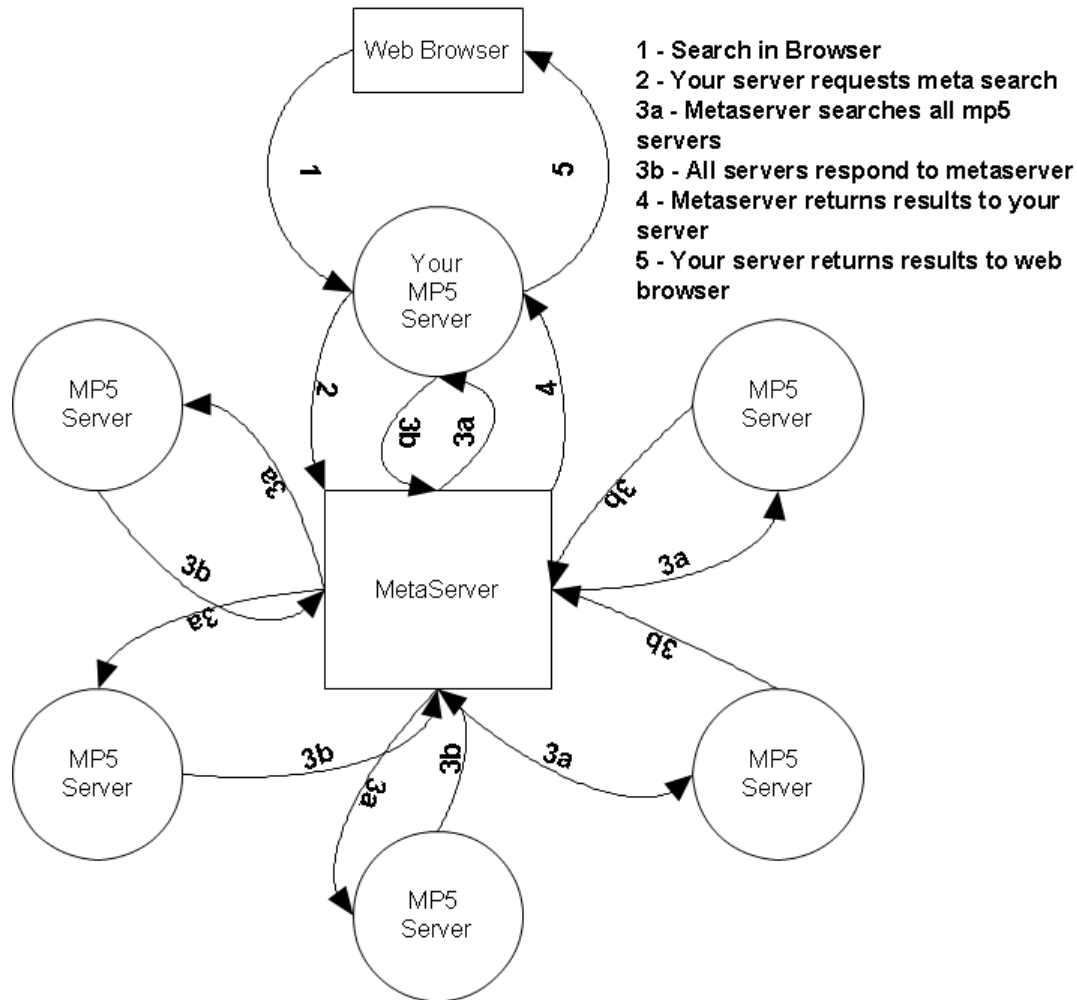
HTTP exchange in MP5

- Server-side exchange has been implemented in `helper.{h,c}`.
 - All you need to do is call them.
- `getHTTPRequest`
 - Converts the request message to a `struct Request`
 - `GET, HEAD` methods are supported
 - Pass this value to `sendHTTPReply`
 - `struct HeaderList` includes the list of headers
- `sendHTTPReply`
 - Generates the response message from the `Request`

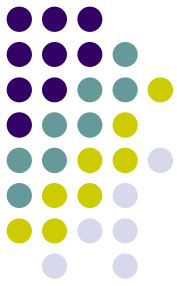
Communications in MP5



Meta-Searching for MP5



Summary



- Socket Programming
 - Library Functions
 - UICI can be implemented through sockets
- Hypertext Transfer Protocol
 - Request
 - Response