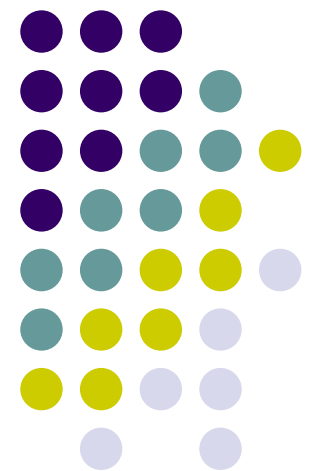


CS241

System Programming

Discussion Section 11
April 24 – April 27





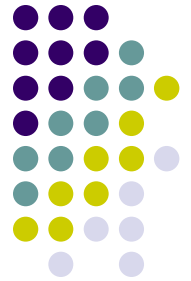
Outline

- Connectionless Socket Programming
 - Connectionless Library Functions
 - UICI UDP Implementation

- MP5 Clarifications

Recall:

Implementation of u_open



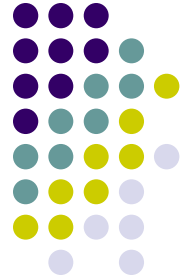
```
int u_open(u_port_t port) {
    int error;
    struct sockaddr_in server;
    int sock;
    int true = 1;

    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;
    }
    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;
}
```

Implementation of u_openudp



```
int u_openudp(u_port_t port) {
    int error;
    int one = 1;
    struct sockaddr_in server;
    int sock;

    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        return -1;
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one)) == -1) {
        error = errno;
        r_close(sock);
        errno = error;
        return -1;
    }
    if (port > 0) {
        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) {
            error = errno;
            r_close(sock);
            errno = error;
            return -1;
        }
    }
    return sock;
}
```

- Returns file descriptor if successful
- port
 - port=0 means client (bind does not occur)
 - port="well-known port" means server



u_open VS. u_openudp

- Only a server needs to `bind` the socket
- No `SIGPIPE` problem
 - A write to a pipe (e.g. TCP socket) generates a `SIGPIPE` signal when there are no active readers.
 - UDP provides no information about active readers.
 - UDP datagram is “sent correctly” when copied into network buffer (no error detection).

POSIX sendto



```
ssize_t sendto(int socket, const void *message, size_t
    length, int flags, const struct sockaddr *dest_addr,
    socklen_t dest_len);
```

- sends `length` bytes, from data pointed to by `message`
- `dest_addr` contains remote host address and remote port number.
- Returns the number of bytes actually sent out if successful, -1 with `errno` set if unsuccessful
- Datagram includes originating port number and source host address
- If connected, still uses `dest_addr` for destination

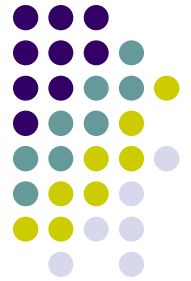


POSIX `recvfrom`

```
ssize_t recvfrom(int socket, void *restrict buffer, size_t length, int flags, struct  
sockaddr *restrict address, socklen_t *restrict address_len);
```

- Must associate `socket` with a port using `bind` or `sendto`
- Blocks until a datagram becomes available
 - Unassociated socket causes indefinite hang
- `buffer` is filled with received data
- `address` is filled with address of sender
- Must fill in `address_len` before calling
 - Why???
- Returns the number of bytes actually read if successful, -1 with `errno` set if unsuccessful

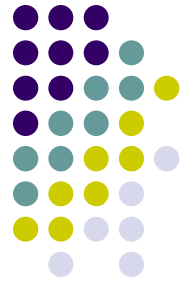
Implementation of `u_sendtohost`



```
ssize_t u_sendtohost(int fd, void *buf, size_t nbytes, char
    *hostn,
                    u_port_t port) {
    struct sockaddr_in remote;

    if (name2addr(hostn, &(remote.sin_addr.s_addr)) == -1) {
        errno = EINVAL;
        return -1;
    }
    remote.sin_port = htons((short)port);
    remote.sin_family = AF_INET;
    return u_sendto(fd, buf, nbytes, &remote);
}
```

- Used to initiate communication with a remote host (specified by `hostn` and `port`)

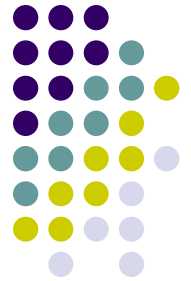


Implementation of u_sendto

```
ssize_t u_sendto(int fd, void *buf, size_t nbytes, u_buf_t *ubufp) {
    int len;
    struct sockaddr *remotep;
    int retval;

    len = sizeof(struct sockaddr_in);
    remotep = (struct sockaddr *)ubufp;
    while (((retval = sendto(fd, buf, nbytes, 0, remotep, len)) == -1)
        && (errno == EINTR)) ;
    return retval;
}
```

- Used to send a reply to a message received using `u_recvfrom` or `u_recvfromtimed`
- `u_buf_t` (`struct sockaddr_in`) structure that was filled by a previous call to `u_recvfrom` or `u_recvfromtimed`
- Basically a restarting `sendto`
- Returns a nonnegative integer corresponding to a socket file descriptor if successful, -1 with `errno` set if unsuccessful



Implementation of `u_recvfrom`

```
ssize_t u_recvfrom(int fd, void *buf, size_t nbytes, u_buf_t *ubufp) {
    int len;
    struct sockaddr *remote;
    int retval;

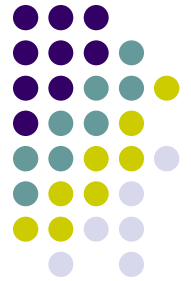
    len = sizeof (struct sockaddr_in);
    remote = (struct sockaddr *)ubufp;
    while (((retval = recvfrom(fd, buf, nbytes, 0, remote, &len)) == -1)
           && (errno == EINTR)) ;
    return retval;
}
```

- Restarting `recvfrom`
- Returned sender information fills the `u_buf_t *ubufp` parameter
- Can call `u_sendto` using this `u_buf_t`
- Returns `-1` with `errno` set if unsuccessful. Note: return value of `0` corresponds to datagram of length `0` (valid).



Recall: UDP is unreliable

- Datagrams can be lost without any errors generated.
- UDP-based protocols for providing reliability use request-reply or request-reply-acknowledge protocols
- Receiver must not block indefinitely
- Which brings us to...<next slide>



Implementation of `u_recvfromtimed`

```
ssize_t u_recvfromtimed(int fd, void *buf, size_t nbytes, u_buf_t *ubufp,
                        double seconds) {
    int len;
    struct sockaddr *remote;
    int retval;
    struct timeval timedone;

    timedone = add2currenttime(seconds);
    if (waitfdtimed(fd, timedone) == -1)
        return (ssize_t)(-1);
    len = sizeof (struct sockaddr_in);
    remote = (struct sockaddr *)ubufp;
    while (((retval = recvfrom(fd, buf, nbytes, 0, remote, &len)) == -1)
            && (errno == EINTR)) ;
    return retval;
}
```

- If successful, returns the number of bytes written into `buf`
- If timeout occurs, returns -1 and sets `errno` to `ETIME`
- `add2currenttime`: converts the time interval into an ending time
- `waitfdtimed`: Implemented using `select` – times out after specified number of seconds



`u_recvfromtimed`

- Suppose you call `u_recvfromtimed` with a timeout of 2 seconds and 10 signals come in 1 second apart. When does `u_recvfromtimed` time out if no data arrives?



`u_recvfromtimed`

- Suppose you call `u_recvfromtimed` with a timeout of 2 seconds and 10 signals come in 1 second apart. When does `u_recvfromtimed` time out if no data arrives?
- Answer:
2 seconds after it's called (timeout is independent of number of times it needs to restart – uses end time computed from `add2currenttime`)

UICI UDP – Receiver host information



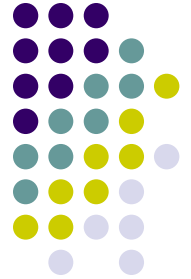
- Functions for examining the receiver host information (as filled in the `u_buf_t` structure)

```
void u_gethostname(u_buf_t *ubufp, char *hostn,  
int hostnsize)
```

```
void u_gethostinfo(u_buf_t *ubufp, char *info,  
int infosize)
```

```
int u_comparehost(u_buf_t *ubufp, char *hostn,  
u_port_t port)
```

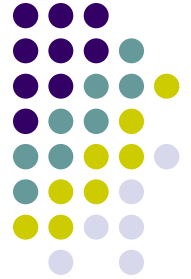
Implementation of `addr2name`



```
void addr2name(struct in_addr addr, char *name, int namelen) {
    struct hostent *hostptr;
    hostptr = gethostbyaddr((char *)&addr, 4, AF_INET);
    if (hostptr == NULL)
        strncpy(name, inet_ntoa(addr), namelen-1);
    else
        strncpy(name, hostptr->h_name, namelen-1);
    name[namelen-1] = 0;
}
```

- Non-reentrant version

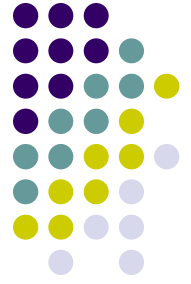
Implementation of name2addr



```
int name2addr(char *name, in_addr_t *addrp) {
    struct hostent *hp;

    if (isdigit((int)(*name)))
        *addrp = inet_addr(name);
    else {
        hp = gethostbyname(name);
        if (hp == NULL)
            return -1;
        memcpy((char *)addrp, hp->h_addr_list[0], hp->h_length);
    }
    return 0;
}
```

- Non-reentrant version



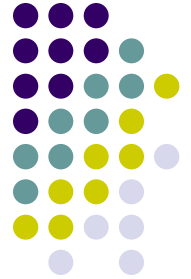
Implementation of `u_gethostname`

```
void u_gethostname(u_buf_t *ubufp, char *hostn,
                  int hostnsize) {
    struct sockaddr_in *remotep;

    remotep = (struct sockaddr_in *)ubufp;
    addr2name(remotep->sin_addr, hostn, hostnsize);
}
```

- Called after `u_recfrom` or `u_recvfromtimed`
- Creates a string containing the remote host name
- Note: If `addr2name` cannot convert the address to a host name, it sets `*hostn` to the dotted-decimal representation

Implementation of `u_gethostinfo`

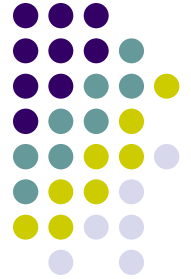


```
void u_gethostinfo(u_buf_t *ubufp, char *info, int infosize) {
    int len;
    int portnumber;

    portnumber = ntohs(ubufp->sin_port);
    len = sprintf(info, infosize, "port number is %d on host ",
                 portnumber);
    info[infosize-1] = 0;          /* in case name did not fit */
    if (len >= infosize) return;
    u_gethostname(ubufp, info+len, infosize-len);
}
```

- Used primarily for debugging
- Creates a printable string with host name and port number

Implementation of `u_comparehost`

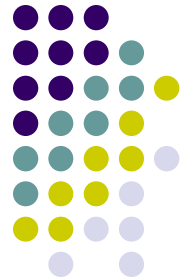


```
int u_comparehost(u_buf_t *ubufp, char *hostn, u_port_t port) {
    in_addr_t addr;
    struct sockaddr_in *remotep;

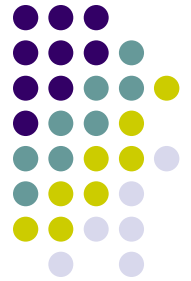
    remotep = (struct sockaddr_in *)ubufp;
    if ((port != ntohs(remotep->sin_port)) ||
        (name2addr(hostn, &addr) == -1) ||
        (memcmp(&(remotep->sin_addr.s_addr), &addr,
                sizeof(in_addr_t)) != 0))
        return 0;
    return 1;
}
```

- Returns 1 if the given host name and port number match the info in `*ubufp` and 0 otherwise
- Does comparison on address obtained from `name2addr` – not on host name itself

UDP vs. TCP

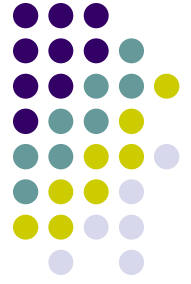


TCP	UDP
Connection-oriented	Each message has destination
Based on byte streams	Based on messages (datagrams)
Delivers streams of bytes in order sent	Delivers messages in order received
Reliable	Unreliable
Returns after write (depends indirectly on receiver status)	Returns after sendto (Does not depend on receiver status)

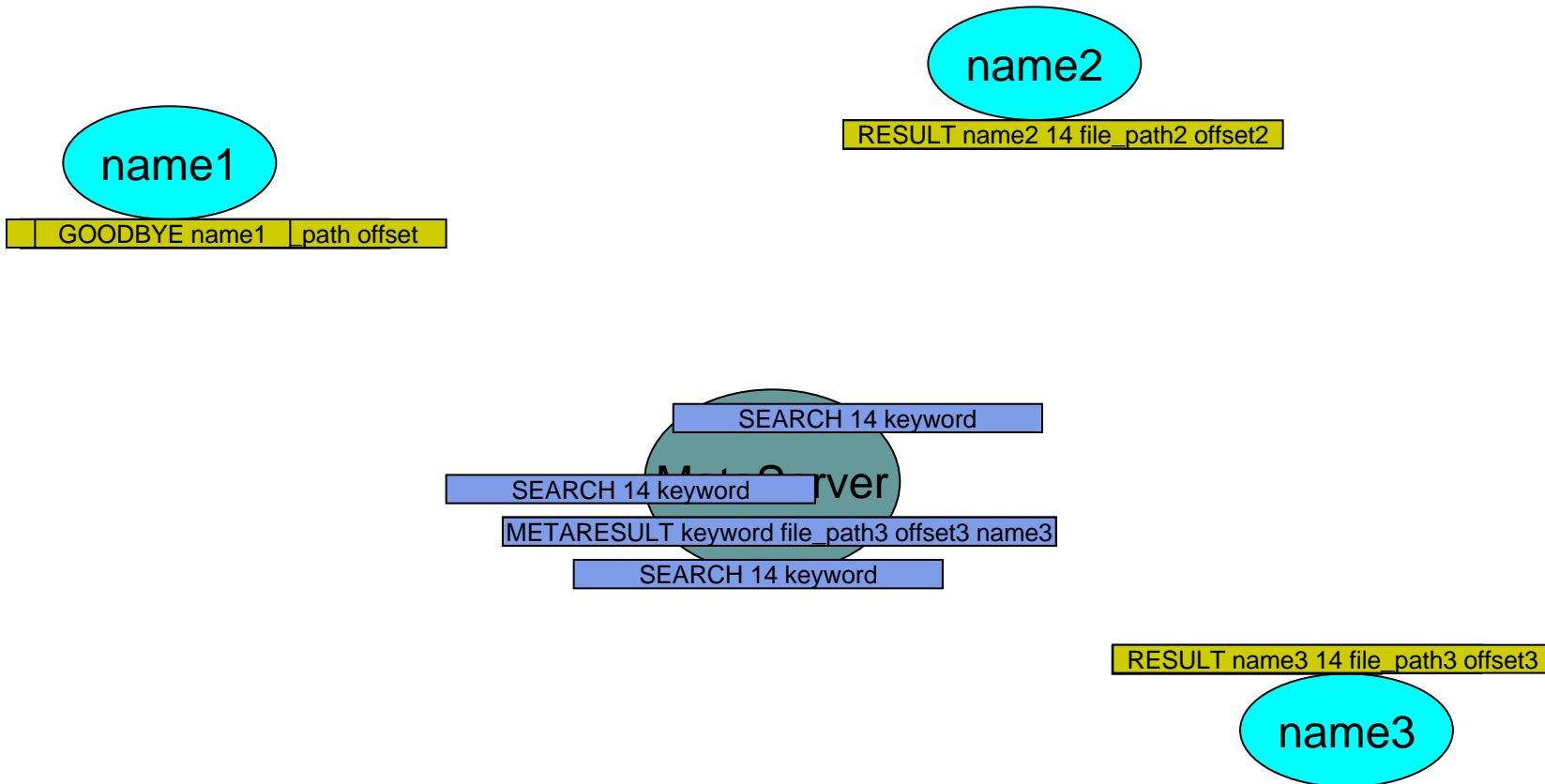


MP5 Clarifications

- resource string
 - Contains resource requested from webserver
- Examples
 - <http://csil-linux52.cs.uiuc.edu:8008/r?s=asdfasdf>
 - resource = `"/r?s=asdfasdf"`
 - <http://csil-linux52.cs.uiuc.edu:8008/index.html>
 - resource = `"/index.html"`



MP5 Datagram Example





Summary

- Connectionless Socket Programming
 - Connectionless Library Functions
 - UICI UDP Implementation

- MP5 Clarifications