

CS241 System Programming UICI Communication (II)

Klara Nahrstedt

Lecture 35

4/17/2006



Content

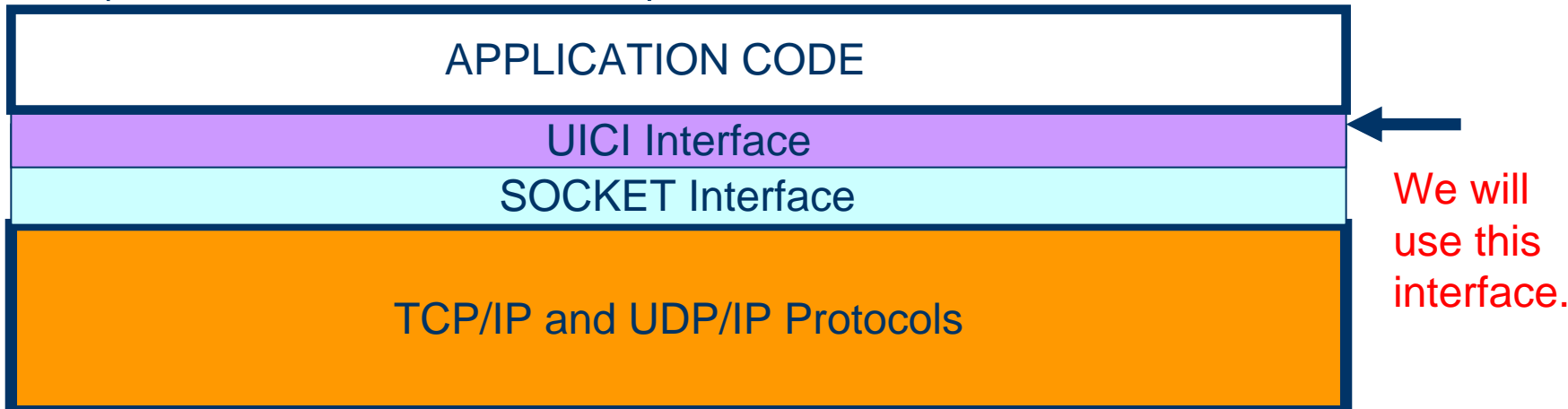
- Universal Internet Communication Interface (UICI)
 - Goal of UICI
 - Interaction of UICI client and server
 - UICI APIs
 - Handling errors
 - UICI Implementation of Server Strategies
 - UICI Clients
 - Relation to Sockets

Administrative Comments

- Deadline for MP4, April 18
- MP5 starts this week, deadline May 1
- Quiz 10, Friday, April 21
 - Covers T: 4.3, 4.4.1-4.4.6 and 4.5.1-4.5.3
- This week, we will cover R&R Chapter 18 and 20

University Internet Communication Interface (UICI)

- Provides a simplified interface to connection-oriented communication in UNIX
- **NOT part of any UNIX standard**
- **Designed by Robins and Robins authors to hide details of underlying network protocols**
- We will use UICI in MP5 so you must upload uici.h (see MP5 instructions)



UICI: A simple connection-oriented client-server implementation

The server does the following:

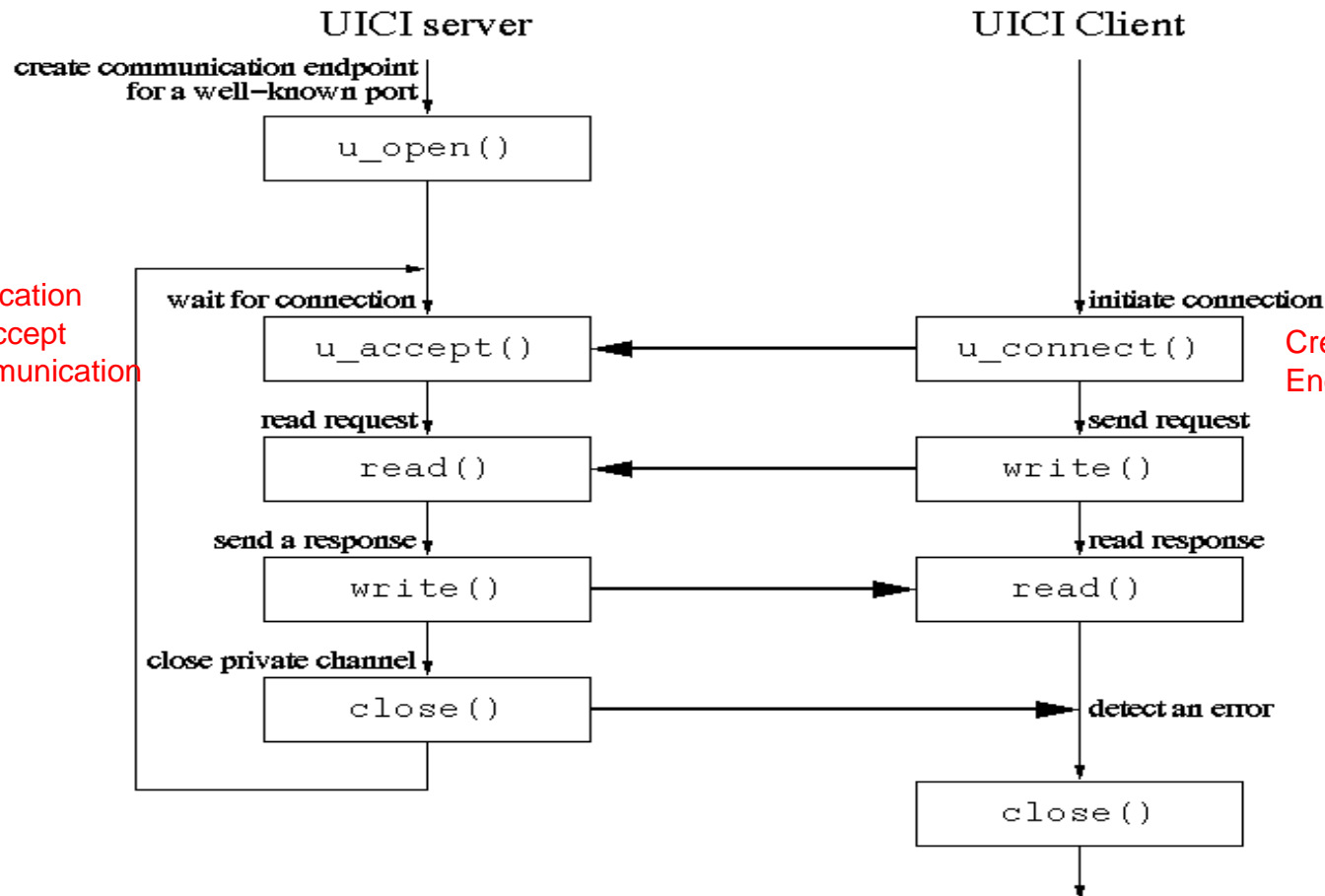
- open a well-known listening port and obtain a **listening file descriptor** (fd) from the port
- wait for a **connection request** from the client on that listening fd
- when a request comes in, **return a communication file descriptor** (to use as a handle for private, two-way client-server communication)
- the server handles the connection by **reading or writing using the communication file descriptor**
- this can be done serially, or by a child while the parent is waiting for another request

UICI: A simple connection-oriented client-server implementation

A client does the following:

- **request a connection** to a particular port on a particular server host
- if successful, server returns a **communication file descriptor** for communication
- communicates by doing **reads and writes** on this file descriptor
- **close** the communication file descriptor

UICI Commands for Client-Server Interaction



UICI Library

| UICI Prototype | Description |
|---|---|
| <code>int u_open(u_port_t port)</code> | Creates a TCP socket bound to port and sets the socket to be passive. Returns a file descriptor for the socket. |
| <code>int u_accept(int fd, char *hostn, int hostnsize)</code> | Waits for connection request on fd. On successful return, hostn has the first hostnsize-1 characters of the client's host name. Returns a communication file descriptor. |
| <code>int u_connect(u_port_t port, char *hostn)</code> | Initiates a connection to server on port port and host hostn. Returns a communication file descriptor. |

Socket Implementation of UICI

| UICI | Sockets | action |
|-------------|----------------|---|
| u_open | socket | create communication endpoint |
| | bind | associate endpoint with a specific port |
| | listen | make endpoint passive listener |
| u_accept | accept | accept connection request from client |
| u_connect | socket | create communication endpoint |
| | connect | request connection from server |

All socket system calls return -1 and set errno.

Handling Errors and Information Exchange

- **Error Handling**

All UICI routines return -1 on error with errno set.

- **Reading and writing**

- After a connection is set up between the client and the server, read and write can be used.
- It is recommended that you use **r_read** and **r_write** instead. These behave better in the presence of signals and r_write will write the number of bytes requested or return an error.
- When doing network communication, it is not unusual for a write call to return with fewer bytes written than requested.

UICI Implementation of Different Server Strategies

- Example shows client sending text to server
 1. Server listening to port
 2. Server listening to port creating processes to communicate
 3. Client accessing port

UICI Implementation of Server Listening to Passive Port (Code from Program 18.1)

```
... /* initialize */
11 portnumber = (u_port_t) atoi(argv[1]);
12 ...listenfd = u_open(portnumber) ...

18 for ( ; ; ) {
19 ... communfd = u_accept(listenfd, client, MAX_CANON)) ...
24     bytescopied = copyfile(communfd, STDOUT_FILENO);
26     ... r_close(communfd) ...
27 }
... /* cleanup */

11 Get a port number in network byte order
12 Open port for requests at port address
19 Get a request for a communication and open communication port
24 Copy all the bytes from the communication
26 Close communication port
```

Server Listening, Creating Process for Communication (Code from Program 18.2)

...

```
22. portnumber = (u_port_t) atoi(argv[1]);  
23. ... u_open(portnumber)  
30. ... u_accept(listenfd, client, MAX_CANON))  
35. ... child = fork()  
38. ... r_close(listenfd)  
42. ... copyfile(communfd, STDOUT_FILENO);  
46. ... r_close(communfd)  
49. ... r_waitpid(-1, NULL, WNOHANG)
```

35. Create a child

38. Main process closes listening port

42. Child process copies data from communication port

46. Child process closes communication port

49. Main process makes sure no children are around when it finishes

UICI Client Code (Code from Program 18.3)

```
14. portnumber = (u_port_t)atoi(argv[2]);  
15. ...u_connect(portnumber, argv[1])  
19. ...copyfile(STDIN_FILENO, communfd);
```

- 14. Converts port to network byte order
- 15. Asks for server port through passive server port number
- 16. Copies all of data from input to remote port.

Compiling network code under Linux and Solaris

- To compile network code under Linux, include the nsl library by putting -lnsl at the end of the compile line as in:
`cc -o server server.c restart.c uici.c uiciname.c -lnsl`
- To compile under Solaris, you also need to include the socket library:
`cc -o server server.c restart.c uici.c uiciname.c -lnsl -lsocket`

Summary

- Client-Server Process Communication
 - Communication Channel
 - Communication Protocols
 - Connectionless vs Connection-oriented
 - Connection-oriented Server Strategies