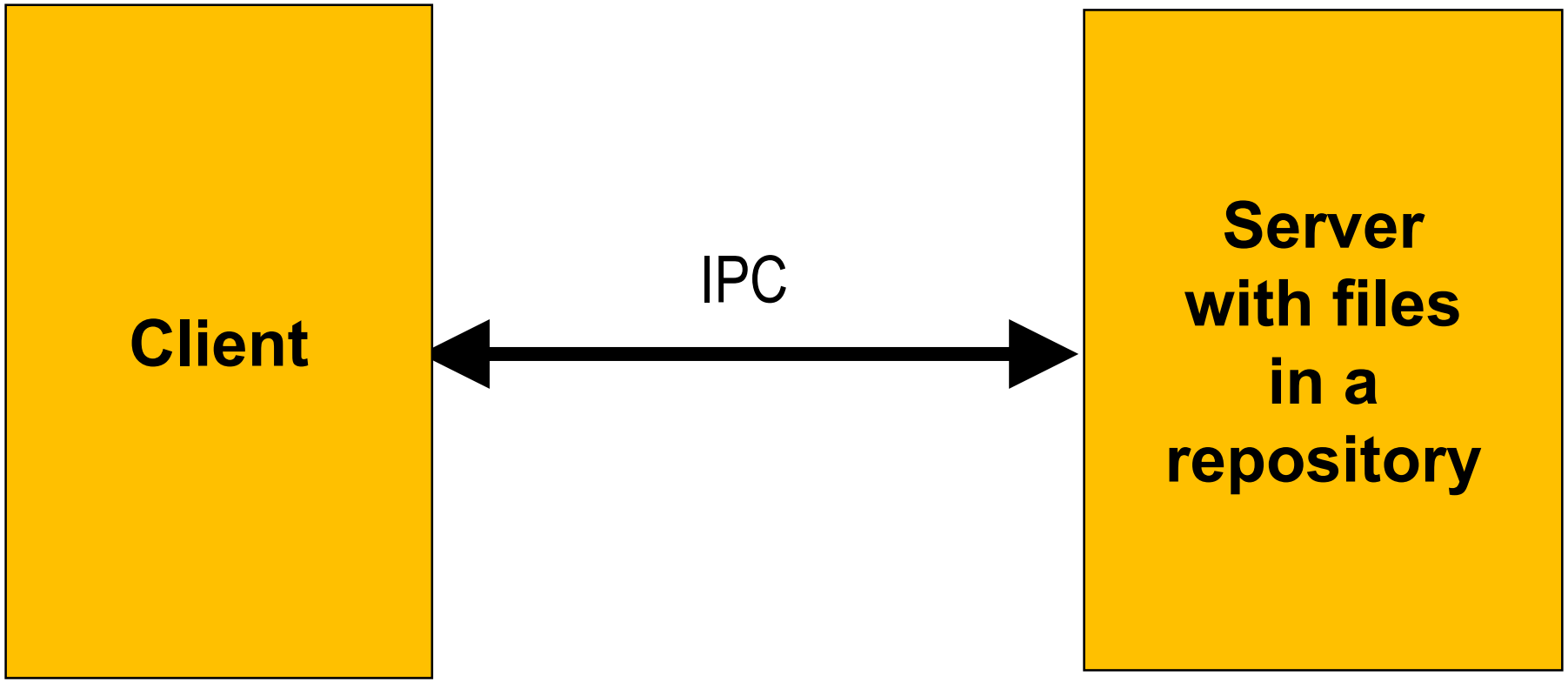
THE SECOND SPRING 2024 COSC 3360 ASSIGNMENT EXPLAINED

THE PROBLEM

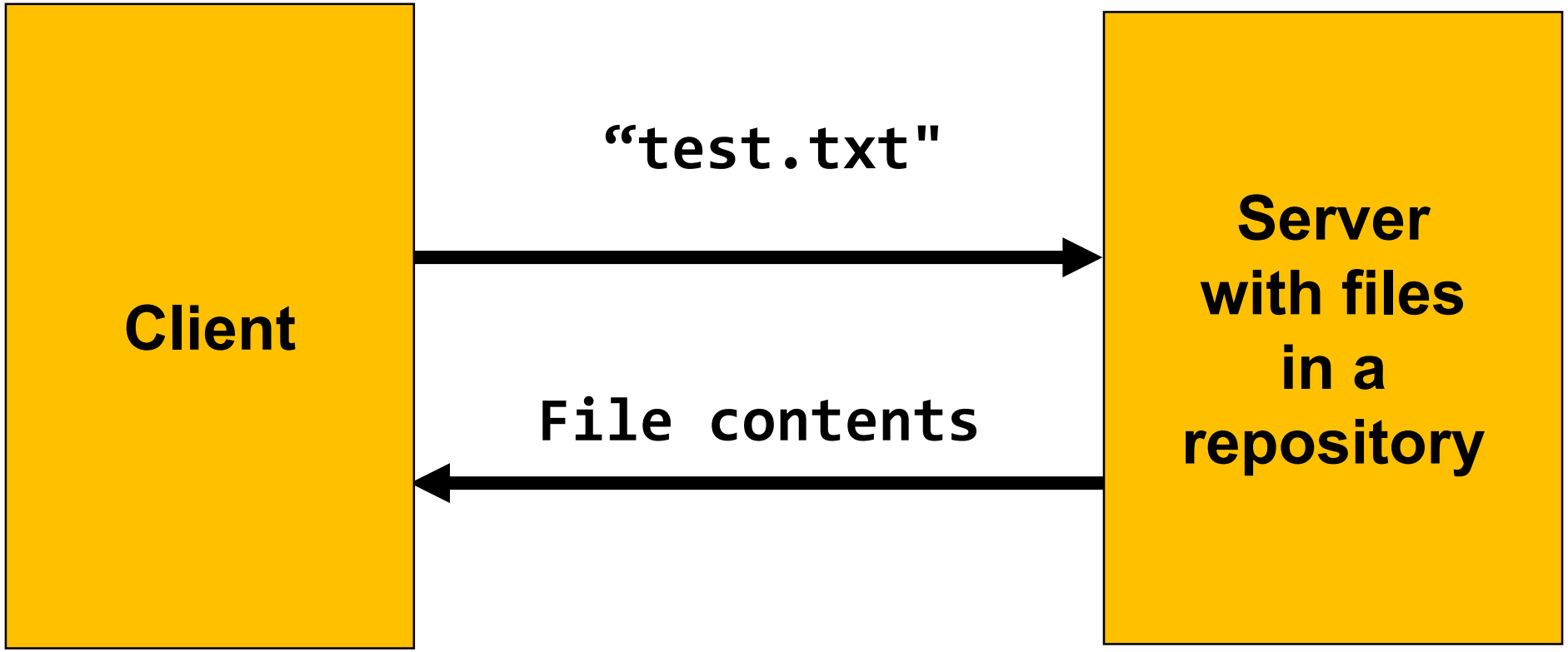
- Build a client-server pair where the clients request file downloads for the server.

- To get credit, your two programs ***must***
  - Be written in C or C++
  - Use stream sockets in the internet domain

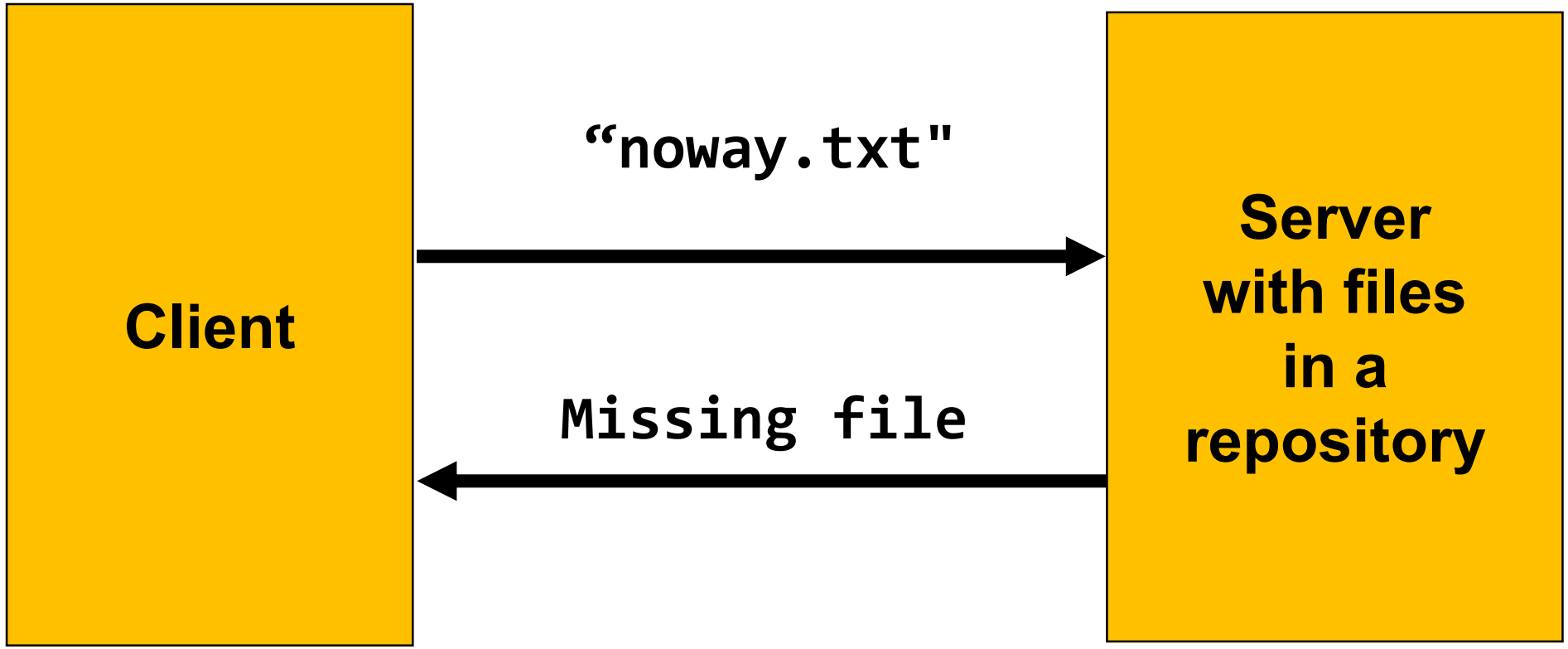
YOUR PROGRAMS



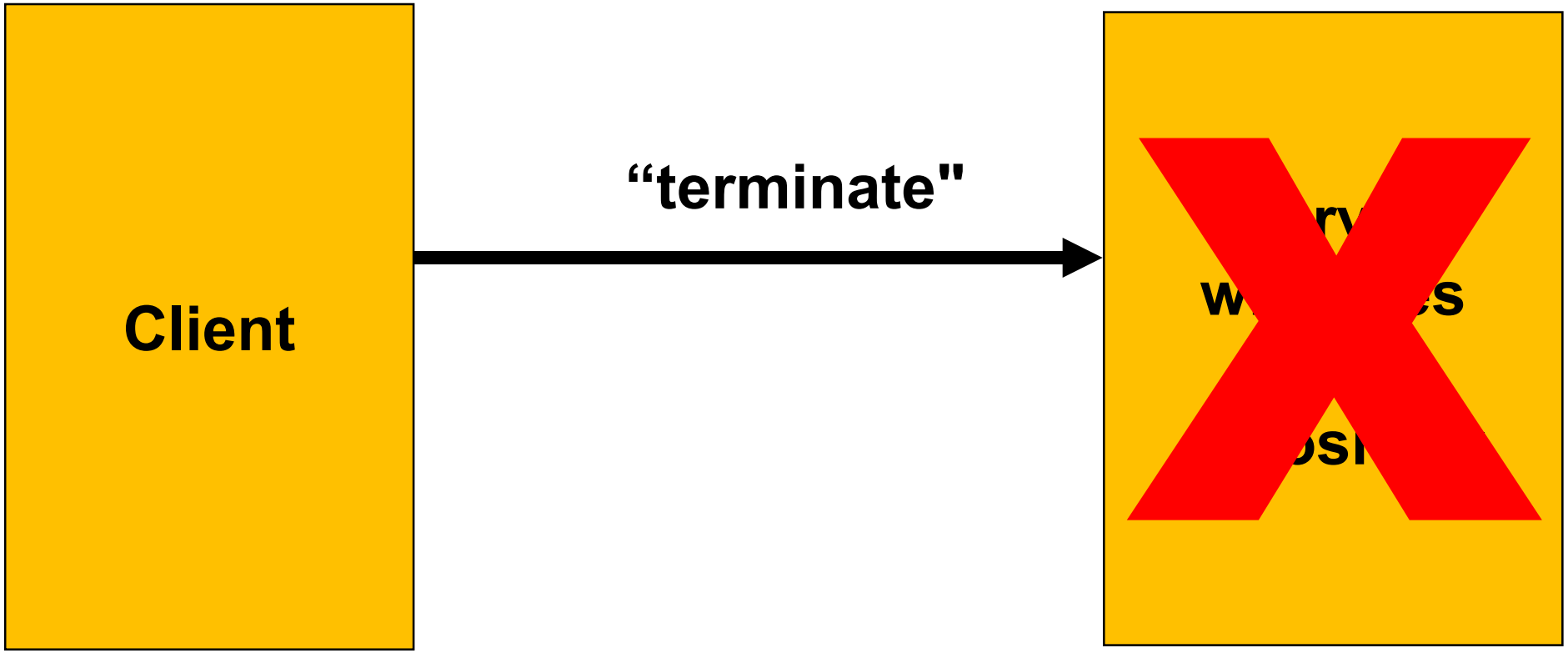
A normal message exchange



The requested file is missing



The client sends a terminate message



Overview

- Will have to implement a very basic talk pair
- Will have to write
  - A ***client*** that will send requests and waits for replies.
  - A ***server*** that will wait for requests from multiple clients

Client side

- Client will
  1. Prompt the user for the server hostname and port number
  2. Create a socket
  3. Prompt the user for a command
  4. Act on the command
  5. Receive the server's reply (optional for **`terminate`** command)
  6. Go back to step 3 unless last the command was **`quit`** or **`terminate`**

Server side

The loop will end when the server receives a `terminate` message

- Server will
  1. Create a socket
  2. Bind an address to that socket
  3. Wait for a request
  4. Fork a child to handle each request\
  5. Either send the file contents or an error message
  6. Terminate the child
  7. Wait for the next client request

Communicating through stream sockets

TCP socket calls (I)

- **`socket(…)`**
  creates a new socket of a given socket type
  (*both client and server sid*es)
- **`bind(…)`**
  binds a socket to a socket address structure (*server side*)
- **`listen(…)`**
  set up buffer size and
  puts a bound TCP socket into listening state (*server side*)

TCP socket calls (II)

- **`connect(…)`**
  requests a new TCP connection from the server (*client side*)
- **`accept(…)`**
  accepts an incoming connect request and creates a new socket associated with the socket address pair of this connection (*server side*)

Accept "magic" (I)

- **`accept(…)`** was designed to implement multithreaded servers
  - Each time it accepts a connect request it creates a ***new socket*** to be used for the duration of that connection
  - Can, if we want, fork a ***child*** to handle that connection
    - ***Not needed this time***

Accept "magic" (II)

New child will
do the work

TCP socket calls (III)

- **`write(…)`**
  sends data to a remote socket
  (*both client and server sides*)
- **`read(…)`**
  receives data from a remote socket
  (*both client and server sides*)
- **`close(…)`**
  terminates a TCP connection
  (*both client and server sides*)

TCP socket calls (IV)

- **`gethostbyname(…)`**
  returns host address structure associated with a given host name

If your client and your server are on
“well-connected” computers, they will both do:

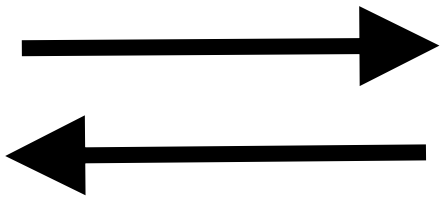
```
gethostname(myname, MAXLEN);
hp = gethostbyname(myname);
```

Summary

- **Client side:**

```
csd = socket(…)

connect(csd, …)
write(csd, …)
read(csd, …)
close(csd)
```



- **Server side:**

```
ssd = socket(…)
bind(…)
listen(…)
newsd = accept(…)
read(newsd, …)
write(newsd, …)
close(newsd)
```

Bad news and good news

- The bad news is that socket calls are somewhat esoteric
  - Might feel you are not fully understanding what you are writing
- The good news is most of these mysterious options are fairly standard

Some examples (I)

- ```
  // create socket
  if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
     return(-1);
  ```

- With datagram sockets (SOCK_DGRAM), everything would be different
  - No **`listen(…)`**, no **`accept(…)`**, no **`connect(…)`**
  - Only **`sendto(…)`** and **`recvfrom(…)`**
  - Message boundaries would be preserved
  ```

Some examples (II)

```
gethostname(myname, MAXHOSTNAME);
// get host address structure
hp = gethostbyname(myname);
sa.sin_family = hp->h_addrtype; // host address
sa.sin_port = htons(portnum); // set port number
//bind address to an existing socket
if (bind(s, &sa, sizeof(struct sockaddr_in)) < 0) {
   close(s);
   return(-1);
} // if
```

Picking a port number

- Your port number should be
  - ***Unique***
    - *Should not interfere with other students' programs*
  - ***Greater than or equal to* 1024**
    - *Lower numbers are reserved for privileged applications*

Some examples (III)

- ```
  // set buffer size for a bound socket
  listen(s, 3);
  ```
- ```
  // request a connection
  // sa must contain address of server
  if (connect(s, &sa, sizeof sa) < 0) {
     close(s);
     return(-1);
  }
  ```

Some examples (IV)

- ```
  // accept a connection and create new socket
  int new_s;
  if ((new_s = accept(s, NULL, NULL)) < 0)
      return(-1)
  ```
- ```
  // send a message
  write(new_s, buffer, nbytes);
  ```
- ```
  // read a message
  read(new_s, buffer, nbytes)
  ```

A fixed number of bytes
  ```

In reality

- No message boundaries on stream sockets
- Client and server exchange ***bytes***
  - Works well if process at receiving end knows ***ahead of time*** how many bytes to expect
    - Sender sends 20 bytes
    - Receiver must read ***exactly*** 20 bytes

Doing networking assignments on your PC

- The host name of a Windows PC does not include its domain
  - **jfparis@Odeon:~$ hostname**
    Odeon
- **hp = gethostbyname("Odeon");**
  does ***not always*** work
- Use instead **localhost**
  hp = gethostbyname(localhost);

Implementation details

The repository files

- Stored in **`~/Repository`**
- File names will not contain spaces

The client messages

- Either
  - `Received file test.txt (256 bytes)`
- or
  - `File test.txt is missing`

- You may choose to print out alternate messages as long as they convey the same information

The server messages

- Either
  - **A client requested the file test.txt**
    Sent 256 bytes
- or
  - **A client requested the file noway.txt**
    That file is missing!
- You may choose to print out alternate messages as long as they convey the same information

The sample program pair

- Will be posted on Teams by March 6
- Will show how to exchange ***variable-size messages*** on TCP ***streams***

Some good tutorials

- https://www.geeksforgeeks.org/udp-server-client-implementation-c/
- https://www.programminglogic.com/sockets-programming-in-c-using-udp-datagrams/
- https://www.softprayog.in/programming/network-socket-programming-using-udp-in-c

- Can lift from them all the code you need.