國立臺灣大學

Assignment 2

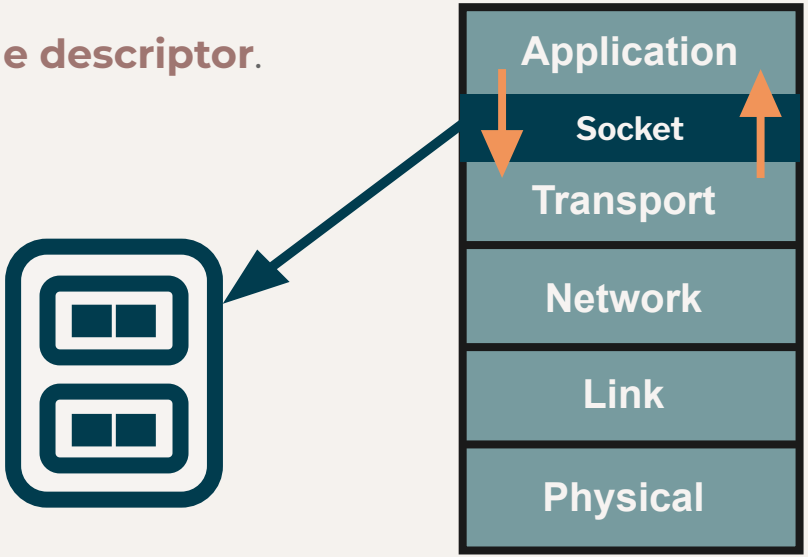
Socket Programming

Prof. Ai-Chun Pang
TA / Zheng-Ying Huang, Chan-Yu Li, Kuang-Hui Huang

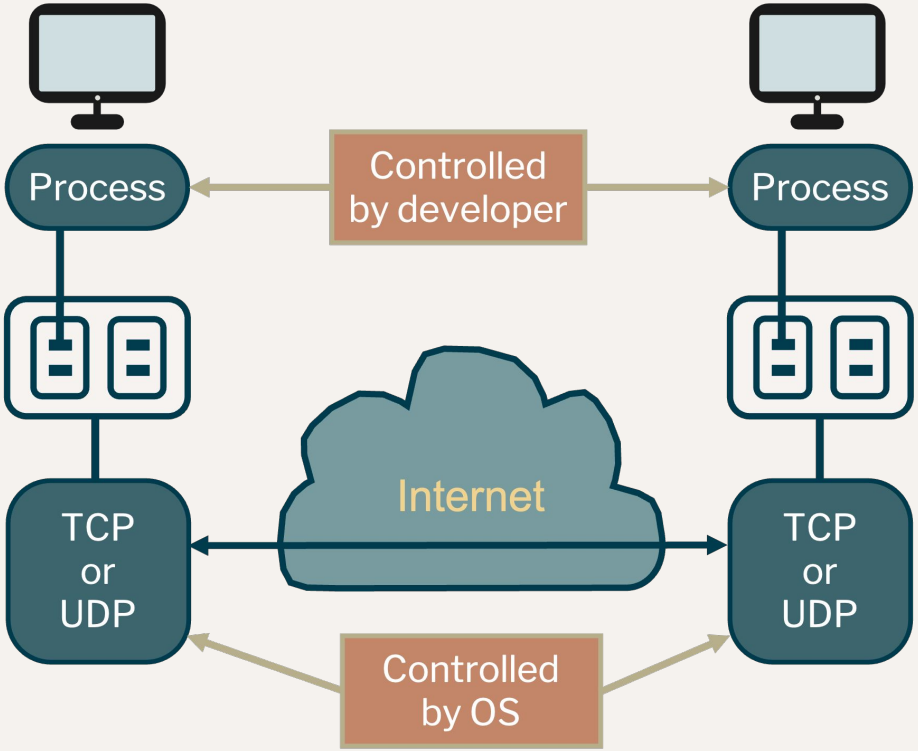
Socket Programming Tutorial

What is Socket?

- **Socket** is the **API for the TCP/IP protocol** stack.
- Provides communication **between the Application layer and Transport layer**.
- Make internet communication **like a file descriptor**.
  - read() and write()
- We will provide sample codes for you.



What is Socket?



File Descriptors

- When we open an existing file or create a new file, the kernel returns a file descriptor to the process.
- If we want to read or write a file, we can access it through the file descriptor.
- Socket in C use file descriptors to send and receive data.

```
int fd = open("example.txt", O_WRONLY | O_CREAT | O_TRUNC);
char *text = "Happy coding time\n";
write(fd, text, strlen(text));
close(fd);
```

```
char *buf[1024];
read(0, buf, 1024);
```

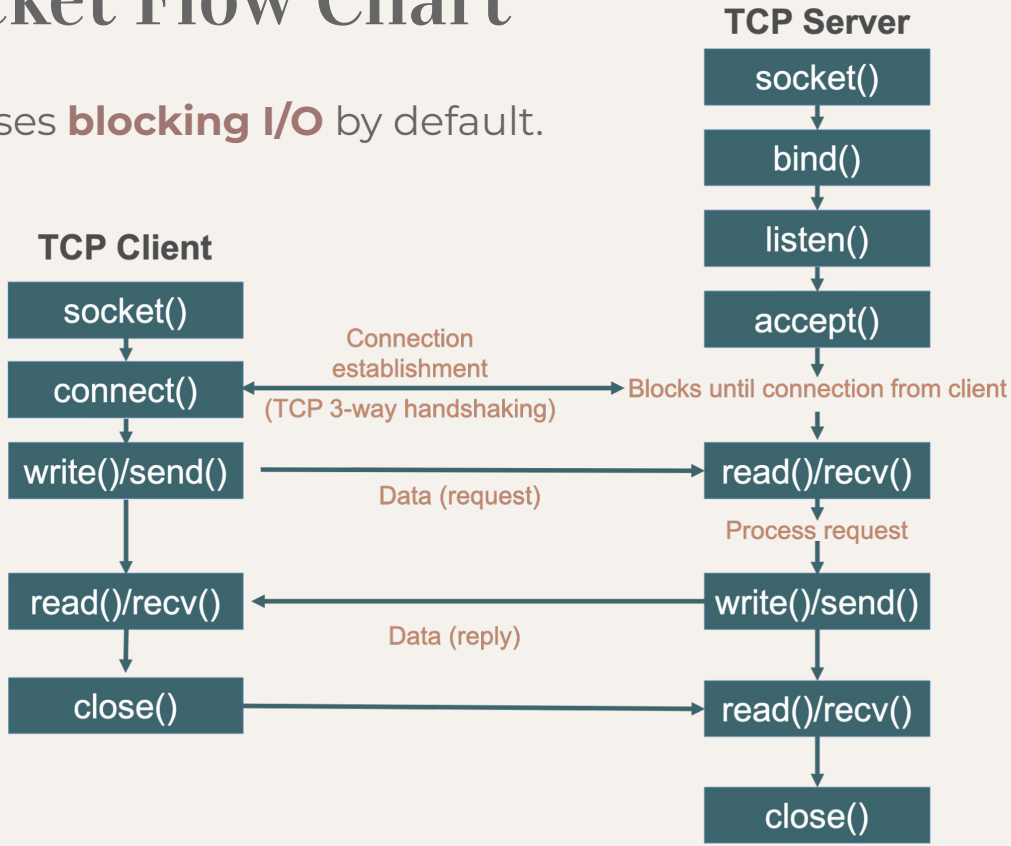
Interger vaule	Name	<unistd.h> symbolic constant	<stdio.h> file stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

TCP Service

- **TCP (Transmission Control Protocol)**
  - **Connection-oriented**
  - **Reliable transport**
  - Flow control
  - Congestion control
- What is Socket-Address?
  - **IP address + Port number**
  - IP address: To find out the machine (Network Layer)
  - Port number: To find out the process (Transport Layer)

TCP Socket Flow Chart

- Socket uses **blocking I/O** by default.



socket()

- **Create** the endpoint for connection.

```
#include <sys/socket.h>
int socket (int domain, int type, int protocol);
```

- domain
  - AF_UNIX/AF_LOCAL: communication between 2 processes on a host. So they can share a file system.
  - AF_INET, AF_INET6: communication between processes on different hosts through the Internet. AF_INET is for **IPv4**, whereas AF_INET6 is for **IPv6**.

socket()

- **Create** the endpoint for connection.

```
#include <sys/socket.h>
int socket (int domain, int type, int protocol);
```

- type
  - SOCK_STREAM: sequential and connection-oriented (TCP)
  - SOCK_DGRAM: datagram (UDP)
- protocol: defined in /etc/protocols, usually set to 0
- return: file descriptor (int)

bind()

- **Bind** the address to the socket.

```
#include <sys/socket.h>
int bind (int sockfd, struct sockaddr *addr, socklen_t len);
```

- sockfd: specifies the socket file descriptor to bind.
- addr
  - specifies the socket address to be associated with the sockfd
  - You can use “struct sockaddr_in*” defined in <netinet/in.h>, and then cast it into “struct sockaddr*”
- len: specifies the size of sockaddr (= **sizeof(struct sockaddr)**)

bind()

```
struct sockaddr_in {
        short sin_family;               // address family. EX:AF_INET
        unsigned short sin_port;        // port number for network
        struct in_addr sin_addr;        // IP address for network
        unsigned char sin_zero[8];      // pad to sizeof(struct sockaddr)
}
```

listen()

- Specify a socket to **listen** for connections.

```
#include <sys/socket.h>
int listen (int sockfd, int backlog);      // returns 0 if it's success; -1 otherwise
```

- sockfd: specifies the socket file descriptor to listen.
- backlog: specifies the number of users allowed in queue.

accept()

- **Accept** the connection on a socket.

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- sockfd: specifies the socket being listened to.
- addr: pointer to the sockaddr. It will be filled in with **the address of the peer socket**.
- **Blocking** until a user **connect()** call is received.
- After accepting the connection, it **creates a new file descriptor** for the client. The original socket is not affected.

connect()

- **Connect** to the socket from client to server.

```
#include <sys/socket.h>
int connect (int sockfd, struct sockaddr *addr, socklen_t len);
```

- The format is the same as **bind()**.

close()

- **Close the file descriptor**.

```
#include <unistd.h>
int close (int sockfd);        // returns 0 if it's success; -1 otherwise
```

read()/recv()

- **Read** data from the socket file descriptor.

```
#include <unistd.h>
ssize_t read (int fd, void *buf, size_t count);
ssize_t recv (int fd, void *buf, size_t len, int flag);
```

- fd: specifies the socket file descriptor to read data from.
- buf: specifies the buffer to contain the received data.
- count/len: specifies the size to receive.
- flag: (**read()** has no this parameter.) It's about some details like blocking/nonblocking.

read()/recv()

- **Read** data from the socket file descriptor.

```
#include <unistd.h>
ssize_t read (int fd, void *buf, size_t count);
ssize_t recv (int fd, void *buf, size_t len, int flag);
```

- Reading data from a file may be
  - Successful, return the number of bytes received
  - EOF. (end of file) (i.e., return = 0)
  - Failed, **errno** is set to indicate the error.
- It may be blocked. (**block I/O**).

write()/send()

- **Write** data to socket file descriptor.

```
#include <unistd.h>
ssize_t write (int fd, const void *buf, size_t count);
ssize_t send (int fd, const void *buf, size_t len, int flags);
```

- fd: specifies the socket file descriptor to send data to.
- buf: specifies the buffer to contain the data to be transmitted.
- count/len: specifies the size to send.
- flag: (write() has no this parameter.) It's about some details.

write()/send()

- **Write** data to socket file descriptor.

```
#include <unistd.h>
ssize_t write (int fd, const void *buf, size_t count);
ssize_t send (int fd, const void *buf, size_t len, int flags);
```

- Writing data to file may be
  - Successful, return the number of bytes written.
  - Failed, **errno** is set to indicate the error.
- It may be blocked. (**block I/O**)

Useful Functions

- Address and port numbers are stored as integers.
  - Different machines implement **different endian**.
  - They may communicate with each other on the network.
- Converting IP address and port number.
  - `htonl()`: for IP address (host -> network)
  - `ntohl()`: for IP address (network -> host)
  - `htons()`: for port number (host -> network)
  - `ntohs()`: for port number (network -> host)

Useful Functions

- An IP address is usually hard to remember.
  - We need to **translate the hostname to IP address**.
- Translate a hostname to IP address.

```
#include <netdb.h>
struct hostent *gethostbyname (const char *name);
```

select()

- **select()** provides you to supervise multiple sockets, telling you which is able to read or write, etc.
- With **select()**, it is possible to achieve Asynchronous Blocking I/O.
- If you want to implement this assignment with **select()**, please refer to this website.

select()

- **Monitor** whether there is at least one fd available.

```
#include <unistd.h>
int select (int nfds, fd_set*, readfds, fd_set* writefds, fd_set* exceptfds, struct timeval* timeout);
```

- nfds: specifies the number of file descriptors to monitor.
- readfds: specifies the pointer to read file descriptor list.
- writefds: specifies the pointer to write file descriptor list.
- exceptfds: specifies the pointer to the error file descriptor list.
- timeout: deadline for **select()**.

select()

```
void FD_SET (int fd, fd_set *set);
void FD_CLR (int fd, fd_set *set);
int FD_ISSET (int fd, fd_set *set);  // return: 1 if it's available, else: 0
void FD_ZERO (fd_set *set);
```

- FD_SET(): Add the file descriptor into the set.
- FD_CLR(): Remove the file descriptor from the set.
- FD_ISSET(): Check if the file descriptor is available.
- FD_ZERO(): Clear the set.

poll()

- **Monitor** whether there is at least one fd available.

```
#include <unistd.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- fds: specifies the pointer to file descriptor list.
- nfds: specifies the number of file descriptors to monitor.
- timeout: deadline for **poll()**.
- **select()** can monitor only file descriptors numbers that are less than FD_SETSIZE (1024)—an unreasonably low limit for many modern applications. All modern applications should instead use **poll()** or **epoll()**.

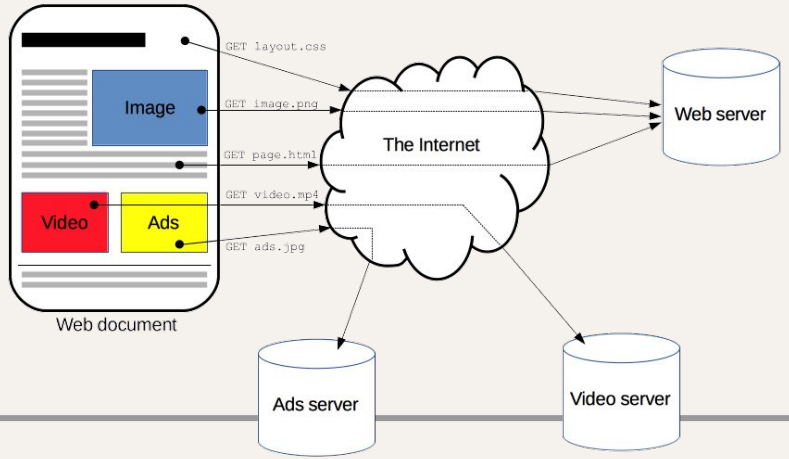
Reference

- Beej's Guide to Network Programming (中文)
- Beej's Guide to Network Programming (English)
- Linux manual page

HyperText Transfer Protocol Tutorial

What is HTTP?

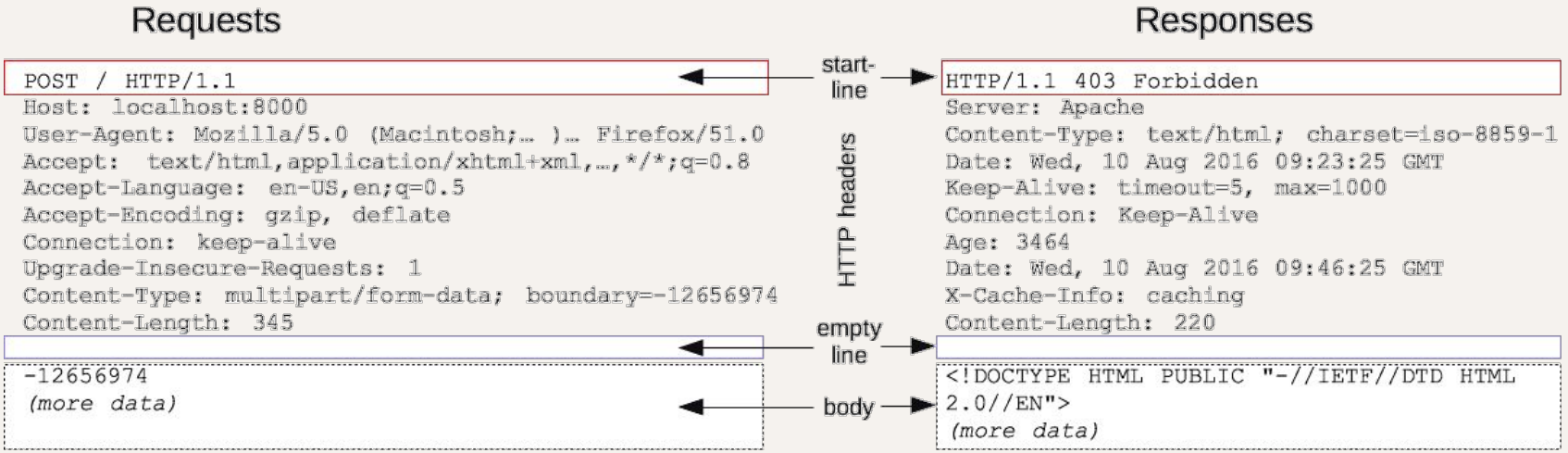
- **HTTP** is an **stateless** application layer protocol using TCP (before HTTP/3).
- A request–response protocol in the client–server model.
- HTTP is designed for sharing multimedia resources.
- Default TCP port is 80. (https: 443)



Message Format

- HTTP messages are how data is exchanged between a server and a client. There are two types of messages:
  - **Request:** sent by the client to trigger an action on the server
  - **Response:** the answer from the server
- HTTP requests and responses, share similar structure and are composed of **start-line**, **headers**, **blank line**, **body**(optional)
- **Newline**
  - In the HTTP 1.1 protocol, information is separated using the CRLF (**\r\n**), not a single LF (**\n**).

Message Format



- HTTP requests and responses, share similar structure and are composed of **start-line**, **headers**, **blank line**, **body**(optional)

Try It Yourself: HTTP Requests

- You can run the command below to send an http request to example.com

```
$ echo -en "GET / HTTP/1.1\r\nHost: example.com\r\nConnection: Close\r\n\r\n" | nc example.com 80
```

- It will send something like:

```
GET / HTTP/1.1\r\n
Host: example.com\r\n
Connection: Close\r\n
\r\n
```

- **Exercise:** Modify client.c to do the same thing above.

Try It Yourself: HTTP Responses

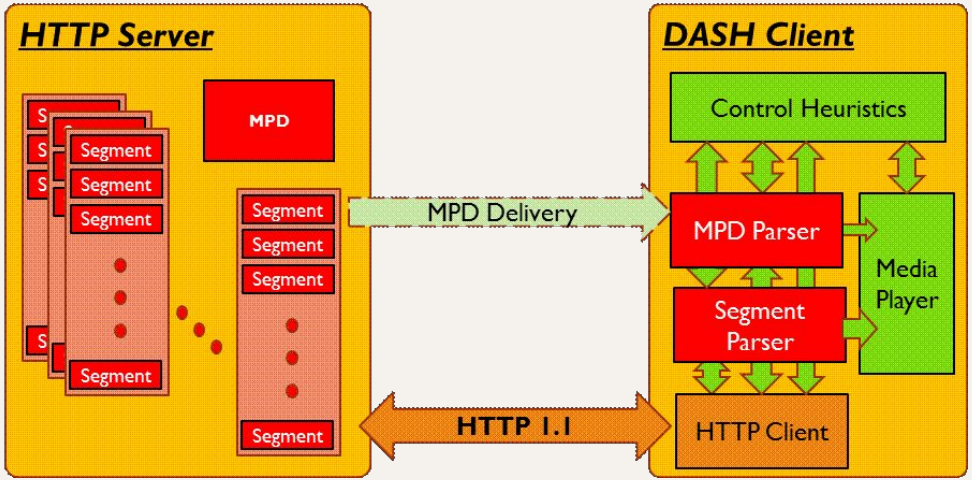
- Modify the code in the example code, **server.c**, setting PORT to 8080, and the message:

```
char *message = "HTTP/1.1 200 OK\r\n"
"Content-Length: 21\r\n"
"Content-Type: text/html\r\n\r\n"
"<h1>Hello World!</h1>";
```

- **Exercise:** How can we keep the server handle new connections?

MPEG-DASH

- Dynamic Adaptive Streaming over **HTTP**
- First adaptive bit-rate HTTP-based streaming solution
- Enable high quality streaming of media content over the Internet delivered from conventional HTTP web servers



Assignment 2 Announcement

Environment Setup

Docker

- We provide a docker config (docker-compose.yml) for you to run our example code. If you use Windows, you will need to install **Windows Subsystem Linux (WSL 2)** first.
- **Please make sure you can compile and run your code well in the provided docker container.**

Docker Installation

- Windows
  - Install Windows Subsystem Linux (WSL): Guide
  - Install Docker Desktop
- Ubuntu
  - Install Docker through your terminal:

    # apt update

    # apt install docker.io
- macOS
  - Install Docker Desktop

Start Your Container

- Clone repository to your host and run the container:

```
$ git clone <your_repository>
$ docker-compose up -d
$ ssh cn@localhost -p 2222
```

- Port 8080/TCP is exposed on the host, allowing you to access your server from outside.

Your Tasks

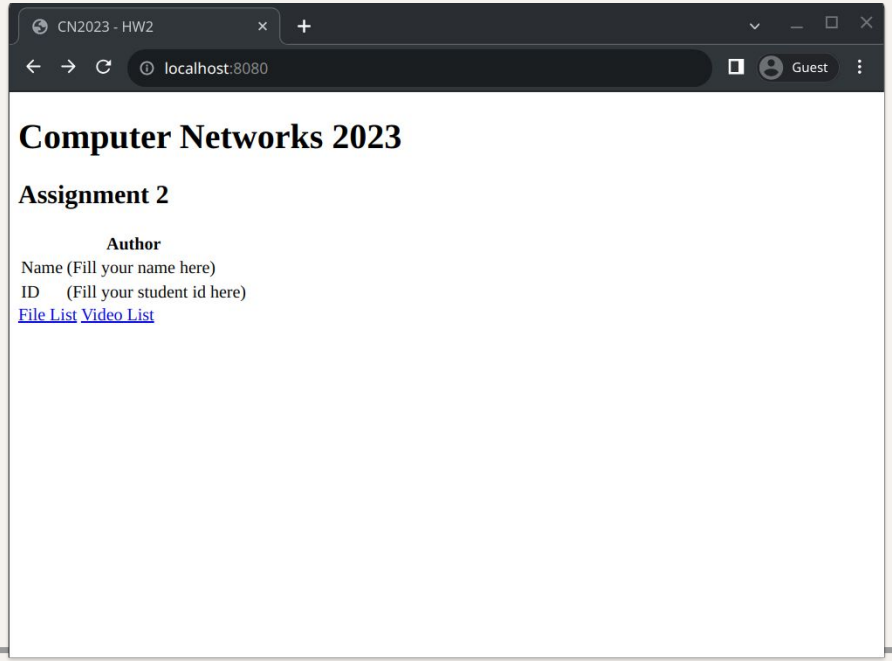
- In this assignment, you need to implement a simple HTTP server/client with the following functions:
  - Server can serve static web page **index.html**, **uploadf.html, uploadv.html, etc.**
  - Server can serve dynamic web pages **listf.rhtml**, **listv.rhtml**, **player.rhtml, etc.**
  - Server can serve text/binary files
  - Client can upload files and videos to server
  - Client can fetch files from server (including DASH videos)
- Your server should be compatible with modern browser.
  (at least supporting browsers based on Chromium)

Routes

- We provide templates for these endpoints:
  - index.html
    - Your name and id
  - uploadf.html, uploadv.html
    - Upload page for files/videos
  - listf.rhtml, listv.rhtml
    - List all the videos/files on the server
  - player.rhtml
    - Video player

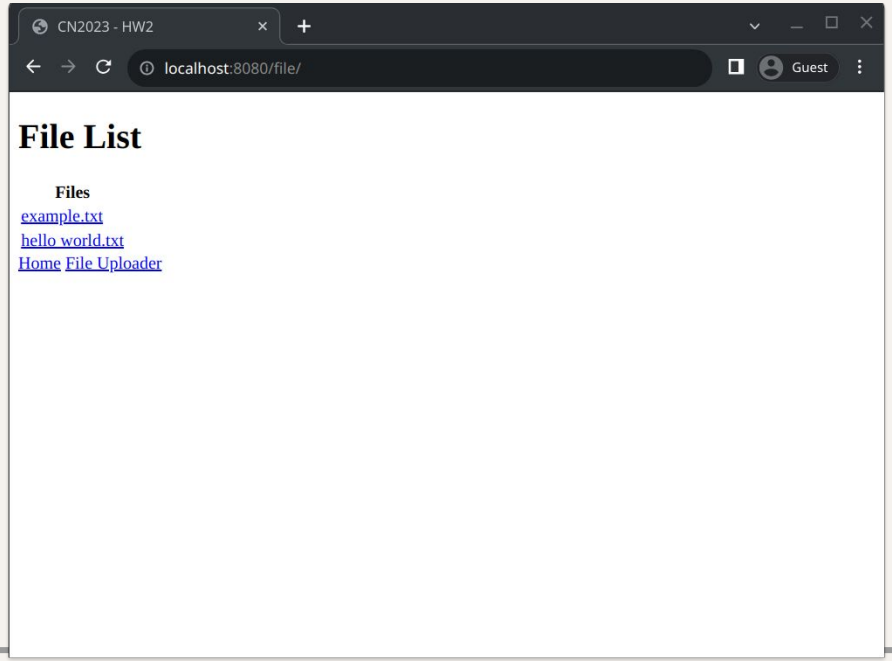
Routes: /

- Using the provided template **index.html**. Fill your name and id in table.



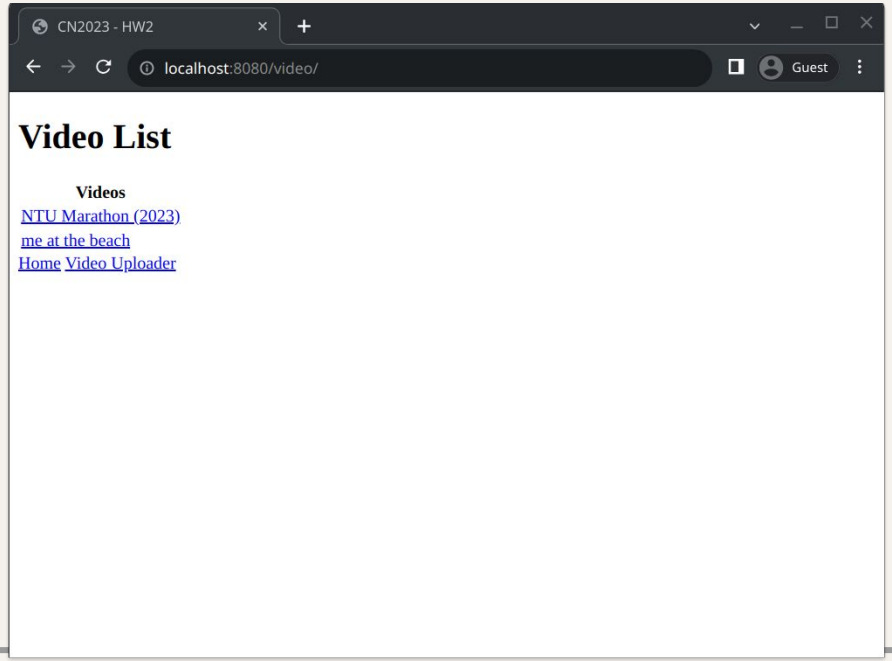
Routes: /file/

- Using the provided template **listf.rhtml**. Shows all the files that the server has.



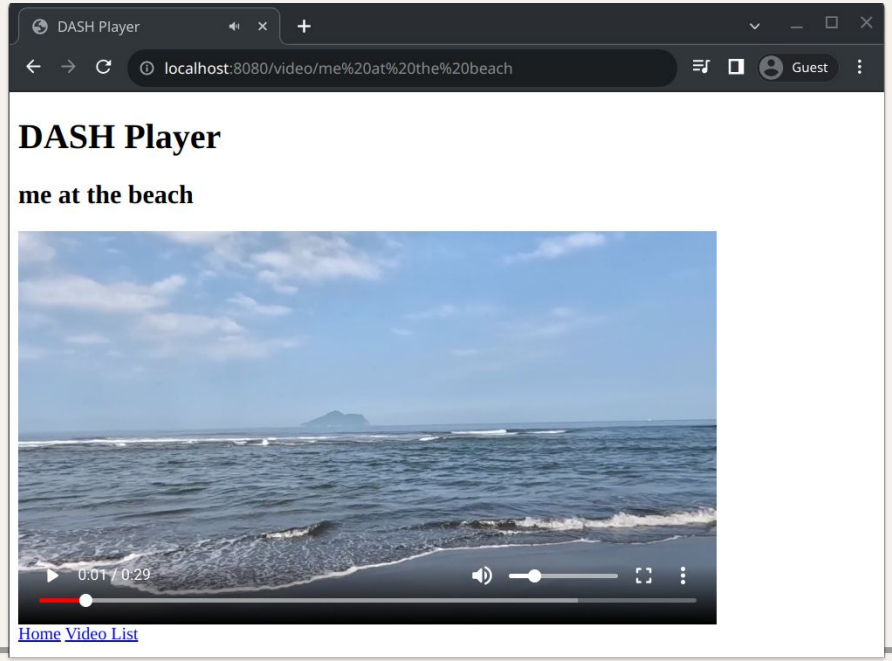
Routes: /video/

- Using the provided template **listv.rhtml**. Shows all the videos that the server has.



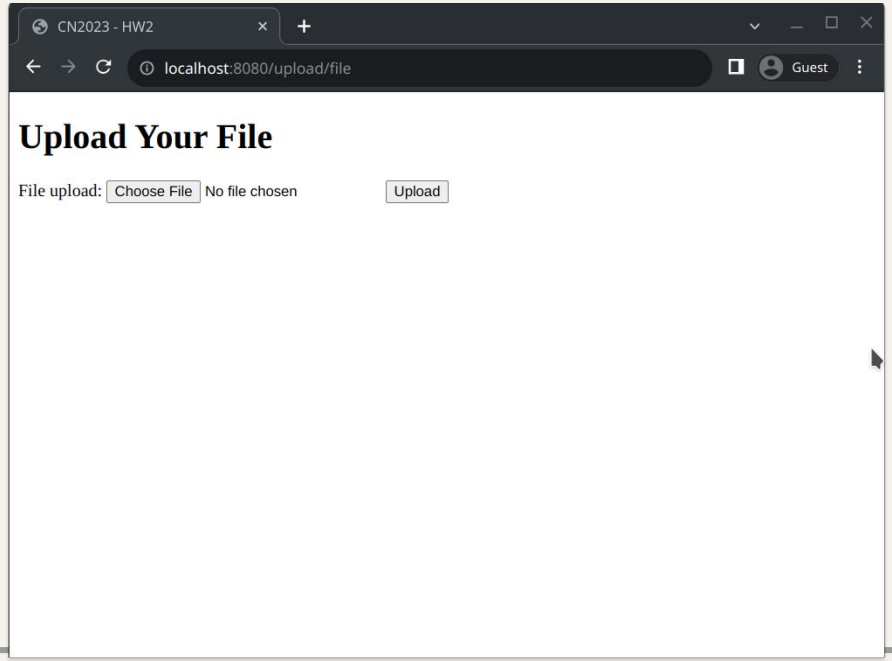
Routes: /video/{videoname}

- Using the provided template **player.rhtml** to play videos.



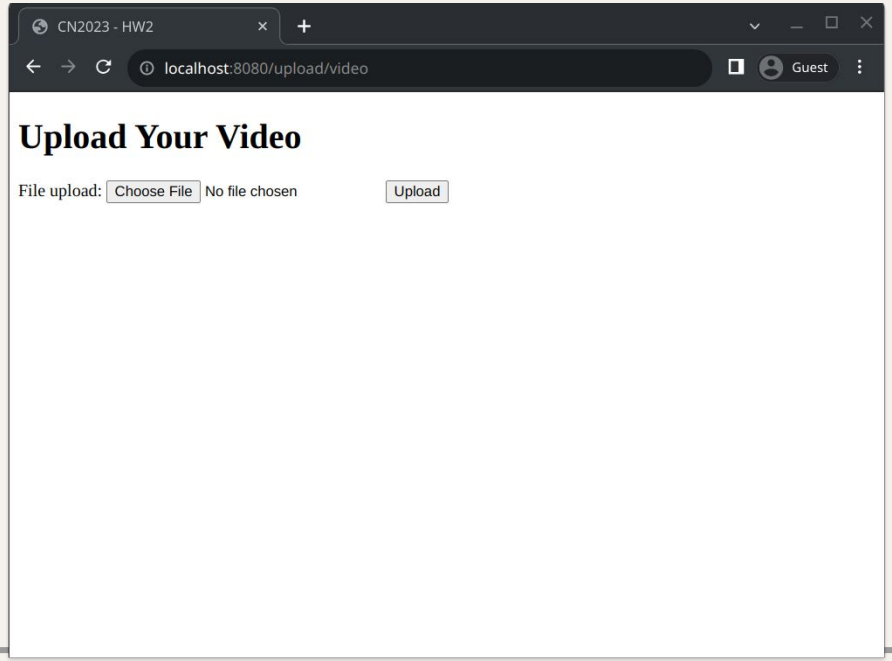
Routes: /upload/file & /api/file

- Upload files to the server.



Routes: /upload/video & /api/video

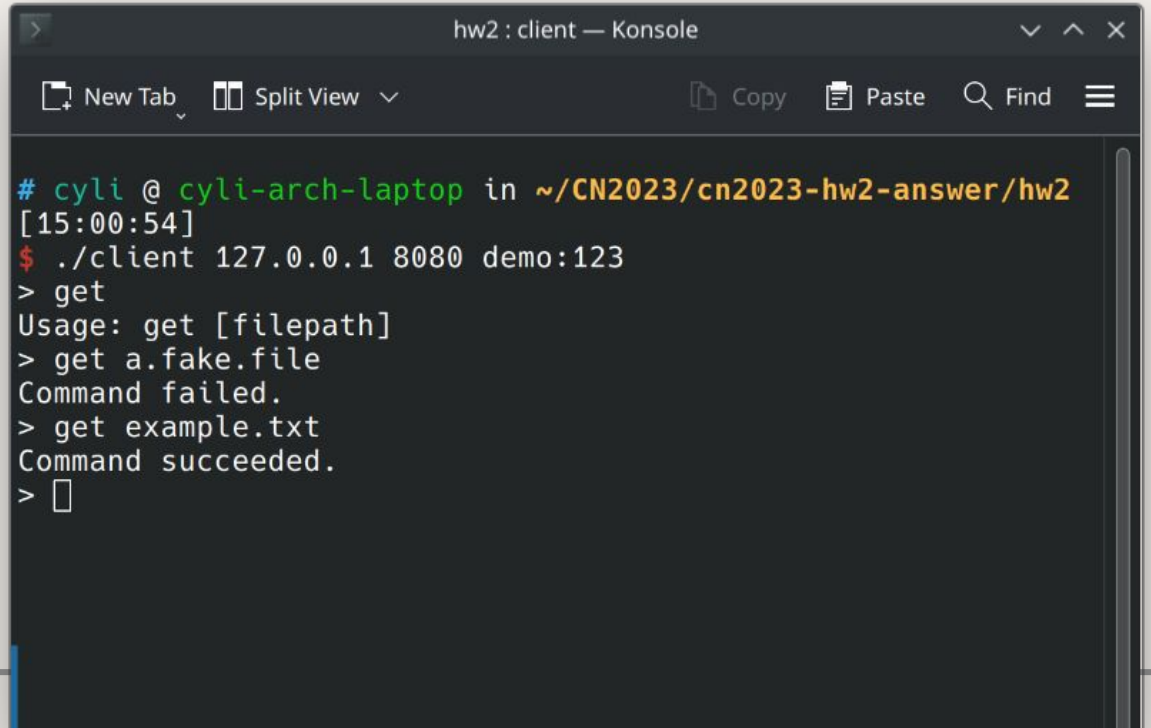
- Upload videos to the server.



Server Demo Time!

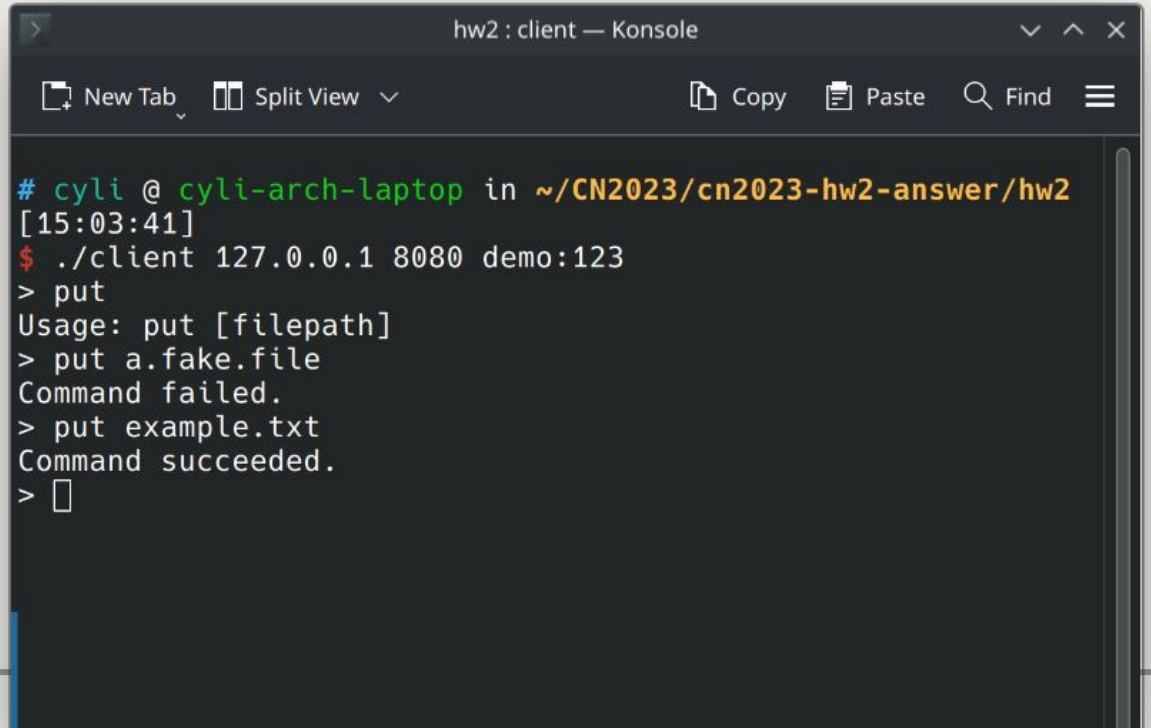
Commands: get {filepath}

- Download files from the server.



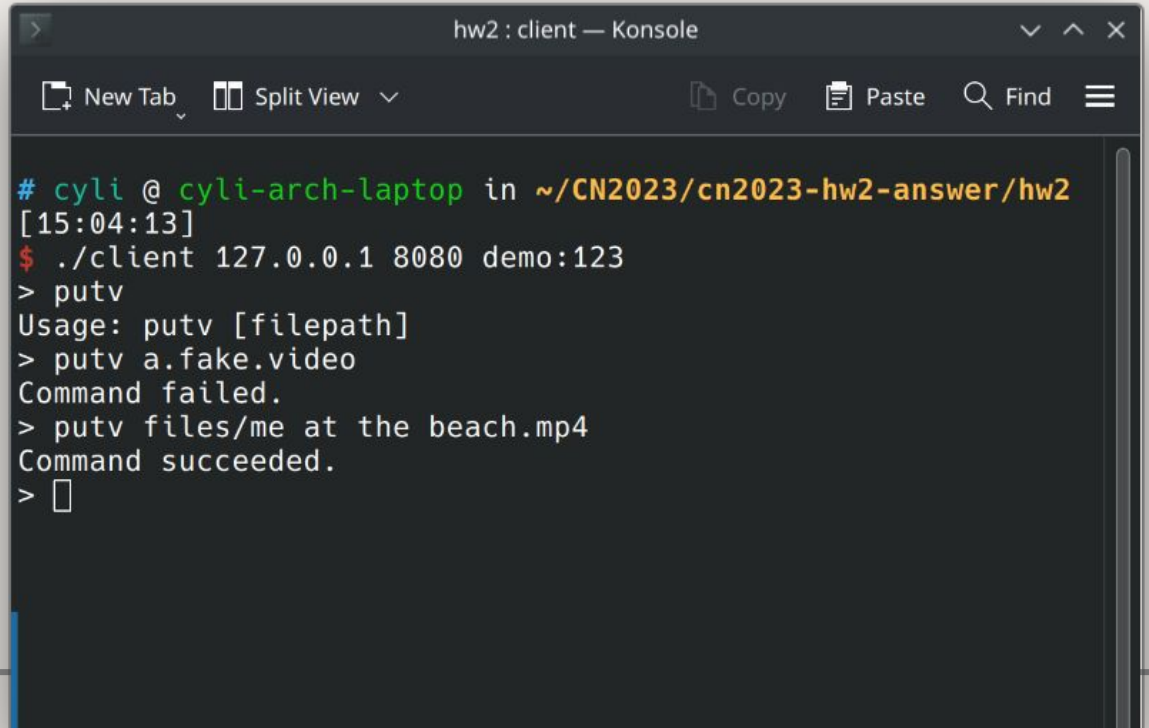
Commands: put {filepath}

- Upload files to the server.



Commands: putv {filepath}

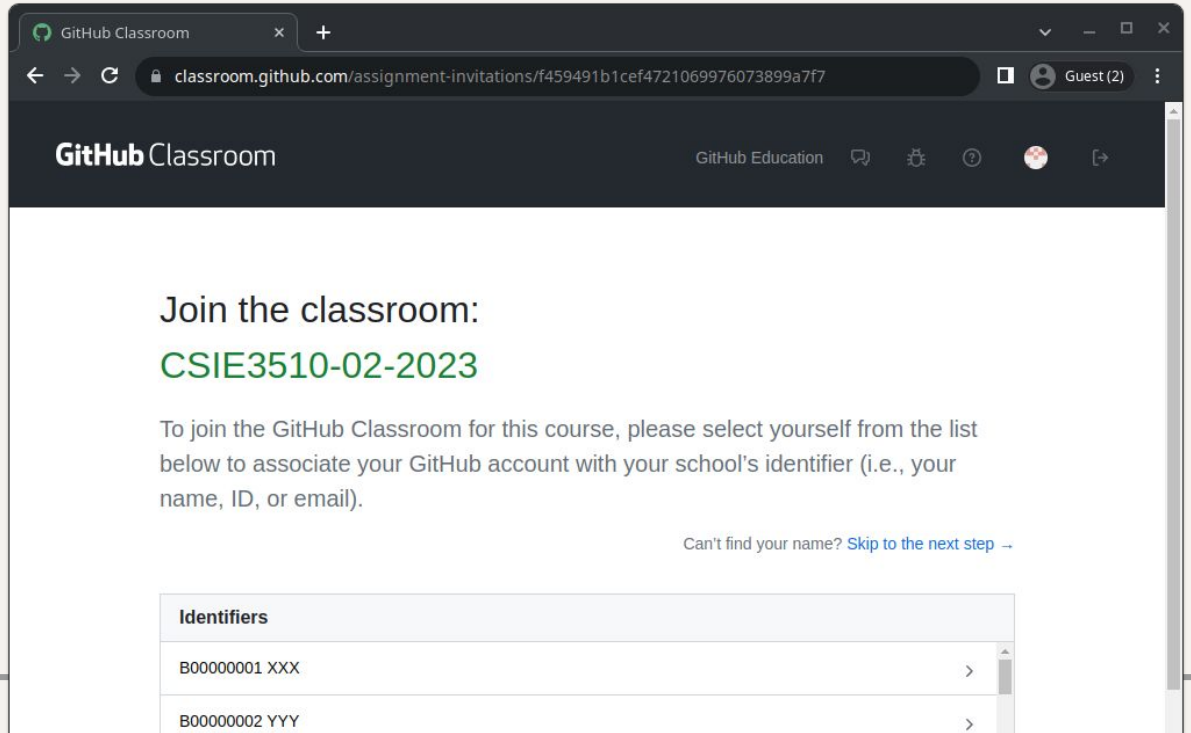
- Upload videos to the server.



Client Demo Time!

Github Classroom

- Get the access of assignment materials via **Github Classroom**.



Bonus

- To better track everyone's progress and improve this course, we encourage you to commit your codes to GitHub Classroom regularly after completing some parts of this assignment
- Make sure your commit messages are meaningful.
- You can decide whether to do the bonus part or not. If you want, please regularly (but not intensively) commit your codes to GitHub Classroom and answer the following questions in your report
    - How did you utilize Git for development in this assignment?
    - What benefits did it bring?
    - Is it a better way for you to submit homework via GitHub Classroom than via traditional ways (e.g, submit a .zip file to NTU Cool)? Why or Why not?
- Once you join this part, we will give you a bonus of 5 points for Assignment 2

Submission

- Report
  - Your report should be a **pdf** file. Submit it to **Gradescope**.
  - PDF file name: <studentID>_hw2.pdf
    - e.g., B10902999_hw2.pdf
- Codes
  - Please push all the source code (i.e., without your report, the video file, and the execution file) to github classroom assigment.
- The penalty for the wrong format is **10 points**.
- **No plagiarism is allowed. A plagiarist will be graded zero.**

Submission

- Deadline
  - Due Date : 23:59:00, November 8th, 2023
  - The penalty for late submission is **20 points per day**.

If You have any Problems…

- You can
  - Ask questions on NTU COOL Discussion Forum
  - Send a mail to TA with the tag **[HW2]** in the title
  - Ask questions in TA hours in R438 by appointment. Google Sheet Link.
- TA Email: ntu.cnta@gmail.com

Happy coding! ●ω●)ค

TA Email: ntu.cnta@gmail.com