Peer-to-Peer Music Sharing System

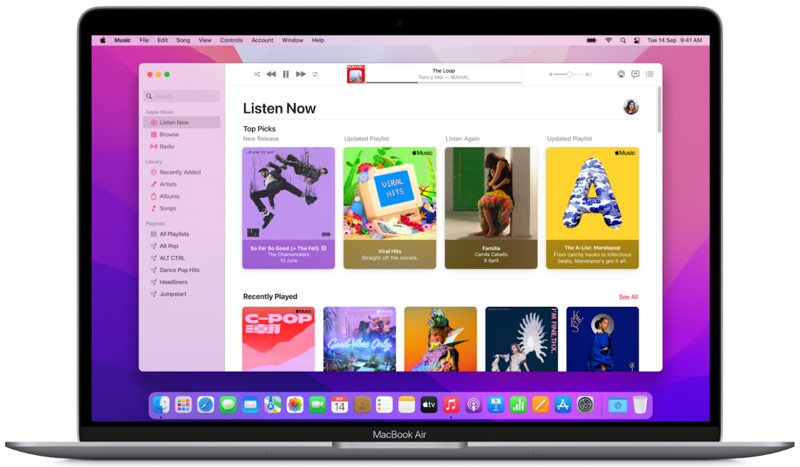
Overview

In this project, you are required to achieve **two** main goals.

- ***(Phase I)*** Firstly, you are required to implement a graphical music player to play WAV audio files, control the playback, display music lyrics, and manage a music library.
- ***(Phase II)*** Secondly, you are required to build a Peer-to-Peer (P2P) system for playing music (in the form of streaming) from remote computers. Each computer works as both client and server, which means you may get the audio data from other computers and share audio data for others to be downloaded.

Evaluation Metrics	***Score (100% in total)***
Basic Requirements: Phase I	30%
Basic Requirements: Phase II	30%
Enhanced Features (Phase I & II)	25%
Demonstration & Report	15%

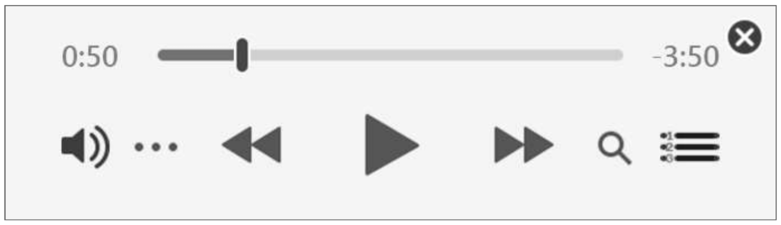
Basic Requirements



Apple Music, https://www.apple.com/hk/en/itunes, is a well-designed music player.

Basic User Interface

Your program shall have a basic user interface. The interface should at least include a play/stop button and a list control of audio files. The user can select the music in the list control to play the audio file. You also need to provide an interface for users to edit the information of audio files.



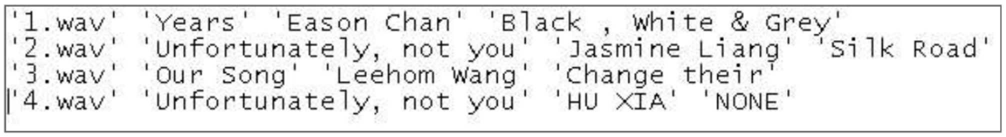
A demonstration of playback controls. You can add any function as you wish, but the controls should contain at least a play/stop (or a play/pause) button.

Music Decoding and Playback

You are required to understand the inside structure of wave format and write your own codes that can open, analyze, and playback a WAV file. ***The fmt sub-chunk and the data sub-chunk of the WAV file must be read and extracted by yourselves, which means you cannot use any third-party libraries/programs***. The sound data should be played fluently and bring the users beautiful music.

Music Management

Your program should have a database that stores music information (e.g., album, title, length) such that the program can detect the music files in the database and then display them in the playlist. Your program should also have a database (or a text file) to store the information of the audio files. The information of the audio files should be manually input and removed by the user or automatically generated.

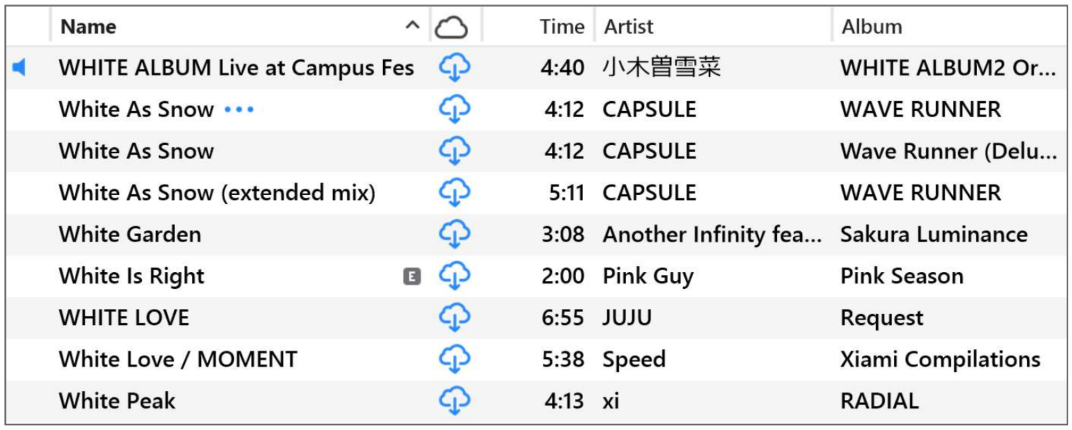


```
'1.wav' 'Years' 'Eason Chan' 'Black , White & Grey'
'2.wav' 'Unfortunately, not you' 'Jasmine Liang' 'Silk Road'
'3.wav' 'Our Song' 'Leehom Wang' 'Change their'
'4.wav' 'Unfortunately, not you' 'HU XIA' 'NONE'
```

A demonstration of music info management using a simple .TXT file.

Information Display

Your program shall be able to display the information of the music, including the music title, singer, and album name according to your database. The corresponding information should be displayed when the user plays a song in the list control. The program should display "None" in certain places if certain information is unavailable.



Name	Time	Artist	Album
WHITE ALBUM Live at Campus Fes	4:40	小木曽雪菜	WHITE ALBUM2 Or...
White As Snow •••	4:12	CAPSULE	WAVE RUNNER
White As Snow	4:12	CAPSULE	Wave Runner (Delu...
White As Snow (extended mix)	5:11	CAPSULE	WAVE RUNNER
White Garden	3:08	Another Infinity fea...	Sakura Luminance
White Is Right E	2:00	Pink Guy	Pink Season
WHITE LOVE	6:55	JUJU	Request
White Love / MOMENT	5:38	Speed	Xiami Compilations
White Peak	4:13	xi	RADIAL

A demonstration of the music info display interface.

Music Searching

Users can type in keywords to search music based on your database. Your program can search the music from the music database according to the keywords. The results should be displayed in the list control of your program. What's more, the user could search from any properties of the music, including music title, singer, and the album as the keywords.

Lyrics Display

Your program should be able to play music and show lyrics. The lyrics file can be simple text files or in LRC format. The location of the lyric file can be maintained by the database, or simply place the file in the same folder as the music file with similar names.

Sample Enhanced Features

You are welcome to come up with more creative ideas to enhance your project!

Support for other music formats

You can use any 3rd-party libs/programs or implement them by yourself to extend your music player to support other audio formats, such as mp3, aac, ogg, and others.

Progress bar

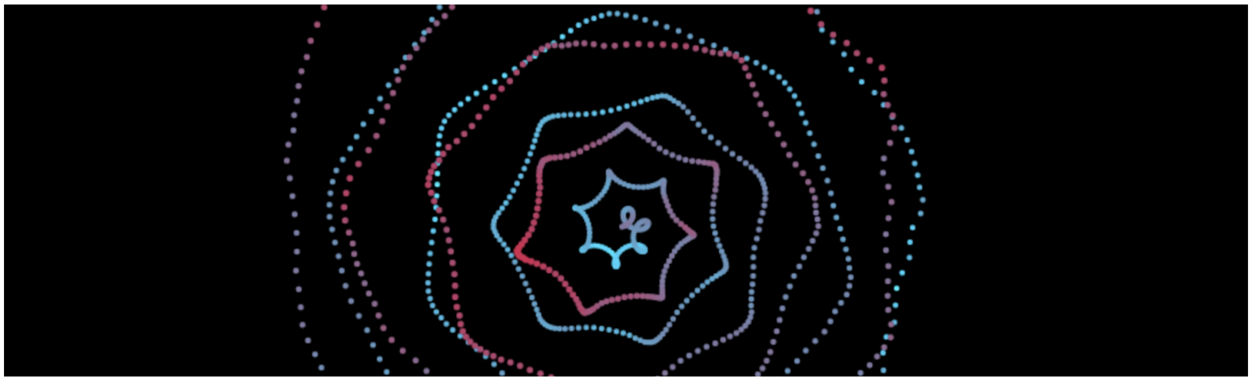
You may implement a progress bar on the GUI to support fast seeking for playback.

Synchronized lyric display

LRC format contains lyrics with the addition of timing information. You may play the music and show the lyrics synchronously. You should download a .LRC file of your favorite song from the internet or type the lyrics text without time-info, and then edit it by yourself.

Visualization

You may create beautiful music visualization effects based on the music. For example, display music spectrum based on Fourier analysis. You may grab some inspiration from here: https://github.com/willianjusten/awesome-audio-visualization



Source: https://soniaboller.github.io/audible-visuals/

Basic Requirements

Network connection

Your program should be able to connect to other PCs using TCP/IP network stack. You can use any method to get the IP address of the connectable PCs except hard coding. (For example, manually inputting IP addresses, using a tracker server, or broadcasting are appropriate). ***The network should support at least three terminals.***

Music Searching (Online)

Besides local music searching, network searching should be supported in this phase. The search interface should be the same one as those in phase 1. What's more, your program should search the audio files not only in the local database but also in the database of the other PC connected to your program. ***All the results from the local and network databases should be displayed in the same list control of the UI***. Identical results (e.g., different computers have the same audio file) must be displayed only once.

Availability Check

As the user may not know where the audio is, the program should check whether the audio file exists locally when the user selects audio from the search results. If the audio file exists, your program shall playback the audio directly. Otherwise, your program will stream the audio file from other computers.

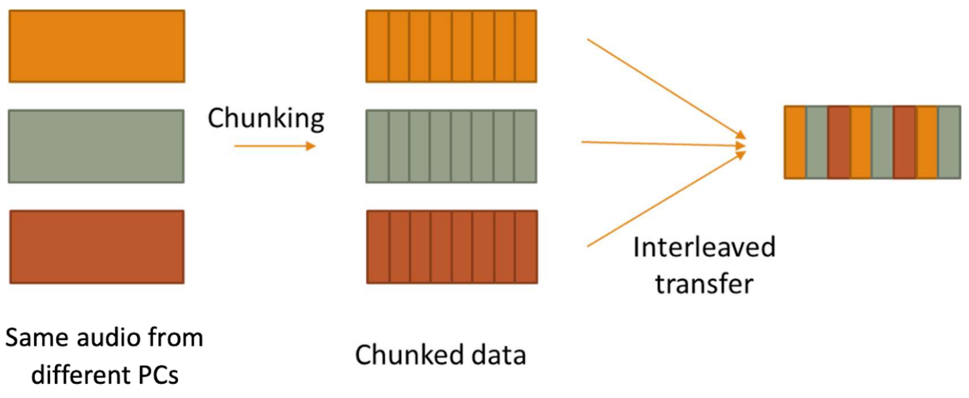
Real-Time Audio Streaming

When the program is streaming from other computers, your program should automatically play the audio as soon as possible when it receives a piece of audio data (after a certain amount of buffering). ***You are only allowed to buffer no more than 50% of a music file before you play the music***.

Peer-to-Peer Playing

Your program should be able to receive one audio file from at least two other computers simultaneously. ***The audio data from different computers should be played in an interleaving way.*** For example, suppose PC1 wants to play a file, and it cannot be found locally. It should get audio data from PC2 and PC3, the file is divided into (at least) 4 parts, and PC1 may get the first part from PC2, the second part from PC3, the third part from PC2, and the fourth part from PC3.

To verify the interleaving feature, you should also implement a function in your project to show your interleaving feature using images. We will give you three images in different colors but have the same file name in bitmap format with the same resolution. Let's say PC1 wants to download this image from other endpoints in an interleaving way; you must achieve that your data are completed but collected from different endpoints. The bottom figure shows an example of an interleaving feature.



Sample Enhanced Features

1. Support streaming other audio formats
2. Support more than three clients
3. Support more than two sources
4. Do any other your own ideas to enhance the system

Demonstration and Report

For this project, you need to do a demonstration of your program and submit a hard copy of the project report in the demonstration.

The report is a brief description, up to 6 pages, to describe your program. You must write down your team number, team member's name and student ID, workload division, and program's operation manual (README). You must also state which third-party libraries have been used in your program and what enhanced features you have implemented.

You need to do a demonstration in front of a tutor. In the demonstration, you should introduce every basic requirement you have fulfilled and every enhanced feature you have implemented clearly and efficiently. Please tell tutors if you have any special requirements on the library, tools, or resources. What's more, you will only have 12-min to demonstrate your program, and a mark will be deducted if you do your demonstration for more than 12 mins.

The following requirements must also be followed:

Before the demonstration:

- You can use any machines, including your PC, in the demonstration venue. Please be well prepared before you do the demonstration (e.g., setting up your environment, downloading the necessary resources, and preparing your own audio files other than .wav format).
- The demonstration starts when you run your program.

During the demonstration:

- You need to demonstrate all your program features.
- Tutors may ask you questions about your program, and your answers will affect your grades.
- You are not allowed to close/restart your program without permission.
- Unstable performance (e.g., No responses or unexpected results) may lead to mark deduction.

Technical Handout - Will be explained in tutorials in detail

Network Connections with Socket + TCP/UDP

Protocol (TCP/UDP)

- TCP/IP (Transmission Control Protocol/Internet Protocol) is the set of protocols that governs communication over the internet. It is a standardized communication protocol used for transferring data between different computers and networks.
- An IP address is a unique identifier assigned to each device on a network that uses the Internet Protocol for communication. It consists of four numbers separated by dots, such as 192.168.0.1. IP addresses can be either static (permanently assigned to a device) or dynamic (assigned by a DHCP server).

What is a port?

- A port is a communication endpoint used in computer networking. Ports are identified by a number and are used to differentiate between different network services running on the same device.

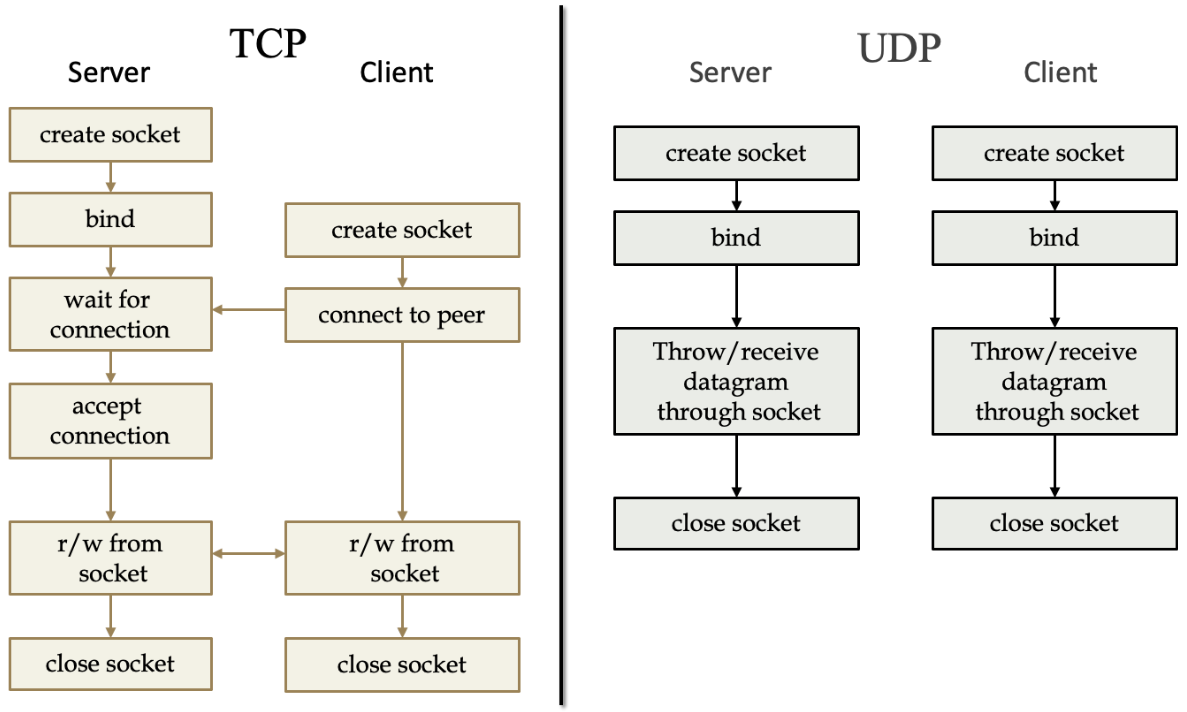
Socket:

- Socket is the endpoint of a communication channel, A network programming interface, abstracting away underlying mechanism
- On the Windows platform, we use WinSock (https://learn.microsoft.com/en-us/windows/win32/winsock/using-winsock)
- Two types of sockets (for two different transport layer protocols): SOCK_STREAM (TCP) / SOCK_DGRAM (UDP)

Some useful functions for Sockets

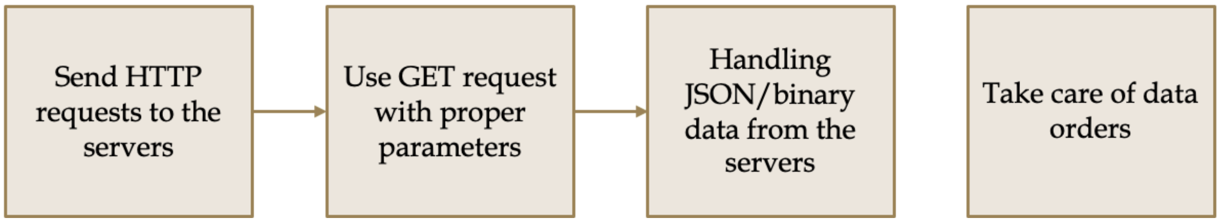
```
// Start-up call - initialize the underlying Windows Sockets DLL
int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData );

// Socket creation - create an endpoint for communication, return a socket.
SOCKET socket(int addr_family, int type, int protocol);

// Socket binding - bind a SOCKET descriptor to a local port and local IP address
int bind(SOCKET socket, const struct sockaddr *address, int address_len);
```

```
// Let the socket to wait for connection requests (stream socket, server-sided)
int listen(SOCKET s, int backlog);

// Try to connect to the server (stream socket, client-sided)
int connect(SOCKET s, const struct sockaddr *remote_addr, int address_len);
// Accepting connection (stream socket, server-sided), after accepting the original
socket, s, remains in "listen" state.
SOCKET accept (SOCKET s, struct sockaddr *addr, int *len);

// Receive or send data with socket
// recvfrom()/sendto(): please refer to the official doc.
int recv(SOCKET s, char *buf, int len, int flags);
int WSAAPI send(SOCKET s, const char *buf, int len, int flags);

// Close a socket - kill the connection
int closesocket(SOCKET s);

// Cleanup - terminates the use of the Windows Sockets DLL
int WSACleanup();

// Error checking - to get the error code after a failed call, the meaning of the
code can be checked in the header file
int WSAGetLastError();
```



The pipeline of TCP/UDP data transfer.

Network Connections with Socket + TCP/UDP


Send HTTP requests to the servers
Use GET request with proper parameters
Handling JSON/binary data from the servers
Take care of data orders

HTTP is a protocol that stands for Hypertext Transfer Protocol. It is widely used to retrieve data from servers using Uniform Resource Identifiers (URI) or URLs. HTTP requests are initiated by clients and responded to by servers. HTTP is a *symmetry network*, a type of network architecture that operates in a way that every node in the network is both a client and server. This allows for more efficient and dynamic communication between nodes.

In HTTP, we can retrieve data using URIs. Here's an example:

http://pc1/get_data?filename=somnus.wav&start=4096&length=4096.

In this example, we request data from the "pc1" node. We are asking for a specific file named "somnus.wav". We also specify the start position and length of the data we want. Similarly, we can request data from other nodes in the network:

http://pc2/get_data?filename=somnus.wav&start=0&length=4096.

This request asks the "pc2" node to send data from the beginning of the file.
We can also check if a file exists in the network using the URL:

http://pc1/check_existence?filename=somnus.wav.

Depending on the request, the server will respond with the appropriate data fragments or status.

- To handle these requests, we will need to implement routing on our server. This means that we will have to define functions to handle different types of requests, such as reading and transferring file fragments or checking the existence of a file.
- When processing these requests, we will also need to extract HTTP GET parameters. In our examples, these are the parameters passed through the URI, such as the filename, start position, and length.

- Once we have processed the request, we will need to respond to the client. For text data, we can use plain text or JSON formats. For binary data, such as audio files, we will need to use the application/octet-stream content type in our header.
- To implement our HTTP server, we can use libraries such as libcurl or Pistache in C++, or Flask in Python.

P2P Client and Multi-Threads.

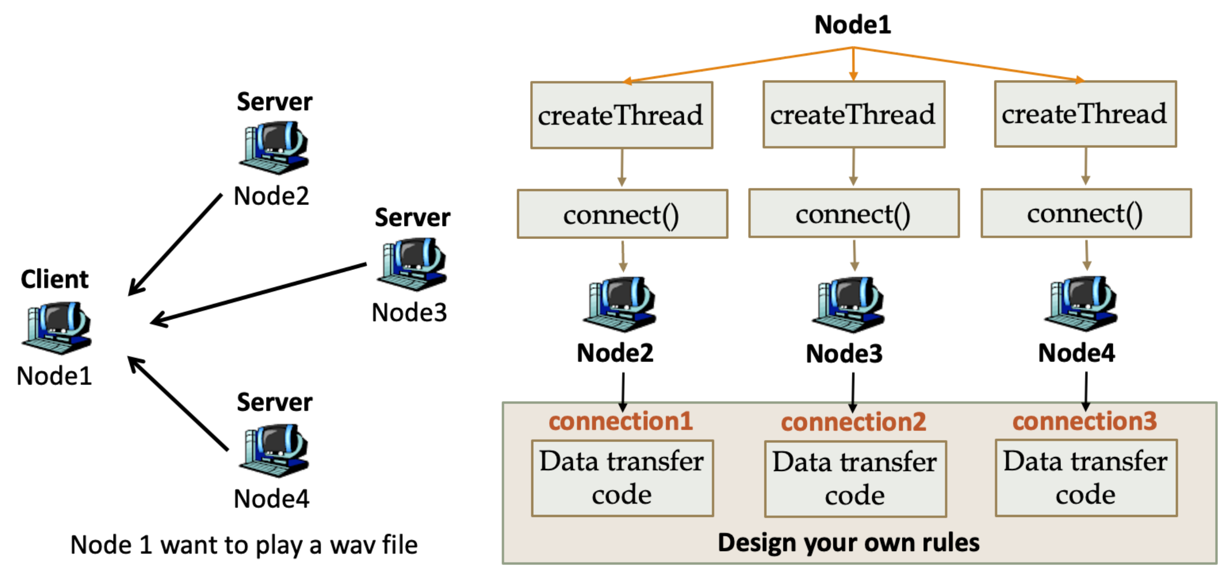
Why do we need Multi-threads? In audio playback, waveOutWrite() blocks everything. We need to do other stuff during the audio playback for real-time response.

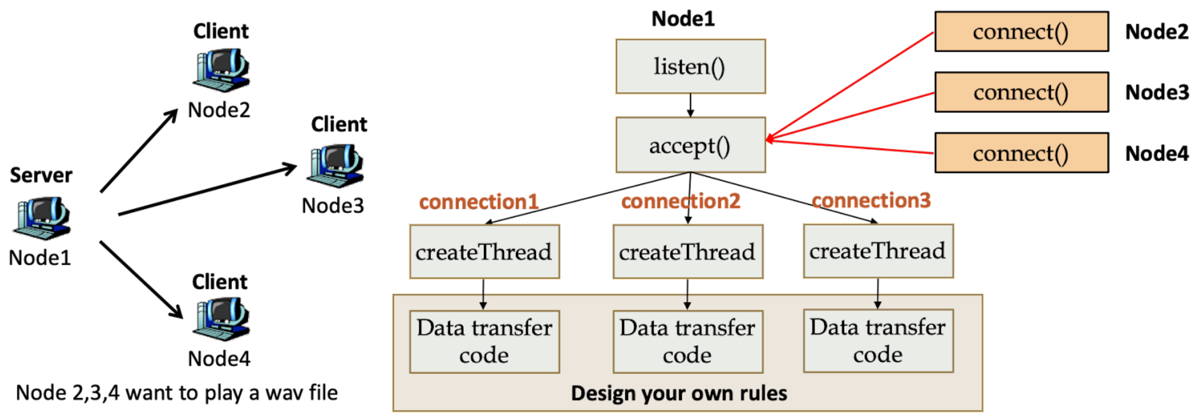
```
while (notEOF){
  loadDataIntoBuffer();       // fread/memcpy or others
  waveOutPrepareHeader(...);
  waveOutWrite(...);
  waitForSingleObject();    // block until playback finished
}
```

Also, in the socket programming, a single thread cannot accept the connection and receive data simultaneously.

```
while (1){
    client_sd = accept(...);  // blocked
    ...
}
while (1) {
    len = recv(...);  // blocked
    ...
}
```

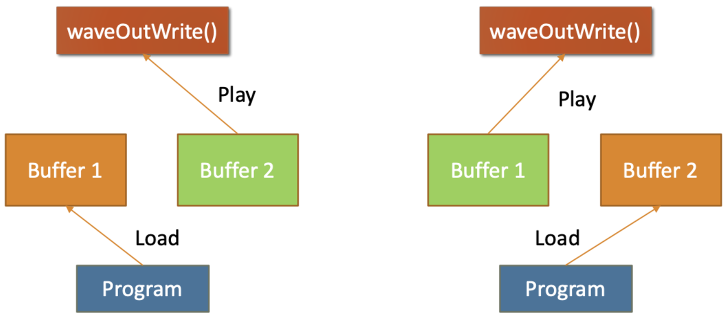
A **thread** is a sequence of such instructions within a program that can be executed independently of other code. The multi-threading design allows an application to do parallel tasks simultaneously.





The P2P process can be represented in the above two figures. When one node is requested for an audio file (***client***), it can create multiple threads to make connections for servers. Then multiple servers will send data chunks to the client for audio playback. When the single node is served as a server, it will listen to multiple connections. Each connection will create a thread and transfer data to each requesting client.

Multiple threads are also helpful for real-time streaming: we can simultaneously read and play with two buffers:



STD::Thread (https://cplusplus.com/reference/thread/thread):

- Add #include <thread> to your source file
- Creation:

```
Std::thread t(void *(*start_routine)(void *), void *arg);
```

*void *(*start_routine)(void *):* the function this thread executes
*void *arg:* arguments to pass to thread function above

- Thread type: **std::thread**
- Join threads: **join()** - Suspends the calling thread to wait for successful termination of the thread specified as the first argument pthread_t thread with an optional *value_ptr data passed from the terminating thread's call to pthread_exit().

Code Example:

```
#include <iostream>
#include <thread>

using namespace std;

void hello(const char* input) {
    cout << input << endl;
}

int main() {
    thread t(hello, "hello world");
    t.join();
    return 0;
}
```

Thread Communications:

Thread Communications are essential in any concurrent program, where multiple threads execute simultaneously.

1. Global variables

Firstly, Global variables can be used to share data between threads. Global variables are variables that are declared outside any function and can be accessed by any function in the program. Here's an example of using a global variable for thread communication:

```cpp
#include <iostream>
#include <thread>
using namespace std;

int globalVar = 0;

void threadFunction() {
    globalVar = 10;
}

int main() {
    thread t(threadFunction);
    t.join();
    cout << "Global variable value: " << globalVar << endl;
    return 0;
}
```

In this example, the `threadFunction()` updates the value of the `globalVar` variable, and the main thread prints its value after the thread has completed execution.

2. Pointers as thread arguments

Secondly, Pointers can be used as thread arguments to share data between threads. Pointers are variables that store the memory addresses of other variables. Here's an example of using pointers for thread communication:

```cpp
void threadFunction(int* ptr) {
    *ptr = 10;
}

int main() {
    int var = 0;
    thread t(threadFunction, &var);
    t.join();
    cout << "Pointer value: " << var << endl;
    return 0;
}
```

In this example, the `threadFunction()` updates the value of the `var` variable indirectly, by using a pointer to its memory location.

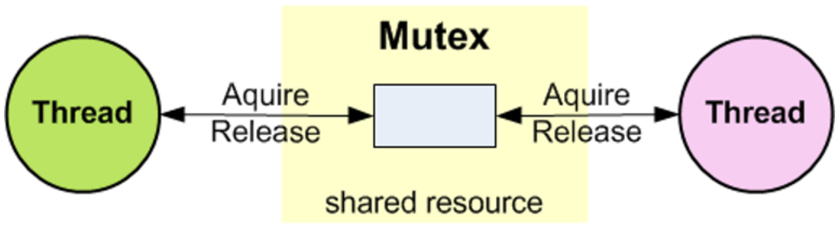
3. Take care of every WRITE operation

Thirdly, it's important to take care of every WRITE operation, since multiple threads may attempt to write to the same variable simultaneously, leading to data inconsistencies. Here's an example of a program that doesn't take care of WRITE operations:

```
int var = 0;

void threadFunction() {
    var++;
}

int main() {
    thread t1(threadFunction);
    thread t2(threadFunction);
    t1.join();
    t2.join();
    cout << "Variable value: " << var << endl;
    return 0;
}
```

In this example, the `threadFunction()` increments the value of the `var` variable. Since two threads are executing simultaneously and both are incrementing the same variable, the final value of the variable can be unpredictable.

4. Mutex: https://en.cppreference.com/w/cpp/thread/mutex



Lastly, a Mutex (short for Mutual Exclusion) can be used to protect shared resources from simultaneous access by multiple threads. A Mutex is a lock that only one thread can hold at a time, preventing other threads from accessing the resource until the lock is released. Here's an example of using a Mutex for thread communication:

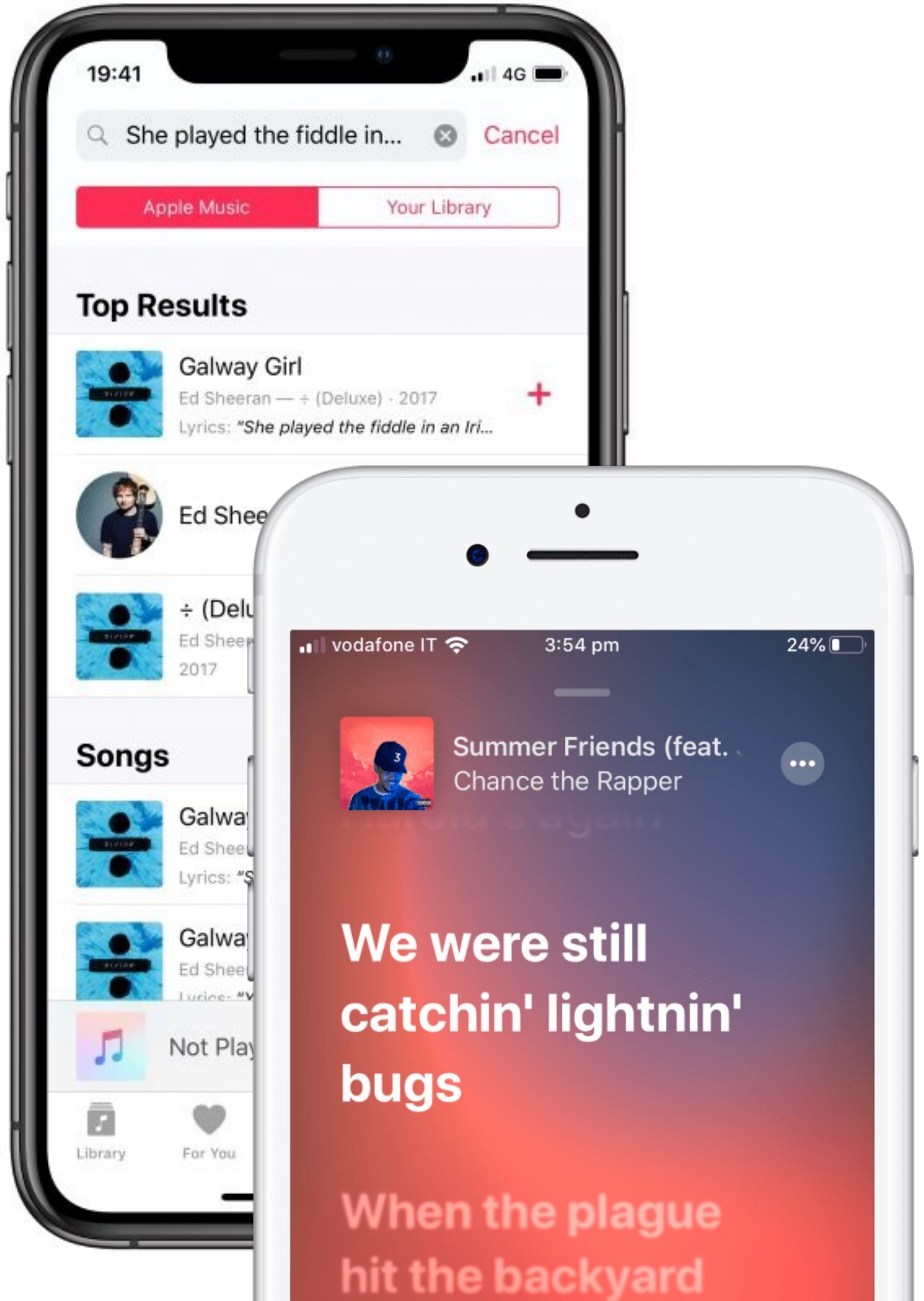
```
int var = 0;
mutex m;

void threadFunction() {
    m.lock();
    var++;
    m.unlock();
}

int main() {
    thread t1(threadFunction);
    thread t2(threadFunction);
    t1.join();
    t2.join();
    cout << "Variable value: " << var << endl;
    return 0;
}
```

In this example, the `threadFunction()` uses a Mutex to protect the var variable from simultaneous access by multiple threads. The `m.lock()` statement acquires the Mutex lock, allowing only one thread to access the shared variable at a time. Once the update is complete, the `m.unlock()` statement releases the lock, allowing other threads to access the variable.

Objective (Basic Requirements)

- Music Searching & Lyrics Display
- Users can type in keywords to search music based on your database. The results should be displayed in the list control of your program.
- Your program should be able to play music and show lyrics. The lyrics file can be simple text files or in LRC format.

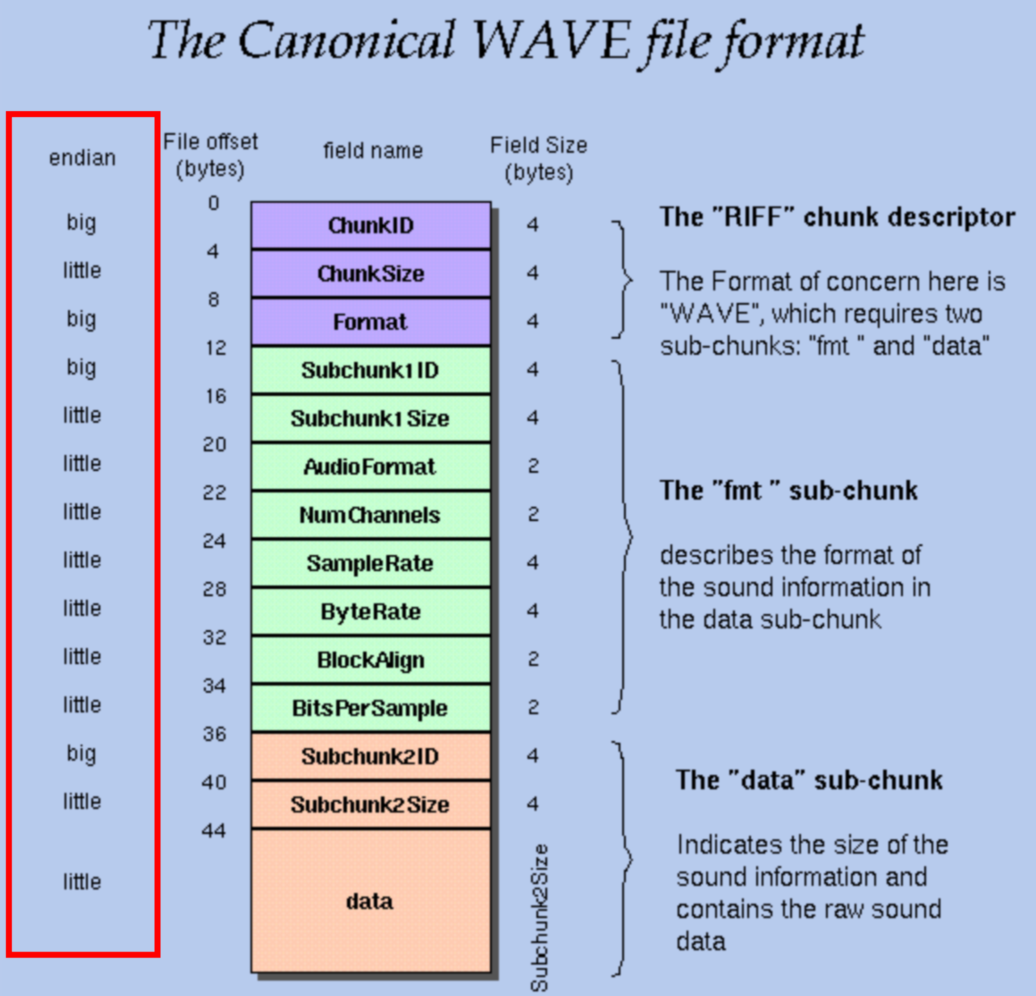
```
[00:12.00]Line 1 lyrics
[00:17.20]Line 2 lyrics
[00:21.10]Line 3 lyrics
...
[mm:ss.xx]last lyrics line
```



19:41
She played the fiddle in...
Cancel
Apple Music
Your Library
Top Results
Galway Girl
Ed Sheeran — ÷ (Deluxe) · 2017
Lyrics: "She played the fiddle in an Iri...
Ed Shee
÷ (Delu
Songs
Not Play
Library
For You
vodafone IT
3:54 pm
24%
Summer Friends (feat.
Chance the Rapper
We were still catchin' lightnin' bugs
When the plague hit the backyard

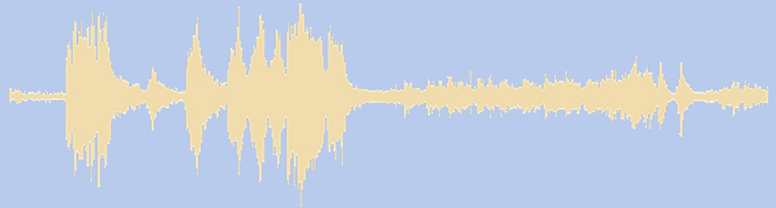
Objective (Basic Requirements)

- ***Music Decoding and Playback***
- You are required to understand the inside structure of wave format and write your own codes that can open, analyze, and playback a WAV file.
- ***The fmt sub-chunk and the data sub-chunk of the WAV file must be read and extracted manually, which means you cannot use any third-party libraries/programs***.
- The sound data should be played fluently and bring the users beautiful music.



The Canonical WAVE file format

endian	File offset (bytes)	field name	Field Size (bytes)	
big	0	ChunkID	4	The "RIFF" chunk descriptor The Format of concern here is "WAVE", which requires two sub-chunks: "fmt " and "data"
little	4	ChunkSize	4	
big	8	Format	4	
big	12	Subchunk1ID	4	The "fmt " sub-chunk describes the format of the sound information in the data sub-chunk
little	16	Subchunk1Size	4	
little	20	AudioFormat	2	
little	22	NumChannels	2	
little	24	SampleRate	4	
little	28	ByteRate	4	
little	32	BlockAlign	2	
little	34	BitsPerSample	2	
big	36	Subchunk2ID	4	The "data" sub-chunk Indicates the size of the sound information and contains the raw sound data
little	40	Subchunk2Size	4	
little	44	data	Subchunk2Size	

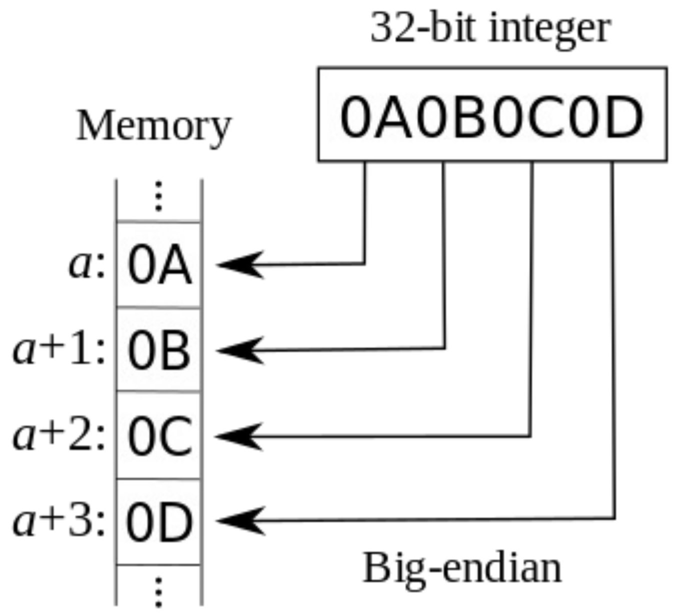


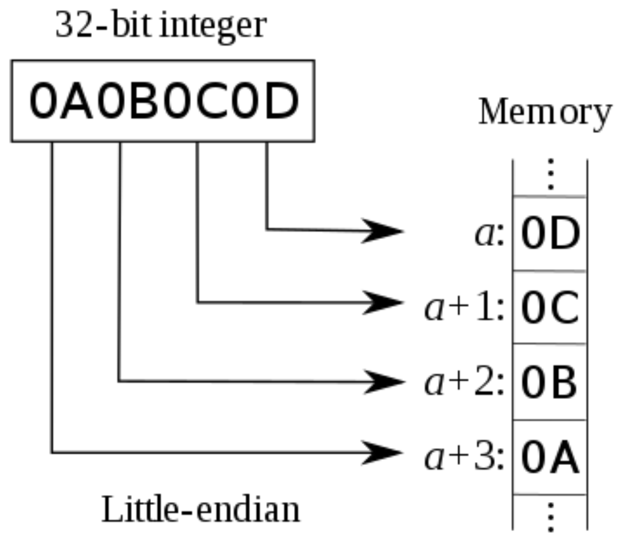
A segment of sampled audio data

Objective (Basic Requirements)

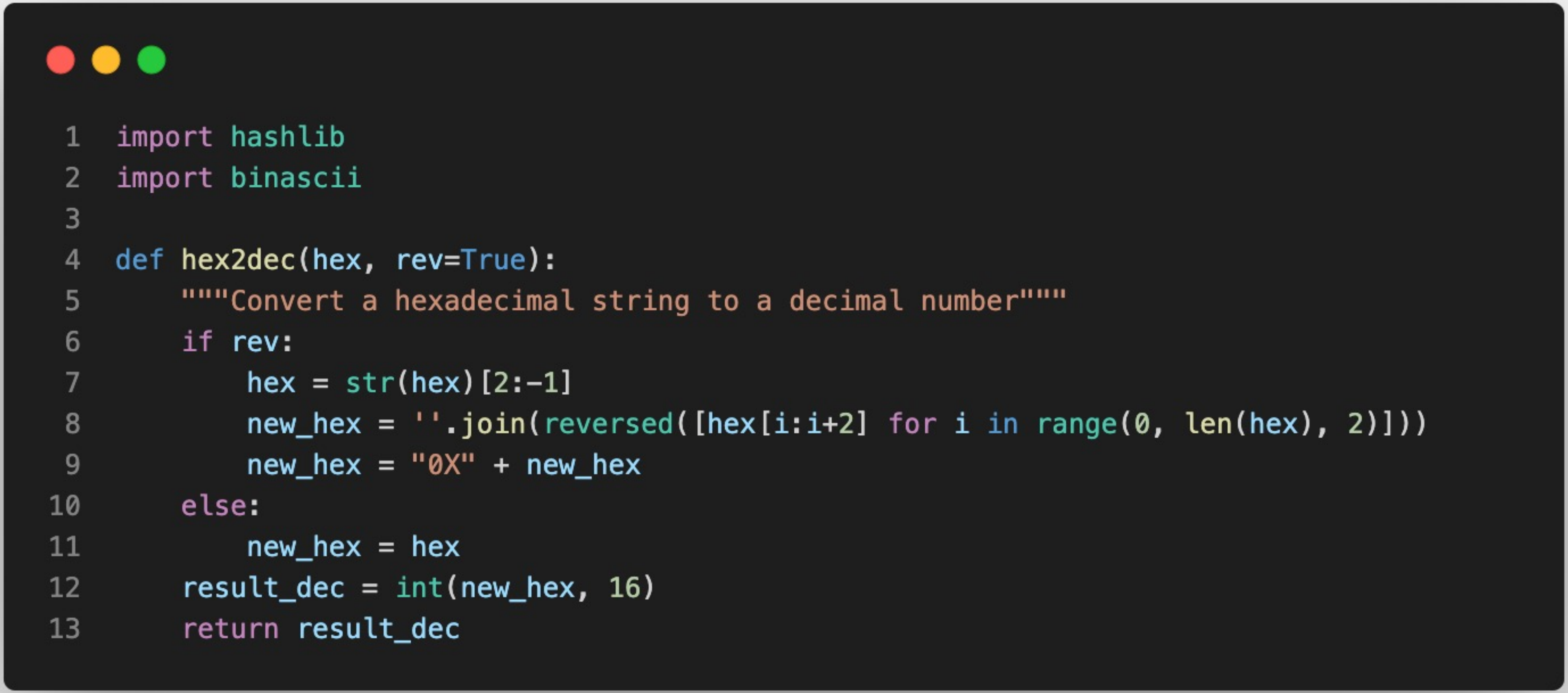
- ***Music Decoding and Playback***
- How to read binary file?
- Endianness, 0x 0A 0B 0C 0D:
  - Big endian: 0A 0B 0C 0D
  - Little endian: 0D 0C 0B 0A

- Tips:
  1. *Follow the data size of each chunk (fseek)*
  2. *Simply ignore unnecessary data*
  3. *Build a routine to convert little-endian data*





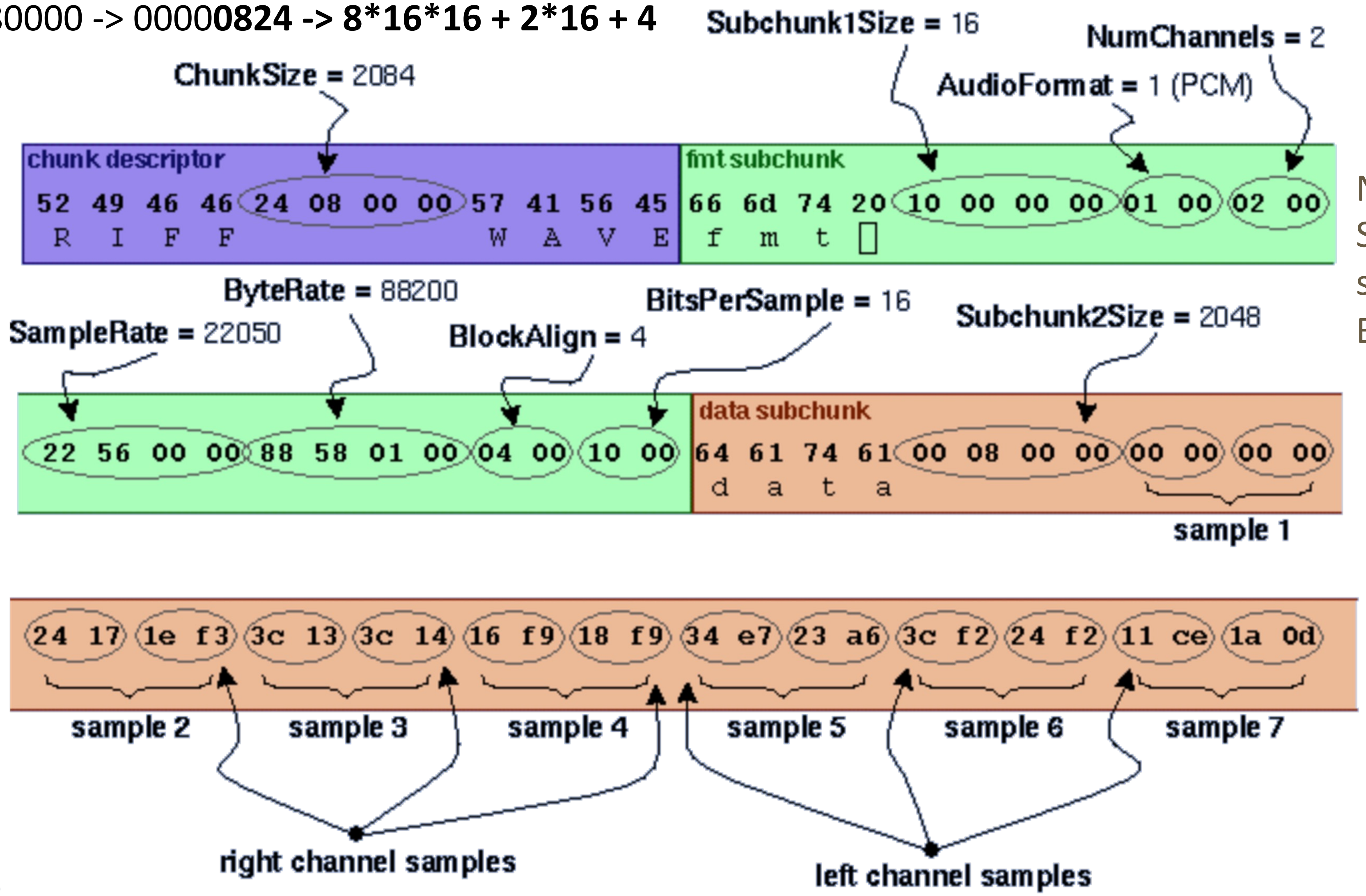
My naïve python code on converting hex code to integer



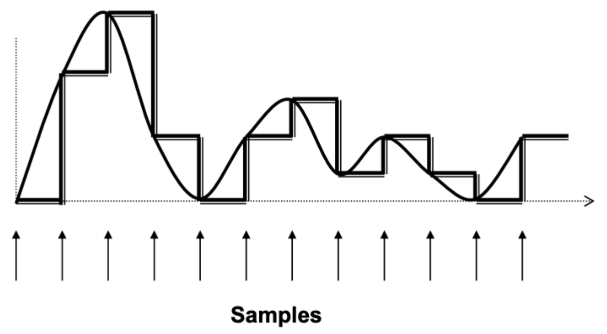
```python
import hashlib
import binascii

def hex2dec(hex, rev=True):
    """Convert a hexadecimal string to a decimal number"""
    if rev:
        hex = str(hex)[2:-1]
        new_hex = ''.join(reversed([hex[i:i+2] for i in range(0, len(hex), 2)]))
        new_hex = "0X" + new_hex
    else:
        new_hex = hex
    result_dec = int(new_hex, 16)
    return result_dec
```

- Format of Uncompressed audio file.
- A python example is given to parse the header

24080000 -> 0000**0824 -> 8*16*16 + 2*16 + 4**

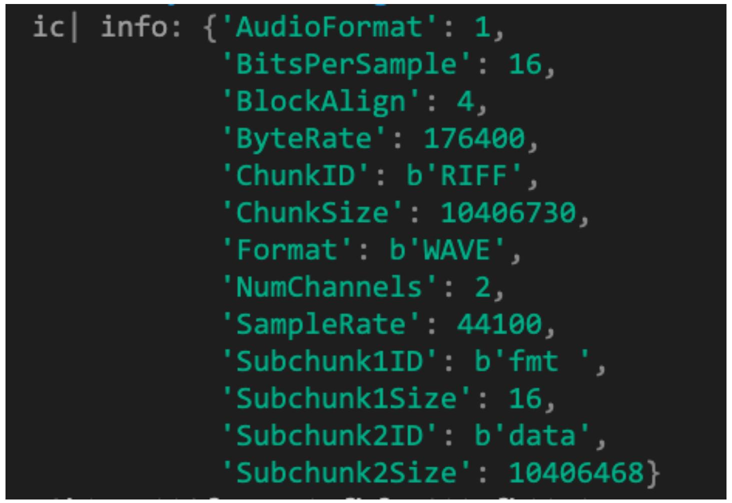


NumChannels: Channels of audio data
SampleRate: number of samples per second
ByteRate: number of bytes per second

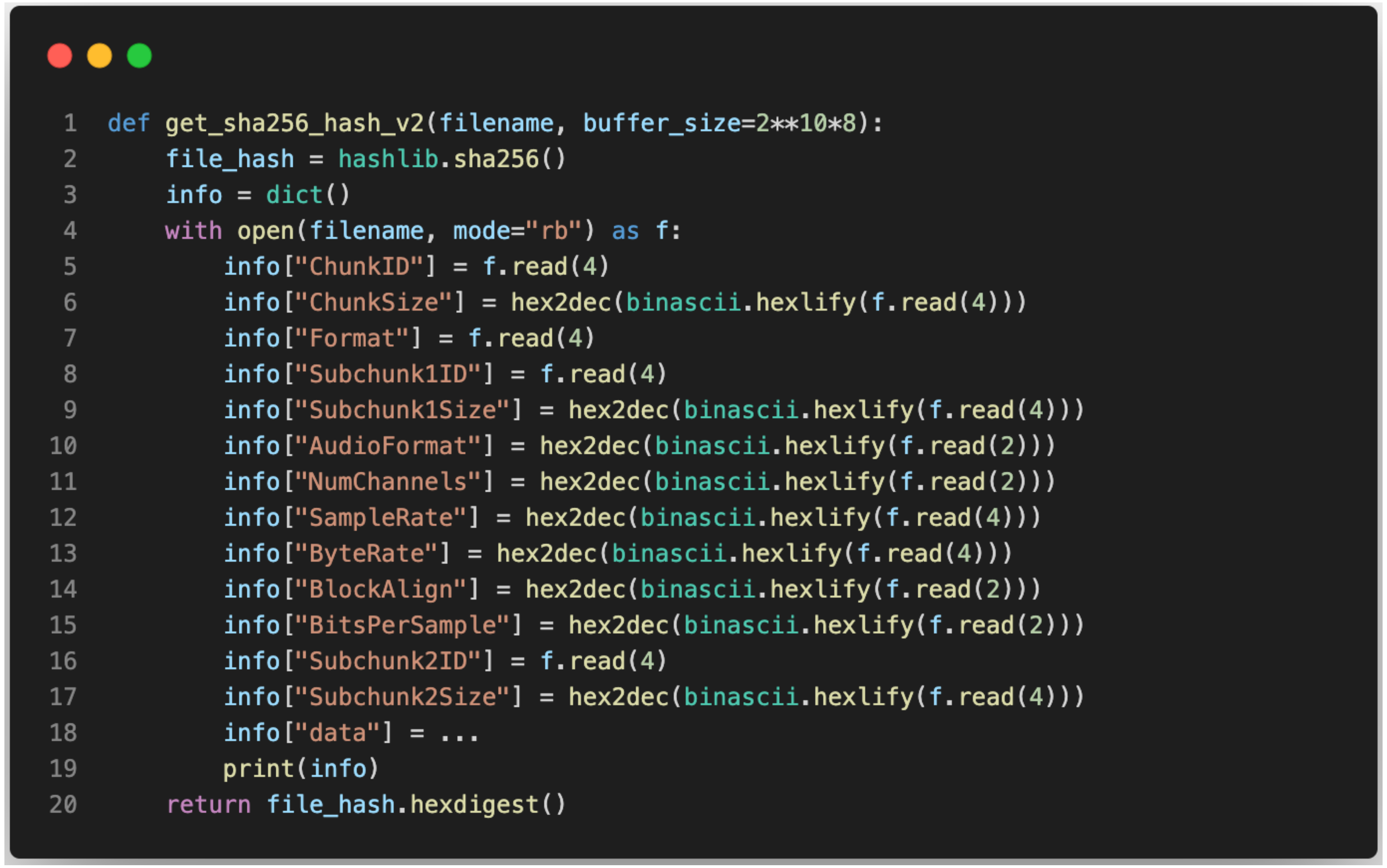


Sample values. Ref: lecture 02

My naïve python code to parse the header.



```
ic| info: {'AudioFormat': 1,
           'BitsPerSample': 16,
           'BlockAlign': 4,
           'ByteRate': 176400,
           'ChunkID': b'RIFF',
           'ChunkSize': 10406730,
           'Format': b'WAVE',
           'NumChannels': 2,
           'SampleRate': 44100,
           'Subchunk1ID': b'fmt ',
           'Subchunk1Size': 16,
           'Subchunk2ID': b'data',
           'Subchunk2Size': 10406468}
```



```python
def get_sha256_hash_v2(filename, buffer_size=2**10*8):
    file_hash = hashlib.sha256()
    info = dict()
    with open(filename, mode="rb") as f:
        info["ChunkID"] = f.read(4)
        info["ChunkSize"] = hex2dec(binascii.hexlify(f.read(4)))
        info["Format"] = f.read(4)
        info["Subchunk1ID"] = f.read(4)
        info["Subchunk1Size"] = hex2dec(binascii.hexlify(f.read(4)))
        info["AudioFormat"] = hex2dec(binascii.hexlify(f.read(2)))
        info["NumChannels"] = hex2dec(binascii.hexlify(f.read(2)))
        info["SampleRate"] = hex2dec(binascii.hexlify(f.read(4)))
        info["ByteRate"] = hex2dec(binascii.hexlify(f.read(4)))
        info["BlockAlign"] = hex2dec(binascii.hexlify(f.read(2)))
        info["BitsPerSample"] = hex2dec(binascii.hexlify(f.read(2)))
        info["Subchunk2ID"] = f.read(4)
        info["Subchunk2Size"] = hex2dec(binascii.hexlify(f.read(4)))
        info["data"] = ...
        print(info)
    return file_hash.hexdigest()
```

Objective (Enhanced Features)

- ***Following are just some basic examples. You are welcome to come up with more creative ideas to enhance your project!***
- Support for other music formats

  MP3? OGG?

  You can use third-party libraries (but please quote them in your code and report)

- Progress bar

  We only require play / stop in the basic requirements

  A progress bar is necessary for playback.

  Try to make it draggable!

Objective (Enhanced Features)

Synchronized lyric display

It is simple! Try to make it display more smoothly?

```
[00:12.00]Line 1 lyrics
[00:17.20]Line 2 lyrics
[00:21.10]Line 3 lyrics
...
[mm:ss.xx]last lyrics line
```

Audio visualization

Visualize the audio data.

Enhancements on the basic features are also welcome! Examples:

A. A better audio data decoder for faster/ high quality data loading

B. You can download data online and update your database automatically

C. A well-designed user interface with smooth-and-easy user interactions

…

Please specify your valuable works in the report!

Scores are based on the overall quality; We are not just counting the number of implemented features and give you scores.

WAV Playback Pipeline

01
Decode WAVE (RIFF) header for bit depth / sample rate / etc.

02
Extract data chunk to a buffer.

03
Play the buffer through system API.

04
Loop though 2-3 until file end.

05
Close the file.

Windows API

- Function: waveOutXXXX()
- Google & MSDN are good references

[C++] Import Libraries.

```
#include "stdafx.h"
#include <Windows.h>
#pragma comment(lib, "winmm.lib")

// For Function: waveOutXXXX()
```

Open An Output Device

- Use waveOutOpen():

[C++] Open Device.

```cpp
HWAVEOUT hwo;
WAVEFORMATEX wfx;
HANDLE wait;
wfx.wFormatTag = WAVE_FORMAT_PCM;
wfx.nChannels = 2;
wfx.nSamplesPerSec = 22050;
// You may set more properties above
wait = CreateEvent(NULL, 0, 0, NULL);
waveOutOpen(&hwo, WAVE_MAPPER, &wfx, (DWORD_PTR)wait, 0L,
CALLBACK_EVENT);
```

Prepare Chunk Data

- Read data into buffer:

[C++] Load Buffer.

```
// assume src is pointing to the data chunk
// otherwise use fseek to locate
int bufSize = 204800;
char *buf = (char*)malloc(bufSize*sizeof(char));
cnt = fread(buf, sizeof(char), bufSize, src);
```

Prepare Chunk Data

- Prepare a header:

[C++] Header Preparation.

```cpp
WAVEHDR wh;
wh.lpData = buf;
wh.dwBufferLength = bufSize;
wh.dwFlags = 0L;
wh.dwLoops = 1L;
waveOutPrepareHeader(hwo, &wh, sizeof(WAVEHDR));
```

Playback from Audio Buffer

- Use waveOutWrite():

[C++] Write data buffer to device.

```cpp
waveOutWrite(hwo, &wh, sizeof(WAVEHDR));
// wait until playback is finished
// useful for buffered playback
WaitForSingleObject(wait, INFINITE);

// do some cleanup
waveOutClose(hwo);
```

Other Useful Functions

- **waveOutSetVolume**(HWAVEOUT *hwo*,DWORD *dwVolume*);
  - between 0xFFFF and 0x0000
  - 0xFFFF is the full volume.
  - The low order word of *dwVolume* is the left-channel volume
  - The high order word is the right-channel volume

- **waveOutGetErrorText**(MMRESULT *mmrError*, LPSTR *pszText*, UINT *cchText*);
  - almost every waveOut calls returns a MMRESULT variable, any value other than MMSYSERR_NOERROR indicates failure
  - *pszText* is the string buffer
  - *cchText* is the size of the buffer

Useful Resources

1. A very initial version of my group project **in JAVA**:
   AudioInputStream to playback audio
   https://github.com/yxwang7/AudioPlayer_CSCI3280

2. A **JS version** of the complete project:
   https://github.com/kyroslee/P2PMusicPlayer-csci3280

3. Windows Multimedia API Documentation (**for C/C++**)
   https://learn.microsoft.com/en-us/windows/win32/api/mmeapi/

4. **Python** Blog for play music in python
   https://realpython.com/playing-and-recording-sound-python/

Objective (Basic Requirements)

- Network connection (25% of basic)

Your program should be able to connect to other PCs using TCP/IP network stack.

Use any method to get the IP address of the connectable PCs except hard coding. (For example, manually inputting IP addresses, using a tracker server, or broadcasting are appropriate).

The network should support at least three terminals.

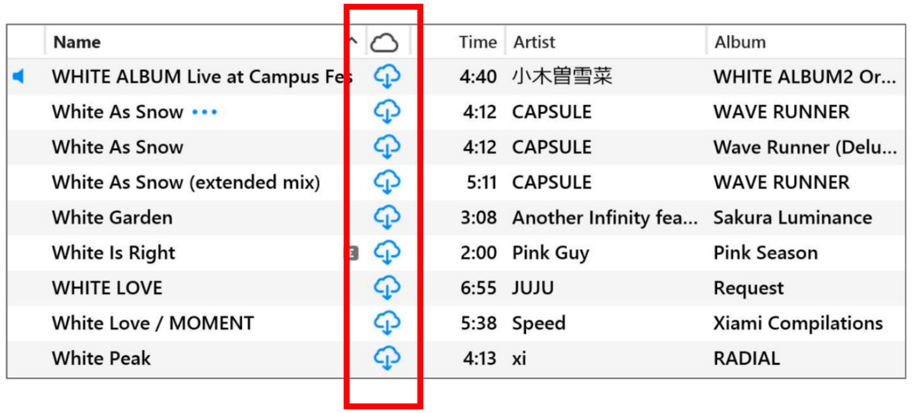
- Online Music Searching (10% of basic)

Besides local music searching, network searching should be supported in this phase.

Not only in the local database but also in the database of the other PC connected to your program.

All the results from the local and network databases should be displayed in the same list control of the UI.

Identical results must be displayed only once.



Name	☁	Time	Artist	Album
WHITE ALBUM Live at Campus Fes	☁	4:40	小木曽雪菜	WHITE ALBUM2 Or...
White As Snow •••	☁	4:12	CAPSULE	WAVE RUNNER
White As Snow	☁	4:12	CAPSULE	Wave Runner (Delu...
White As Snow (extended mix)	☁	5:11	CAPSULE	WAVE RUNNER
White Garden	☁	3:08	Another Infinity fea...	Sakura Luminance
White Is Right	☁	2:00	Pink Guy	Pink Season
WHITE LOVE	☁	6:55	JUJU	Request
White Love / MOMENT	☁	5:38	Speed	Xiami Compilations
White Peak	☁	4:13	xi	RADIAL

Music on other nodes: searching "white", information can be listed locally.

Objective (Basic Requirements)

- Availability Check (5% of basic)

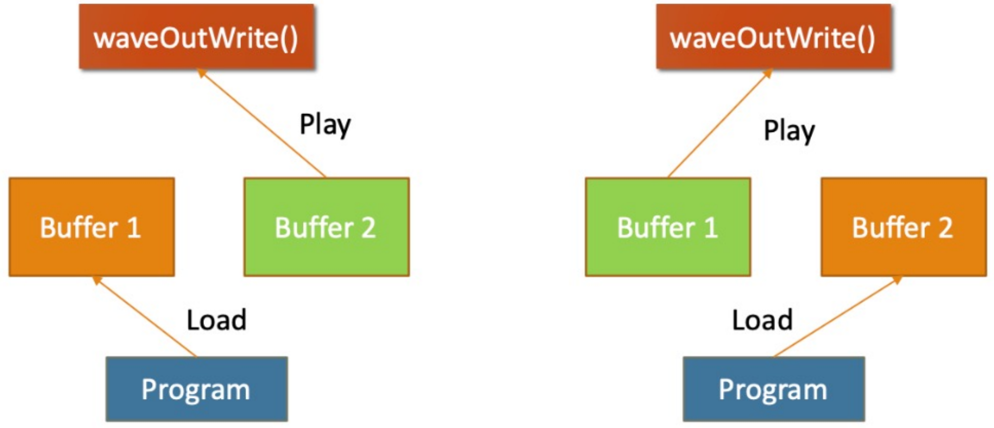
As the user may not know where the audio is, the program should check whether the audio file exists locally when the user selects audio from the search results.

If the audio file exists, playback the audio directly. Otherwise, stream the audio file from other computers.

- Real-Time Audio Streaming (20% of basic)

When the program is streaming from other computers, your program should automatically play the audio as soon as possible when it receives a piece of audio data (after a certain amount of buffering). ***You are only allowed to buffer no more than 50% of a music file before you play the music***.

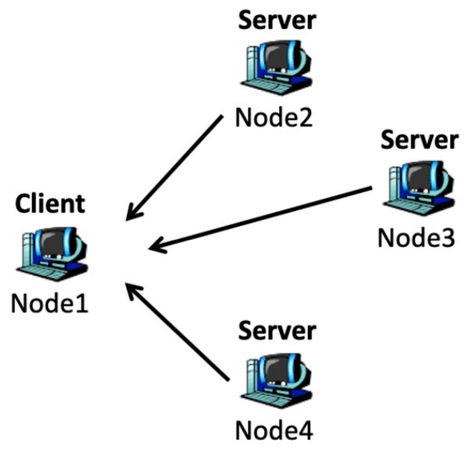
Single buffer size <= 5% of the music OR streaming delay <50ms will be regard as enhanced.



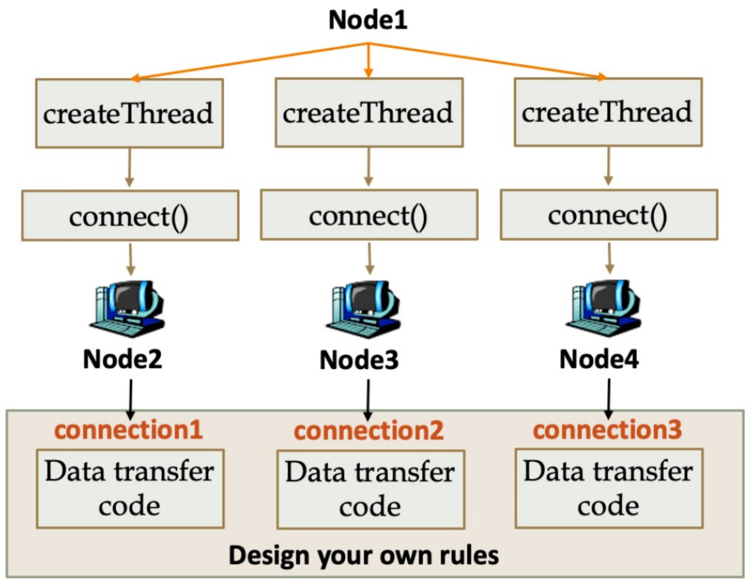
Objective (Basic Requirements)

- Peer-to-Peer Playing (40% of basic)

Your program should be able to receive one audio file from at least two other computers simultaneously. ***The audio data from different computers should be played in an interleaving way.***



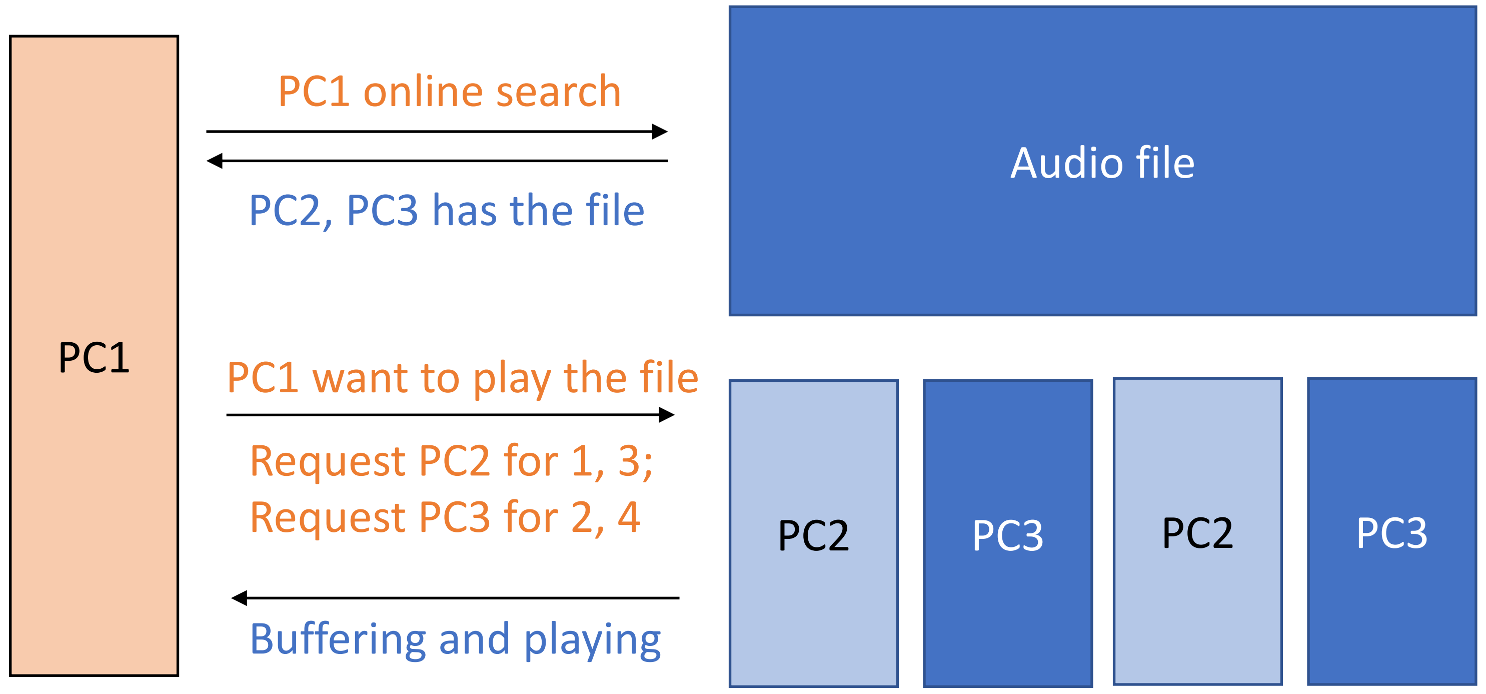
Node 1 want to play a wav file



Objective (Basic Requirements)

- Peer-to-Peer Playing (40% of basic)

For example, suppose PC1 wants to play a file, and it cannot be found locally. It should get audio data from PC2 and PC3, the file is divided into (at least) 4 parts, and PC1 may get the first part from PC2, the second part from PC3, the third part from PC2, and the fourth part from PC3.

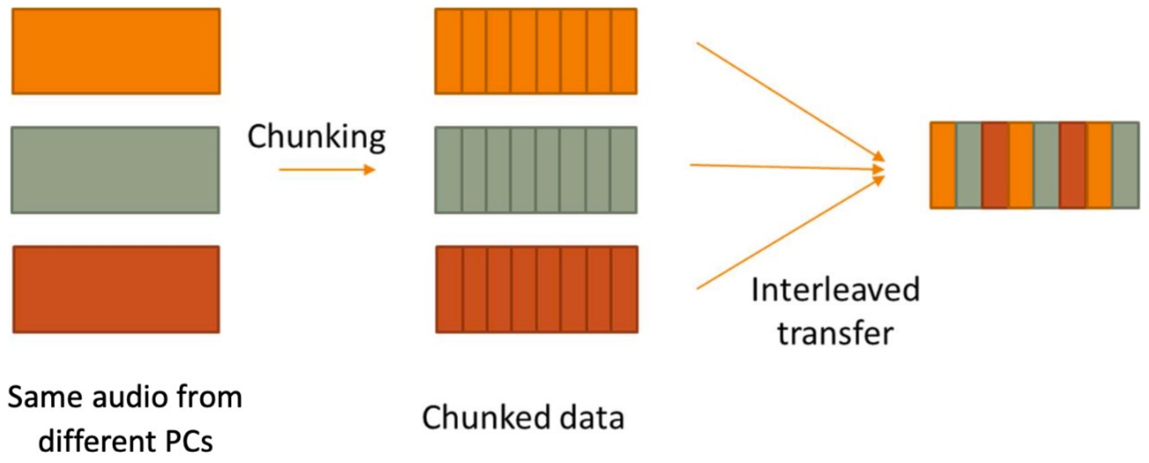


Objective (Basic Requirements)

- Peer-to-Peer Playing (40% of basic)

To verify the interleaving feature, you should also implement a function in your project to show your interleaving feature using images.

Let's say PC1 wants to download this image from other endpoints in an interleaving way; you must achieve that your data are completed but collected from different endpoints.



Objective (Enhanced Features)

To get full mark (for a group of 4), a set of enhancements is provided (both Phase 1&2):

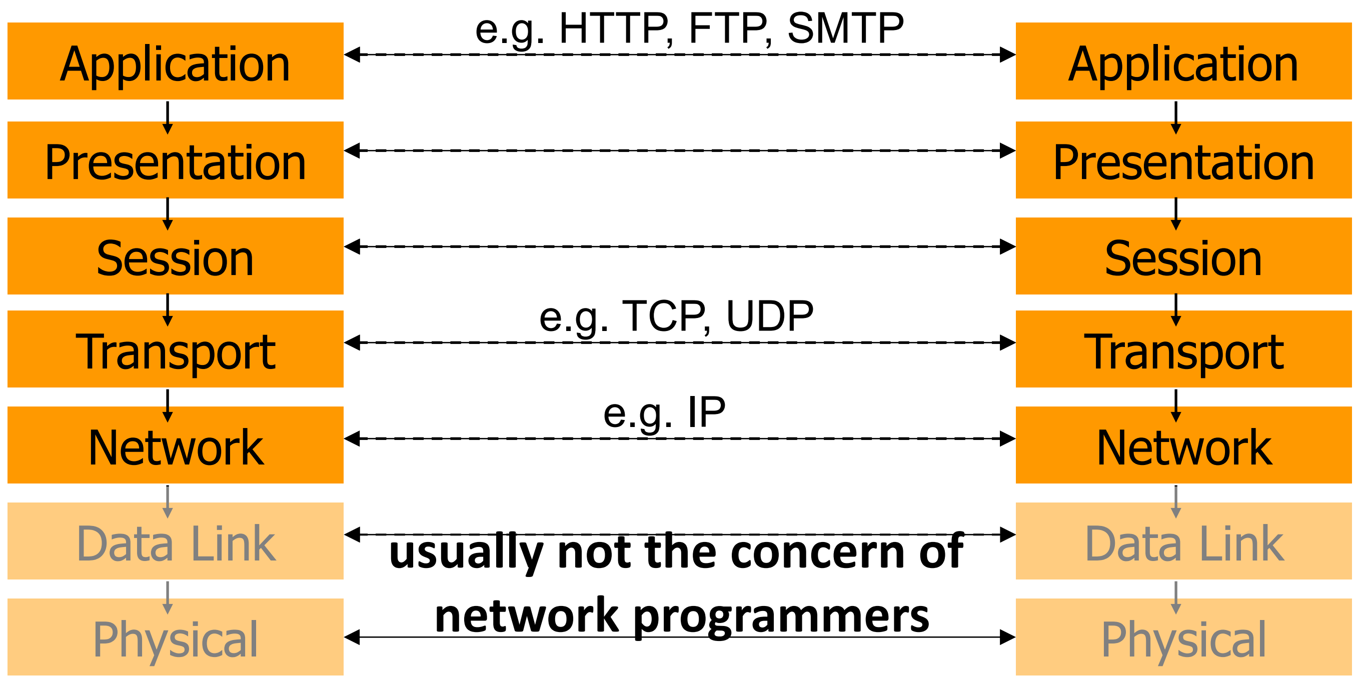
1. Synchronized lyric display
2. Progress Bar
3. Support more than 2 server & clients in Network & P2P
4. 1 other features that enhance the overall quality: e.g.,
   - A pleasing UI control
   - Network error handling: example: one of two servers down during the data transfer
   - Support other audio formats
5. 1 other NOVEL feature

The above is NOT the golden rule, you can follow your designed plan.

For basic / enhanced features, mark will be deducted if the feature quality is not satisfying

The Network Model

- Networks are organized as a series of layers (or levels)
- The rules to communicate are called protocol
- Seven layers (OSI reference model):



The Network Model

- Connection basics:
    - Protocol (TCP/UDP)
    - Local IP address, local port
    - Remote IP address, remote port

  IP address is in the form 123.23.23.22

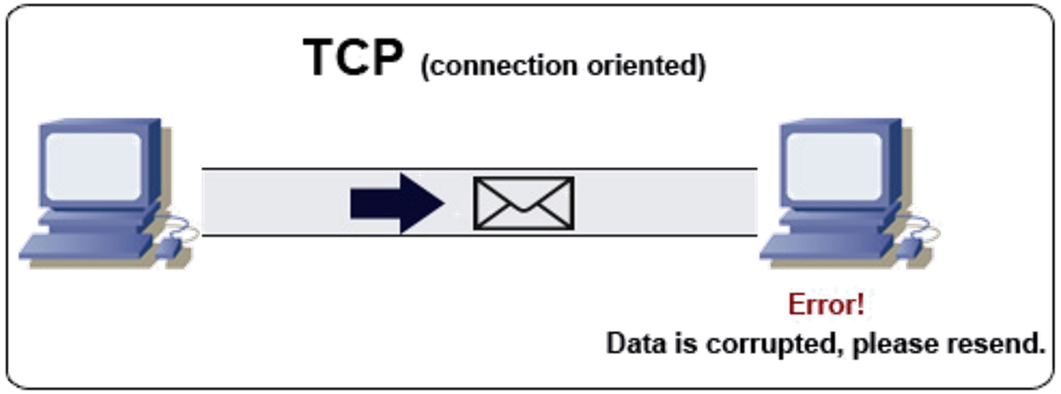
- What is a port?
    - Application-specific or process-specific
    - A 16-bit integer for identification (0 to 65535)
    - Need to use a local port for sending data

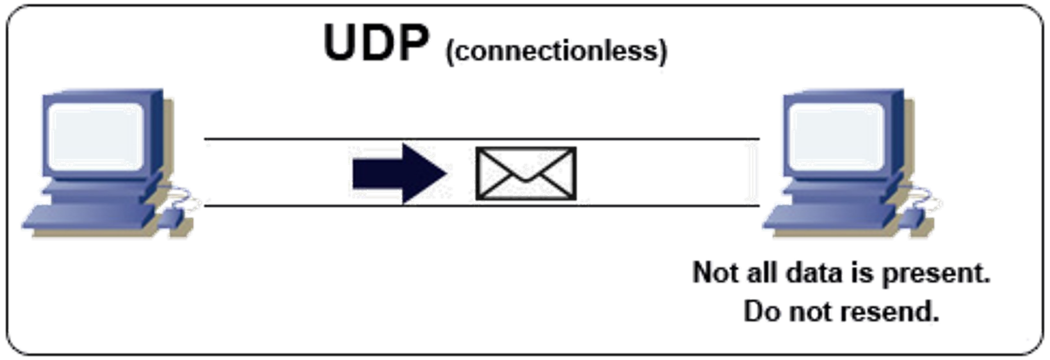
Socket

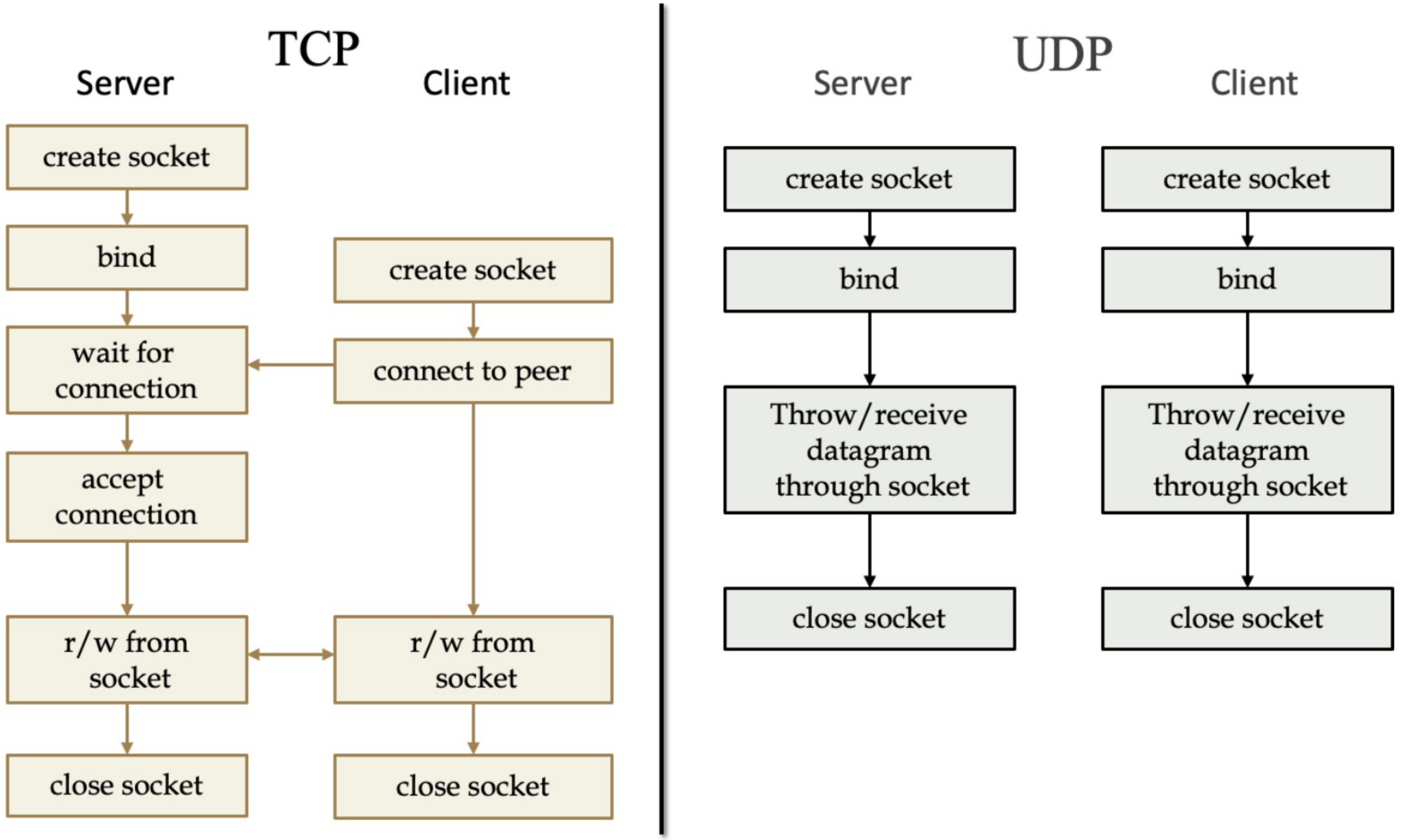
- Socket is the endpoints of a communication channel
- A network programming interface, abstracting away underlying mechanism
- On Windows platform, we use WinSock.
- Two types of sockets (for two different transport layer protocols)
    - `SOCK_STREAM (TCP) [Correctness Concern]`
    - `SOCK_DGRAM (UDP) [Speed Concern]`

TCP vs UDP

TCP	UDP
Sequenced	Unsequenced
Reliable	Unreliable
Connection-oriented	Connectionless
Virtual circuit	Low overhead
Acknowledgements	No acknowledgment
Windowing flow control	No windowing or flow control







The pipeline of TCP/UDP data transfer.

Call the Socket

- Start-up call – initialize the underlying Windows Sockets DLL

  int **WSAStartup** (WORD *wVersionRequested*, LPWSADATA *lpWSAData*);

- Socket creation – create an endpoint for communication, return a socket.

$$\text{SOCKET }\textbf{socket}(\underbrace{\text{int }\textit{addr\_family},}_{\text{“AF\_INET”}}\quad\underbrace{\text{int }\textit{type},}_{\substack{\text{“SOCK\_STREAM”/}\\\text{“SOCK\_DGRAM”}}}\quad\underbrace{\text{int }\textit{protocol}}_{\text{“0”}});$$

- Socket binding – bind a SOCKET descriptor to a local port and local IP address

$$\text{int }\textbf{bind}(\text{SOCKET }\underset{\uparrow\ \text{the descriptor}}{\textit{socket}}, \underbrace{\text{const struct sockaddr *}\textit{address}}_{\text{local IP address+port}}, \underbrace{\text{int }\textit{address\_len}}_{\text{length of sockaddr}});$$

Call the Socket

- Let the socket to wait for connection requests (stream socket, server-sided)
  $\text{int } \textbf{listen}(\underbrace{\text{SOCKET } s}_{\text{the descriptor}}, \text{ int } \textit{backlog});$
- Try to connect to the server (stream socket, client-sided)
  int connect(SOCKET s, const struct sockaddr *remote_addrint address_len);
- Accepting connection (stream socket, server-sided), after accept the original socket, s, remains in “listen” state.
-

$\boxed{\text{Just like bind()!}}$

$\text{SOCKET } \underbrace{\text{accept}}_{\text{returns a new socket descriptor for the new connection}} \text{ (SOCKET s, } \underbrace{\text{struct sockaddr *addr,}}_{\text{set to NULL if you don't want it}} \text{ int } \underbrace{\text{*len}}_{\text{length of the structure, (remember the *)}});$

```c
// Start-up call - initialize the underlying Windows Sockets DLL
int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData );

// Socket creation - create an endpoint for communication, return a socket.
SOCKET socket(int addr_family, int type, int protocol);

// Socket binding - bind a SOCKET descriptor to a local port and local IP address
int bind(SOCKET socket, const struct sockaddr *address, int address_len);

// Let the socket to wait for connection requests (stream socket, server-sided)
int listen(SOCKET s, int backlog);

// Try to connect to the server (stream socket, client-sided)
int connect(SOCKET s, const struct sockaddr *remote_addr, int address_len);

// Accepting connection (stream socket, server-sided), after accepting the
// original socket, s, remains in "listen" state.
SOCKET accept (SOCKET s, struct sockaddr *addr, int *len);
```

Call the Socket

```
// Receive or send data with socket
// recvfrom()/sendto(): For UDP.
int recv(SOCKET s, char *buf, int len, int flags);
int WSAAPI send(SOCKET s, const char *buf, int len, int flags);

// Close a socket - kill the connection
int closesocket(SOCKET s);

// Cleanup - terminates the use of the Windows Sockets DLL
int WSACleanup();

// Error checking - to get the error code after a failed call,
the meaning of the code can be checked in the header file
int WSAGetLastError();
```

Blocking and Multi-tasking

- Some network I/O calls are in BLOCKING mode,
  - i.e. the program waits until a call completes
  - e.g. accept(), connect(), recv(), send(), sendto(), recvfrom()…
- How to serve multiple clients?
  - FIFO serving
  - Set the socket in non-blocking mode by ioctlsocket()
  - Use select() to wait for any incoming activities on all the sockets
  - Spawn a new thread/process for each connection request by pthread_create()
  - …

Another Idea: HTTP

- Symmetry network
- Every node is both client and server
- In HTTP, we can retrieve data using URLs. Here's an example:

http://pc1/get_data?filename=somnus.wav&start=4096&length=4096.

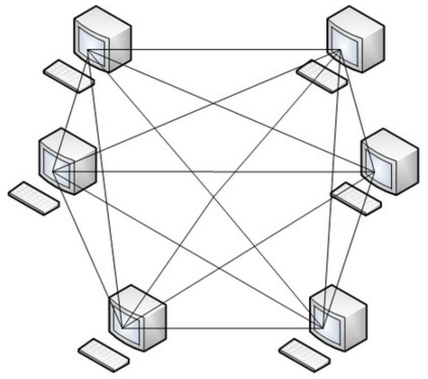
In this example, we request data from the "pc1" node. We are asking for a specific file named "somnus.wav". We also specify the start position and length of the data we want.
Similarly, we can request data from other nodes in the network:

http://pc2/get_data?filename=somnus.wav&start=0&length=4096.

This request asks the "pc2" node to send data from the beginning of the file.
We can also check if a file exists in the network using the URL:

http://pc1/check_existence?filename=somnus.wav.

- Server send data fragments / status



Implementing HTTP Server

- Listening to port 80 (or others)
- Routing
  - /get_data -> a function to read and transfer file fragment
  - /check_existence -> a function to check existence of a file
  - … some others
- Parameters: Extracting HTTP GET parameters
- Server response
  - Text data: using plain text/json
  - Binary data: using application/octet-stream in your header
- Suggested Library: libmircohttpd(C++) / Pistache(C++) / flask(python)

Implementing HTTP Client

- Send HTTP requests to the servers
- Use GET request with proper parameters
- Handling JSON/binary data from the servers
- Take care of data orders
- Suggested Library: libcurl(C) / Pistache(C++) / urllib3 (python)

Network Error Handling

- If a network function call fails, use **WSAGetLastError()** to get the error number
- If a socket creation call fails, the returned value would be INVALID_SOCKET
- For other function calls, the returned value SOCKET_ERROR indicates a failure
- To make your program more robust, you should handle the errors if they ever happened

Multiple-Threads

Why Need Multiple Threads?

- Audio playback
  - waveOutWrite() blocks everything:

```
while (notEOF){
  loadDataIntoBuffer();  // fread/memcpy or
others
  waveOutPrepareHeader(...);
  waveOutWrite(...);
  waitForSingleObject();    // block until
playback finished
}
```

How to do some other stuff during playback?
How to make UI responsive during playback?

Why Need Multiple Threads?

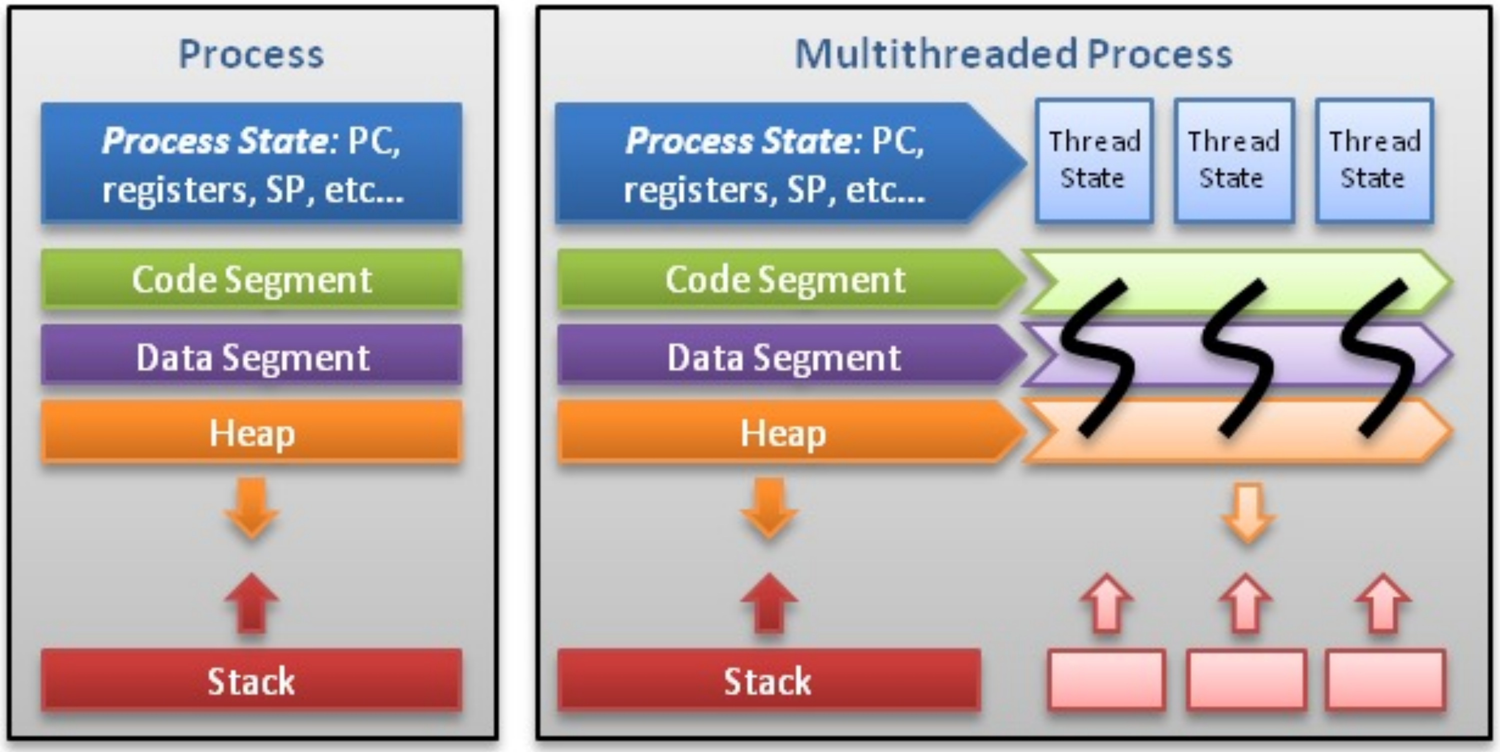
- Socket programming
  - Code is executed sequentially

```
while (1){
    client_sd = accept(...);  // blocked
    ...
}
while (1) {
    len = recv(...);  // blocked
    ...
}
```

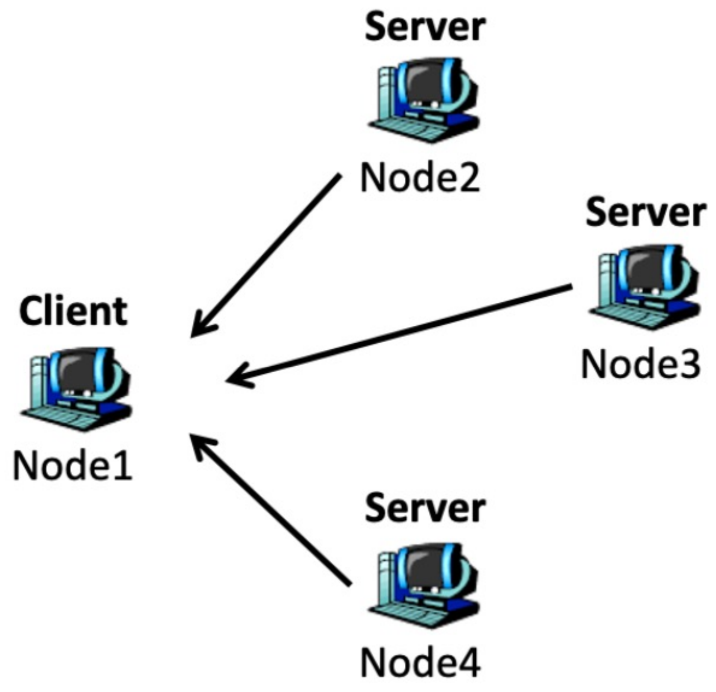
Can you accept connection and receive data simultaneously?

Threads

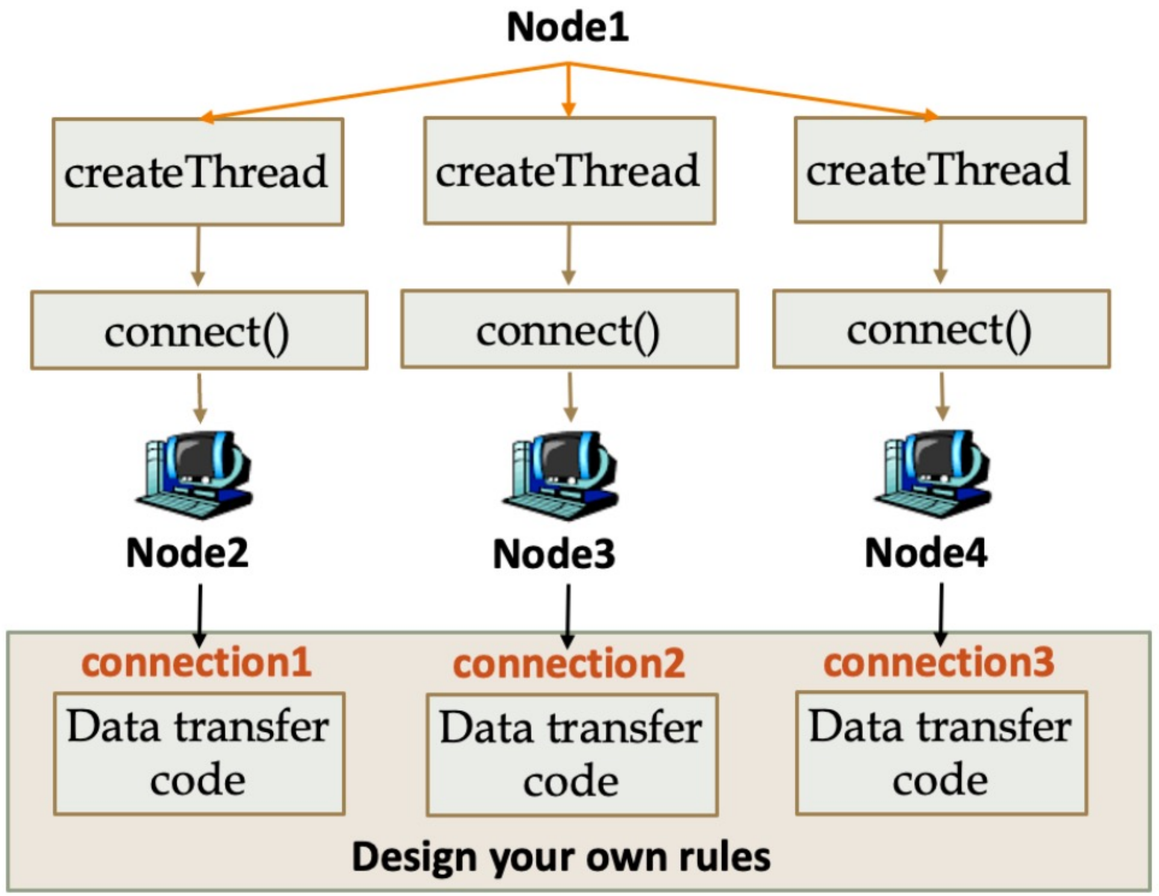
- A **thread** is a sequence of such instructions within a program that can be executed independently of other code.
- **Multi-threading** design allows an application to do parallel tasks simultaneously.



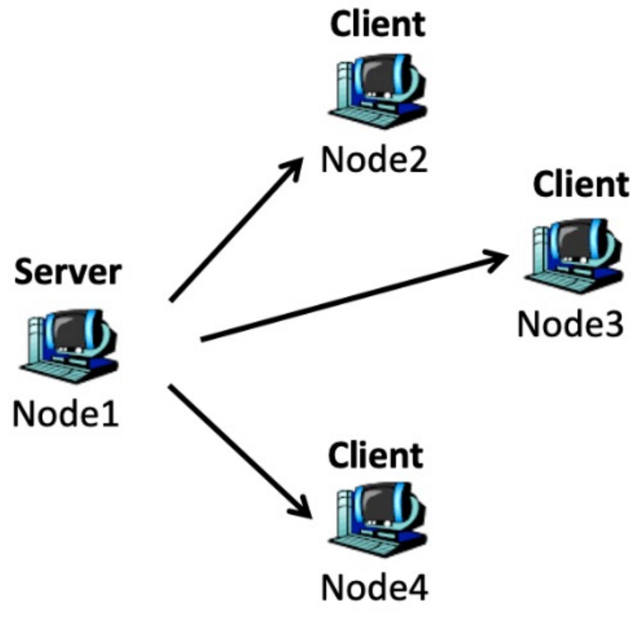
P2P Client (TCP)



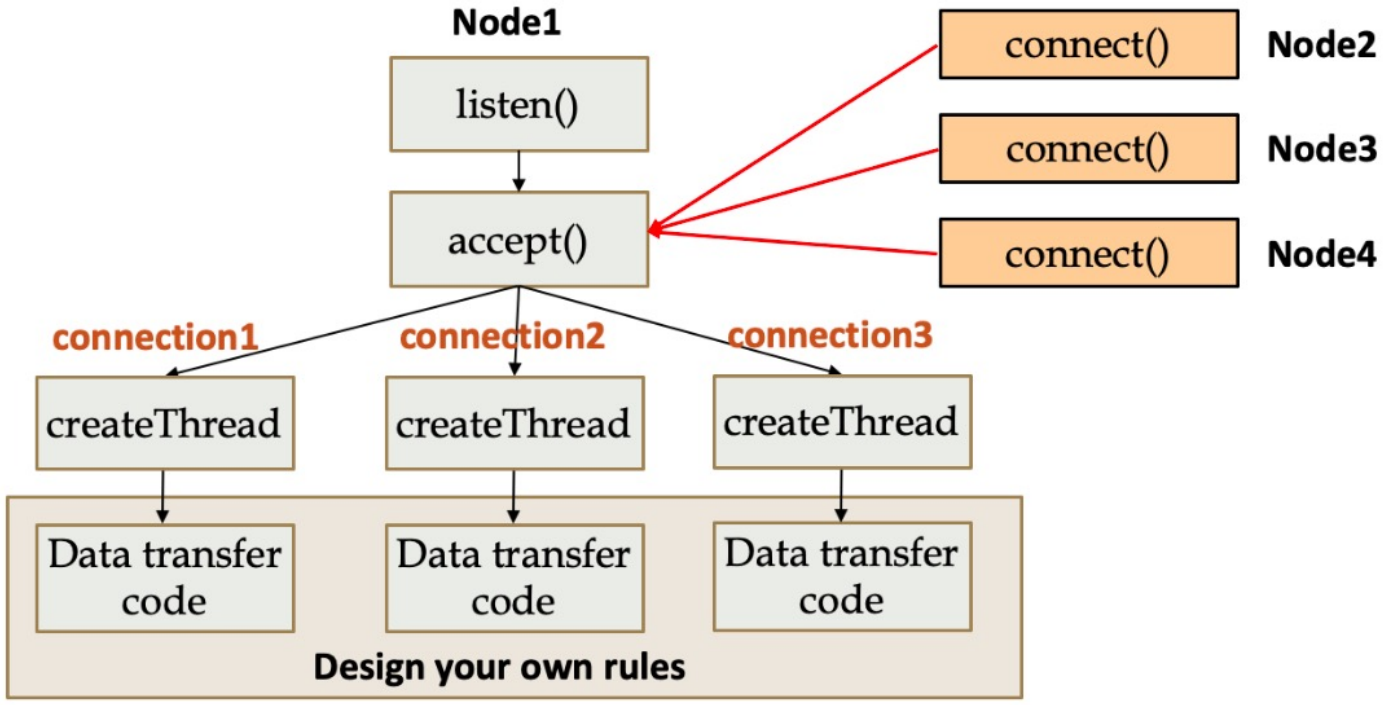
Node 1 want to play a wav file



P2P Client (TCP)

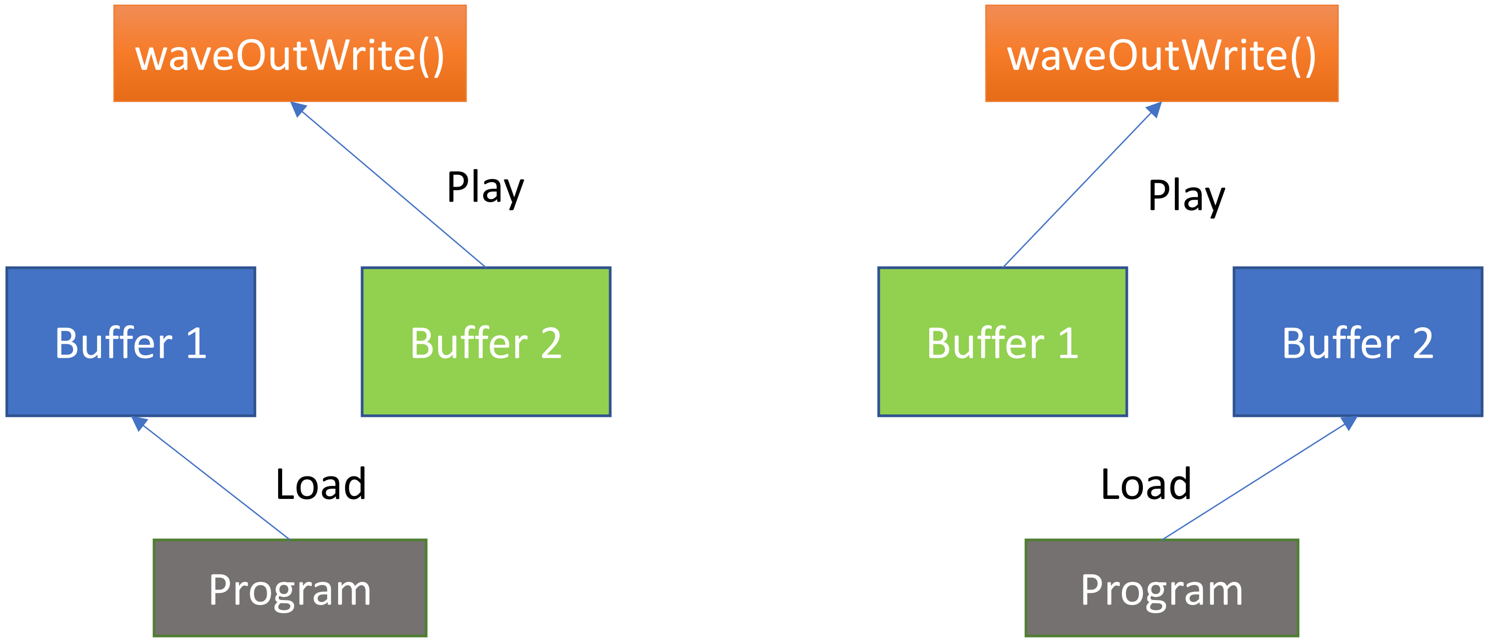


Node 2,3,4 want to play a wav file



Page Flipping

- To minimize the audio playback delay:



STD::Thread

- Add #include <thread> to your source file
- Thread type: **std::thread**

Creation

```
Std::thread t(void *(*start_routine)(void *), void *arg);
```

*void *(*start_routine)(void *):* the function this thread executes *void *arg:* arguments to pass to thread function above

Join threads: **join()**

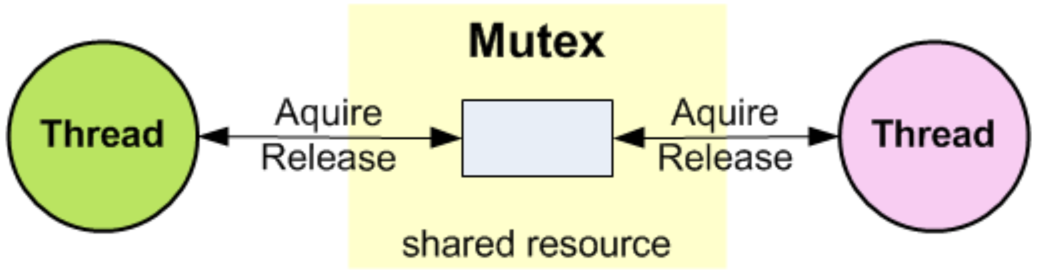
- Suspends the calling thread to wait for successful termination of the thread specified as the first argument pthread_t thread with an optional *value_ptr data passed from the terminating thread's call to pthread_exit().

Code Example:

```
#include <iostream>
#include <thread>

using namespace std;

void hello(const char* input) {
    cout << input << endl;
}

int main() {
    thread t(hello, "hello world");
    t.join();
    return 0;
}
```

Thread Communications

- Global variables
- Pointers as thread arguments
- Take care of every WRITE operations
- Mutex



Examples are in the specification.

Additional References for JS\Python

Python-p2p (TCP):

https://github.com/GianisTsol/python-p2p

Python HTTP server/client:

https://www.tutorialspoint.com/python_network_programming/python_http_client.htm

Python multi-thread:

https://realpython.com/intro-to-python-threading/#working-with-many-threads

JS HTTP requests:

https://kinsta.com/knowledgebase/javascript-http-request/;

https://github.com/kyroslee/P2PMusicPlayer-csci3280/blob/master/src/main/httpSrv.js

JS multi-thread (workers):

https://www.loginradius.com/blog/engineering/adding-multi-threading-to-javascript-using-web-workers

JS p2p:

https://medium.com/@carloslfu/make-a-p2p-connection-in-10-minutes-57d9559fd1c

Please specify libraries/related functions you used in the report.

Thank you!

Tutorial Plan about Project:

Network (TCP/IP, Socket, HTTP) / Peer-to-Peer (Multiple-Threads): Apr 6th

~~Multi-Threads Cont. /~~

[TBA, if necessary] Q&A: Apr 13th

Details are already provided in the specification, please check!

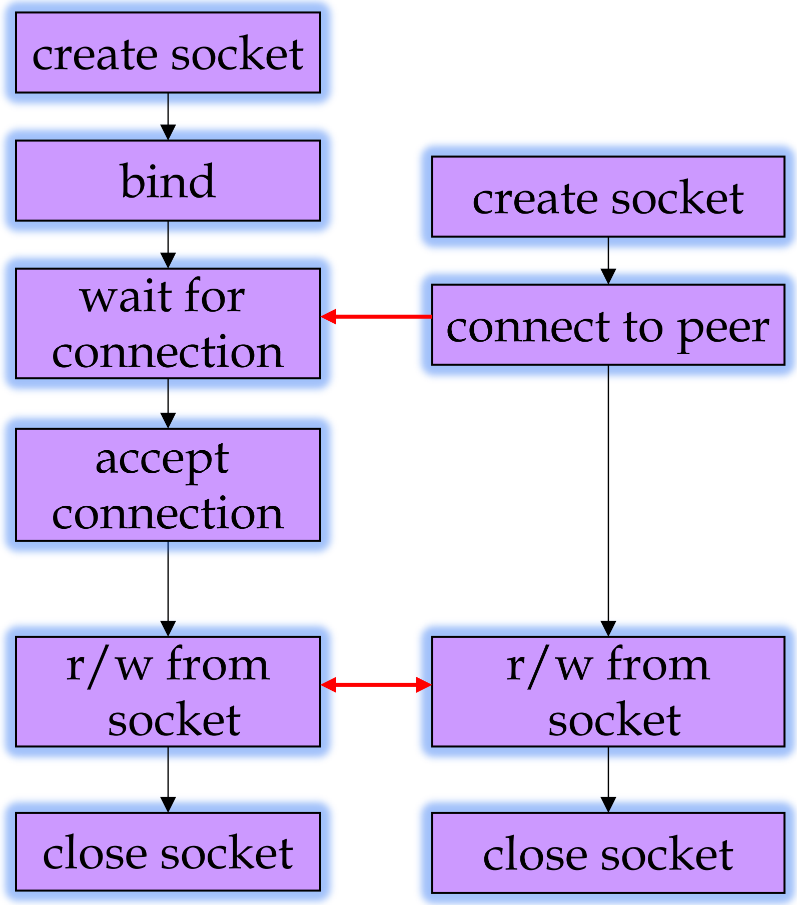
Be responsible to your teammates!

ZHANG Yuechen

zhangyc@link.cuhk.edu.hk

TCP Server

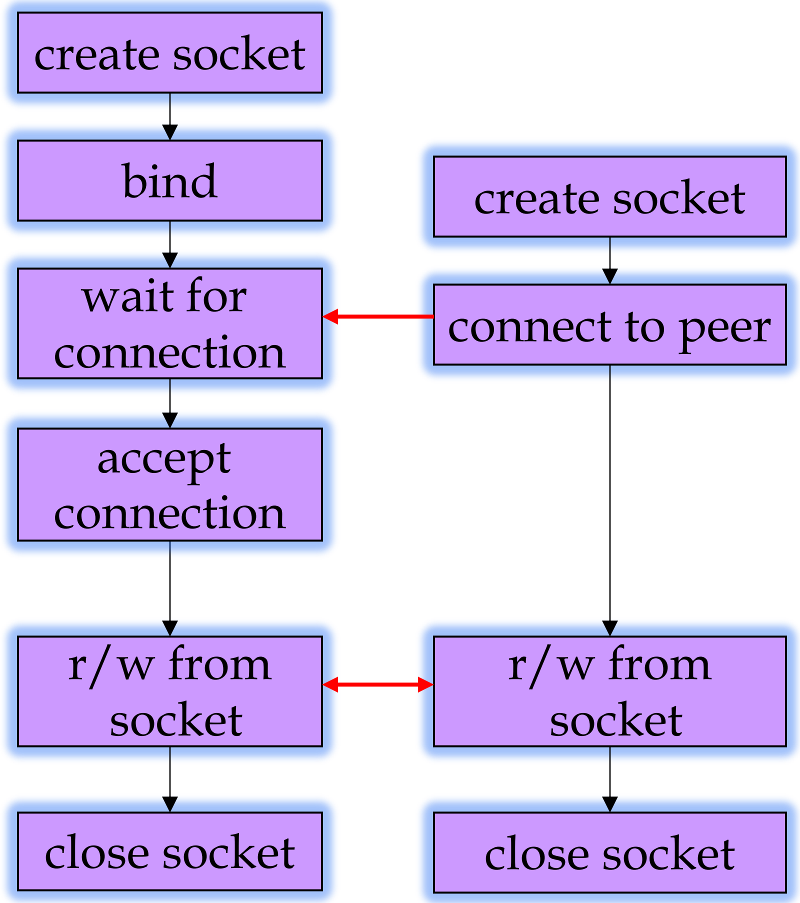
```
#include <winsock.h>

#define MY_PORT 3434

int main() {
   SOCKET listen_sock, new_sock;
   struct sockaddr_in my_addr;
   int dummy;
   char *buffer ="How old are you?\0";
   WSADATA wsaData;

   WSAStartup(MAKEWORD(2,1),&wsaData);

   listen_sock = socket(AF_INET, SOCK_STREAM, 0);

   my_addr.sin_family = AF_INET;
   my_addr.sin_port = htons(MY_PORT);
   my_addr.sin_addr.s_addr = INADDR_ANY;

   bind(listen_sock, (struct sockaddr *)&my_addr, sizeof(struct
sockaddr));
   listen(listen_sock, SOMAXCONN);
   new_sock = accept(listen_sock, NULL, &dummy);
   send(new_sock, buffer, strlen(buffer), 0);

   closesocket(new_sock);
   closesocket(listen_sock);
   WSACleanup();

   return 0;
}
```



connection-oriented
(SOCK_STREAM)

TCP Client

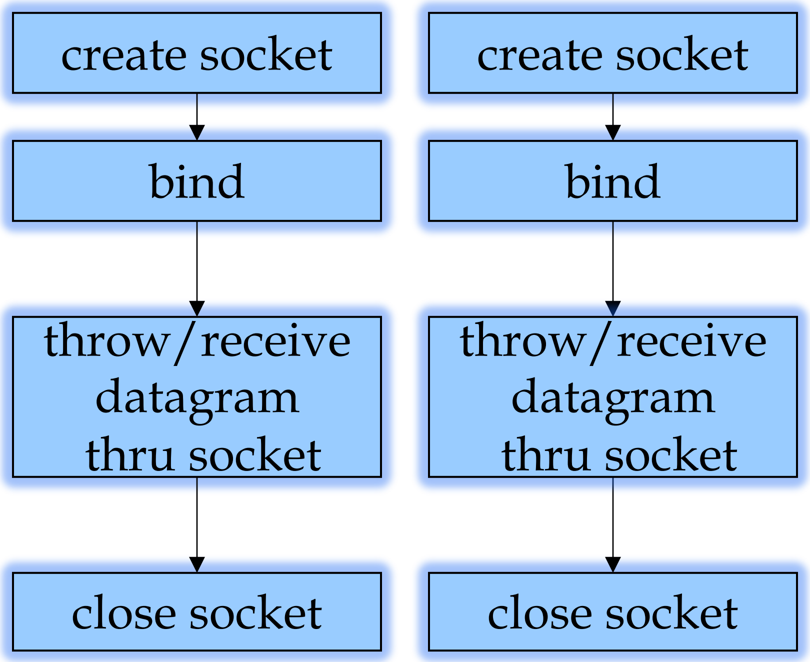
```
#include <winsock.h>
#include <stdio.h>

#define MY_PORT 3434

int main() {
   SOCKET conn_sock;
   struct sockaddr_in remote_addr;
   int bytes_recvd;
   char buffer[100];
   WSADATA wsaData;

   WSAStartup(MAKEWORD(2,1),&wsaData);

   conn_sock = socket(AF_INET, SOCK_STREAM, 0);

   remote_addr.sin_family = AF_INET;
   remote_addr.sin_port = htons(MY_PORT);
   remote_addr.sin_addr.s_addr = inet_addr("137.189.90.38");

   connect(conn_sock, (struct sockaddr *)&remote_addr, sizeof
(struct sockaddr));

   bytes_recvd = recv(conn_sock, buffer, sizeof(buffer), 0);

   printf("Received (%d bytes): \"%s\"\n", bytes_recvd, buffer);

   closesocket(conn_sock);
   WSACleanup();

   return 0;
}
```



connection-oriented
(SOCK_STREAM)

```
#include <winsock.h>
#include <stdio.h>
#define MY_PORT 3434
int main() {
   SOCKET udp_sock;
   struct sockaddr_in remote_addr, local_addr;
   int bytes_recvd, dummy;
   char buffer[100];
   WSADATA wsaData;

   WSAStartup(MAKEWORD(2,1),&wsaData);
   udp_sock = socket(AF_INET, SOCK_DGRAM, 0);

   local_addr.sin_family = AF_INET;
   local_addr.sin_port = htons(MY_PORT);
   local_addr.sin_addr.s_addr = INADDR_ANY;

   bind(udp_sock, (sockaddr *)&local_addr, sizeof(struct
sockaddr_in));
   bytes_recvd = recvfrom(udp_sock, buffer, sizeof(buffer), 0,
NULL, &dummy);
   buffer[bytes_recvd] = '\0';
   printf("Received (%d bytes): \"%s\"\n", bytes_recvd, buffer);
   closesocket(udp_sock);
   WSACleanup();
   return 0;
}
```

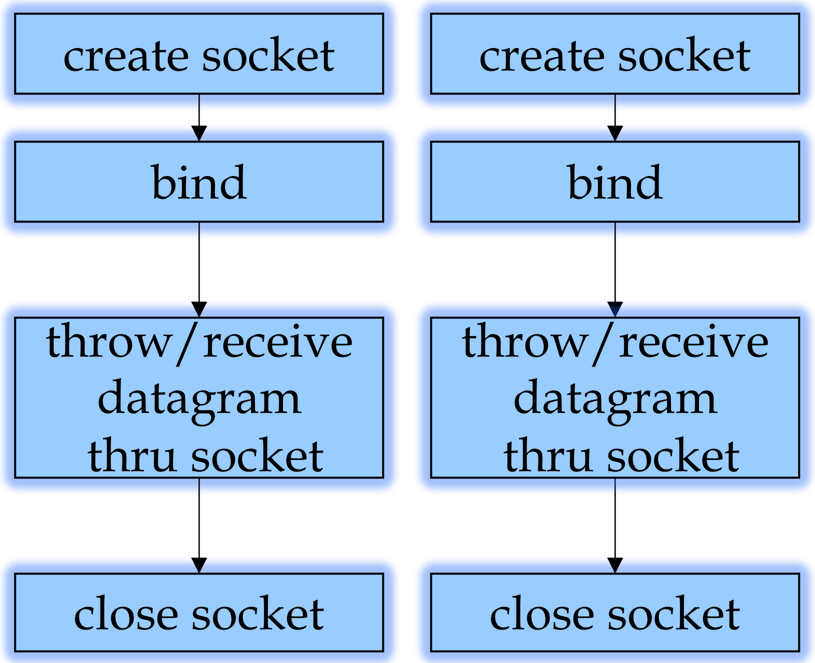
UDP Server


create socket
bind
throw/receive datagram thru socket
close socket
create socket
bind
throw/receive datagram thru socket
close socket

connectionless
(SOCK_DGRAM)

UDP Client

```
#include <winsock.h>
#define MY_PORT 3434
int main() {
   SOCKET udp_sock;
   struct sockaddr_in remote_addr, local_addr;
   char *buffer = "How do you do?";
   WSADATA wsaData;
   WSAStartup(MAKEWORD(2,1),&wsaData);
   udp_sock = socket(AF_INET, SOCK_DGRAM, 0);

   local_addr.sin_family = AF_INET;
   local_addr.sin_port = htons(MY_PORT);
   local_addr.sin_addr.s_addr = INADDR_ANY;

   bind(udp_sock, (sockaddr *)&local_addr, sizeof(struct
sockaddr_in));

   remote_addr.sin_family = AF_INET;
   remote_addr.sin_port = htons(MY_PORT);
   remote_addr.sin_addr.s_addr = inet_addr("137.189.90.38");

   for (int i = 0; i < 10; i++)
        sendto(udp_sock, buffer, strlen(buffer), 0, (struct sockaddr
*)&remote_addr, sizeof(struct sockaddr_in));
   closesocket(udp_sock);
   WSACleanup();
   return 0;
}
```


create socket
create socket
bind
bind
throw/receive datagram thru socket
throw/receive datagram thru socket
close socket
close socket

connectionless
(SOCK_DGRAM)