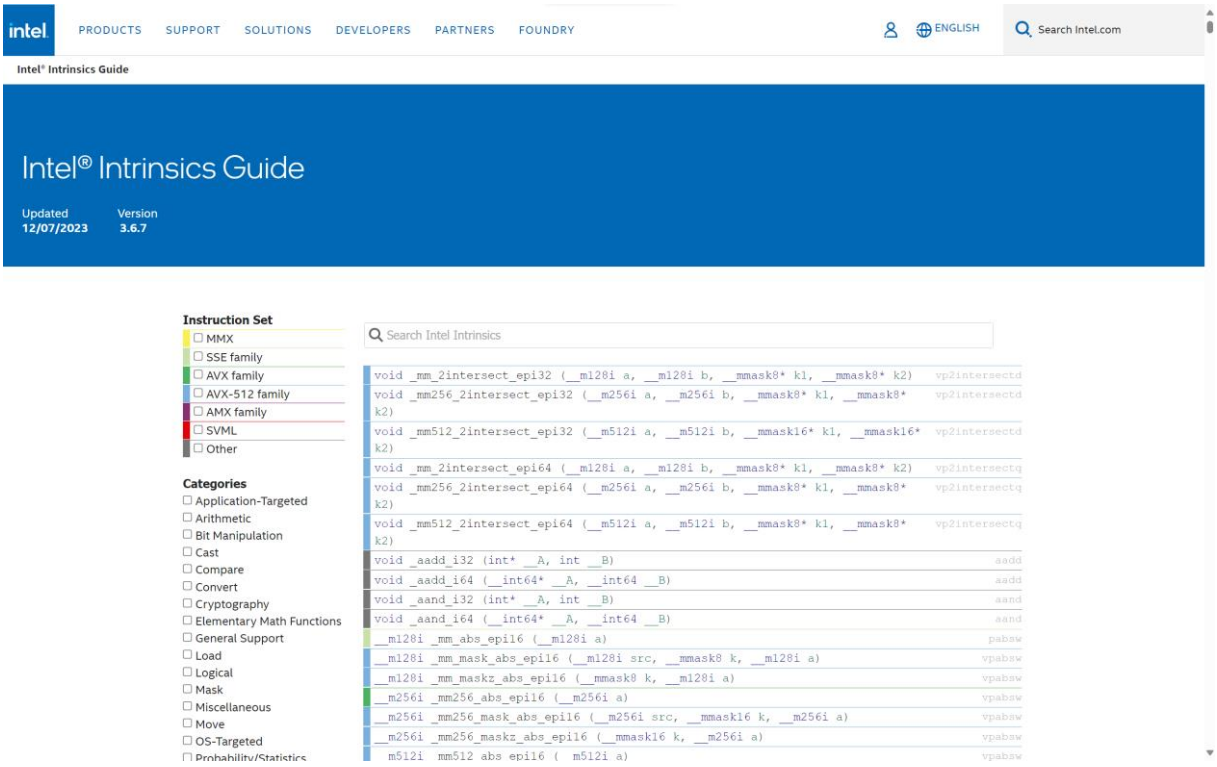
CS120 Final Report

葛书源 信息科学与技术学院　许家睿 信息科学与技术学院

Fundemental Designs:

As is known to all, C++ provides direct control over hardware, allowing programmers to manage computing resources more efficiently, whose capability enables C++ to generate efficient machine code, achieving outstanding performance. Additionally, C++ offers a rich Standard Template Library (STL) that includes various generic data structures and algorithms. The STL enhances code reusability while providing efficient implementations.

C++ is chosen as our CS120 projects' programming language owing to its delivering outstanding performance given features like static-typed data and convenience to further optimization acquired from ready-made frameworks such as OpenMP and SIMD instruction sets(in terms of the concrete and actual appliance, the Intel Intrinsics) which provides convenient multi-threaded and parallel operations and could be instrumental for upper implementation.



In the context of exploring subsequent projects involving schemes for latency, we made several attempts further on distinctive sound card drivers beyond ASIO4ALL. Specifically, we delved into WASAPI(Windows Audio Session API) which is a audio interface provided by Microsoft having underlying access to audio devices and produces theoretically lower latency while achieveing

exclusive output simultaneously.

关于 WASAPI

项目 • 2023/06/13 • 3 个参与者 反馈

Windows 音频会话 API (WASAPI) 使客户端应用程序能够管理应用程序和 音频终结点设备之间的音频数据流。

头文件 Audioclient.h 和 Audiopolicy.h 定义 WASAPI 接口。

每个音频流都是 音频会话的成员。 通过会话抽象，WASAPI 客户端可以将音频流标识为一组相关音频流的成员。 系统可以将会话中的所有流作为单个单元进行管理。

音频引擎是 用户模式音频组件， 应用程序通过该组件共享对音频终结点设备的访问。 音频引擎在终结点缓冲区和终结点设备之间传输音频数据。 若要通过呈现终结点设备播放音频流，应用程序会定期将音频数据写入呈现终结点缓冲区。 音频引擎混合来自各种应用程序的流。 若要从捕获终结点设备录制音频流，应用程序会定期从捕获终结点缓冲区读取音频数据。

WASAPI 由多个接口组成。 其中第一个是 IAudioClient 接口。 若要访问 WASAPI 接口，客户端首先通过调用 IMMDevice：：Activate 方法获取对音频终结点设备的 IAudioClient 接口的引用，并将参数 *iid* 设置为 REFIID IID_IAudioClient。 客户端调用 IAudioClient：：Initialize 方法以初始化终结点设备上的流。 初始化流后，客户端可以通过调用 IAudioClient：：GetService 方法获取对其他 WASAPI 接口的引用。

如果客户端应用程序使用的音频终结点设备无效，WASAPI 中的许多方法返回错误代码 AUDCLNT_E_DEVICE_INVALIDATED。 通常，应用程序可以从此错误中恢复。 有关详细信息，请参阅 从Invalid-Device错误中恢复。

Project 1 & Project 2:

Project 1 :

For Project 1, we experimented on various modulation and corresponding demodulation methods for preamble and physical payload, within which we found most are hard-pressed to comply with acoustic transmission requirement. For instance, FSK-generated signal in frequency domain after FFT transition was dissimilar to the ideal case and couldn't offer a limpid guidance for integrating a feasible method from clues after inumerable attempts, hence we came up against enormous setback.

As regards to the most suitable one, PSK is prefered in this case given its relative stability in comparison with FSK and ASK. It is to be regretted that we didn't notice the provided sample code and that has led to our inability after much miscellaneous and duplicated labour to discover a mature method for detecting preambles and it wasn't until Project 2 that it was tackled.

Project 2 : equipments represent setbacks

We were chuffed for swapping from sound to electric singals and line-coding(initially basic Manchester encoding chosen) in Project 2, where one 0/1 bit corresponds to four samples under 48'000

sampling rate in the prime design. Despite singals audio-cabled, the major obstacle yet appeared to be the physical equipments, which exhibits wierd characteristics: issues such as warming up sound card and prolonged high level would lead to autonomous convergence of electric signal to 0, nevertheless those cases could be manually solved at implementation level, supposing we introduce sufficient jumps and slicing data segments into minor payloads; still among those problems some were rather uncontrollable and perpetuated till the end of the semester, such as that the oscillation shrunk proportionally to a minimal value without abnormal and inappropriate operation and was posing significant variance subjected to whether the power supply was plugged in or not, which expended us much redundant and repetitive effort adjusting thresholds. We figured the proper reason for those were equipment related, diversities in laptops for instance.

We didn't arrive at a flawless and gratifying CSMA/CD protocal within the physical layer until Project 4, when we had eventually and entirely adapted to devices. Surely, we are still subjected to the prior unpredictable issues persisting. In the final version of physical layer, we utilized 4B/5B encoding method and one transmitted bit corresponed to 2 samples(48'000 sampling rate) with shift correction. We attained most aspects of CSMA/CD protocal: carrier sense, ACK timeout, and still more managed to implement timeout retrasmission, which overall can bear the load of mutual transmission containing redirected traffic from virtual net adaptor to physical interface including the bulk of packets(ICMP, TCP and UDP in IPV4, ARP).

Suggestions:

The two projects' existence concerning its correlation to and irreplaceability for subsequent projects is indisputable, yet it is undeniable that they were somehow too troublesome in practical implementation and we conquerred untold hardships to get over it. I suppose more concrete documentation apart from JUCE library introductions, such as more tangible drafts and skeleton frameworks, forums with students of CS120 from previous semesters participating in case any concrete assistance is required, documentations of probable problems and remedies bound up with specified in different programming languages and so forth.

An anology is that the JUCE libruary takes much over time while compiling and generating targets in Visual Studio, hence to mitigate this we forged its static linked lib and saved substantially over half a minute for each compilation, which could came as an aid for students of CS120 thereafter. Furthermore, our discussions unveiled a noteworthy observation that the DEBUG mode performance of proceeding JUCE libruary in Visual Studio is inferior to that in RELEASE mode, which could serve as experience of values.

In summary, if I were to retake this course, my greatest hope would be that when we were implementing physical layer functions, a more flat path could be paved rather than being hindered incessantly, nonetheless we managed to overcome the obstacle.

Project 3:

The IP layer is the cornerstone of the Internet, enabling communication among different computers and network devices globally. By assigning a unique IP address to each device, the IP layer achieves a worldwide unique identification, facilitating seamless connectivity between devices worldwide. Additionally, it provides an abstraction of protocol independence, allowing different transport layer protocols (such as TCP, UDP) to share the same network infrastructure. This flexibility makes the IP layer suitable for various types of applications and services, laying the groundwork for the upcoming project content.

The substances in Project 3 allowed us to penetrate further into computer network, which, as far as I am concerned, is process-delightful. We were using npcap to simulate virtual network cards in the successive two projects.

Technical Details:

We implemented a substantial part in this project, including ICMP packet generation, router logic with respect to various protocals in network layer and transport layer as we dipped into in courses, NAT table in routers, incoorperating virtual network devices and so on.

ICMP Echo served as a good start for us to enter the domain of network layer and get familiar with usages of requisite test tools like Wireshark. Equally, it was helping us overcome any difficulties in validating the implementation of IP protocal. We had met up with little setbacks given the stability of campus WLAN and usability of npcap, therefore once a packet due to appear was not collected, it was undisputed that something was wrong with checksum. For our NAT table, it met our expectations and was running in consistence with true static NAT conversion logic, compatible to TCP/UDP and ICMP for Task 3.

For the virtual network devices, it demanded a high-performance physical interface to deal with distribution of massive traffic from virtual net adaptors, yet we managed to fulfiled it. There is solely one thing regretful that we our CSMA/CD, in accordance with our demand, involves timeout transmission and therefore took a slightly longer time than a simplified version during our expirimental procedure and didn't satisfy the Task 8(Router) time limit.

Suggestions:

The design and practical contents of Project 3 is impeccable and I don't think much room for improvement yet remains, and I deemed that I found my pleasure in it. Therefor no suggestion would I pose on it.

Project 4:

Among Project 4, we might assert that a naive yet functional network had been achieved, which was truly encouraging and was giving a sense of achievement. In Project 4, we stepped further into the transmit layer, familiarizing ourselves to DNS(UDP) and HTTP(TCP), once again delving into concepts in CS120 lectures and understanding through hands-in practice.

For Task 1, we firstly implemented DNS pakcet producing and resolving methods by manual, which can properly interpret the DNS response and thereafter ping the acquired IP address unwindingly. Checksum as before is sheerly the only one which took some time to debug. Afterwards, we made a sucessful attempt to utilize the system's ping method to check reachability through our route program.

For Task 2, we did came across a minor problem. It is that we initially tried to utilize the system curl again, yet it is not feasible in terms of modification of squence numbers, e.t.c, the sequence number of DNS packets captured on node 2 should be something like 0x12345678, and exploiting the system curl to resolve the problem brought about by TCP connection oriented and reliable transmission somehow deviated from the initial intention of Task 2. Therefore we had to implement a naive triple handshake protocal and a abbreviate and tailored sliding window to accept unordered incoming HTTP packets. The last unresolved hard nut to crack is Part 5, where we were supposed to browsing the web, yet commonly didn't receive reply often from http://www.example.com. We make attempts such as monopolizing one specific port before running the main program but it wasn't to avail. In spite of the opinion that we err on TCP format conventions, we assumed that we were with the absense of sufficient awareness of TCP deeper insights like Protocal Stack and so on, and that left over for us a subject worthy of pondered over in the future.

Suggestion:

Once again, I found myself deeply engrossed in the intricacies of Project 4, much like in Project

3. The essence of Project 4 was a substantial extension of knowledge acquired from lectures. The prospect of upgrading from conventional sound cards to more efficient equipment, allowing us to leverage the protocols we implemented and explore additional functionalities, such as utilizing UDP for rendering a video stream, remained nothing more than a tantalizing notion. In summary, the realization of the network layer and transmission layer proved to be a delightful experience.