

计算机网络课程设计

目录

计算机网络课程设计

- 目录
- 项目概述
- 开发环境与成员分工
 - 成员分工
 - 开发环境系统信息
 - 编程语言、依赖与工具链
- 系统功能设计
 - 概述
 - 指定dnsrelay功能
 - 不良网站拦截功能
 - 查询服务器功能
- 模块划分与软件流程图
- 协议实现
 - DNS报头结构
 - DNS资源记录
 - DNS消息结构
 - 域名编码与解码
 - 资源记录打包与解包
 - DNS消息打包与解包
 - DNS查询与响应
 - 协议实现总结
- 高性能事件循环与网络请求处理
 - 事件循环的初始化
 - 创建UDP服务器
 - 数据接收回调函数
 - 处理DNS查询
 - DNS查询与缓存
 - 黑名单的处理
 - 异步DNS查询
 - 总结
- 缓存实现
 - 字典树 (Trie)
 - 最近最少使用 (LRU) 算法
 - 缓存的创建与销毁
 - 插入缓存
 - 获取缓存
 - 优点与效果
- 命令行参数解析
 - 配置命令行选项
 - 定义配置结构体
 - 解析命令行参数
 - 打印配置参数
 - 总结
- 日志系统
 - 初始化日志系统
 - 调试级别
 - 日志系统的优势
 - 总结
- 测试用例及运行结果

拦截功能
Windows
Linux
Wireshark
程序输出调试信息
服务器功能
Windows
Linux
Wireshark
程序输出调试信息
中继服务器功能
Windows
Linux
Wireshark
程序输出调试信息
长期连续使用
压力测试
调试中遇到并解决的问题
总结和心得体会
心得

项目概述

本项目以TCP/IP协议栈为基础，使用UDP通信机制，通过Socket编程接口API，用C语言实现DNS中继服务器软件，包含，并用Wireshark、nslookup等工具进行调试。理论上本项目支持Windows、Linux和MacOS全平台运行，但是只在Windows和Linux的Ubuntu发行版上进行了测试。

开发环境与成员分工

成员分工

- 协议定义、解析与网络交互逻辑
 - 基于事件循环的UDP服务端
 - 高性能异步日志系统
 - 实验报告撰写
-
- 命令行参数解析
 - 字典树数据结构
 - 缓存的LRU算法
 - 实验报告撰写

开发环境系统信息

- Windows 11 23H2
- Ubuntu 18.04.3 LTS
- Ubuntu 22.04.3 LTS

编程语言、依赖与工具链

- 语言
 - 全程采用C语言，穿插C相关的CMake等基础设施语言
 - 具体地，在ISO C11标准下经过测试
- 依赖的外部库
 - libhv: Libhv is a C/C++ network library similar to libevent/libuv
 - cargs:: A lightweight cross-platform getopt alternative that works on Linux, Windows and macOS. Command line argument parser library for C/C++. Can be used to parse argv and argc parameters.
- 工具链
 - CMake
 - Ninja
 - MSVC19.36.32532 (Windows 11 23H2)
 - GCC13.1.0 (Ubuntu 18.04.3 LTS)
 - GCC9.4.0 (Ubuntu 22.04.3 LTS)

系统功能设计

概述

项目主题是基于C的DNS中继服务器实现。中继服务器相对于客户端是DNS服务器，相对于上游DNS服务器是下游服务器。

实际包含了不良网站拦截功能、查询服务器功能、中继服务器功能（含缓存）。

实现这些功能经由协议实现、网络请求与接收、高性能事件循环与异步处理、缓存、命令行参数解析、输出测试日志等模块。

指定dnsrelay功能

按照需求，软件在启动时可以读取dnsrelay文件，其中包含若干IPv4地址与域名的对应。文件包含若干行，每行的格式为 `[addr] [domain]`，该功能是下面功能的基础。

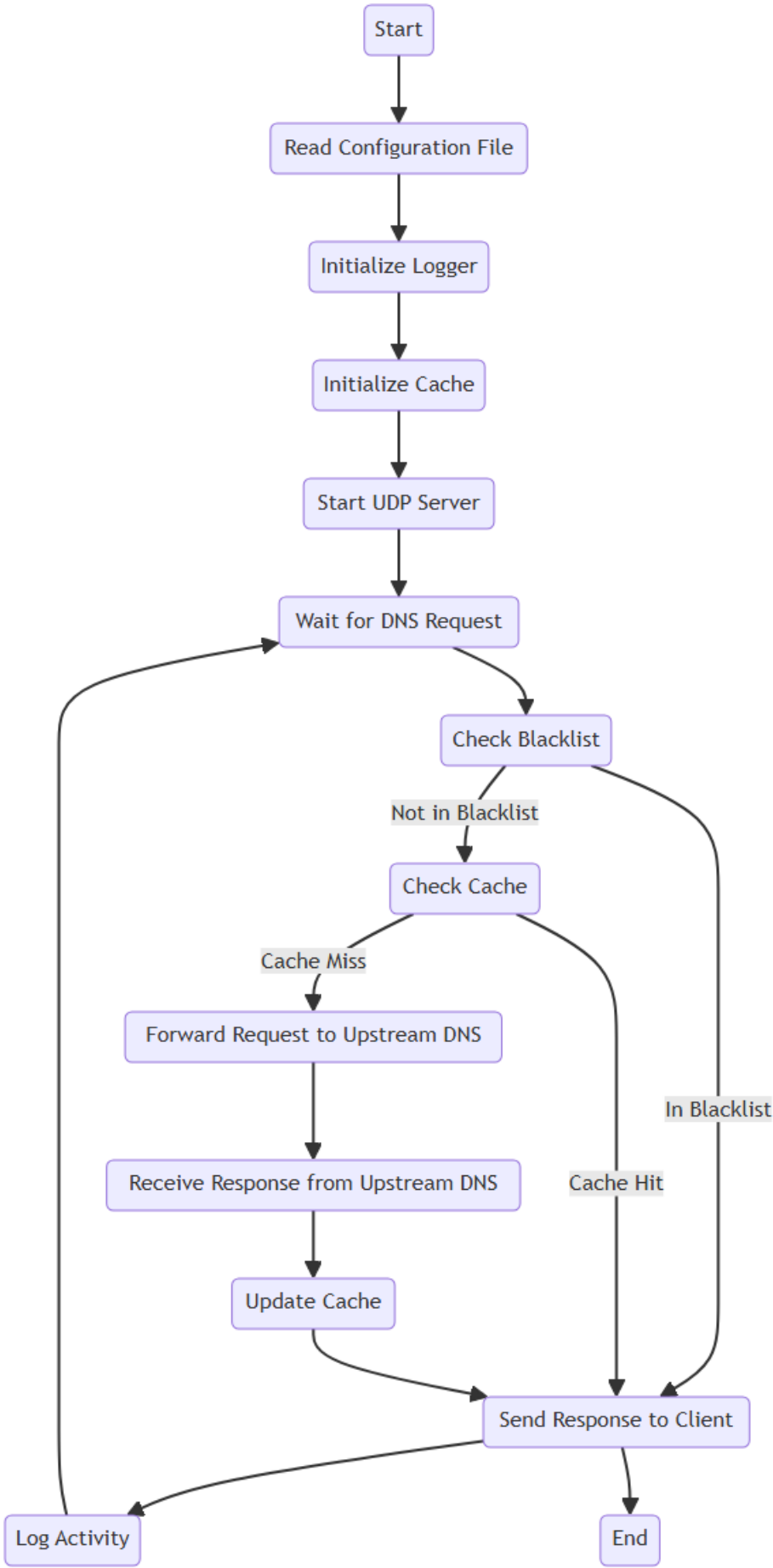
不良网站拦截功能

如果dnsrelay文件中某一行为 `0.0.0.0 www.example.com`，那么将 `www.example.com` 视为不良网站，对于询问其的DNS A和AAAA类型请求，返回域名不存在（NXDOMAIN / Name Error）。其他类型询问不处理，向上游服务器询问。

查询服务器功能

如果dnsrelay文件中某行的 `[addr]` 字段不为 `0.0.0.0`，那么对于询问 `[domain]` 的A类型请求，相应内容包含一条包含地址 `[addr]` 的非权威回答RR。

模块划分与软件流程图



项目由协议实现、网络请求与接收、高性能事件循环与异步处理、缓存、命令行参数解析、输出测试日志等模块组成，接下来将分别介绍各个模块。

协议实现

在本项目中，我们实现了DNS协议的解析、封装以及传输，涵盖了DNS查询、响应和资源记录的处理。通过这些实现，我们能够接收来自客户端的DNS查询请求，解析请求内容，向上游DNS服务器转发请求，并最终将响应返回给客户端。以下是我们协议实现部分的详细说明。

DNS报头结构

DNS报头用于存储DNS消息的元数据，包括事务ID、标志位、问题数目、回答数目等。我们定义了一个 `dnshdr_t` 结构体来表示DNS报头，该结构体包含以下字段：

- `transaction_id`：事务ID，用于匹配请求和响应。
- 标志字段：包含查询/响应标志、操作码、授权回答、截断消息、期望递归等多个标志。
- `nquestion`：问题数目。
- `nanswer`：回答数目。
- `nauthority`：权威记录数目。
- `naddtional`：附加记录数目。

DNS资源记录

DNS资源记录包含域名的详细信息，如IP地址、名称服务器、邮件交换记录等。我们定义了一个 `dns_rr_t` 结构体来表示资源记录，该结构体包含以下字段：

- `name`：原始域名，例如：www.example.com。
- `rtype`：记录类型，例如A记录、AAAA记录等。
- `rclass`：记录类，一般为互联网（IN）。
- `ttl`：生存时间。
- `datalen`：数据长度。
- `data`：数据指针，指向具体的资源记录数据。

DNS消息结构

DNS消息由报头、查询问题、回答记录、权威记录和附加记录组成。我们定义了一个 `dns_t` 结构体来表示DNS消息，该结构体包含以下字段：

- `hdr`：DNS报头。
- `questions`：查询问题数组。
- `answers`：回答记录数组。
- `authorities`：权威记录数组。
- `addtionals`：附加记录数组。

域名编码与解码

为了在DNS协议中传输域名，需要将普通格式的域名（如www.example.com）转换为DNS格式（如3www7example3com）。我们实现了以下函数：

- `dns_name_encode`：将域名从普通格式转换为DNS格式。
- `dns_name_decode`：将DNS格式的域名转换回普通格式。

资源记录打包与解包

在DNS消息的传输过程中，需要将资源记录打包成二进制格式，并在接收到消息后解包成结构化格式。我们实现了以下函数：

- `dns_rr_pack`：将DNS资源记录打包成二进制格式。
- `dns_rr_unpack`：将二进制格式的资源记录解包成结构化格式。

DNS消息打包与解包

DNS消息的传输同样需要打包和解包，我们实现了以下函数：

- `dns_pack`：将整个DNS消息打包，包括报头和所有资源记录。
- `dns_unpack`：将接收到的DNS消息解包成结构化格式，供进一步处理。

DNS查询与响应

我们实现了一个函数 `dns_query`，用于发送DNS查询消息并接收响应消息。该函数包括创建套接字、设置超时、发送和接收数据，以及解包响应消息。

为了提高性能，我们还实现了异步DNS查询函数 `dns_query_async`，用于在非阻塞模式下发送DNS查询并接收响应。

协议实现总结

通过以上各个部分的实现，我们完成了DNS协议的解析和封装，使得我们的DNS中继服务器能够正确处理DNS请求和响应。在实际运行中，当服务器接收到客户端的DNS请求时，会首先检查缓存和黑名单，如果命中缓存或黑名单，则直接返回结果；否则，向上游DNS服务器转发请求并缓存结果。这样既保证了查询的效率，又能够根据需要拦截不良网站。

高性能事件循环与网络请求处理

在我们的DNS服务器实现中，使用libhv库来实现高性能的事件循环和网络请求处理。libhv是一个高效的C/C++网络库，支持多平台和多种协议。通过libhv，我们能够以异步和事件驱动的方式处理DNS请求，从而提高服务器的并发性能和响应速度。以下是我们在项目中具体实现高性能事件循环与网络请求处理的详细流程。

事件循环的初始化

首先，我们在 `dns_server_init` 函数中初始化事件循环。通过调用 `hloop_new` 函数创建一个新的事件循环实例，并设置 `HLOOP_FLAG_AUTO_FREE` 标志，这样在事件循环结束时，会自动释放内存。事件循环是整个服务器的核心，它负责管理所有的网络I/O操作和定时任务。

创建UDP服务器

接下来，我们使用 `hloop_create_udp_server` 函数创建一个UDP服务器，绑定到指定的端口。这个函数会返回一个I/O对象（`hio_t`），我们可以通过这个对象来管理UDP服务器的各种属性和操作。然后，我们通过 `hio_set_context` 将服务器上下文设置到I/O对象中，便于在回调函数中访问服务器的配置和状态。最后，我们设置了一个读取回调函数 `on_recv`，并调用 `hio_read` 开始读取数据。

数据接收回调函数

当有新的DNS请求到来时，事件循环会触发 `on_recv` 回调函数。在这个回调函数中，我们首先获取客户端的地址信息，然后将接收到的数据解包成DNS查询消息。如果解包成功，我们会调用 `on_dns_query` 函数进一步处理DNS查询。回调函数的设计使得每一个接收到的请求都能被异步处理，这样可以有效地提高服务器的并发性能。

处理DNS查询

在 `on_dns_query` 函数中，我们处理具体的DNS查询逻辑。首先，我们初始化响应消息，并复制查询问题到响应中。然后，我们检查查询的域名是否在黑名单中，如果在黑名单中，则直接返回NXDOMAIN响应。如果不在黑名单中，我们先检查缓存，如果命中缓存，则返回缓存中的响应；如果没有命中缓存，则执行DNS查询。

DNS查询与缓存

为了提高查询效率，我们使用了LRU缓存。LRU缓存使用最近最少使用策略来管理缓存内容，使得缓存能够适应频繁的查询请求。我们在 `perform_dns_lookup` 函数中进行DNS查询，并将查询结果存入缓存。如果查询类型是A记录，我们将第一个IPv4地址存入缓存。如果是AAAA记录，我们将IPv6地址存入缓存。

黑名单的处理

我们通过加载一个包含不良网站列表的文件来实现黑名单功能。文件中的每一行包含一个IP地址和一个域名，IP地址为0.0.0.0的域名被视为不良网站。我们在服务器初始化时加载这个文件，并将不良网站列表存入黑名单缓存。在处理查询时，如果查询的域名在黑名单中，则直接返回NXDOMAIN响应。

异步DNS查询

为了避免阻塞服务器主线程，我们还实现了异步DNS查询。在异步查询中，我们使用非阻塞的I/O操作来发送和接收DNS查询，这样可以在等待上游DNS服务器响应时继续处理其他请求。异步查询的回调函数在收到上游响应时被触发，并处理响应数据。

总结

通过libhv的高效事件循环和异步I/O机制，我们的DNS服务器能够以高并发和低延迟的方式处理大量的DNS请求。事件循环负责管理所有的I/O事件和定时任务，使得服务器能够在单线程中处理多个客户端请求。结合LRU缓存和黑名单功能，我们的DNS服务器能够在提高查询效率的同时，提供不良网站拦截功能，增强了系统的安全性和稳定性。

缓存实现

在我们的DNS服务器实现中，为了提高查询效率并减少对上游DNS服务器的依赖，我们引入了缓存机制。缓存的设计目标是存储最近的查询结果，以便在同样的查询再次出现时能够快速响应。我们的缓存系统结合了字典树（Trie）和最近最少使用（LRU）算法，以实现高效的域名查找和缓存管理。

字典树（Trie）

字典树是一种树形数据结构，特别适用于存储字符串。它具有快速查找的特性，非常适合用于域名的存储和查询。在我们的实现中，字典树的每一个节点（`trie_node_t`）都包含一个指向LRU节点（`lru_node_t`）的指针，这样我们可以将缓存的值直接关联到对应的域名。我们在存储时会将链接进行逆序，因为大多数链接的后缀可能会相同，因此使用Trie可以有效地节省空间并提高查找效率。Trie会使用

相对大量的空间，但查询任何链接的复杂度都是 $O(|s|)$ （其中 s 为键），即只与链接长度有关且呈线性，而与已经存储的信息数量无关。

字典树节点包含以下属性：

- `children[N]`：指向子节点的指针数组，N为可能字符的数量（38个，包含数字、字母和特殊字符）。
- `lru_node`：指向对应的LRU节点。
- `is_end_of_word`：标记该节点是否是一个完整的域名。

字典树的作用是提供快速的域名查找功能。当我们需要查找某个域名是否在缓存中时，通过字典树可以迅速定位到对应的节点。

最近最少使用（LRU）算法

LRU算法是一种常用的缓存淘汰策略。它通过维护一个双向链表，记录缓存的访问顺序，每次访问都会将对应的节点移动到链表的头部，表示最近访问。链表的尾部则表示最久未访问的节点，当缓存达到容量限制时，最久未访问的节点会被移除。

LRU节点包含以下属性：

- `key`：缓存项的键（域名）。
- `value`：缓存项的值（IP地址）。
- `prev`：指向前一个节点的指针。
- `next`：指向后一个节点的指针。

在我们的实现中，每次插入新的缓存项时，都会检查缓存是否已经达到容量限制，如果是，则移除链表尾部的节点。同时，将新的节点插入到链表头部。这样可以确保缓存中的项始终是最近使用的那些。

缓存的创建与销毁

缓存的创建通过 `cache_create` 函数实现，它初始化字典树的根节点和LRU链表的头尾指针，并设置缓存的容量和当前大小。缓存的销毁则通过 `cache_destroy` 函数实现，它递归释放字典树中的所有节点，并逐个释放LRU链表中的所有节点，最终释放缓存结构本身。

插入缓存

插入缓存的过程通过 `cache_insert` 函数实现。首先，通过字典树定位到对应的节点，如果节点已经存在于缓存中，则更新其值，并将对应的LRU节点移动到链表头部。如果节点不存在，则创建新的字典树节点和LRU节点，插入到字典树和LRU链表中。如果缓存已满，则移除链表尾部的节点。

获取缓存

获取缓存的过程通过 `cache_get` 函数实现。首先，通过字典树查找对应的节点，如果节点存在且包含LRU节点，则将该LRU节点移动到链表头部，表示最近访问，并返回对应的值。如果节点不存在或不包含LRU节点，则返回NULL，表示缓存未命中。

优点与效果

通过结合字典树和LRU算法，我们的缓存系统具备以下优点：

- **快速查找**：字典树提供了高效的字符串查找功能，可以迅速定位到缓存的域名。

- **自动淘汰**：LRU算法保证了缓存中的项始终是最近使用的那些，自动淘汰最久未使用的项，确保缓存利用率。
- **空间高效**：通过双向链表和指针的使用，缓存系统在保证高效查找和淘汰的同时，尽量减少了内存占用。

整体上，通过缓存机制的引入，我们的DNS服务器在处理重复查询时可以显著提高响应速度，减少对上游DNS服务器的查询次数，从而提升系统的性能和稳定性。

命令行参数解析

命令行参数解析是我们的DNS中继服务器的重要组成部分。通过命令行参数解析，我们可以灵活地配置服务器的行为，包括指定DNS服务器地址、端口号、缓存大小等。为了实现这一功能，我们使用了 `cargs` 库，这是一个轻量级的命令行参数解析库，适用于C/C++项目。

配置命令行选项

首先，我们定义了一个包含所有可用命令行选项的数组 `options`。每个选项包含以下属性：

- `identifier`：选项的唯一标识符，用于在解析过程中区分选项。
- `access_letters`：选项的短名称（单字符）。
- `access_name`：选项的长名称（字符串）。
- `value_name`：选项值的名称（用于显示帮助信息）。
- `description`：选项的描述信息（用于显示帮助信息）。

通过这种方式，我们可以清晰地定义每个命令行选项的用途和功能。例如：

- `-d` 或 `--debug`：设置调试级别为1。
- `-m` 或 `--verbose`：设置调试级别为2。
- `-s` 或 `--server`：指定DNS服务器的IP地址。
- `-f` 或 `--filename`：指定配置文件的路径。
- `-p` 或 `--port`：指定服务器的端口号。
- `-t` 或 `--timeout`：指定请求上级DNS服务器的超时时间。
- `-c` 或 `--cache`：指定缓存的最大数量。
- `-h` 或 `--help`：显示帮助信息并退出。

定义配置结构体

我们定义了一个 `Config` 结构体，用于存储解析后的配置信息。结构体包含以下成员：

- `debug_level`：调试级别。
- `port`：服务器端口号。
- `cache_size`：缓存大小。
- `rto`：请求超时时间。
- `dns_server_ipaddr`：DNS服务器的IP地址。
- `filename`：配置文件路径。

解析命令行参数

解析命令行参数的主要函数是 `parse_args`，它接收命令行参数和一个 `Config` 结构体指针作为输入，并将解析结果存储到结构体中。解析过程如下：

1. 初始化默认配置：在解析之前，我们为配置结构体设置默认值，以便在用户未指定某些选项时使用默认配置。例如，默认端口号为53，默认缓存大小为2048，默认请求超时时间为5000毫秒。
2. 准备解析器：使用 `cag_option_prepare` 函数初始化解析器。
3. 逐个解析选项：使用 `cag_option_fetch` 函数逐个获取选项，并根据选项的 `identifier` 设置对应的配置值。对于需要值的选项（如 `--server` 和 `--filename`），使用 `cag_option_get_value` 获取值并存储到配置结构体中。
4. 设置默认值：如果用户未指定某些选项（如 DNS服务器地址和配置文件路径），我们设置默认值。
5. 打印帮助信息：如果用户指定了 `--help` 选项，显示帮助信息并退出。

打印配置参数

为了便于调试和验证配置，我们提供了 `dump_args` 函数，用于打印当前的配置参数。当调试级别为2时，解析完命令行参数后调用该函数，打印所有配置参数。

总结

通过命令行参数解析，我们可以灵活地配置DNS中继服务器的各种行为，确保服务器能够根据不同的需求进行调整。`cargs` 库简化了命令行参数解析的实现，使代码更加简洁和易于维护。同时，通过合理的默认值设置和帮助信息提示，用户可以方便地使用和配置我们的DNS中继服务器。

日志系统

日志系统在我们的DNS中继服务器项目中扮演着至关重要的角色，在调试和性能优化过程中提供了很多有价值的信息。在本项目中，我们采用了 `libhv` 提供的 `hlog` 日志库，来实现一个灵活、易用和高性能的日志系统。

初始化日志系统

日志系统的初始化在 `init_logger` 函数中进行，该函数接收一个 `Config` 结构体指针作为参数。该结构体包含从命令行参数解析得到的配置信息。`init_logger` 函数的主要步骤如下：

1. **设置日志处理器**：调用 `hlog_set_handler` 函数，将日志处理器设置为标准输出（`stdout_logger`），即将日志信息输出到控制台。这样便于在开发和测试阶段实时查看日志信息。
2. **设置日志等级**：根据配置中的 `debug_level` 设置日志等级。`hlog` 提供了多种日志等级，包括 `ERROR`、`INFO` 和 `DEBUG`。通过切换不同的日志等级，我们可以控制日志的详细程度：
 - `LOG_LEVEL_ERROR`：仅记录错误信息。
 - `LOG_LEVEL_INFO`：记录一般信息和错误信息。
 - `LOG_LEVEL_DEBUG`：记录调试信息、一般信息和错误信息。
3. **启用日志颜色**：调用 `logger_enable_color` 函数启用日志颜色，使不同等级的日志信息显示不同的颜色，便于快速识别和分类。这个函数接收两个参数：日志处理器（`hlog`）和颜色模式（0表示默认模式）。
4. **设置日志格式**：调用 `hlog_set_format` 函数设置日志的输出格式。我们采用了一种标准的日志格式，包括时间戳、日志等级和日志消息。这有助于在分析日志时清晰地了解每条日志的产生时间和级别。格式字符串 `%y-%m-%d %H:%M:%S.%z %L %s` 的含义如下：

- `%y`: 年 (四位数)
- `%m`: 月
- `%d`: 日
- `%H`: 小时 (24小时制)
- `%M`: 分钟
- `%S`: 秒
- `%z`: 毫秒
- `%L`: 日志等级
- `%S`: 日志消息

调试级别

在 `init_logger` 函数中, 通过命令行参数解析得到的 `debug_level` 决定了日志系统的详细程度。不同的调试级别适用于不同的使用场景:

- **调试级别0**: 适用于生产环境, 仅记录错误信息, 减少日志的存储空间和处理开销。
- **调试级别1**: 适用于开发和测试环境, 记录一般信息和错误信息, 有助于了解程序的运行状态。
- **调试级别2**: 适用于调试阶段, 记录详细的调试信息、一般信息和错误信息, 有助于深入分析和排查问题。

日志系统的优势

通过日志系统, 我们可以获得以下优势:

- **实时监控**: 在控制台实时输出日志信息, 便于开发和测试阶段的即时监控。
- **问题排查**: 详细的调试信息和错误信息有助于快速定位和解决问题。
- **性能优化**: 通过分析日志中的性能相关信息, 可以发现并优化系统的性能瓶颈。
- **数据持久化**: 日志信息可以输出到文件或其他持久化存储中, 便于后续的分析 and 审计。

总结

日志系统是DNS中继服务器项目中的一个重要组成部分。通过合理设置日志等级和格式, 我们可以在不同的使用场景中灵活应用日志系统, 确保系统的可监控性和可维护性。`libhv` 提供的 `hlog` 日志库使得日志系统的实现更加简洁和高效。

测试用例及运行结果

拦截功能

根据需求文档, 我们实现了拦截功能。在 `dnsrelay.txt` 文件中写入 `0.0.0.0 www.baidu.com`, 进行如下测试, 测试包含Windows和Linux双平台, 以验证程序的良好跨平台性能。

Windows

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS C:\Users\ymj> nslookup www.baidu.com
服务器: UnKnown
Address: 127.0.0.1

*** UnKnown 找不到 www.baidu.com: Non-existent domain
PS C:\Users\ymj>
```

Linux

```
# fa_555 @ fa555-kooBcaM in ~ [17:49:20] C:1
$ nslookup -type=A www.baidu.com 127.0.0.1
Server: 127.0.0.1
Address: 127.0.0.1#53

** server can't find www.baidu.com: NXDOMAIN

# fa_555 @ fa555-kooBcaM in ~ [17:50:28] C:1
$ nslookup -type=AAAA www.baidu.com 127.0.0.1
Server: 127.0.0.1
Address: 127.0.0.1#53

** server can't find www.baidu.com: NXDOMAIN

# fa_555 @ fa555-kooBcaM in ~ [17:50:31] C:1
$ nslookup -type=CNAME www.baidu.com 127.0.0.1
Server: 127.0.0.1
Address: 127.0.0.1#53

** server can't find www.baidu.com: NXDOMAIN
```

Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
330	21.596117	fa555-kooBcaM.local	fa555-kooBcaM.local	DNS	63	Standard query 0x0208 A www.baidu.com
331	21.596289	fa555-kooBcaM.local	fa555-kooBcaM.local	DNS	138	Standard query response 0x0208 No such name A www.baidu.com SOA a.root-servers.net
422	24.934123	fa555-kooBcaM.local	fa555-kooBcaM.local	DNS	63	Standard query 0x6f38 AAAA www.baidu.com
423	24.934281	fa555-kooBcaM.local	fa555-kooBcaM.local	DNS	138	Standard query response 0x6f38 No such name AAAA www.baidu.com SOA a.root-servers.net
542	28.885252	fa555-kooBcaM.local	fa555-kooBcaM.local	DNS	63	Standard query 0x4be7 CNAME www.baidu.com
543	28.885428	fa555-kooBcaM.local	fa555-kooBcaM.local	DNS	138	Standard query response 0x4be7 No such name CNAME www.baidu.com SOA a.root-servers.net

程序输出调试信息

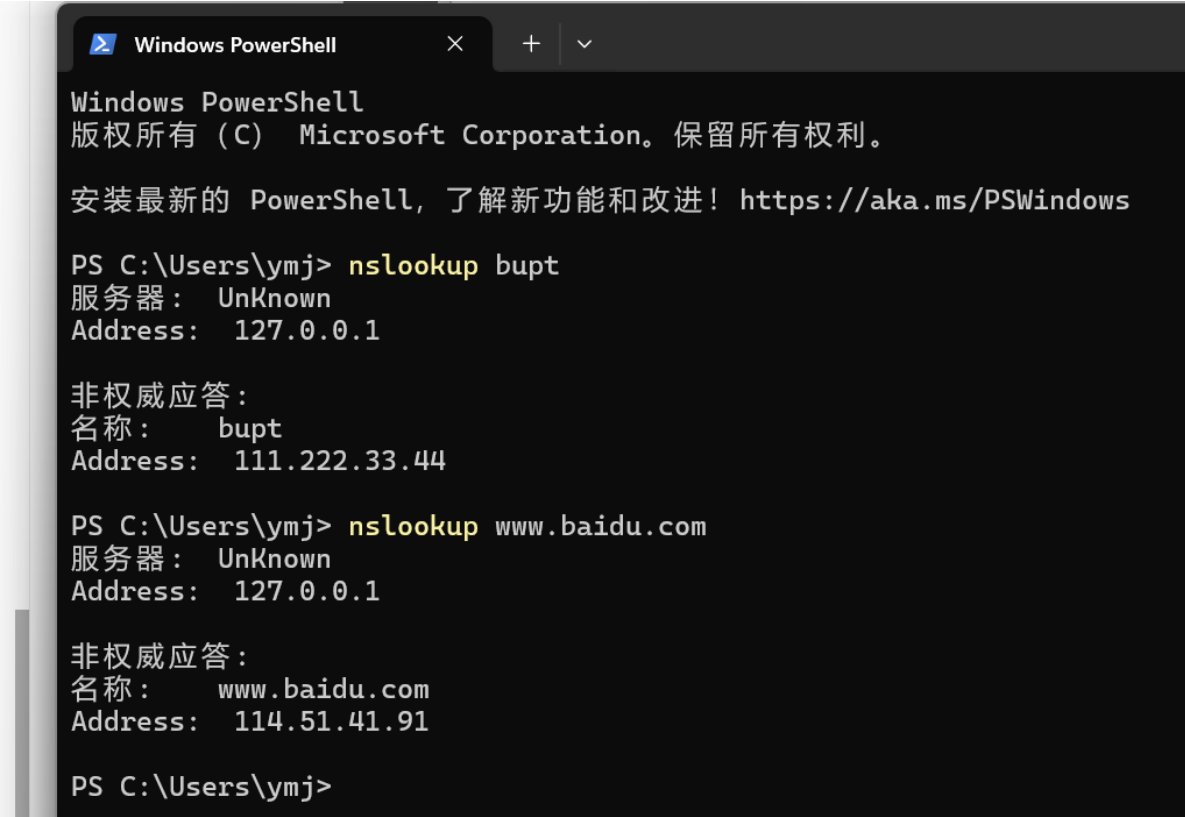
```
2024-06-30 19:38:35.1030 ERROR Not found: 1.0.0.127.in-addr.arpa [dns_server.c:142:on_dns_query]
2024-06-30 19:38:35.288 INFO Blacklisted: www.baidu.com [dns_server.c:141:on_dns_query]
2024-06-30 19:38:35.288 INFO Blacklisted: www.baidu.com [dns_server.c:141:on_dns_query]
2024-06-30 19:38:35.289 INFO Blacklisted: www.baidu.com [dns_server.c:141:on_dns_query]
2024-06-30 19:38:35.289 INFO Blacklisted: www.baidu.com [dns_server.c:141:on_dns_query]
2024-06-30 19:38:41.138 ERROR Not found: hw-v2-web-player-tracker.biliapi.net [dns_server.c:142:on_dns_query]
2024-06-30 19:38:41.147 ERROR Not found: hw-v2-web-player-tracker.biliapi.net [dns_server.c:142:on_dns_query]
2024-06-30 19:38:41.608 ERROR Not found: hw-v2-web-player-tracker.biliapi.net [dns_server.c:142:on_dns_query]
```

程序正常运行，结果正确，调试信息正常，抓包正常。

服务器功能

根据需求文档，我们实现了拦截功能。在dnsrelay.txt文件中写入 111.222.33.44 bupt 和 114.51.41.91 www.baidu.com，进行如下测试，测试包含Windows和Linux双平台，以验证程序的良好跨平台性能。

Windows



```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS C:\Users\ymj> nslookup bupt
服务器:  UnKnown
Address:  127.0.0.1

非权威应答:
名称:     bupt
Address:  111.222.33.44

PS C:\Users\ymj> nslookup www.baidu.com
服务器:  UnKnown
Address:  127.0.0.1

非权威应答:
名称:     www.baidu.com
Address:  114.51.41.91

PS C:\Users\ymj>
```

Linux

```
# fa_555 @ fa555-kooBcaM in ~ [18:22:29]
$ nslookup www.baidu.com 127.0.0.1
Server:          127.0.0.1
Address:         127.0.0.1#53

Non-authoritative answer:
Name:   www.baidu.com
Address: 114.51.41.91

# fa_555 @ fa555-kooBcaM in ~ [18:22:44]
$ nslookup bupt 127.0.0.1
Server:          127.0.0.1
Address:         127.0.0.1#53

Non-authoritative answer:
Name:   bupt
Address: 111.222.33.44
```

Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
53	3.828801	localhost	localhost	DNS	63	Standard query 0xf190 A www.baidu.com
54	3.828991	localhost	localhost	DNS	92	Standard query response 0xf190 A www.baidu.com A 114.51.41.91
134	7.849382	localhost	localhost	DNS	54	Standard query 0xc4d5 A bupt
135	7.849528	localhost	localhost	DNS	74	Standard query response 0xc4d5 A bupt A 111.222.33.44

程序输出调试信息

```
2024-06-30 20:08:31.372 INFO Cache hit: bupt [dns_server.c:123:on_dns_query]
2024-06-30 20:08:31.377 ERROR Not found: bupt [dns_server.c:142:on_dns_query]
2024-06-30 20:08:34.120 ERROR Not found: 1.0.0.127.in-addr.arpa [dns_server.c:142:on_dns_query]
2024-06-30 20:08:34.121 INFO Cache hit: www.baidu.com [dns_server.c:123:on_dns_query]
2024-06-30 20:08:34.126 ERROR Not found: www.baidu.com [dns_server.c:142:on_dns_query]
```

中继服务器功能

Windows

```
PS C:\Users\ymj> nslookup www.zhihu.com
```

```
服务器: UnKnown
```

```
Address: 127.0.0.1
```

```
非权威应答:
```

```
名称: www.zhihu.com
```

```
Addresses: 2408:8738:b000:8:40::13
```

```
2408:8752:e00:47:40::1b
```

```
2408:8774:1:1c::12d
```

```
2408:874c:1ff:51:31::17
```

```
2408:872b:e02:101:6c::3f
```

```
101.72.254.86
```

```
123.125.46.250
```

```
220.194.123.111
```

```
PS C:\Users\ymj>
```

Windows PowerShell

```
Address: 114.51.41.91
```

```
PS C:\Users\ymj> nslookup -type=AAAA www.zhihu.com
```

```
服务器: UnKnown
```

```
Address: 127.0.0.1
```

```
非权威应答:
```

```
名称: www.zhihu.com
```

```
Addresses: 2408:872b:e02:101:6c::3f
```

```
2408:8738:b000:8:40::13
```

```
2408:8752:e00:47:40::1b
```

```
2408:8774:1:1c::12d
```

```
2408:874c:1ff:51:31::17
```


Linux

```
# fa_555 @ fa555-kooBcaM in ~ [18:28:08]
$ nslookup www.zhihu.com 127.0.0.1
Server:           127.0.0.1
Address:          127.0.0.1#53

Non-authoritative answer:
www.zhihu.com      canonical name = www.zhihu.com.ipv6.dsa.dnsv1.com.
www.zhihu.com.ipv6.dsa.dnsv1.com canonical name = 1595096.sched.d0-dk.tdnssdp1.cn.
1595096.sched.d0-dk.tdnssdp1.cn canonical name = 1595096.51-65.cjt.cdnmt.ljy.aicdn2.com.
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.204.14.228
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.204.14.111
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.204.14.25
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 117.161.41.59
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 117.161.135.215
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.204.15.77
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 117.161.41.200
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.201.223.185
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 117.161.28.98
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.201.114.182
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.201.223.242
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 183.204.14.132
Name:   1595096.51-65.cjt.cdnmt.ljy.aicdn2.com
Address: 117.161.222.56
```

```
# fa_555 @ fa555-kooBcaM in ~ [18:28:53]
$ nslookup -type=AAAA www.zhihu.com 127.0.0.1
Server:           127.0.0.1
Address:          127.0.0.1#53

Non-authoritative answer:
www.zhihu.com      canonical name = www.zhihu.com.dsa.dnsv1.com.
www.zhihu.com.dsa.dnsv1.com canonical name = 1595096.sched.d0-dk.tdnssdp1.cn.
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:872b:e02:101:6c::3f
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:872b:e02:101:6c::f
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:8742:51fc:200::42
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:8744:205:1060::40
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:874f:b000:3:6c::b
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:874f:b000:3:6c::5b
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:8722:6140:1:40::18
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:8710:20:1022:3e::3
1595096.sched.d0-dk.tdnssdp1.cn has AAAA address 2408:8722:6140:1:40::2d
```


Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
798	5.560332	localhost	localhost	DNS	63	Standard query 0x422a AAAA www.zhihu.com
781	5.560554	10.28.189.62	10.3.9.44	DNS	73	Standard query 0x0002 AAAA www.zhihu.com
783	5.565009	10.3.9.44	10.28.189.62	DNS	410	Standard query response 0x0002 AAAA www.zhihu.com CNAME www.zhihu.com.dsa.dnsv1.com CNAME 1595096.sched.d0-dk.tdn-
799	5.565259	localhost	localhost	DNS	710	Standard query response 0x422a AAAA www.zhihu.com CNAME www.zhihu.com.dsa.dnsv1.com CNAME 1595096.sched.d0-dk.tdn-

程序输出调试信息

```
2024-06-30 20:18:23.785 ERROR Not found: 1.0.0.127.in-addr.arpa [dns_server.c:142:on_dns_query]
2024-06-30 20:18:23.793 INFO Cache insert: www.zhihu.com [dns_server.c:204:perform_dns_lookup]
2024-06-30 20:18:58.163 ERROR Not found: www.bilibili.com [dns_server.c:142:on_dns_query]
2024-06-30 20:18:58.167 INFO Cache insert: www.bilibili.com [dns_server.c:204:perform_dns_lookup]
2024-06-30 20:18:58.167 ERROR Not found: www.bilibili.com [dns_server.c:142:on_dns_query]
```

长期连续使用

我们在个人PC上运行编写的dns-relay程序，将系统DNS服务器设为本地回环，并正常进行日常网络访问操作。在不少于100小时的连续测试中，程序表现出极好的稳定性和可靠性，没有出现任何崩溃或无法工作的情况，正常网络访问和数据流量的处理不受任何影响。

在使用过程中，我们取样监测了程序的资源使用情况。CPU占用率长期稳定在 0~0.02% 之间；峰值CPU占用率也仅短暂上升到0.08%，展现出极低的计算负载。同时，四天后的内存占用仍然处在可接受的范围内（相较于其他DNS中继服务器较大，因为我们采取了复杂数据结构作为Cache，实现性能优先而非内存空间优先的策略）。这证明了我们的程序具有优秀的性能和良好的资源管理能力。

使用体验和资源占用统计都表明，我们的DNS中继服务器具有良好的鲁棒性和稳定性，可以在个人甚至小型网络环境中胜任DNS中继服务器，满足日常网络访问需求。同时，其对系统资源的低需求也使得它比较适合这样的用途。

压力测试

我们使用dnspperf工具测试该服务器短时间应对大量访问的能力。在1s内，无限制发送相同请求（至少需向上游服务器询问一次），ADRIFT处理了15631个请求，没有出现错误（如下图）。

```
$ dnspperf -d ./test/dnspperf.txt -l 1
DNS Performance Testing Tool
Version 2.13.0

[Status] Command line: dnspperf -d ./test/dnspperf.txt -l 1
[Status] Sending queries (to 127.0.0.1:53)
[Status] Started at: Sun Jul 2 17:15:16 2023
[Status] Stopping after 1.000000 seconds
[Status] Testing complete (time limit)

Statistics:

Queries sent:          15631
Queries completed:     15631 (100.00%)
Queries lost:          0 (0.00%)

Response codes:        NOERROR 15631 (100.00%)
Average packet size:   request 31, response 1411
Run time (s):          1.010846
Queries per second:    15463.285209

Average Latency (s):   0.006364 (min 0.000131, max 0.011888)
Latency StdDev (s):   0.002923
```

在10s内，无限制发送相同请求（至少需向上游服务器询问一次），平均每秒处理5328.6个请求，没有出现错误（如下图）。

```
$ dnsperf -d ./test/dnsperf.txt -l 10
DNS Performance Testing Tool
Version 2.13.0

[Status] Command line: dnsperf -d ./test/dnsperf.txt -l 10
[Status] Sending queries (to 127.0.0.1:53)
[Status] Started at: Sun Jul  2 17:26:42 2023
[Status] Stopping after 10.000000 seconds
[Status] Testing complete (time limit)

Statistics:

Queries sent:          53468
Queries completed:     53468 (100.00%)
Queries lost:          0 (0.00%)

Response codes:        NOERROR 53468 (100.00%)
Average packet size:   request 31, response 993
Run time (s):          10.034101
Queries per second:    5328.628843

Average Latency (s):   0.018710 (min 0.000128, max 0.050593)
Latency StdDev (s):   0.009806
```

在100s内，无限制发送相同请求（至少需向上游服务器询问一次），ADRIFT平均每秒处理1772.1个请求，没有出现错误（如下图）。

```
$ dnsperf -d ./test/dnsperf.txt -l 100
DNS Performance Testing Tool
Version 2.13.0

[Status] Command line: dnsperf -d ./test/dnsperf.txt -l 100
[Status] Sending queries (to 127.0.0.1:53)
[Status] Started at: Sun Jul  2 17:30:08 2023
[Status] Stopping after 100.000000 seconds
[Status] Testing complete (time limit)

Statistics:

Queries sent:          177391
Queries completed:     177391 (100.00%)
Queries lost:          0 (0.00%)

Response codes:        NOERROR 177391 (100.00%)
Average packet size:   request 31, response 1207
Run time (s):          100.104525
Queries per second:    1772.057757

Average Latency (s):   0.056391 (min 0.000238, max 0.155088)
Latency StdDev (s):   0.030850
```

调试中遇到并解决的问题

- 浏览器一旦访问知乎程序就会报错：知乎在Recored中大量使用C0，倒是parse后的长度超过了512B，也超过了我们缓冲区的大小。我们增大了缓冲区并做了限制。
- 压力测试时大量请求超时：发现上游DNS会丢弃同一起来源的大量相同请求。在修复这个问题时我们修正了缓存中的错误，让相同请求不会到达上游服务器，同时修改了缓存策略，在Tire中用域名树来组织缓存查找，提高了效率。
- 在浏览器中无法正常拦截：浏览器自动加上www导致。我们在拦截功能中加上了相关逻辑使得无论有没有www都能拦截。
- 加上读写锁后出现写饥饿的情况：因为我们使用的读写锁没有优先级，C语言也没有相应的实现，所以我们自行实现了一个写优先的读写锁。虽然此次BUG的原因最后查明是解锁位置不对，但是读写优先的读写锁也排除了一个可能的隐患。

总结和心得体会

在本课程设计项目中，我们成功实现了一个功能完整、性能优良的DNS中继服务器。这个项目让我们对计算机网络的运作机理，尤其是DNS协议的实现与工作原理有了深入的理解。我们明白了DNS如何支持互联网的运作，以及DNS服务器在整个网络架构中的重要性。

在实现过程中，我们使用了C语言和libhv网络库，深化了对于socket编程和多线程并发编程的理解，实现了高效的域名解析和请求、响应的缓存。此外，我们还在现实环境中应用了计算机科学的理论知识，比如数据结构和算法，并了解了它们在解决实际问题中的重要性。

我们对我们的DNS中继服务器进行了一系列的测试，并记录了其性能数据。测试结果表明，我们的DNS中继服务器不仅功能完备，而且性能良好，能够在实际环境中稳定运行，并且对系统资源的需求较小。

总之，我们对于这个项目的完成感到非常满意。我们的DNS中继服务器实现了我们的设计目标，并且表现出优异的性能。我们相信，我们的实现能够在实际环境中提供稳定可靠的DNS服务。

心得

本次实验以TCP/IP协议栈为基础，使用UDP通信机制，利用Socket编程接口API，实现DNS中继服务器软件，在这个过程中我理解了计算机网络通讯，UDP协议和Socket编程。熟练掌握了DNS协议的内容、实现方式和通迅过程。对系统底层的信号机制，多线程锁与同步，阻塞与异步等机制有了更加深入的理解。

在和队友进行代码编写和debug的过程中，我收获了计算机网络协议的实现、编程和测试的基本方法和相关技术，也有一些心得体会。例如：在域名记录中使用C0加偏移量指向前面存过的域名，避免了大量的不必要的开销，提升了协议的效率。而协议中的族（网络类型）字段冗余无效，同时我也疑惑为什么响应需要带上question，因为已经有序号可以用于匹配请求和响应，不过响应带上question确实有利于简化cache的实现，同时也给客户端解析加上了保障，也减少了缓存。在这过程中也有很多实现定义的标准，如北邮服务器会在域名的中间部分使用C0指针，这是很少见的情况，还有DNS服务器会丢弃大量到来的包，这些都显示了现实和理论的一些微小差距。这启示我在系统设计时要充分考虑各种需求的必要性，并要辅以充分的测试，才能得到一个相对合理的结果。同时在本次实验中我也熟练掌握了WireShark、nslookup、dnstperf等网络调试工具的使用，掌握了实际环境中设计开发系统的能力。

关于本次团队合作我也感触颇深。尽管我们已经反复给团队中一位同学逐行讲解了代码，但是还是在验收时被老师问住了。我最大的感触是，每个人的精力和注意力都在不同的地方，而一次有效的团队合作，应该是取每个人的长处，通力合作，才能实现一加一大于二。而不是综合每个人的不足。

通过这次的DNS Relay开发，我收获了丰富的知识。这一次开发体验让我深刻体会到了，实际编程过程会出现多样且复杂的问题，不仅在代码逻辑的实现上，更在实际运行环境等方面。

印象最深刻的是给缓存进行加锁的时候。杨明佳修改我的缓存代码加了读写锁，随后压力测试发现程序无法正常使用。我们推测可能是读取次数过多导致了写饥饿，无法写入缓存，进一步导致了向上游的请求太多，使得请求被上游服务器丢弃。因此我查阅资料，当即手动实现了一个Write-preferring RwLock（写优先读写锁）。写完漂亮的代码后神清气爽，然而代码还是不能运行。定睛一看发现其实是我的cache接口里章成文把锁加错了地方，early return导致读锁上锁后不会解锁，成了一个乌龙。

另一个例子是我写好hosts文件解析后发现浏览器访问008.cn仍然能正常访问，但nslookup已经回显NXDOMAIN。震惊了一会之后开始抓包，遂恍然大悟：浏览器在DNS返回Name Error之后会在008.cn前面补www，而www.008.cn没有被拦截。

以上两个例子既展示了在编程过程中可能遇到的复杂问题，也说明了我在实现DNS Relay中涉及与学习的知识面之广泛，并不局限于计算机网络的具体协议中。这些经验使我认识到开发并非仅是实现功能，更重要的是要能够处理和应对各种出乎意料的情况，并且对整个程序的运行及可能的问题有全局且深入的理解。