15-213, Fall 20XX
15-213 秋季 20xx

Proxy Lab: Writing a Caching Web Proxy
Proxy Lab: Writing a Caching Web Proxy 代理实验室: 编写缓存 Web 代理

Assigned: Thu, Nov 19
分配时间: 11 月 19 日，星期四

Due: Tue, Dec 8, 11:59 PM
截止日期: 12 月 8 日，星期二，11:59 PM

Last Possible Time to Turn In: Fri, Dec 11, 11:59 PM
最后上交时间: 12 月 11 日，星期五，11:59 PM

1 Introduction
引言

A Web proxy is a program that acts as a middleman between a Web browser and an end server. Instead of contacting the end server directly to get a Web page, the browser contacts the proxy, which forwards the request on to the end server. When the end server replies to the proxy, the proxy sends the reply on to the browser.
Web 代理是在 Web 浏览器和终端服务器之间充当中间人的程序。浏览器不直接联系终端服务器获取网页，而是联系代理，代理将请求转发到终端服务器。当终端服务器回复代理时，代理会将回复发送到浏览器。

Proxies are useful for many purposes. Sometimes proxies are used in firewalls, so that browsers behind a firewall can only contact a server beyond the firewall via the proxy. Proxies can also act as anonymizers: by stripping requests of all identifying information, a proxy can make the browser anonymous to Web servers. Proxies can even be used to cache web objects by storing local copies of objects from servers then responding to future requests by reading them out of its cache rather than by communicating again with remote servers.
代理有很多用途。有时代理被用于防火墙，所以防火墙后的浏览器只能通过代理与防火墙外的服务器联系。代理也可以作为匿名者: 通过删除所有识别信息的请求，代理可以使浏览器对 Web 服

务器是匿名的。代理甚至可以用来缓存 web 对象，方法是从服务器存储对象的本地副本，然后通过从缓存中读取对象来响应未来的请求，而不是再次与远程服务器通信。

In this lab, you will write a simple HTTP proxy that caches web objects. For the first part of the lab, you will set up the proxy to accept incoming connections, read and parse requests, forward requests to web servers, read the servers' responses, and forward those responses to the corresponding clients. This first part will involve learning about basic HTTP operation and how to use sockets to write programs that communicate over network connections. In the second part, you will upgrade your proxy to deal with multiple concurrent connections. This will introduce you to dealing with concurrency, a crucial systems concept. In the third and last part, you will add caching to your proxy using a simple main memory cache of recently accessed web content.
在这个实验室中，您将编写一个简单的 HTTP 代理来缓存 web 对象。对于实验室的第一部分，您将设置代理来接受传入的连接、读取和解析请求、将请求转发给 web 服务器、读取服务器的响应，并将这些响应转发给相应的客户机。第一部分包括学习基本的 HTTP 操作，以及如何使用套接字编写通过网络连接进行通信的程序。在第二部分，你将升级你的代理来处理多个并发连接。这将向你介绍如何处理并发，这是一个至关重要的系统概念。在第三部分，也是最后一部分，你将使用最近访问的网页内容的一个简单的主内存缓存为你的代理添加缓存。

2 Logistics
物流

This is an individual project.
这是一个单独的项目。

3 Handout instructions
讲义说明

SITE-SPECIFIC: Insert a paragraph here that explains how the instructor will hand out the **proxylab-handout.tar** file to the students.
SITE-SPECIFIC: 在这里插入一个段落，说明教师将如何向学生分发 proxylab-handout.tar 文件。

Copy the handout file to a protected directory on the Linux machine where you plan to do your work, and then issue the following command:
将讲义文件复制到 Linux 机器上的一个受保护的目录，然后发出以下命令:

```
linux> tar xvf proxylab-handout.tar
Linux > tar xvf proxylab-讲义
```

This will generate a handout directory called proxylab-handout. The README file describes the various files.
这将生成一个名为 proxylab-handout 的讲义目录。 README 文件描述了各种文件。

4 Part I: Implementing a sequential web proxy
第一部分: 实现顺序 web 代理

The first step is implementing a basic sequential proxy that handles HTTP/1.0 GET requests. Other requests type, such as POST, are strictly optional.
第一步是实现一个处理 HTTP/1.0 GET 请求的基本顺序代理。其他请求类型，比如 POST，是严格可选的。

When started, your proxy should listen for incoming connections on a port whose number will be specified on the command line. Once a connection is established, your proxy should read the entirety of the request from the client and parse the request. It should determine whether the client has sent a valid HTTP request; if so, it can then establish its own connection to the appropriate web server then request the object the client specified. Finally, your proxy should read the server's response and forward it to the client.
启动时，代理应该侦听端口上的传入连接，该端口的号码将在命令行中指定。一旦连接建立，你的代理应该从客户端读取整个请求并解析请求。它应该确定客户端是否发送了一个有效的 HTTP 请求; 如果是，它可以建立自己到适当 web 服务器的连接，然后请求客户端指定的对象。最后，你的代理应该读取服务器的响应并将它转发给客户端。

4.1 HTTP/1.0 GET requests
4.1 HTTP/1.0 GET 请求

When an end user enters a URL such as http://www.cmu.edu/hub/index.html into the address bar of a web browser, the browser will send an HTTP request to the proxy that begins with a line that might resemble the following:

当终端用户在 web 浏览器的地址栏中输入一个 URL (如 HTTP://www.cmu.edu/hub/index. html) 时，该浏览器将向代理发送一个 HTTP 请求，该请求的起始行可能类似于以下内容:

GET http://www.cmu.edu/hub/index.html HTTP/1.1
获取 HTTP://www.cmu.edu/hub/index. html HTTP/1.1

In that case, the proxy should parse the request into at least the following fields: the hostname, www.cmu.edu; and the path or query and everything following it, /hub/index.html. That way, the proxy can deter-mine that it should open a connection to www.cmu.edu and send an HTTP request of its own starting with a line of the following form:
在这种情况下，代理应该至少将请求解析为以下字段: 主机名，www.cmu.edu; 路径或查询以及其后的所有内容,/hub/index. html。这样，代理就可以决定它是否应该打开一个到 www.cmu.edu 的连接，并自己发送一个 HTTP 请求，以下面的一行开头:

GET /hub/index.html HTTP/1.0
GET/hub/index.html HTTP/1.0

Note that all lines in an HTTP request end with a carriage return, '\r', followed by a newline, '\n'. Also important is that every HTTP request is terminated by an empty line: "\r\n".
请注意，HTTP 请求中的所有行都以一个回车符“ r”结束，后面跟着一个换行符“ n”。同样重要的是，每个 HTTP 请求都以一个空行结束: “ r n”。

You should notice in the above example that the web browser's request line ends with HTTP/1.1, while the proxy's request line ends with HTTP/1.0. Modern web browsers will generate HTTP/1.1 requests, but your proxy should handle them and forward them as HTTP/1.0 requests.
您应该注意到，在上面的示例中，web 浏览器的请求行以 HTTP/1.1 结束，而代理的请求行以 HTTP/1.0 结束。现代的 web 浏览器会生成 HTTP/1.1 请求，但是你的代理应该处理这些请求并将它们转发为 HTTP/1.0 请求。

It is important to consider that HTTP requests, even just the subset of HTTP/1.0 GET requests, can be incredibly complicated. The textbook describes certain details of HTTP transactions, but you should refer to RFC 1945 for the complete HTTP/1.0 specification. Ideally your HTTP request parser will be fully robust according to the relevant sections of RFC 1945, except for one detail: while the specification allows for multiline request fields, your proxy is not required to properly handle them. Of course, your proxy should never prematurely abort due to a malformed request.
重要的是要考虑到 HTTP 请求，即使只是 HTTP/1.0 GET 请求的子集，也可能是难以置信的复杂。教科书描述了 HTTP 事务的某些细节，但是你应该参考 rfc1945 了解完整的 HTTP/1.0 规范。理想情况下，根据 rfc1945 的相关章节，HTTP 请求解析器将是完全健壮的，除了一个细节: 虽然规范允许多行请求字段，但是您的代理不需要正确处理它们。当然，你的代理永远不应该因为一个错误的请求而过早中止。

4.2 Request headers
4.2 请求头

The important request headers for this lab are the Host, User-Agent, Connection, and Proxy-Connection headers:
这个实验室的重要请求头是 Host、 User-Agent、 Connection 和 proxy-Connection 头:

> Always send a Host header. While this behavior is technically not sanctioned by the HTTP/1.0 specification, it is necessary to coax sensible responses out of certain Web servers, especially those that use virtual hosting.
> 总是发送一个主机头。虽然这种行为在技术上并未得到 HTTP/1.0 规范的认可，但是有必要诱导某些 Web 服务器做出合理的响应，特别是那些使用虚拟主机的服务器。
>
> The Host header describes the hostname of the end server. For example, to access http://www. cmu.edu/hub/index.html, your proxy would send the following header:
> Host 标头描述了终端服务器的主机名。例如，访问 http://www。Cmu.edu/hub/index. html，你的代理会发送下面的头信息:
>
> > Host: www.cmu.edu
> > 主机: www.cmu.edu
>
> It is possible that web browsers will attach their own Host headers to their HTTP requests. If that is the case, your proxy should use the same Host header as the browser.
> Web 浏览器可能会在 HTTP 请求中附加自己的主机头。如果是这样的话，你的代理应该使用和浏览器相同的主机头。

You may choose to always send the following User-Agent header:

你可以选择总是发送下面的 User-Agent 头:

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:10.0.3)
用户代理: Mozilla/5.0(X11; Linux x86 _ 64; rv: 10.0.3)
Gecko/20120305 Firefox/10.0.3
Gecko/20120305 Firefox/10.0.3

The header is provided on two separate lines because it does not fit as a single line in the writeup, but your proxy should send the header as a single line.
标题在两个独立的行上提供，因为它不适合作为写作中的一个单独的行，但是您的代理应该作为一个单独的行发送标题。

The User-Agent header identifies the client (in terms of parameters such as the operating system and browser), and web servers often use the identifying information to manipulate the content they serve. Sending this particular User-Agent: string may improve, in content and diversity, the material that you get back during simple telnet-style testing.
User-Agent 头标识客户端(根据操作系统和浏览器等参数)，web 服务器通常使用标识信息来操作它们所服务的内容。发送这个特殊的 User-Agent: 字符串可以在内容和多样性方面改善你在简单的 telnet 风格测试中得到的材料。

Always send the following Connection

header: Connection: close

始终发送以下 Connection 头: Connection:

close

Always send the following Proxy-Connection header:
始终发送以下代理连接头:

Proxy-Connection: close
代理-连接: 关闭

The Connection and Proxy-Connection headers are used to specify whether a connection will be kept alive after the first request/response exchange is completed. It is perfectly acceptable (and suggested) to have your proxy open a new connection for each request. Specifying close as the value of these headers alerts web servers that your proxy intends to close connections after the first request/response exchange.
Connection 和 proxy-Connection 头用于指定在第一次请求/响应交换完成后，连接是否保持活动状态。让你的代理为每个请求打开一个新的连接是完全可以接受的(并且建议)。指定 close 作为这些头的值警告 web 服务器，你的代理打算在第一次请求/响应交换之后关闭连接。

For your convenience, the values of the described User-Agent header is provided to you as a string constant in proxy.c.
为了方便起见，所描述的 User-Agent 头的值在 proxy.c 中作为字符串常量提供给您。

Finally, if a browser sends any additional request headers as part of an HTTP request, your proxy should forward them unchanged.
最后，如果一个浏览器发送任何额外的请求头作为 HTTP 请求的一部分，你的代理应该转发他们不变。

4.3 Port numbers
4.3 端口号

There are two significant classes of port numbers for this lab: HTTP request ports and your proxy's listening port.
这个实验室有两类重要的端口号: HTTP 请求端口和代理的监听端口。

The HTTP request port is an optional field in the URL of an HTTP request. That is, the URL may be of the form, http://www.cmu.edu:8080/hub/index.html, in which case your proxy should connect to the host www.cmu.edu on port 8080 instead of the default HTTP port, which is port 80. Your proxy must properly function whether or not the port number is included in the URL.
HTTP 请求端口是 HTTP 请求 URL 中的一个可选字段。也就是说，URL 的格式可能是 HTTP://www.cmu.edu: 8080/hub/index. html，在这种情况下，代理应该在端口 8080 上连接到主机 www.cmu.edu，而不是默认的 HTTP 端口 80。无论 URL 中是否包含端口号，你的代理都必须正常工作。

The listening port is the port on which your proxy should listen for incoming connections. Your proxy should accept a command line argument specifying the listening port number for your proxy. For example, with the following command, your proxy should listen for connections on port 15213:
侦听端口是代理应该侦听传入连接的端口。你的代理应该接受一个命令行参数来指定你的代理的监听端口号。例如，使用下面的命令，你的代理应该侦听端口 15213 上的连接:

linux> ./proxy 15213
Linux >/proxy 15213

You may select any non-privileged listening port (greater than 1,024 and less than 65,536) as long as it is not used by other processes. Since each proxy must use a unique listening port and many people will simultaneously be working on each machine, the script port-for-user.pl is provided to help you pick your own personal port number. Use it to generate port number based on your user ID:
您可以选择任何非特权侦听端口(大于 1,024 和小于 65,536)，只要它不被其他进程使用。由于每个代理必须使用一个唯一的侦听端口，而且许多人将同时在每台机器上工作，因此提供了脚本 port-for-user.pl 来帮助您选择自己的个人端口号。使用它根据你的用户 ID 生成端口号:

```
linux> ./port-for-user.pl droh
Linux > ./port-for-user.pl droh
droh: 45806
Droh: 45806
```

The port, p, returned by port-for-user.pl is always an even number. So if you need an additional port number, say for the Tiny server, you can safely use ports p and p + 1.
Port-for-user.pl 返回的端口 p 总是一个偶数。所以如果你需要一个额外的端口号，比如 Tiny 服务器，你可以安全地使用端口 p 和 p + 1。

Please don’t pick your own random port. If you do, you run the risk of interfering with another user.
请不要选择你自己的随机端口，如果你这样做了，你会冒着干扰其他用户的风险。

5 Part II: Dealing with multiple concurrent requests
第二部分: 处理多个并发请求

Once you have a working sequential proxy, you should alter it to simultaneously handle multiple requests. The simplest way to implement a concurrent server is to spawn a new thread to handle each new connection request. Other designs are also possible, such as the prethreaded server described in Section 12.5.5 of your textbook.
一旦你有了一个可以工作的顺序代理，你应该修改它来同时处理多个请求。实现并发服务器最简单的方法是生成一个新的线程来处理每个新的连接请求。其他的设计也是可能的，比如教科书 12.5.5 节中描述的预处理服务器。

Note that your threads should run in detached mode to avoid memory leaks.
注意，线程应该以分离模式运行以避免内存泄漏。

The open clientfd and open listenfd functions described in the CS:APP3e textbook are based on the modern and protocol-independent getaddrinfo function, and thus are thread safe.
在 CS: APP3e 教科书中描述的开放客户端 fd 和开放监听 fd 函数基于现代的和协议无关的 getaddrinfo 函数，因此是线程安全的。

6 Part III: Caching web objects
第三部分: 缓存 web 对象

For the final part of the lab, you will add a cache to your proxy that stores recently-used Web objects in memory. HTTP actually defines a fairly complex model by which web servers can give instructions as to how the objects they serve should be cached and clients can specify how caches should be used on their behalf. However, your proxy will adopt a simplified approach.
在实验室的最后一部分，您将向代理添加一个缓存，将最近使用过的 Web 对象存储在内存中。HTTP 实际上定义了一个相当复杂的模型，通过这个模型，web 服务器可以指导如何缓存它们服务的对象，客户端可以指定如何代表它们使用缓存。然而，你的代理将采用一种简化的方法。

When your proxy receives a web object from a server, it should cache it in memory as it transmits the object to the client. If another client requests the same object from the same server, your proxy need not reconnect to the server; it can simply resend the cached object.
当您的代理从服务器接收到一个 web 对象时，它应该在将该对象传输给客户端时将其缓存在内存中。如果另一个客户端从同一个服务器请求相同的对象，你的代理不需要重新连接到服务器；它可以简单地重新发送缓存的对象。

Obviously, if your proxy were to cache every object that is ever requested, it would require an unlimited amount of memory. Moreover, because some web objects are larger than others, it might be the case that one giant object will consume the entire cache, preventing other objects from being cached at all. To avoid those problems, your proxy should have both a maximum cache size and a maximum cache object size.
显然，如果你的代理缓存每一个被请求的对象，它将需要无限量的内存。此外，由于某些 web 对象比其他对象大，可能会出现这样的情况: 一个巨大的对象将占用整个缓存，从而阻止其他对象被缓存。为了避免这些问题，你的代理应该同时拥有最大缓存大小和最大缓存对象大小。

6.1 Maximum cache size
6.1 最大缓存大小

The entirety of your proxy's cache should have the following maximum size:
整个代理缓存的最大大小应该如下:

MAX_CACHE_SIZE = 1 MiB
最大缓存 _ 大小 = 1mib

When calculating the size of its cache, your proxy must only count bytes used to store the actual web objects; any extraneous bytes, including metadata, should be ignored.
在计算其缓存的大小时，代理必须只计算用于存储实际 web 对象的字节数; 应该忽略包括元数据在内的任何无关字节。

6.2 Maximum object size
6.2 最大对象大小

Your proxy should only cache web objects that do not exceed the following maximum size:
你的代理应该只缓存不超过以下最大大小的 web 对象:

MAX_OBJECT_SIZE = 100 KiB
物体大小 = 100kb

For your convenience, both size limits are provided as macros in proxy.c.
为了方便起见，两个大小限制都在 proxy.c 中以宏的形式提供。

The easiest way to implement a correct cache is to allocate a buffer for each active connection and accumulate data as it is received from the server. If the size of the buffer ever exceeds the maximum object size, the buffer can be discarded. If the entirety of the web server's response is read before the maximum object size is exceeded, then the object can be cached. Using this scheme, the maximum amount of data your proxy will ever use for web objects is the following, where T is the maximum number of active connections:
实现正确缓存的最简单方法是为每个活动连接分配一个缓冲区，并在从服务器接收数据时累积延迟数据。如果缓冲区的大小超过了对象的最大大小，缓冲区就会被丢弃。如果在超过最大对象大小之前读取了整个 web 服务器的响应，那么对象可以被缓存。使用这个方案，代理对 web 对象使用的最大数据量如下，其中 t 是活动连接的最大数量:

MAX_CACHE_SIZE + T * MAX_OBJECT_SIZE
MAX _ cache _ size + t * MAX _ object _ size

6.3 Eviction policy
6.3 驱逐政策

Your proxy's cache should employ an eviction policy that approximates a least-recently-used (LRU) eviction policy. It doesn't have to be strictly LRU, but it should be something reasonably close. Note that both reading an object and writing it count as using the object.
你的代理缓存应该使用一个近似于最近使用的(LRU)驱逐策略的驱逐策略。它不一定是严格的 LRU，但它应该是一些合理的接近。请注意，读取和写入一个对象都被认为是使用该对象。

6.4 Synchronization
6.4 同步

Accesses to the cache must be thread-safe, and ensuring that cache access is free of race conditions will likely be the more interesting aspect of this part of the lab. As a matter of fact, there is a special requirement that multiple threads must be able to simultaneously read from the cache. Of course, only one thread should be permitted to write to the cache at a time, but that restriction must not exist for readers.
对缓存的访问必须是线程安全的，确保缓存访问不受竞态条件的影响可能是实验室这一部分更有趣的方面。事实上，有一个特殊的要求，即多个线程必须能够同时从缓存中读取数据。当然，一次只允许一个线程写入缓存，但是对于读取器来说这个限制是不存在的。

As such, protecting accesses to the cache with one large exclusive lock is not an acceptable solution. You may want to explore options such as partitioning the cache, using Pthreads readers-writers locks, or using semaphores to implement your own readers-writers solution. In either case, the fact that you don't have to implement a strictly LRU eviction policy will give you some flexibility in supporting multiple readers.
因此，使用一个大的独占锁来保护对缓存的访问是不可接受的解决方案。您可能希望探索诸如分区缓存、使用 Pthreads 读写器锁或使用信号量实现自己的读写器解决方案等选项。无论哪种情况，你都不必实现严格的 LRU 驱逐策略，这将给你支持多个读取器带来一定的灵活性。

7 Evaluation
评估

This assignment will be graded out of a total of 70 points:
这项作业的总分为 70 分:

BasicCorrectness: 40 points for basic proxy operation (autograded)
基本正确性: 基本代理操作 40 分(自动评分)

Concurrency: 15 points for handling concurrent requests (autograded)
并发性: 处理并发请求的 15 点(自动分级)

Cache: 15 points for a working cache (autograded)
缓存: 工作缓存 15 点(自动分级)

7.1 Autograding
7.1 自动评分

Your handout materials include an autograder, called driver.sh, that your instructor will use to assign scores for BasicCorrectness, Concurrency, and Cache. From the proxylab-handout directory:
你的讲义材料包括一个叫做 driver.sh 的自动评分工具，你的老师会用它来给 BasicCorrectness，Concurrency 和 Cache 打分。来自 proxylab-handout 目录:

```
linux> ./driver.sh
Linux > ./driver.sh
```

You must run the driver on a Linux machine.
您必须在 Linux 机器上运行驱动程序。

7.2 Robustness
7.2 稳健性

As always, you must deliver a program that is robust to errors and even malformed or malicious input. Servers are typically long-running processes, and web proxies are no exception. Think carefully about how long-running processes should react to different types of errors. For many kinds of errors, it is certainly inappropriate for your proxy to immediately exit.
像往常一样，你必须交付一个程序，它对错误，甚至错误或恶意输入都是健壮的。服务器通常是长时间运行的进程，web 代理也不例外。仔细考虑长时间运行的进程应该对不同类型的错误做出怎样的反应。对于许多类型的错误，你的代理立即退出当然是不合适的。

Robustness implies other requirements as well, including invulnerability to error cases like segmentation faults and a lack of memory leaks and file descriptor leaks.
鲁棒性还意味着其他需求，包括对分段错误、缺少内存泄漏和文件描述符泄漏等错误情况的不可破坏性。

8 Testing and debugging
测试和调试

Besides the simple autograder, you will not have any sample inputs or a test program to test your implementation. You will have to come up with your own tests and perhaps even your own testing harness to help you debug your code and decide when you have a correct implementation. This is a valuable skill in the real world, where exact operating conditions are rarely known and reference solutions are often unavailable.
除了简单的自动校正器，您将不会有任何样品输入或测试程序来测试您的实现。您将不得不提出您自己的测试，甚至您自己的测试工具来帮助您调试代码并决定何时有正确的实现。在现实世界中，这是一项很有价值的技能，因为在现实世界中，精确的操作条件很少为人所知，参考解决方案也常常不可用。

Fortunately there are many tools you can use to debug and test your proxy. Be sure to exercise all code paths and test a representative set of inputs, including base cases, typical cases, and edge cases.
幸运的是，你可以使用很多工具来调试和测试你的代理。一定要练习所有的代码路径并测试一组代表性的输入，包括基本用例、典型用例和边缘用例。

8.1 Tiny web server
8.1 微型网络服务器

Your handout directory the source code for the CS:APP Tiny web server. While not as powerful as thttpd, the CS:APP Tiny web server will be easy for you to modify as you see fit. It’s also a reasonable starting point for your proxy code. And it’s the server that the driver code uses to fetch pages.

你的讲义目录 CS 的源代码: APP Tiny web 服务器。虽然没有 thttpd 那么强大，但是 CS: APP Tiny web 服务器很容易修改，只要你觉得合适。它也是你代理代码的一个合理的起点。它是驱动程序代码用来获取页面的服务器。

8.2 telnet
8.2 telnet

As described in your textbook (11.5.3), you can use telnet to open a connection to your proxy and send it HTTP requests.
正如你的教科书(11.5.3)中所描述的，你可以使用 telnet 打开一个连接到你的代理并发送 HTTP 请求。

8.3 curl
8.3 卷

You can use curl to generate HTTP requests to any server, including your own proxy. It is an extremely useful debugging tool. For example, if your proxy and Tiny are both running on the local machine, Tiny is listening on port 15213, and proxy is listening on port 15214, then you can request a page from Tiny via your proxy using the following curl command:
你可以使用 curl 向任何服务器生成 HTTP 请求，包括你自己的代理。它是一个非常有用的调试工具。例如，如果您的代理和 Tiny 都在本地计算机上运行，Tiny 正在监听端口 15213，而 proxy 正在监听端口 15214，那么您可以使用以下 curl 命令通过代理从 Tiny 请求页面:

```
linux> curl -v --proxy http://localhost:15214 http://localhost:15213/home.html
Curl-v -- 代理 http://localhost: 15214 http://localhost: 15213/home. html

*   About to connect() to proxy localhost port 15214 (#0)
About to connect () to proxy localhost port 15214(# 0)

*       Trying 127.0.0.1... connected
尝试连接 127.0.0.1
```

```
*   Connected to localhost (127.0.0.1) port 15214 (#0) > GET
http://localhost:15213/home.html HTTP/1.1
连接到 localhost (127.0.0.1)端口 15214(# 0) > GET HTTP://localhost:
15213/home. html HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu)...
> User-Agent: curl/7.19.7(x86 _ 64-redhat-linux-gnu) ...
> Host: localhost:15213
主机: localhost: 15213
>   Accept: */*
接受: */*
>   Proxy-Connection: Keep-Alive
代理连接: Keep-Alive
>
*   HTTP 1.0, assume close after body < HTTP/1.0
200 OK
HTTP 1.0，假设关闭后体 < HTTP/1.0200 确定
< Server: Tiny Web Server
服务器: 微型 Web 服务器
< Content-length: 120
< 内容长度: 120
< Content-type: text/html
内容类型: 文本/html
<
<html>
< html >
<head><title>test</title></head>
< 标题 > 测试 </title > </head >
<body>
< 身体 >
<img align="middle" src="godzilla.gif"> Dave
O'Hallaron
大卫·奥哈拉伦
</body>
</body >
</html>
</html >
*   Closing connection #0
关闭连接 # 0
```

8.4 netcat

8.4 netcat

netcat, also known as nc, is a versatile network utility. You can use netcat just like telnet, to open connections to servers. Hence, imagining that your proxy were running on catshark using port 12345 you can do something like the following to manually test your proxy:

Netcat，也称为 nc，是一个多功能的网络实用程序。你可以像 telnet 一样使用 netcat 打开到服务器的连接。因此，假设你的代理使用 12345 端口在 catshark 上运行，你可以做以下事情来手动测试你的代理:

```
sh> nc catshark.ics.cs.cmu.edu 12345
Sh > nc catshark.ics.cs.cmu.edu 12345
GET http://www.cmu.edu/hub/index.html HTTP/1.0
```

HTTP://www.cmu.edu/hub/index. html HTTP/1.0

HTTP/1.1 200 OK
HTTP/1.1200 OK
...
...

In addition to being able to connect to Web servers, netcat can also operate as a server itself. With the following command, you can run netcat as a server listening on port 12345:
除了能够连接到 Web 服务器之外，netcat 本身也可以作为服务器进行操作。通过下面的命令，你可以运行 netcat 作为服务器监听端口 12345:

sh> nc -l 12345
Sh > nc-l 12345

Once you have set up a netcat server, you can generate a request to a phony object on it through your proxy, and you will be able to inspect the exact request that your proxy sent to netcat.
一旦设置了 netcat 服务器，就可以通过代理在其上生成对虚假对象的请求，并且可以检查代理发送给 netcat 的确切请求。

8.5 Web browsers
8.5 网页浏览器

Eventually you should test your proxy using the most recent version of Mozilla Firefox. Visiting About Firefox will automatically update your browser to the most recent version.
最后，你应该使用最新版本的 Mozilla Firefox 来测试你的代理。访问 About Firefox 会自动将你的浏览器更新到最新版本。

To configure Firefox to work with a proxy, visit
要配置 Firefox 使用代理服务器，请访问

> Preferences>Advanced>Network>Settings
> 首选项 > 高级 > 网络 > 设置

It will be very exciting to see your proxy working through a real Web browser. Although the functionality of your proxy will be limited, you will notice that you are able to browse the vast majority of websites through your proxy.
看到您的代理通过真正的 Web 浏览器工作将是非常令人兴奋的。尽管代理的功能有限，但是你会发现你可以通过你的代理浏览绝大多数的网站。

An important caveat is that you must be very careful when testing caching using a Web browser. All modern Web browsers have caches of their own, which you should disable before attempting to test your proxy's cache.
一个重要的警告是，在使用 Web 浏览器测试缓存时你必须非常小心。所有现代的 Web 浏览器都有自己的缓存，在测试代理缓存之前你应该禁用它。

9 Handin instructions
交出指示

The provided Makefile includes functionality to build your final handin file. Issue the following command from your working directory:
提供的 Makefile 包含构建最终 handin 文件的功能。从工作目录发出以下命令:

> linux> make handin
> Linux > make handin

The output is the file ../proxylab-handin.tar, which you can then handin.
输出是文件. ./proxylab-handin.tar，然后可以将其交付。

> SITE-SPECIFIC: Insert a paragraph here that tells each student how to hand in their **proxylab-handin.tar** solution file.
> SITE-SPECIFIC: 在这里插入一个段落，告诉每个学生如何提交他们的 proxylab-handin.tar 解决方案文件。

> Chapters 10-12 of the textbook contains useful information on system-level I/O, network program-ming, HTTP protocols, and concurrent programming.
> 教科书的第 10-12 章包含了关于系统级 i/o、网络编程、 HTTP 协议和并发编程的有用信息。

> RFC 1945 (http://www.ietf.org/rfc/rfc1945.txt) is the complete specification for the HTTP/1.0 protocol.
> Rfc1945(HTTP://www.ietf.org/RFC/rfc1945.txt)是 HTTP/1.0 协议的完整规范。

10 Hints
提示

As discussed in Section 10.11 of your textbook, using standard I/O functions for socket input and output is a problem. Instead, we recommend that you use the Robust I/O (RIO) package, which is provided in the csapp.c file in the handout directory.
正如教科书第 10.11 节所讨论的，使用标准 i/o 函数进行套接字输入和输出是一个问题。相反，我们建议你使用 Robust i/o (RIO)包，它是在讲义目录的 csapp.c 文件中提供的。

The error-handling functions provide in csapp.c are not appropriate for your proxy because once a server begins accepting connections, it is not supposed to terminate. You'll need to modify them or write your own.
在 csapp.c 中提供的错误处理功能不适用于您的代理，因为一旦服务器开始接受连接，它就不应该终止。你需要修改它们或者自己编写。

You are free to modify the files in the handout directory any way you like. For example, for the sake of good modularity, you might implement your cache functions as a library in files called cache.c and cache.h. Of course, adding new files will require you to update the provided Makefile.
您可以自由地以任何方式修改讲义目录中的文件。例如，为了更好的模块化，你可以在 cache.c 和 cache.h 文件中以库的形式实现缓存函数。当然，添加新文件需要更新提供的 Makefile。

As discussed in the Aside on page 964 of the CS:APP3e text, your proxy must ignore SIGPIPE signals and should deal gracefully with write operations that return EPIPE errors.
正如在 CS: APP3e 文本第 964 页的 Aside 中所讨论的，代理必须忽略 SIGPIPE 信号，并且应该优雅地处理返回 EPIPE 错误的写操作。

Sometimes, calling read to receive bytes from a socket that has been prematurely closed will cause read to return -1 with errno set to ECONNRESET. Your proxy should not terminate due to this error either.
有时，从套接字中调用 read 以接收字节(该套接字已过早关闭)将导致 read 返回 -1，errno 设置为 ECONNRESET。你的代理也不应该因为这个错误而终止。

Remember that not all content on the web is ASCII text. Much of the content on the web is binary data, such as images and video. Ensure that you account for binary data when selecting and using functions for network I/O.
记住，并不是所有的网络内容都是 ASCII 文本。网络上的大部分内容都是二进制数据，比如图片和视频。确保在选择和使用网络 i/o 函数时考虑到二进制数据。

Forward all requests as HTTP/1.0 even if the original request was HTTP/1.1.
即使原始请求是 HTTP/1.1，也将所有请求转发为 HTTP/1.0。

Good luck!
祝你好运！

10