15-213, 20XX 가을

프록시 랩: 캐싱 웹 프록시 작성

할당됨: 11월 19일 목요일

마감: 12월 8일 화요일 오후 11시 59분

제출 가능한 마지막 시간: 12월 11일 금요일 오후 11시 59분

1. 소개

웹 프록시는 웹 브라우저와 최종 서버 사이에서 중개자 역할을 하는 프로그램입니다. 웹 페이지를 얻기 위해 최종 서버에 직접 접속하는 대신, 브라우저는 요청을 최종 서버로 전달하는 프록시에 접속합니다. 최종 서버가 프록시에 응답하면 프록시는 응답을 브라우저로 보냅니다.

프록시는 다양한 목적에 유용합니다. 방화벽 뒤에 있는 브라우저가 프록시를 통해서만 방화벽 외부의 서버에 접속할 수 있도록 프록시가 방화벽에서 사용되는 경우도 있습니다. 프록시는 익명화 장치 역할도 할 수 있습니다. 요청에서 모든 식별 정보를 제거함으로써 프록시는 브라우저를 웹 서버에 익명으로 만들 수 있습니다. 프록시는 서버에서 개체의 로컬 복사본을 저장한 다음 원격 서버와 다시 통신하는 대신 캐시에서 해당 복사본을 읽어 향후 요청에 응답함으로써 웹 개체를 캐시하는 데에도 사용할 수 있습니다.

이 실습에서는 웹 개체를 캐시하는 간단한 HTTP 프록시를 작성합니다. 실습의 첫 번째 부분에서는 들어오는 연결을 수락하고, 요청을 읽고 구문 분석하고, 요청을 웹 서버에 전달하고, 서버의 응답을 읽고, 해당 응답을 해당 클라이언트에 전달하도록 프록시를 설정합니다. 첫 번째 부분에서는 기본 HTTP 작업과 소켓을 사용하여 네트워크 연결을 통해 통신하는 프로그램을 작성하는 방법을 학습합니다. 두 번째 부분에서는 여러 동시 연결을 처리하기 위해 프록시를 업그레이드합니다. 이는 중요한 시스템 개념인 동시성을 다루는 방법을 소개합니다. 세 번째이자 마지막 부분에서는 최근에 액세스한 웹 콘텐츠의 간단한 주 메모리 캐시를 사용하여 프록시에 캐싱을 추가합니다.

2 물류

이것은 개별 프로젝트입니다.

3 유인물 지침

특정 사이트: 강사가 학생들에게 Proxylab-handout.tar 파일을 배포하는 방법을 설명하는 단락을 여기에 삽입하십시오.

작업을 수행하려는 Linux 시스템의 보호된 디렉터리에 유인물 파일을 복사한 후 다음 명령을 실행합니다.

```
리눅스> tar xvf Proxylab-handout.tar
```

그러면 Proxylab-handout이라는 유인물 디렉토리가 생성됩니다. README 파일은 다양한 파일을 설명합니다.

4 1부: 순차 웹 프록시 구현

첫 번째 단계는 HTTP/1.0 GET 요청을 처리하는 기본 순차 프록시를 구현하는 것입니다. POST와 같은 기타 요청 유형은 엄격히 선택 사항입니다.

시작되면 프록시는 명령줄에 번호가 지정되는 포트에서 들어오는 연결을 수신해야 합니다. 연결이 설정되면 프록시는 클라이언트의 요청 전체를 읽고 요청을 구문 분석해야 합니다. 클라이언트가 유효한 HTTP 요청을 보냈는지 확인해야 합니다. 그렇다면 적절한 웹 서버에 대한 자체 연결을 설정한 다음 클라이언트가 지정한 개체를 요청할 수 있습니다. 마지막으로 프록시는 서버의 응답을 읽고 이를 클라이언트에 전달해야 합니다.

4.1 HTTP/1.0 GET 요청

최종 사용자가 웹 브라우저의 주소 표시줄에 http://www.cmu.edu/hub/index.html과 같은 URL을 입력하면 브라우저는 다음과 같은 줄로 시작하는 HTTP 요청을 프록시에 보냅니다. 다음과 유사합니다.

```
http://www.cmu.edu/hub/index.html HTTP/1.1 받기
```

이 경우 프록시는 요청을 최소한 다음 필드로 구문 분석해야 합니다. 호스트 이름, www.cmu.edu; 경로 또는 쿼리와 그 뒤에 오는 모든 항목, /hub/index.html. 그런 식으로 프록시는 www.cmu.edu에 대한 연결을 열고 다음 형식의 줄로 시작하는 자체 HTTP 요청을 보내야 하는지 결정할 수 있습니다.

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

HTTP 요청의 모든 줄은 캐리지 리턴 '\r'로 끝나고 그 뒤에 줄바꿈 '\n'이 옵니다. 또한 중요한 점은 모든 HTTP 요청이 빈 줄("\r\n")로 종료된다는 것입니다.

위의 예에서 웹 브라우저의 요청 라인은 HTTP/1.1로 끝나는 반면 프록시의 요청 라인은 HTTP/1.0으로 끝나는 것을 확인해야 합니다. 최신 웹 브라우저는 HTTP/1.1 요청을 생성하지만 프록시는 이를 처리하고 HTTP/1.0 요청으로 전달해야 합니다.

HTTP/1.0 GET 요청의 하위 집합일지라도 HTTP 요청은 엄청나게 복잡할 수 있다는 점을 고려하는 것이 중요합니다. 교과서에서는 HTTP 트랜잭션의 특정 세부 사항을 설명하지만 전체 HTTP/1.0 사양은 RFC 1945를 참조해야 합니다. 이상적으로 HTTP 요청 파서는 한 가지 세부 사항을 제외하고 RFC 1945의 관련 섹션에 따라 완전히 강력합니다. 사양에서는 여러 줄의 요청 필드를 허용하지만 프록시는 이를 적절하게 처리할 필요가 없습니다. 물론 잘못된 요청으로 인해 프록시가 조기에 중단되어서는 안 됩니다.

4.2 요청 헤더

이 실습에서 중요한 요청 헤더는 Host, User-Agent, Connection 및 Proxy-Connection 헤더입니다.

- 항상 Host 헤더를 보냅니다. 이 동작은 기술적으로 HTTP/1.0 사양에 의해 승인되지 않지만 특정 웹 서버, 특히 가상 호스팅을 사용하는 웹 서버에서 합리적인 응답을 유도해야 합니다.

  Host 헤더는 최종 서버의 호스트 이름을 설명합니다. 예를 들어 http://www에 액세스하려면 cmu.edu/hub/index.html에서 프록시는 다음 헤더를 보냅니다.

  ```
  호스트: www.cmu.edu
  ```

  웹 브라우저가 자체 Host 헤더를 HTTP 요청에 첨부할 수 있습니다. 이 경우 프록시는 브라우저와 동일한 Host 헤더를 사용해야 합니다.

- 항상 다음 User-Agent 헤더를 보내도록 선택할 수 있습니다.

  ```
  사용자 에이전트: Mozilla/5.0(X11; Linux x86_64; rv:10.0.3)
  게코/20120305 Firefox/10.0.3
  ```

  헤더는 쓰기에서 한 줄로 맞지 않기 때문에 별도의 두 줄로 제공되지만 프록시는 헤더를 한 줄로 보내야 합니다.

  User-Agent 헤더는 운영 체제 및 브라우저와 같은 매개변수 측면에서 클라이언트를 식별하며, 웹 서버는 식별 정보를 사용하여 제공하는 콘텐츠를 조작하는 경우가 많습니다. 이 특정 User-Agent: 문자열을 보내면 간단한 텔넷 스타일 테스트 중에 반환되는 자료의 내용과 다양성이 향상될 수 있습니다.

- 항상 다음 연결 헤더를 보냅니다.

  ```
  연결: 닫기
  ```

- 항상 다음 프록시 연결 헤더를 보냅니다.

프록시 연결: 닫기

Connection 및 Proxy-Connection 헤더는 첫 번째 요청/응답 교환이 완료된 후 연결이 유지되는지 여부를 지정하는 데 사용됩니다. 프록시가 각 요청에 대해 새 연결을 열도록 하는 것은 완벽하게 허용되며 제안됩니다. 이러한 헤더 값으로 close를 지정하면 첫 번째 요청/응답 교환 후 프록시가 연결을 닫으려고 한다는 것을 웹 서버에 알립니다.

귀하의 편의를 위해 설명된 User-Agent 헤더의 값은 Proxy.c에서 문자열 상수로 제공됩니다.

마지막으로, 브라우저가 HTTP 요청의 일부로 추가 요청 헤더를 보내는 경우 프록시는 이를 변경하지 않고 전달해야 합니다.

4.3 포트 번호

이 실습에는 HTTP 요청 포트와 프록시의 수신 대기 포트라는 두 가지 중요한 포트 번호 클래스가 있습니다.

HTTP 요청 포트는 HTTP 요청 URL의 선택적 필드입니다. 즉, URL은 http://www.cmu.edu:8080/hub/index.html 형식일 수 있으며, 이 경우 프록시는 포트 8080 대신 호스트 www.cmu.edu에 연결해야 합니다. 기본 HTTP 포트는 포트 80입니다. 포트 번호가 URL에 포함되어 있는지 여부에 관계없이 프록시가 제대로 작동해야 합니다.

수신 포트는 프록시가 들어오는 연결을 수신해야 하는 포트입니다. 프록시는 프록시의 수신 대기 포트 번호를 지정하는 명령줄 인수를 허용해야 합니다. 예를 들어 다음 명령을 사용하면 프록시는 포트 15213에서 연결을 수신해야 합니다.

리눅스> ./proxy 15213

다른 프로세스에서 사용하지 않는 한 권한이 없는 수신 포트(1,024보다 크고 65,536보다 작음)를 선택할 수 있습니다. 각 프록시는 고유한 수신 포트를 사용해야 하고 많은 사람들이 각 시스템에서 동시에 작업하므로 개인 포트 번호를 선택하는 데 도움이 되는 port-for-user.pl 스크립트가 제공됩니다. 이를 사용하여 사용자 ID를 기반으로 포트 번호를 생성합니다.

리눅스> ./port-for-user.pl droh droh: 45806

port-for-user.pl이 반환하는 포트 p는 항상 짝수입니다. 따라서 Tiny 서버와 같이 추가 포트 번호가 필요한 경우 포트 p 및 p + 1을 안전하게 사용할 수 있습니다.

자신의 임의의 포트를 선택하지 마십시오. 그렇게 하면 다른 사용자를 방해할 위험이 있습니다.

5 2부: 여러 동시 요청 처리

작동하는 순차 프록시가 있으면 여러 요청을 동시에 처리하도록 이를 변경해야 합니다.
동시 서버를 구현하는 가장 간단한 방법은 각각의 새로운 연결 요청을 처리하기 위해 새 스레드를 생성하는 것입니다. 교과서의 섹션 12.5.5에 설명된 사전 스레드 서버와 같은 다른 설계도 가능합니다.

- 메모리 누수를 방지하려면 스레드가 분리 모드에서 실행되어야 합니다.
- CS:APP3e 교과서에 설명된 open clientfd 및 open listeningfd 함수는 현대적이고 프로토콜 독립적인 getaddrinfo 함수를 기반으로 하므로 스레드로부터 안전합니다.

6 파트 III: 웹 객체 캐싱

실습의 마지막 부분에서는 최근에 사용한 웹 개체를 메모리에 저장하는 프록시에 캐시를 추가합니다. HTTP는 실제로 웹 서버가 자신이 제공하는 개체를 캐시하는 방법에 대한 지침을 제공하고 클라이언트가 캐시를 대신 사용하는 방법을 지정할 수 있는 상당히 복잡한 모델을 정의합니다. 그러나 프록시는 단순화된 접근 방식을 채택합니다.

프록시는 서버에서 웹 개체를 받으면 개체를 클라이언트에 전송할 때 메모리에 캐시해야 합니다. 다른 클라이언트가 동일한 서버에서 동일한 객체를 요청하는 경우 프록시는 서버에 다시 연결할 필요가 없습니다. 단순히 캐시된 개체를 다시 보낼 수 있습니다.

분명히 프록시가 요청된 모든 객체를 캐시하려면 무제한의 메모리가 필요할 것입니다. 또한 일부 웹 개체는 다른 개체보다 크기 때문에 하나의 거대한 개체가 전체 캐시를 소비하여 다른 개체가 전혀 캐시되지 않는 경우가 있을 수 있습니다. 이러한 문제를 방지하려면 프록시에 최대 캐시 크기와 최대 캐시 개체 크기가 모두 있어야 합니다.

6.1 최대 캐시 크기

전체 프록시 캐시의 최대 크기는 다음과 같아야 합니다.

MAX_CACHE_SIZE = 1MiB

캐시 크기를 계산할 때 프록시는 실제 웹 개체를 저장하는 데 사용된 바이트만 계산해야 합니다. 메타데이터를 포함한 모든 외부 바이트는 무시되어야 합니다.

6.2 최대 개체 크기

프록시는 다음 최대 크기를 초과하지 않는 웹 객체만 캐시해야 합니다.

MAX_OBJECT_SIZE = 100KiB

귀하의 편의를 위해 두 가지 크기 제한이 모두 Proxy.c에 매크로로 제공됩니다.

올바른 캐시를 구현하는 가장 쉬운 방법은 각 활성 연결에 대해 버퍼를 할당하고 서버에서 수신되는 데이터를 축적하는 것입니다. 버퍼 크기가 최대 개체 크기를 초과하면 버퍼가 삭제될 수 있습니다. 최대 개체 크기를 초과하기 전에 웹 서버의 응답 전체를 읽으면 개체가 캐시될 수 있습니다. 이 구성표를 사용하면 프록시가 웹 개체에 사용하는 최대 데이터 양은 다음과 같습니다. 여기서 T는 최대 활성 연결 수입니다.

```
MAX_CACHE_SIZE + T * MAX_OBJECT_SIZE
```

6.3 퇴거 정책

프록시 캐시는 LRU(최근 사용 횟수) 제거 정책에 가까운 제거 정책을 사용해야 합니다. 엄격하게 LRU일 필요는 없지만 합리적으로 가까운 것이어야 합니다. 객체를 읽는 것과 쓰는 것은 모두 객체를 사용하는 것으로 간주됩니다.

6.4 동기화

캐시에 대한 액세스는 스레드로부터 안전해야 하며 캐시 액세스에 경합 조건이 없는지 확인하는 것이 이 실습 부분에서 더 흥미로운 측면일 것입니다. 실제로 여러 스레드가 캐시에서 동시에 읽을 수 있어야 한다는 특별한 요구 사항이 있습니다. 물론 한 번에 하나의 스레드만 캐시에 쓸 수 있도록 허용되어야 하지만 판독기에는 이러한 제한이 존재해서는 안 됩니다.

따라서 하나의 큰 배타적 잠금으로 캐시에 대한 액세스를 보호하는 것은 허용 가능한 솔루션이 아닙니다. 캐시 분할, Pthreads 판독기-작성기 잠금 사용 또는 세마포어 사용과 같은 옵션을 탐색하여 고유한 판독기-작성기 솔루션을 구현할 수 있습니다. 두 경우 모두 엄격하게 LRU 퇴거 정책을 구현할 필요가 없다는 사실은 여러 판독기를 지원하는 데 어느 정도 유연성을 제공합니다.

7 평가

이 과제는 총 70점 만점으로 평가됩니다.

- 기본정확도: 기본 프록시 작업 40점(자동 등급)
- 동시성: 동시 요청 처리를 위한 15포인트(자동 등급)
- 캐시: 작동 중인 캐시의 경우 15포인트(자동 등급)

7.1 자동 채점

유인물 자료에는 강사가 기본 정확성, 동시성 및 캐시에 대한 점수를 할당하는 데 사용할 드라이버.sh라는 자동 채점기가 포함되어 있습니다. Proxylab-handout 디렉토리에서:

```
리눅스> ./driver.sh
```

Linux 시스템에서 드라이버를 실행해야 합니다.

7.2 견고성

언제나 그렇듯, 오류는 물론 잘못된 형식이나 악의적인 입력에도 견고한 프로그램을 제공해야 합니다.
서버는 일반적으로 장기 실행 프로세스이며 웹 프록시도 예외는 아닙니다. 장기 실행 프로세스가 다양한 유형의 오류에 어떻게 반응해야 하는지 신중하게 생각해 보세요. 많은 종류의 오류에 대해 프록시가 즉시 종료되는 것은 확실히 부적절합니다.

견고성은 분할 오류, 메모리 누수 및 파일 설명자 누수 부족과 같은 오류 사례에 대한 취약성을 포함하여 다른 요구 사항도 의미합니다.

8 테스트 및 디버깅

간단한 자동 그레이더 외에 구현을 테스트하기 위한 샘플 입력이나 테스트 프로그램이 없습니다. 코드를 디버깅하고 올바른 구현이 언제 있는지 결정하는 데 도움이 되는 자체 테스트는 물론 자체 테스트 도구도 마련해야 합니다. 이는 정확한 작동 조건을 거의 알 수 없고 참조 솔루션을 사용할 수 없는 현실 세계에서 귀중한 기술입니다.

다행히도 프록시를 디버그하고 테스트하는 데 사용할 수 있는 도구가 많이 있습니다. 모든 코드 경로를 실행하고 기본 사례, 일반적인 사례 및 극단적 사례를 포함한 대표적인 입력 세트를 테스트하십시오.

8.1 작은 웹 서버

CS:APP Tiny 웹 서버의 소스 코드가 담긴 유인물 디렉토리입니다. thttpd만큼 강력하지는 않지만 CS:APP Tiny 웹 서버는 원하는 대로 쉽게 수정할 수 있습니다. 이는 또한 프록시 코드의 합리적인 시작점이기도 합니다. 그리고 드라이버 코드가 페이지를 가져오는 데 사용하는 서버입니다.

8.2 텔넷

교과서(11.5.3)에 설명된 대로 텔넷을 사용하여 프록시에 대한 연결을 열고 HTTP 요청을 보낼 수 있습니다.

8.3 컬

컬을 사용하여 자체 프록시를 포함한 모든 서버에 대한 HTTP 요청을 생성할 수 있습니다. 매우 유용한 디버깅 도구입니다. 예를 들어 프록시와 Tiny가 모두 로컬 컴퓨터에서 실행되고 Tiny가 포트 15213에서 수신하고 프록시가 포트 15214에서 수신하는 경우 다음 컬 명령을 사용하여 프록시를 통해 Tiny에서 페이지를 요청할 수 있습니다.

linux> 컬 -v --proxy http://localhost:15214 http://localhost:15213/home.html * 프록시 로컬 호스트 포트 15214(#0)에 연결()하려고 합니다.

* 127.0.0.1 시도 중... 연결됨

```
* localhost(127.0.0.1) 포트 15214(#0)에 연결됨
> GET http://localhost:15213/home.html HTTP/1.1 > 사용자 에이전트: 컬/7.19.7 (x86_64-
redhat-linux-gnu)...
> 호스트: localhost:15213
> 수락: */*
> 프록시 연결: Keep-Alive
>
* HTTP 1.0, body 이후 close 가정 < HTTP/1.0 200 OK < Server: Tiny
Web Server < Content-length:
120 < Content-type: text/html


<
<html>
<head><title>테스트</title></head> <body> <img

align="middle" src="godzilla.gif">
데이브 오할라론
</body> </
html>
* 연결 #0 종료
```

8.4 넷캣

nc라고도 알려진 netcat은 다목적 네트워크 유틸리티입니다. Telnet과 마찬가지로 netcat을 사용하여 서버에 대한 연결을 열 수 있습니다. 따라서 프록시가 포트 12345를 사용하여 catshark에서 실행되고 있다고 가정하면 다음과 같은 작업을 수행하여 프록시를 수동으로 테스트할 수 있습니다.

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

HTTP/1.1 200 확인
 ...
```

netcat은 웹 서버에 연결할 수 있을 뿐만 아니라 서버 자체로도 작동할 수 있습니다. 다음 명령을 사용하면 포트 12345에서 수신 대기하는 서버로 netcat을 실행할 수 있습니다.

```
sh> nc -l 12345
```

netcat 서버를 설정한 후에는 프록시를 통해 가짜 개체에 대한 요청을 생성할 수 있으며 프록시가 netcat에 보낸 정확한 요청을 검사할 수 있습니다.

8.5 웹 브라우저

결국 최신 버전의 Mozilla Firefox를 사용하여 프록시를 테스트해야 합니다. Firefox 정보를 방문하면 브라우저가 자동으로 최신 버전으로 업데이트됩니다.

프록시와 작동하도록 Firefox를 구성하려면 다음을 방문하세요.

환경설정>고급>네트워크>설정

실제 웹 브라우저를 통해 프록시가 작동하는 모습을 보는 것은 매우 흥미로울 것입니다. 프록시의 기능은 제한되어 있지만 프록시를 통해 대부분의 웹사이트를 탐색할 수 있다는 것을 알게 될 것입니다.

중요한 주의 사항은 웹 브라우저를 사용하여 캐싱을 테스트할 때 매우 주의해야 한다는 것입니다. 모든 최신 웹 브라우저에는 자체 캐시가 있으므로 프록시 캐시를 테스트하기 전에 비활성화해야 합니다.

9 Handin 지침

제공된 Makefile에는 최종 Handin 파일을 빌드하는 기능이 포함되어 있습니다. 작업 디렉터리에서 다음 명령을 실행합니다.

리눅스> make handin

출력은 ../proxylab-handin.tar 파일이며, 이를 전달할 수 있습니다.

특정 사이트: 각 학생에게 Proxylab-handin.tar 솔루션 파일을 제출하는 방법을 알려주는 단락을 여기에 삽입하십시오.

- 교과서 10~12장은 시스템 수준 I/O, 네트워크 프로그램에 대한 유용한 정보를 담고 있습니다.
  ming, HTTP 프로토콜 및 동시 프로그래밍.
- RFC 1945(http://www.ietf.org/rfc/rfc1945.txt)는 다음의 전체 사양입니다.
  HTTP/1.0 프로토콜.

10개의 힌트

- 교과서의 섹션 10.11에서 설명한 것처럼 소켓 입력 및 출력에 표준 I/O 기능을 사용하는 것은 문제입니다. 대신 handout 디렉터리의 csapp.c 파일에 제공되는 RIO(Robust I/O) 패키지를 사용하는 것이 좋습니다.

- csapp.c에서 제공하는 오류 처리 기능은 서버가 연결 수락을 시작하면 종료되지 않으므로 프록시에 적합하지 않습니다. 이를 수정하거나 직접 작성해야 합니다.

- 유인물 디렉토리에 있는 파일을 원하는 대로 자유롭게 수정할 수 있습니다. 예를 들어, 우수한 모듈성을 위해 캐시 함수를 캐시.c 및 캐시.h라는 파일의 라이브러리로 구현할 수 있습니다. 물론 새 파일을 추가하려면 제공된 Makefile을 업데이트해야 합니다.

- CS:APP3e 텍스트 964페이지의 Aside에서 설명한 대로 프록시는 SIGPIPE 신호를 무시해야 하며 EPIPE 오류를 반환하는 쓰기 작업을 적절하게 처리해야 합니다.
- 때로는 조기에 닫힌 소켓에서 바이트를 수신하기 위해 read를 호출하면 errno가 ECONNRESET으로 설정된 read가 -1을 반환하게 됩니다. 이 오류로 인해 프록시가 종료되어서는 안 됩니다.
- 웹의 모든 콘텐츠가 ASCII 텍스트는 아니라는 점을 기억하십시오. 웹 콘텐츠의 대부분은 이미지, 비디오와 같은 바이너리 데이터입니다. 네트워크 I/O용 기능을 선택하고 사용할 때 이진 데이터를 고려해야 합니다.
- 원래 요청이 HTTP/1.1이더라도 모든 요청을 HTTP/1.0으로 전달합니다.

행운을 빌어요!