**2021 SYSTEM PROGRAMMING  
Lab6 Report – Proxy Lab**

자유전공학부 2012-13311 안 효 지

**0. 이 랩의 목적**

1) 프록시를 직접 구현함으로써 client와 server의 역할을 모두 처리할 수 있다.

2) concurrent server로도 구현해봄으로써 thread와 thread safe 에 대해 알 수 있다.

3) 이미 알고 있는 cache를 프록시에 구현해봄으로써 프록시의 유용함을 알 수 있다.

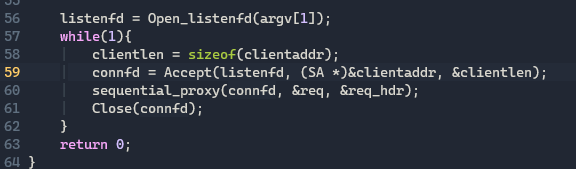
**1. How to implement**

**<Part A-sequential proxy>**

**1. main() : client로부터 request를 받는다.**

1) listenfd와 connfd를 선언한다.

2) 아래와 같이 listenfd를 열고 accept하기 위해 계속 기다린다.



3) accept가 되면 해당 connfd를 sequential\_proxy()라는 함수로 넘겨주어 처리되도록 한다.

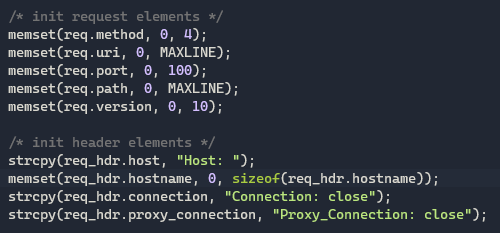
**2. void\* sequential\_proxy(): request를 parsing하여 server에게 보낼 request를 만든다.**

**1) Request, Request\_header structure**

- Request structure에는 method(GET), uri, port, path, version이 들어가도록 하였다.

- Request\_header에는 hostname, connection, proxy\_connection이 들어가도록 하였다. Hostname은 request로 들어오는 정보이나 서버로 request를 보낼 때에는 header에 들어가는 정보여서 request가 아닌 request\_header에 들어가도록 하였다.

**2) init structure elements**



- 현재 설명하고 있는 함수는 thread가 시작되는 함수이다. 따라서 thread가 시작될 때 앞서 설명한 structure의 모든 element들을 초기화 시켜준다. 그래야 기존 메모리에 남아있던 trash value들이 리셋되어 다음 작업을 할 때 영향을 미치지 않는다.

- request의 원소는 0으로, request\_header의 원소는 hostname을 제외하고는 다 정해진 값을 갖고있기 때문에 각각이 해당하는 값으로 hard coding하여 설정해주었다.

**3) parsing request line**



- 아주 간단하게 sscanf를 사용하여 method, uri, version으로 나누어주었다. 그 후 method가 GET인 경우만 다룰 수 있도록 설정해주었다.

**4) parsing uri**





- uri를 파싱하여 server에게 보낼 request를 만든다.

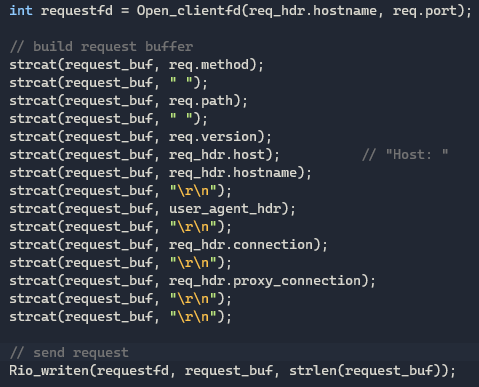
- `http://`로 시작하지 않으면 exit을 한다.

- `http://` 이후에 `/`로 시작하는 것이 있으면 그 이후의 값들을 받아와 path에 넣는다. 만약 없으면 기본 path는 `/` 로 설정해준다.

- uri에 `:`가 있으면 port의 값을 :뒤의 값으로 설정해준다. 없으면 기본 80.

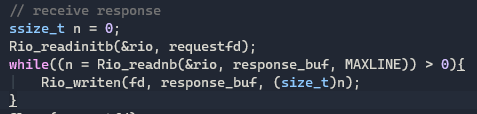
- hostname은 `http://` 이후부터 `/`, 혹은 `:`가 시작되기 전까지로 한다.  이와 같이 명시적으로 input에 `/`가 없더라도 client로부터 proxy에 들어올 때에는 이 형태로 들어오므로 고려하지 않아도 된다.

**5) make request**



버퍼에 request 형식에 따라 request line과 request header를 넣고 서버에게 request를 보낸다.

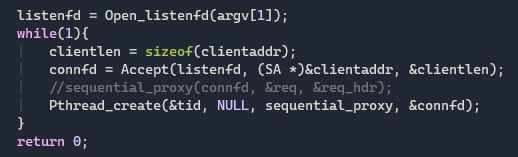
**3. 서버에게 response가 오면 그걸 받아서 client에게 다시 보내준다.**



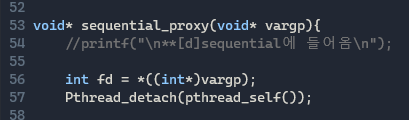
**<Part B – Concurrent proxy>**

Concurrent proxy를 구현하기 위해 Posix thread를 이용하였다. Part A에서 구현한 코드를 아주 조금만 수정하면 되어서 큰 어려움은 없었다.

1. main함수의 while문에 있던 sequential\_proxy() 호출 대신, Pthread\_create를 호출하여 sequential\_proxy를 thread에서 실행되도록 만든다.



2. 하지만 이렇게 하면, 기존 sequential\_proxy()에는 connfd, request\_line, request\_header, 이렇게 총 3개의 param을 설정 해놓았던 것을 한 개의 param으로 줄여야 했다. 따라서 원래는 그냥 main function에서 request\_line과 request\_header를 선언하고 init해준 후, thread에 arg로 전달을 해주었는데, 굳이 그럴 필요가 없는 것 같아 sequential\_proxy()로 옮겨주었다.



3. 그 후, thread에 들어가면 Pthread\_detach(pthread\_self())를 통해 자기 자신을 분리한다.

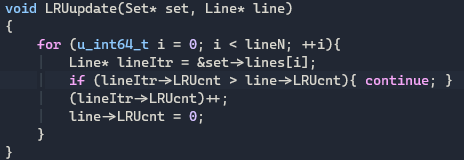
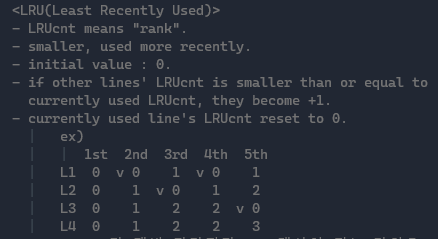
4. 마지막에  fd를 닫는 것을 깜빡하여 concurrent가 제대로 동작되지 않았던 것을 수정하였다.

**2. What was difficult**

① 처음에 디버깅하는 방법을 잘 몰라서 어려웠다. 평소 하나하나 printf문 써가면서 디버깅하면서 코딩을 하는 습관이 있는데, lab ppt에 나와있는 curl, telnet을 어떻게 사용할 지 잘 모르기도 했고, 그것은 다 완성한 후에 디버깅을 하는 용도로 사용하는 것이라고 생각해서 처음부터 사용할 생각 자체를 하지 못했다. 그래서 ./proxy portnum 만 친 상태에서 어떻게 client가 서버에게 request를 보낼 지 삽질을 하고 있었다. 하다하다 안 되어 다른 분들께 디버깅하는 방법만 질문했더니 터미널 하나를 더 키고나서 curl을 쓰면 된다고 했다. 그 답변을 듣고 curl에 대해 ppt에서 본 것 같아 찾아보니 프록시를 사용하여 curl을 이용하는 방법이 ppt자료에 그대로 나와있는 것을 발견하였다. 몇 시간 버리긴 했지만 이 방법을 알고 난 후부터는 진도가 빨라져서 괜찮았다.

② 그리고 web 내용 자체를 잘 이해를 못하고 있던 상태여서 시작이 더 어려웠다. 하지만 과제를 하고, 디버깅을 하면서 하나하나 실험을 하다 보니 이제 구조가 어떻게 되는지는 이해를 하였다. 내가 client로서 터미널에 curl –proxy localhost:4500 <http://www.sk.co.kr> 을 치면 이것이 내가 작성한 proxy server로 가고, 거기서 [www.sk.co.kr](http://www.sk.co.kr)로 다시 request를 보내는 것이라는 것을 알았다.

③ 또한 처음에 아무 생각 없이 concurrent proxy를 구현하면서 global var을 사용하였다. ./driver.sh로는 점수가 제대로 나왔다. 하지만 global static으로 하였더니 점수가 0점이 나왔다. global이든 global static이든 thread unsafe한 것은 마찬가지일텐데 왜 결과가 다르게 나오는 지 잘 모르겠지만, 어쨌든 thread safe하게 만들기 위해 global/static global 대신 local var로 변경해주었다.

 LRU를 두 가지 방법으로 구현했었다. 표에 나와있는 값은 update 후의 값이다.  
첫 번째 방법은 initial LRUcnt는 0이고, 가장 최근에 use되었을수록 0에 가까운 값을 가지도록 구현하였다. LRU의 범위는 [0, E)이다. 2nd에서 L1이 사용된 경우, 기존의 L1 LRU인 0보다 작거나 같은 LRU들은 +1씩 해주고 L1 LRU는 0으로 초기화를 해준다. 5th를 예로 들어보면, L3가 사용되어야 하기 때문에 그 전 L3의 LRU인 2보다 작거나 같은 L1, L2, L4에 1씩 더해주고, L3는 0으로 리셋을 시킨다. 이러면 evict당해야 하는 라인은 다른 라인들보다 큰 LRUcnt를 갖게 된다.   
처음에는 line->LRUcnt =0을 for문 밖에 위치시켰었다. 그랬더니 LRU의 업데이트가 제대로 이루어지지 않았다. 2nd처럼 L1부터 iteration을 돌면, 나 자신과 비교를 하기 때문에 작거나 같은 것에 해당되어 +1이 되어버린다. 따라서 기준이 망가지므로 그 다음 라인과 LRU값을 비교할 때 잘못된 계산을 하게 된다. 그래서 for문 안으로 넣어서 해결을 하였고, 지금 생각해보면 그냥 pointer로 역참조를 하는 것이 아니라 그냥 value로 값을 받아놓았어도 된다는 생각이 든다. 하지만, 여전히 남아있는 이 방법의 문제는 4th에서 보이는 것처럼 E-1보다 작은 최댓값을 보이는 경우가 있다는 것이다. 이러한 경우에는 모든 라인의 LRU를 비교하여 Max 값을 찾아내야 하는데 이는 너무나 비효율적이다. 따라서, evict 대상이 0을 갖도록 다시 설계를 한 것이다.

**3. What was surprising**

① thread를 끝낼 때에는 열려있는 fd를 다 닫아주어야 한다는 것은 알고 있었지만, 깜빡하고 닫아주지 않았더니 concurrent 점수가 갑자기 0점이 나왔다. 뭐든지 할당한 후에는 닫아주어야 한다는 교훈을 다시금 얻었다.

② request를 보낼 때 어떻게 보내는 지 굉장히 추상적으로 느껴졌었는데 직접 구현해보니 그냥 버퍼에 담고 rio를 이용해 써주기만 하면 된다는 것을 알았다. 유닉스는 모든 것을 파일로 여긴다는 사실도 잘 와 닿지 않았었는데 조금 더 구체화된 개념으로 머리에 들어온 것 같다. 간단하면서도 신기했다.

③ 과제를 하기 전에는 정말로 networking programming이 복잡하고 붕 뜬 개념으로 느껴졌었는데, 과제를 하면서 훨씬 구체적으로 이해할 수 있게 되었다.

**4. Result screen shot**

