시스템프로그래밍 실습

# 과제 5 proxy lab

2015-16828 김민규

## 1. 서론

간단한 프록시 서버를 구현하는 과제였습니다. cache.c에서 캐시를 구현하였고, proxy.c에서 캐시가 지원되는 프록시를 만들었습니다.

## 2. 본론

### 2.1. cache.c 구현

먼저 CachedItem이라는 struct를 만들었습니다. 이 struct는 hostname, pathname등 연결 정보와 그 연결로 받아온 데이터를 저장하는 struct입니다. 다음으로, CachedItem struct로 링크드 리스트를 만들어 캐시를 구현하였습니다. 이 때, 제일 나중에 추가된 아이템은 제일 끝에 위치하게 하여, 가장 앞에 있는 아이템이 LRU가 되도록 하였습니다.

#### 2.1.1. free_CachedItem 함수

cache용량이 넘치면 LRU순서대로 아이템을 캐시에서 제거해야 하는데, 이 때 제거된 아이템을 free해주는 함수입니다.

#### 2.1.2. cache_init 함수

캐시에 관련된 데이터들의 초기화를 위한 함수입니다.

#### 2.1.3. remove_LRU 함수

링크드 리스트 맨 앞의 아이템을 제거하고, 메모리 누수를 방지하기 위해 제거된 아이템을 free_CachedItem함수를 이용해 free합니다.

#### 2.1.4. create_CI함수

CachedItem struct를 하나 만들기 위한 함수로, hostname, pathname등의 연결 정보와 그 연결로 받아온 데이터를 argument로 받아 CachedItem을 하나 만듭니다.

#### 2.1.5. insert_cache함수

위에서 create_CI로 만든 CachedItem을 캐시 링크드 리스트의 맨 끝에 추가하기 위한 함수입니다. 추가하고 난 후의 사이즈가 MAX_CACHE_SIZE보다 크다면, remove_LRU함수를 이용해 LRU 아이템을 제거합니다.

#### 2.1.6. isSame함수

CachedItem 하나와 hostname, pathname등의 연결 정보를 비교하여, 같다면 1, 다르면 0을 반환합니다.

#### 2.1.7. find_cache함수

위에서 만든 isSame함수를 이용해, 링크드 리스트 전체를 훑으며 현재 연결 정보와 동일한 아이템이 있는지 확인합니다. 만약 있다면, 그 아이템을 move_to_end(2.1.8에서 서술)을 이용해 캐시 링크드 리스트 맨 뒤로 옮기고 그 아이템을 반환합니다. 없다면 NULL을 반환합니다.

### 2.1.8. move_to_end함수
링크드 리스트 안에 있는 CachedItem을 받아 그 아이템을 링크드 리스트 맨 뒤로 옮겨 주는 역할을 합니다. 제일 최근에 사용한 아이템을 제일 끝에 옮기는 규칙을 cache hit상황에서도 유지하기 위함입니다.

### 2.1.9. printCachedItems함수
디버깅 용도로 만든 함수로, 캐시 링크드 리스트에서 각각의 pathname을 출력합니다.

## 2.2. proxy.c 구현
먼저 Request라는 struct를 만들었습니다. method, uri, version, hostname, pathname, port를 각각 분리해 담기 위한 struct입니다.

### 2.2.1. main 함수
먼저 Open_listenfd로 소켓을 하나 만듭니다. 그다음 클라이언트로부터 연결 요청이 오면, Accept함수를 이용해 connfd에 이를 받은 후 thread를 하나 만들어 그 연결을 처리하게 합니다.  이를 무한정 반복합니다.

### 2.2.2. handle_client 함수
main에서 만든 thread를 핸들링하기 위한 함수입니다. 우선 Pthread_detatch로 메모리 누수를 방지합니다. 그 다음 connfd에서 Rio로 첫 요청을 읽어들입니다. 그 다음 parse_request(2.2.3)로 이 요청을 method,uri,version으로 나눠 Request struct에 담습니다. method가 GET인 상황만 가정하고, GET이 아니라면 종료하게 하였습니다. 이제, parse_uri(2.2.4)함수로 uri를 hostname과 pathname, port등으로 세부적으로 분리해 Request struct에 담습니다. 그 다음, get_from_cache(2.2.6)함수로 이 연결이 이미 캐싱되어 있다면 그 값을 그대로 connfd에 적고, 아니라면 get_from_server(2.2.7)함수로 서버에서 받아 와서 그 값을 캐싱하고 connfd에 적습니다.

### 2.2.3. parse_request 함수
sscanf를 이용해 단순히 띄어쓰기 구분으로 request를 파싱합니다.

### 2.2.4. parse_uri 함수
uri를 파싱하기 위한 함수입니다. 먼저 처음의 http://를 떼네고, ':'를 만나기 전까지를 hostname으로 합니다. 그 다음 ':'가 존재한다면, 그 뒤는 port번호로 파싱하였습니다. 만약 포트 번호가 따로 없는 경우, 기본 포트인 80을 넣었습니다.

### 2.2.5. parse_header 함수
연결하고자 하는 서버에 request를 보낼 때 적절한 헤더를 만들기 위한 함수입니다. 미리 정의된 header들을 붙이고,  추가로 client에서 보낸 헤더들이 있다면 같이 붙여 헤더를 만듭니다.

### 2.2.6. get_from_cache 함수
연결해 얻고자 했던 파일이 이전에 이미 해서 캐시에 존재한다면, 서버에 연결하지 않고 바로 캐시에서 그 파일을 전송하게 하는 함수입니다. find_cache(2.1.7)함수를 써 캐시에서 그 연결을 찾고, 만약 있다면 연결된 fd에 바로 그 파일을 쓰고 1을 반환하고, 아니라면 0을 반환합니다.

### 2.2.7. get_from_server 함수
get_from_cache에서 0을 반환하였다면, 캐시에서 찾지 못했다는 뜻이므로 get_from_server가 실행됩니다. 연결 정보를 가지고, Open_clientfd로 소켓을 열고 거기서 파일을 읽어 옵니다. 읽어온 파일을 connfd에 차례로 씁니

다. 그리고 또한 create_CI와 insert_cache함수를 이용해 캐시에 읽어온 파일을 기록합니다. 이 때 캐시에 데이터를 쓰는 과정은 csapp.h의 P함수와 V함수를 이용해 thread-safe하게 보호합니다.

# 3. 결론과 의견

Rio와 서버, 소켓, 그리고 thread-safe한 코드까지 이제껏 배운 것들을 다양하게 활용하는 과제인 만큼 까다로운 점도 많았습니다. 처음에는 프록시 서버와 타이니 서버를 동시에 실행시켜 놓고 거기에 curl등을 이용해 요청을 보내 디버깅해야 한다는 것을 이해하지 못 해 헤맸습니다. 다음으로 어려웠던 점은, 거의 모든 작업들을 포인터를 가지고 해야 했기 때문에 걸핏하면 segmentation fault가 떠서 디버깅하기 까다로웠습니다.