Proxy Lab

2019-11730 신현지

Goal of Proxy Lab

Proxy Lab에서는 웹 객체를 캐시하는 간단한 HTTP proxy를 작성할 것이다.

1. 기본적인 HTTP operation과 소켓을 사용하여 Sequential web proxy를 구현한다.
2. concurrent connection을 지원하도록 업데이트 한다.
3. 웹 객체를 캐싱하도록 업데이트 한다.

Implement

Part 1

1. 커맨드라인에서 프록시서버 `port`를 가져온다.
2. `Open_listenfd(port)`로 `listenfd`를 연다.
3. `Accept(listenfd, (SA *) &clientaddr, &clientlen)`로 `connfd`를 받는다.
4. `Rio_readinitb(&rio, connfd)`로 초기 설정을 한다.
5. 첫 번째 `Rio_readlineb(&rio, buf, MAXLINE)`로 읽은 HTTP Requests를 파싱한다.
    - `GET <uri> <version>` 형태임을 가정한다.
    - `uri`에서 `hostname`과 `port`를 추출한다.
    - `uri`에서 scheme과 `host`를 제외한 뒷 부분을 `path`로 파싱한다.
6. `/r/n`이 나올 때까지 읽어 header를 파싱한다.
    - Host, User-Agent, Connection, Proxy-Connection은 덮어쓸 것이므로 무시한다.
    - 이외의 header는 key, value 값으로 나누어 저장한다.
7. `Open_clientfd(hostname, port)`로 `clientfd`를 연다.
8. `Rio_readinitb(&rio, clientfd)`로 초기 설정을 한다.
9. `Rio_writen(clientfd, buf, strlen(buf))`를 반복해 request를 보낸다.
10. `Rio_readnb(&rio, buf, MAXLINE)`를 반복해 response를 받는다.
11. `Rio_writen(connfd, buf, n)`를 반복해 클라이언트에게 response를 보낸다.
12. `Close(connfd)`로 연결을 끊는다.

Part 2

1. `pthread_create(&tid, NULL, proxy_thread, connfdp)`로 스레드를 생성한다.

- `connfdp`는 Race Condition을 피하기 위해 `Malloc`으로 할당한다.

2. `proxy_thread`에서는 `connfdp`를 `connfd`로 바꾸고 `Free`한다.
3. `Pthread_detach(pthread_self())`로 스레드를 분리한다.
4. Part 1과 같이 진행한다.

Part 3

1. `cache_init()`로 캐시를 초기화한다.

- `head`, `tail`을 `NULL`로 초기화한다.
- `cache_size`를 0으로 초기화한다.
- `pthread_mutex_init(&cache_mutex, NULL)`로 뮤텍스를 초기화한다.

2. request를 받으면 캐시에 있는지 확인한다.

- `cache_find()`로 캐시에 있는지 확인한다.
- mutex로 캐시에 동시에 접근하는 것을 막는다.
- 캐시에 없으면 Part 1과 같이 진행한다.

3. 웹 객체를 한 번 가져오면 캐시에 저장한다.

- `cache_insert()`로 캐시에 저장한다.
- 캐시가 가득 차면 `cache_evict()`로 가장 오래된 웹 객체를 삭제한다.
- mutex로 캐시에 동시에 접근하는 것을 막는다.

4. `cache_free()`로 캐시를 해제한다.

- `cache_entry`를 `Free`한다.
- `pthread_mutex_destroy(&cache_mutex)`로 뮤텍스를 해제한다.

What I learned

- HTTP 요청과 응답의 기본적인 구조를 이해하게 되었으며, HTTP 헤더와 메서드의 역할에 대해 배울 수 있었다. 또한, URI를 파싱하는 방법을 배워 웹 요청을 처리하는 방법에 대해 이해할 수 있었다.
- pthread 라이브러리를 사용하여 concurrent programming을 했다. 동시에 여러 연결을 처리하는 방법을 배웠고, race condition을 피하는 방법에 대해 배울 수 있었다.
- 캐시를 구현하면서 LRU 알고리즘을 배웠다. 캐시를 구현하는 방법과 캐시를 사용하는 이유에 대해 배울 수 있었다.
- 여러 스레드가 공유 데이터에 동시에 접근하는 것을 방지하기 위해 뮤텍스를 사용하는 방법을 배웠다. 캐시에 대한 동시 접근을 제어하기 위해 뮤텍스를 사용하여 synchronization 문제를 해결했다.
- 소켓 프로그래밍을 통한 클라이언트와 서버 간의 네트워크 통신 방법을 이해하였다. 또한, Robust I/O 패키지를 사용하여 데이터를 안정적으로 읽고 쓰는 방법을 배웠다.

What was difficult

- 캐시를 구현하면서 LRU 알고리즘을 적용하는 과정이 어려웠다. 특히, 캐시의 크기를 제한하고, 가장 오래된 웹 객체를 제거하는 방법을 구현하는 것이 어려웠다. 또한, 캐시에 동시에 접근하는 것을 방지하기 위해 뮤텍스를 올바르게 사용하는 것이 어려웠다. 캐시의 동작이 알맞게 작동하는 것인지 테스트하는 것이 어려웠다.
- concurrent programming을 하면서 문제가 될 수 있는 부분을 찾는 것이 어려웠다. 한 번 실행하는 것으로는 에러가 발생하지 않을 수도 있기 때문에 여러 번 실행하면서 문제가 발생하는지 확인하는 것이 어려웠다.