System Programming

Programming Assignment #5 Proxy Lab

2013-11826 임주경

- Goal

이번 과제 Proxy Lab 에서는 간단한 HTTP proxy 를 구현한다. Proxy 는 GET 요청만을 처리하도록 하고, 클라이언트의 요청을 서버에 전달하여, 서버의 응답을 다시 클라이언트에게 전달하도록 구현한다. 기초적인 HTTP operation 을 이해하고, 네트워크 프로그래밍에서 socket 을 어떻게 사용해야 할지, concurrent 한 programming 과 caching 을 이용한 proxy 를 구현하며, 네트워크 프로그래밍의 이해를 도운다.

- Part #1. Implementing a sequential web proxy

우선, 기존 뼈대코드에 정의된 static const char * 형 변수 user_agent_hdr 를 비롯해, header parsing 에 사용될 같은 타입 변수들을 정의하였다. host_hdr, connection_hdr, proxy_connection_hdr, request_hdr, user_agent_macro, host_macro, connection_macro, proxy_connection_macro 와 같으며, 과제 스펙에 맞게 선언하였다.

Main 의 내부에는 강의자료의 echo server 를 참고하여 구현하였다. file descriptor 로 사용할 int 형

변수 listenfd, connfd 를 정의하였고, listenfd 는 Open_listenfd 함수의 반환 값으로 저장하였다. 구조체 타입 sockaddr_storage 를 클라이언트 주소로 정의했으며, while 문 내부에서 connfd 는 Accept 함수의 반환 값을 저장하였다.

doit 함수는 connection fd 를 인자로 받아서 클라이언트의 HTTP 명령을 처리한다. rio_t 변수 clientRio 와 endServerRio 로 클라이언트와 엔드 서버의 데이터를 buffer 에 저장할 수 있도록 하였다. 우선 Rio_readlineb 함수를 이용해 버퍼에 클라이언트의 request 를 저장한다. 이때, request 에 담긴 URI 는 parseURI 함수로 parsing 을 거쳐 host name, path, port 를 얻어낸다. parsing 을 통해 얻어낸 위의 값들을 이용해 end server 에 전송할 HTTP 헤더를 만든다. 이를 위해, buildHttpHdr 함수를 구현하였다. endServerfd 값은 Open_clientfd(hostname, portStr) 값으로 저장한다. 다음은 Rio_writen 함수를 이용해서, end server 에 위에서 얻은 HTTP 헤더값을 인자로 사용하고, while 문 내부에서 end server 로부터 받은 메시지를 클라이언트에 Rio_writen 함수를 이용해 전송한다. 더 이상 서버로부터 받은 메시지가 없으면 while 문은 종료되며, Close 함수로 서버를 닫는다. parseURI 함수는 인자로 받은 URI 를 이용하고, HTTP 명령어 구현이므로, port 값은 80 으로 설정한다. URI 로부터 해당 조건에 맞는 분기문을 거쳐 host name, path 값을 얻어낸다.

buildHttpHdr 함수는 클라이언트 rio 값을 인자로 받아, string 함수를 이용하여, end server http header 버퍼에 과제 스펙에 맞는 헤더 값을 저장한다.

- Part #2. Dealing with multiple concurrent requests

Concurrent 한 구현을 위해 thread 를 이용하였다. main 내부에서 while 문에서 매 connection 마다, Pthread_create 함수로 multi-Thread connection 을 구현하였다.

이때, Thread 함수는 강의자료를 참고하여, detach 하며, 각각은 Part 1 에서 구현한 doit 함수를 수행하도록 한다.

- Part #3. Caching web objects

과제에서 정의된 MAX_CACHE_SIZE 와 MAX_OBJECT_SIZE 값을 고려해 MAX_CACHE_NUM 을 10 으로 정의하였다. 새로운 구조체 타입 cacheBlock 을 선언하여, char 형 배열 obj, url 과 int 형 변수 isEmpty, LRU, rCnt 그리고, sem_t rMutex, wMutex 를 멤버로 갖는다. 이때, cache 간에 thread-safe 한 synchronization 구현을 위해, readers-writers (readers favor) 구현을 하였으며, rCnt 는 read count, rMutex, wMutex 는 각각 read mutex, write mutex 를 위한 멤버이다. 다음, 이러한 cache block 10 개로 이루어진 구조체 cache set 을 정의하였고, 글로벌 변수 cache set 구조체 변수 cache 를 선언하였다. part 1, part 2 에서 추가된 점은 doit 함수에서 URI 가 cache 에 존재하는지 확인하고, 존재할 경우 cache 를 통해 read, write 가 이루어지며, 이때, eviction policy 는 LRU policy 로 구현하였다. Caching 을 위해 구현한 함수는 cacheInit 으로 cache block 들의 값을 초기화 해주고, beforeRead, afterRead 와 beforeWrite, afterWrite 함수로 편의상 read, write 전후의 mutex 를 이용한 synchronization 구현을 선택하였다. 이 외에 해당 URL 이 cache 에 존재하는지 확인하는 cacheFind 함수와, eviction 을 위한 cacheEvict, cacheLRU , cacheURI 함수를 구현하였다.

- 마치며

이번 과제 구현에서 어려웠던 점은 thread-safe 구현을 위해, mutex 사용이 익숙하지 않아서 이 개념을 이해하는데 오래 걸렸다. 그래도 강의중 배운 readers-writers 구현을 이용해서 synchronization 구현을 이룰 수 있었고, 네트워크 프로그래밍은 처음이어서 해당하는 Header 의 형식을 숙지하는 것이 난해하였다. 과제 구현하는 과정을 통해서 네트워크 프로그래밍과 concurrent 하면서 synchronization 에 지장을 주지 않는 프로그래밍을 조금은 이해를 할 수 있게 되었다.