EE488

과제3 REPORT

20160542 이현지

- 전반적인 description
  - compile 방법: make
  - 총 제출 파일 수: client.c, client.h, client2.c, client2.h, server.c, server.h, threadpool.c, threadpool.h, tc_malloc.c, tc_malloc.h, makefile (11개)
  - 실행코드
    - server) ./server [abs_path] [port]
    - client1) ./client1 [ip] [port] [threadnum] [requestnum] [searchword]
    - client2) ./client2 [ip] [port]
      - mininet> 가 나오면 "search [searchword]" 입력
      - 없으면 ERROR: file is not in server
      - 존재하면 filename and line format에 맞게 나옵니다
  - tc_malloc을 추가하여 구현했습니다.

- server.c / server.h + threadpool.c / threadpool.h

- 실행코드) ./server [abs_path] [port]
- 과제 1번에 있던 structure와 함수를 가져와서 bootstrapping과 searching이 가능하게 하였다. 바뀐 사항이라면 이전에는 searching에서 찾으면 바로 print하게 하였는데, 이번에는 buffer 저장한 다음, 이를 client에 넘겨서 client에서 print되게 하였다.
- lecture note I/O multiplexing code를 기반으로 코드를 작성하였다. listen_fd라면 새로운 connection을 만들어주고, 아니라면 threadpool에 새로 일을 추가 하게 구현하였다.
- threadpool에서 10개의 thread로 pool을 만들어주고, FIFO 형식으로 thread가 일을 갖고 진행할 수 있게 했다.
- main function에서는 client와의 connection을 I/O multiplexing을 통해 구현하였고, client로 부터 읽어오는 경우, protocol에 맞게 해석하여 다 읽어올

때까지 while문을 돌렸고, write하는 경우 pagesize를 넘기는 경우는 고려하지 않아도 된다하여 while문을 제외하고 client에 search한 결과를 바로 보내주는 방식으로 구현하였다. client와 동일하게 protocol에 맞게 전달하였다.

- create_threadpool() 함수를 선언하여 threadpool이 처음 만들어지게 하였고, add_work_threadpool() 함수를 통해 새로운 worker이 threadpool에 추가되게 만들었다.
- worker에 대한 함수들도 만들어주었는데, 새로운 worker을 initialize 해주는 create_worker, 해당 worker을 없애주는 destroy_worker, 첫 번째 worker을 돌려주는 get_first_worker() 그리고 worker이 condition에 맞게 돌아갈 수 있도록 work_worker을 통해 working thread 개수를 조정해주었다.
- 또한, mutex lock을 이용하여 thread 간에 엇갈리는 상황을 방지해주었다.

- client.c / client.h

- 실행코드) ./client1 [ip] [port] [threadnum] [requestnum] [searchword]
- client1에 대한 코드로, client2와 다른 점은 input에 thread 개수와 각 thread에 대한 request 수를 설정해주어 이에 맞게 구현을 하였다.
- protocol에 맞게 찾고자 하는 단어를 server에 전달하였고, server로부터 전달된 값이 잘 전달되어 오는 것을 확인할 수 있었다.
- client1의 경우에 대해서는 server로부터 돌아온 값을 print하면 너무 느려서 print 코드는 제외했습니다.
- main function에서 c_arg structure에 필요한 인자를 넣어서 connection function에 전달하였고, 생성한 thread들이 connection 함수를 실행하게 하였다.

- client2.c / client2.h

- 실행코드) ./client2 [ip] [port]
- 실행코드) mininet> search [searchword]
- 과제 1에서 한 것과 같이 mininet을 만들어서 server 내에 있는 file에서 원하는 단어를 찾을 수 있게 하였다.
- server에 원하는 단어를 전달해주면, server에서 hw1에서 구현한 bootstrapping과 searching을 통해 찾은 문서와 줄을 돌려주어 이를 print하였다. 해당 단어가 server에 존재하지 않으면 protocol 내의 MSG 값을 통해 detect 하여 “ERROR: file is not in server”가 print 되게 하였다.

- hw1과 동일한 print 결과가 나오는 것을 확인할 수 있었다.
- client1과 동일하게 main function에서 필요한 인자들을 c_arg를 넣어서 connection 함수에 전달해주었다.

- tc_malloc.c / tc_malloc.h

- 2번 과제에서 구현한 tc_malloc을 가져와서 사용하였다. 자세한 내용은 과제2 report에 있으므로 생략하겠습니다.
- client와 server 둘 다에 대해 main function에서 central_init() / thread_init()을, 그리고 thread를 생성할 때 thread_init()을 해주어 각각의 thread에 대해 thread_local_cache를 만들어주어 tc_malloc이 가능하게 했습니다.