시스템프로그래밍 과제 보고서

Proxy 2-1

수업 명: 시스템프로그래밍 월5수6

담당 교수: 김태석 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

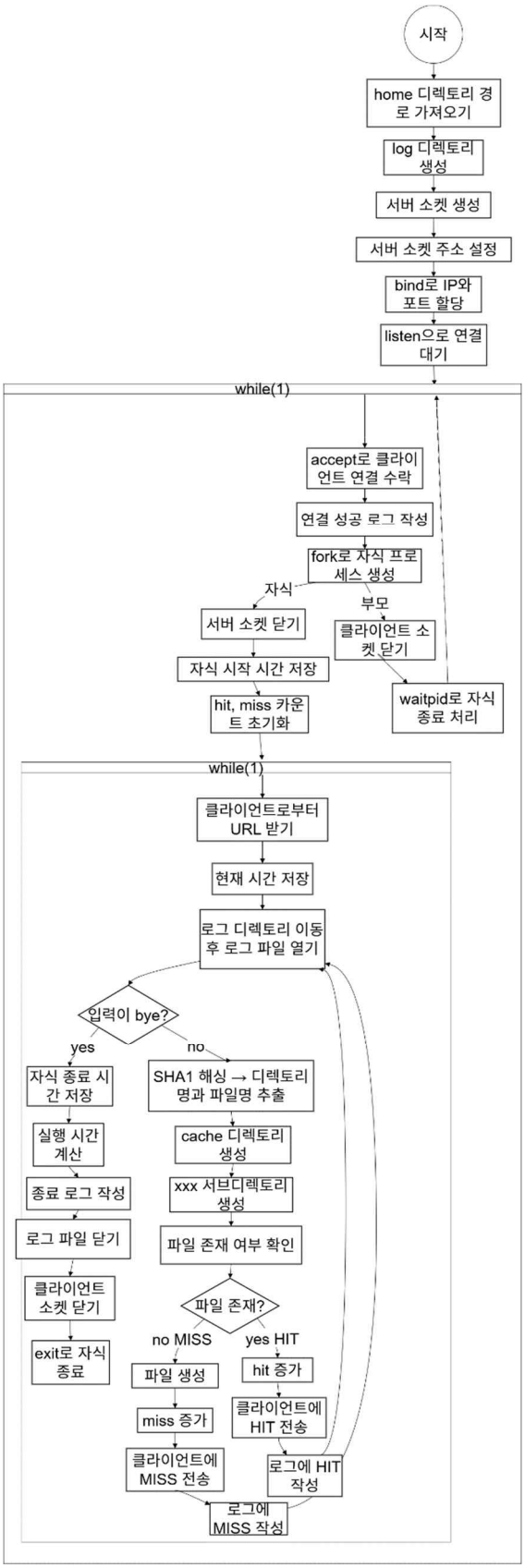
이름: 최현진

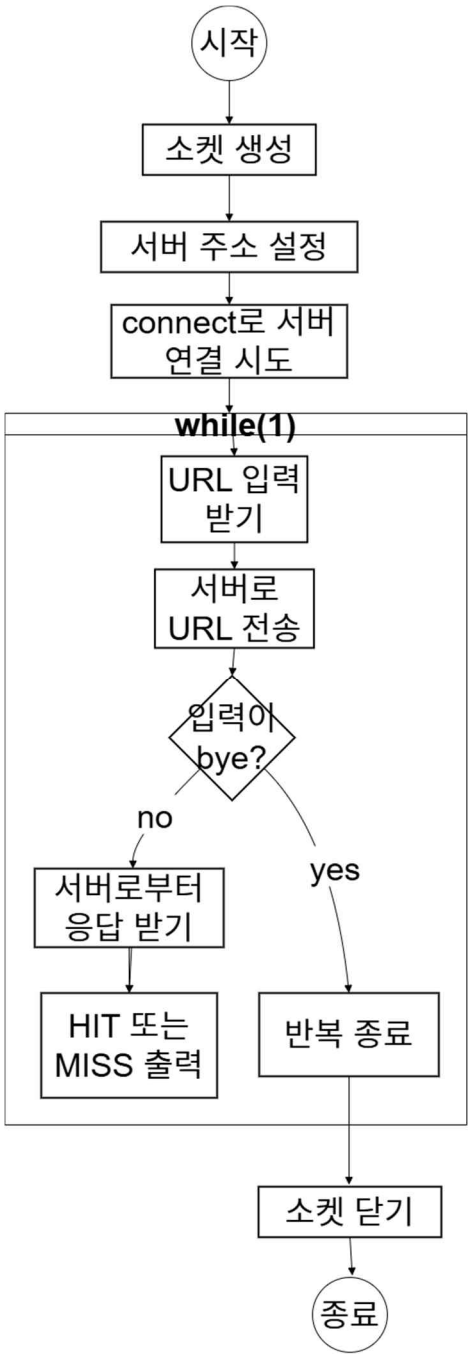
제출일: 2025.05.01

Introduction

이번 Proxy 2-1 과제는 Proxy 1-2에서 구현한 기능을 기반으로, 클라이언트의 동시 접속을 처리할 수 있는 멀티 프로세스 서버 구조를 구현하는 실습 과제이다. 서버는 클라이언트로부터의 연결 요청을 수락한 뒤, 각 연결마다 새로운 서브 프로세스를 생성하여 연산을 독립적으로 처리한다. 서브 프로세스는 Proxy 1-2와 동일하게 SHA1 해시를 통해 캐시 디렉토리 및 파일을 생성하고, 해당 캐시 파일의 존재 여부에 따라 HIT 또는 MISS 여부를 판단하여 결과를 클라이언트에 전송한다. 또한 모든 요청은 로그 파일에 기록되며, "bye" 명령이 입력되면 서브 프로세스는 종료 로그를 남기고 종료된다. 클라이언트는 사용자로부터 URL을 입력 받아 연결 되어있는 서버의 서브 프로세스로 URL을 전송한다. 서버는 fork(), accept(), waitpid() 등을 활용하여 클라이언트 요청을 병렬적으로 처리하며, 로그 출력을 위해 fopen(), fprintf(), fflush() 등을 활용한다. 네트워크 소켓 프로그래밍, 프로세스 제어를 아우르는 시스템 프로그래밍의 핵심 개념을 실습할 수 있다.

Flow Chart





server.c client.c

Pseudo code

(server.c) main:

```
home 디렉토리 경로 가져오기
log 디렉토리(~/logfile) 생성

서버 소켓 생성
서버 소켓 주소 설정
bind()로 IP/PORT 할당
listen()으로 연결 대기

무한 반복
    accept()로 클라이언트 연결 수락
    연결 성공 로그 작성

    fork()로 자식 프로세스 생성
    자식이면
        서버 소켓 닫기
        자식 시작 시간 저장
        hit, miss 카운트 초기화

        무한 반복
            클라이언트로부터 URL 받기
            현재 시간 저장
            로그 디렉토리로 이동하고 로그 파일 열기

            입력이 "bye"면
                자식 종료 시간 저장
                실행 시간 계산
                종료 로그 작성
                로그 파일 닫기
                클라이언트 소켓 닫기
                exit(0)로 자식 종료

            URL SHA1 해싱
            앞 3자리 → 디렉토리명
            뒤 37자리 → 파일명
```

~/cache 디렉토리 생성
~/cache/xxx 디렉토리 생성
거기로 이동해서 파일 있는지 확인

파일 없으면(MISS)
creat()로 파일 생성
miss++
클라이언트에 MISS 전송
로그에 MISS 작성

파일 있으면(HIT)
hit++
클라이언트에 HIT 전송
로그에 HIT 작성

로그 파일 닫기

부모면
클라이언트 소켓 닫기
waitpid()로 좀비 자식 처리

프로그램 종료

(client.c) main:

소켓 생성
서버 주소 설정
connect()로 서버 연결
 	
무한 반복
  사용자 입력 받기
  URL 서버로 전송
 	
  "bye"면 반복 종료
 	
  서버 응답(HIT/MISS) 받기
  화면에 출력
 	
소켓 닫기
종료

결과 화면

1. proxy 2-1

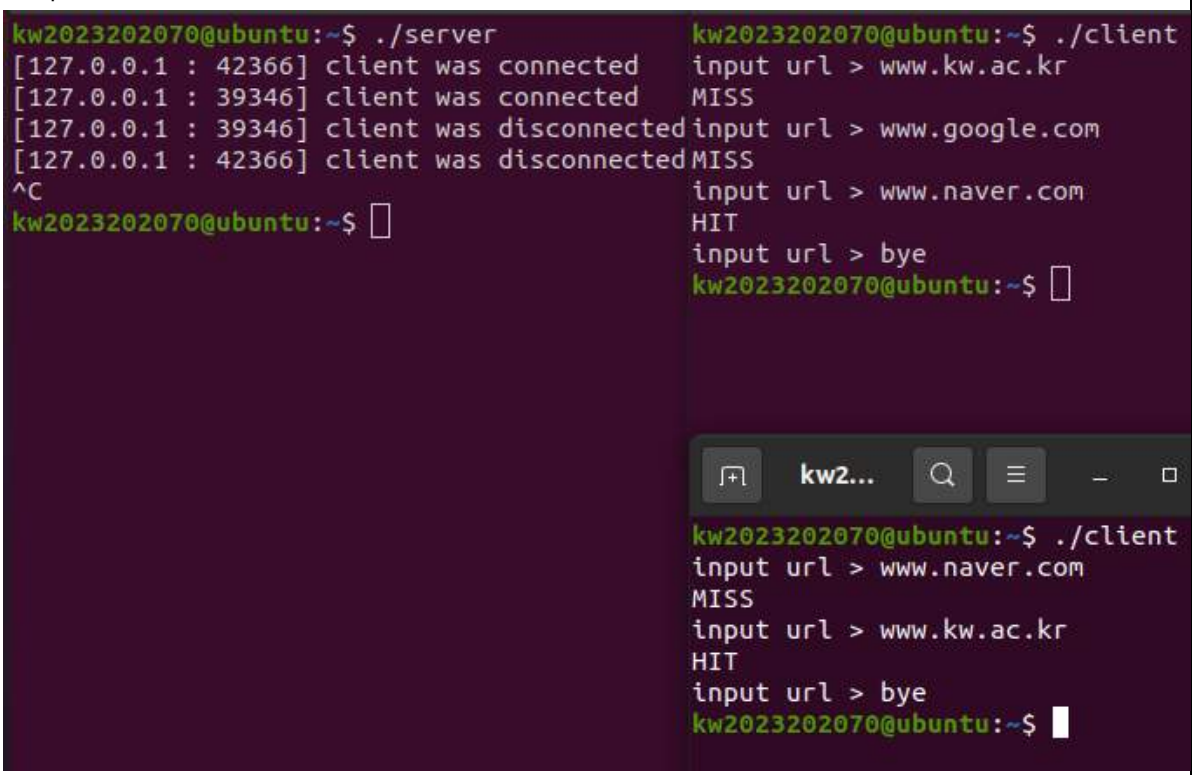
1. make

$ ls: Makefile과 소스 코드를 작성하였다.

$ make: make를 실행하여 컴파일을 수행하였고, gcc 컴파일 명령어가 정상적으로 실행되었다.

$ ls: 컴파일 결과, 실행 파일 server와 client가 생성된 것을 확인했다.

2. operation



Server: 두 클라이언트가 각각 연결되어 서브 프로세스가 생성되었고, 연결된 포트 번호는 42366(우상단), 39346(우하단)이다.

Client 1: www.kw.ac.kr 입력 → 캐시가 없어 MISS

Client 2: www.naver.com 입력 → 캐시가 없어 MISS

www.kw.ac.kr 입력 → Client 1이 캐시를 생성한 이후라 HIT

bye 입력 → 서버 연결을 끊고 종료

Server: 39346 클라이언트와 연결이 끊어지고, client was disconnected 메시지가 출력된다.

Client 1: www.google.com 입력 → 캐시가 없어 MISS

www.naver.com 입력 → 캐시가 이미 존재하므로 HIT

bye 입력 → 서버 연결을 끊고 종료

Server: 42366 클라이언트와 연결이 끊어지고, client was disconnected 메시지가 출력된다.

Ctrl + C → 서버 프로세스를 강제로 종료

3. logfile

```
kw2023202070@ubuntu:~$ cat ./logfile/logfile.txt
[MISS] ServerPID : 2915 | www.kw.ac.kr - [2025/05/01, 00:20:13]
[MISS] ServerPID : 2917 | www.naver.com - [2025/05/01, 00:20:18]
[HIT] ServerPID : 2917 | e00/0f293fe62e97369e4b716bb3e78fababf8f90 - [2025/05/01, 00:20:22]
[HIT] www.kw.ac.kr
[Terminated] ServerPID : 2917 | run time : 16sec. #request hit : 1, miss : 1
[MISS] ServerPID : 2915 | www.google.com - [2025/05/01, 00:20:29]
[HIT] ServerPID : 2915 | fed/818da7395e30442b1dcf45c9b6669d1c0ff6b - [2025/05/01, 00:20:33]
[HIT] www.naver.com
[Terminated] ServerPID : 2915 | run time : 29sec. #request hit : 1, miss : 2
```

$ cat ./logfile/logfile.txt: logfile.txt을 출력한 결과를 확인했다.

ServerPID: 2915는 client1와, 2917은 client2와 연결된 서버 서브 프로세스 id를 뜻한다.

2. operation의 결과화면의 시간 순서 동작에 맞게 서브 프로세스에서 hit/miss 결과를 client에 전송한 결과로, client1의 1번의 hit와 1번의 miss가 기록되었고, client 2의 1번의 hit와 2번의 miss가 기록되었다.

각 client가 bye를 입력했을 때는 종료 로그로 해당 client의 동작 시간과 hit/miss 개수가 출력된다.

4. cache directory

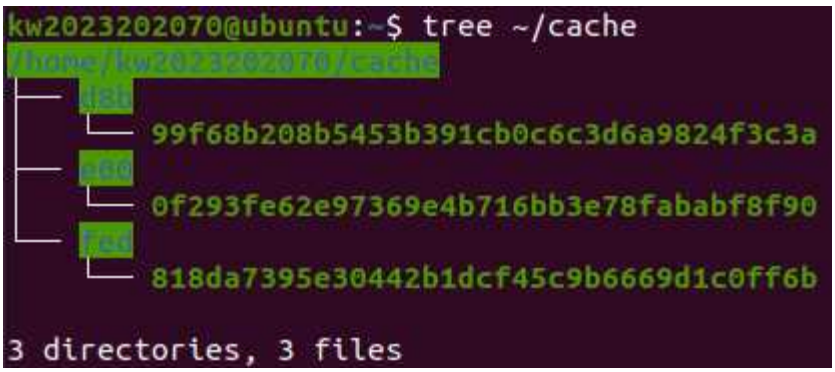
```
kw2023202070@ubuntu:~$ ls -R ~/cache
/home/kw2023202070/cache:
d8b  e00  fed

/home/kw2023202070/cache/d8b:
99f68b208b5453b391cb0c6c3d6a9824f3c3a

/home/kw2023202070/cache/e00:
0f293fe62e97369e4b716bb3e78fababf8f90

/home/kw2023202070/cache/fed:
818da7395e30442b1dcf45c9b6669d1c0ff6b
kw2023202070@ubuntu:~$
```

$ ls -R ~/cache: ls 명령어를 –R 옵션을 통해 재귀적으로 실행하여 ~/cache 디렉토리와 그 하위 디렉토리까지 출력하여 캐시 디렉토리와 파일들이 생성됨을 확인했다.



$ tree ~/cache/: ~/cache 구조 확인 결과, SHA1 해시된 url의 앞 3글자를 이름으로 하여 디렉토리가 생성되었다. 그 디렉토리의 하위에는 나머지 37글자 이름으로 파일이 3개 생성되었다.

고찰

이번 과제를 수행하면서 클라이언트와 서버 간의 소켓 통신 흐름을 전체적으로 이해할 수 있었다. 특히 서버에서는 socket(), bind(), listen(), accept() 순서로 동작하고, 클라이언트는 socket()과 connect()를 통해 서버에 연결된다는 구조를 직접 코드로 구현하며 흐름이 확실히 잡혔다. 이론 수업 때 단순히 개념만 알고 있었던 함수들이었지만, 이번 과제를 통해 어떤 순서로 호출되어야 하고, 어떤 역할을 하는지 몸으로 익힐 수 있었다.

또한 로그 파일을 처리하는 부분에서 시행착오가 많았는데, 처음에는 메인 프로세스에서 로그 파일을 열고 자식 프로세스들이 공유하도록 했더니, 동시에 접근할 때 로그 순서가 꼬이는 문제가 발생했다. 이를 해결하기 위해 각 자식 프로세스에서 별도로 로그 파일을 열고 fprintf() 다음에는 반드시 fflush()를 사용해서 버퍼에 남은 내용을 바로 기록하도록 수정했다. 이런 과정을 거치며 파일 스트림의 동작 방식과 버퍼링 문제도 함께 익힐 수 있었다.

전반적으로 Proxy 1-2에서 사용했던 SHA1 해시, 캐시 디렉토리 생성, HIT/MISS 판별 로직은 그대로 활용할 수 있어서 구현 자체는 익숙했다. 자식 프로세스를 만들고 종료를 waitpid()로 수거하는 방식은 새롭게 사용해봤다.

이번 과제를 통해 동시성 환경에서 어떤 문제가 발생할 수 있고 그것을 어떻게 안정적으로 처리해야 하는지 고민해보는 계기가 되었고, 2-2 과제부터는 더 복잡할 텐데 기초를 탄탄히 잡아두는 계기가 되었다.

Reference

시스템프로그래밍 이론 및 실습 자료 참고하였습니다.