시스템프로그래밍
Proxy #2-1

담당교수 : 최상호 교수님(목4)

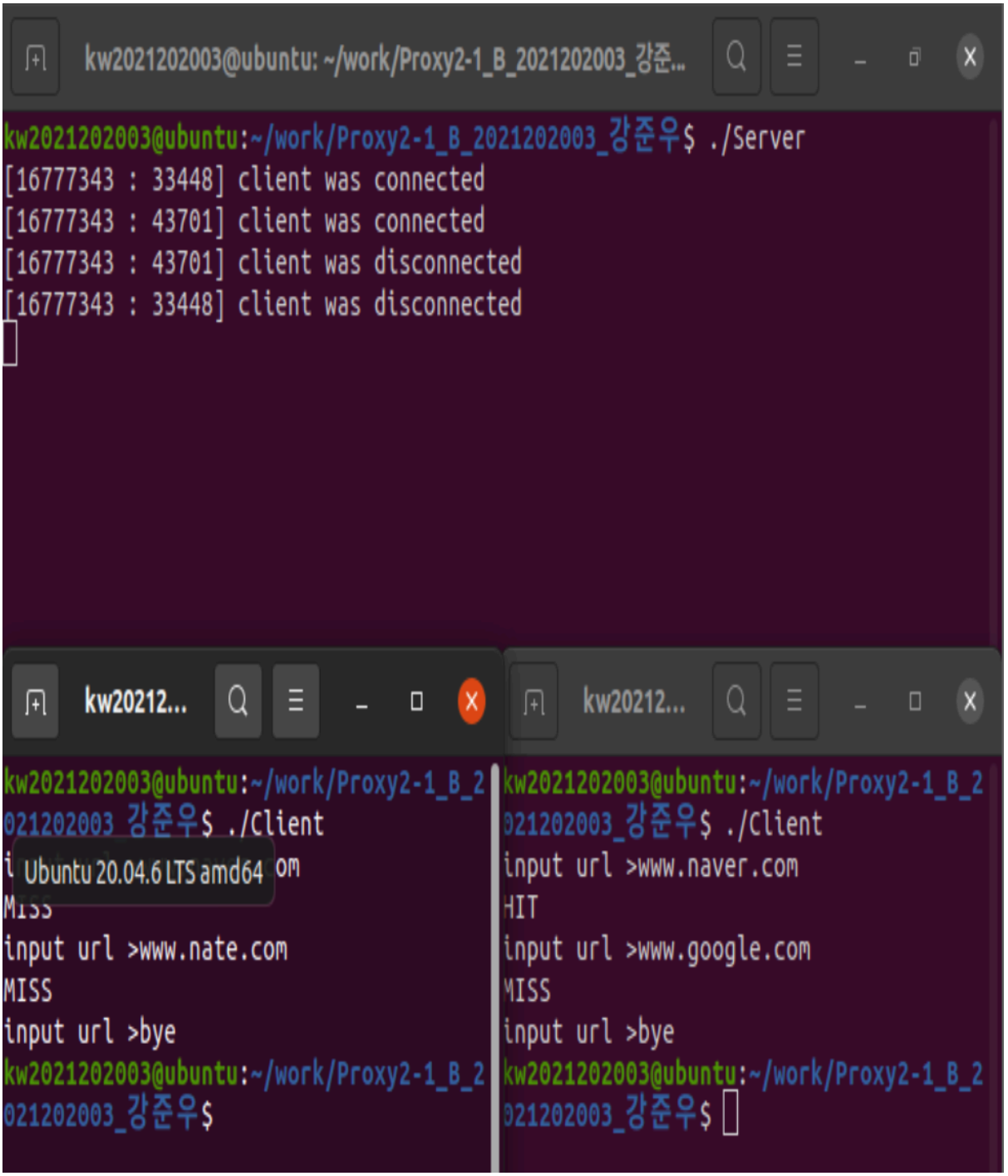
2021202003 강준우

Proxy #2-1

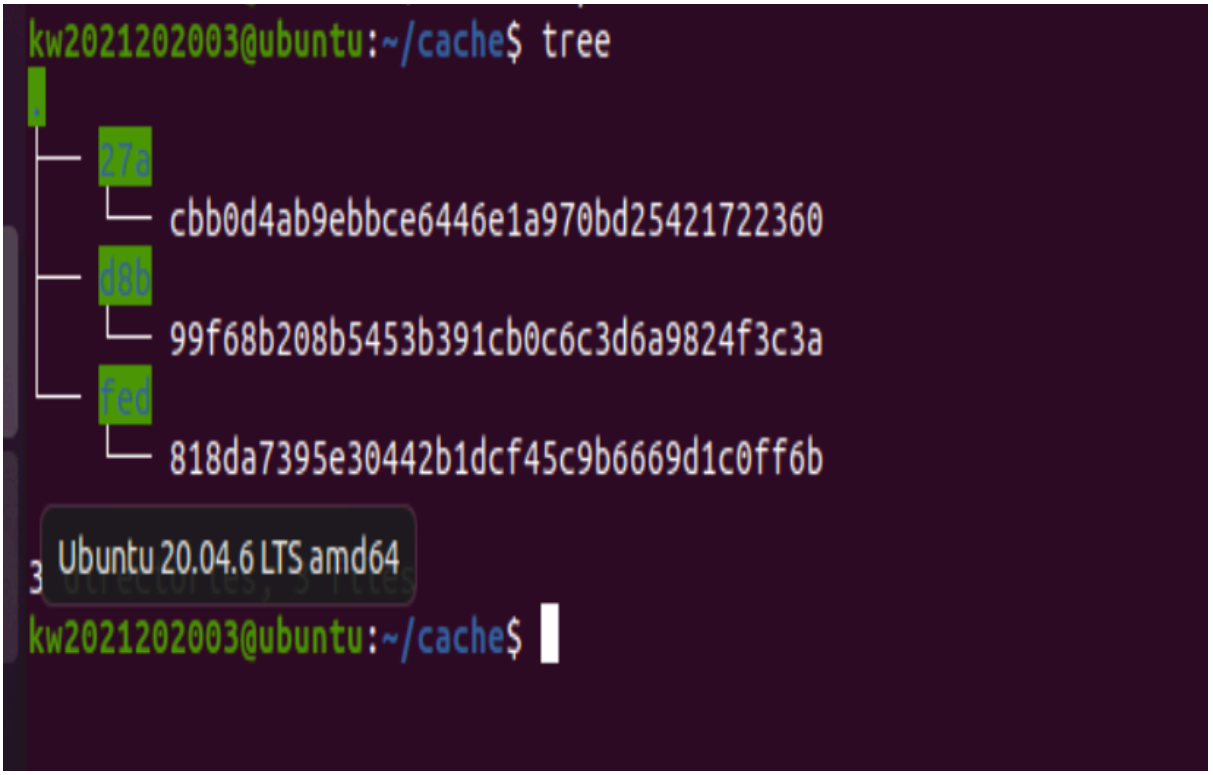
Introduction

이번 과제에서는 기존의 Proxy#1-2를 바탕으로 클라이언트와 서버를 각각 나눠서 구현합니다. 이 때 서버는 클라이언트의 접속을 확인하고 URL을 수신받아 해시화 및 파일 생성 hitmiss를 판단합니다.
이후 hit miss에 대한 내용을 다시 클라이언트에게 송신합니다. 클라이언트에서는 URL을 입력하고 hitmiss에 대한 정보를 서버로부터 받는 작업을 수행합니다.

결과화면



서버에서는 연결과 비연결을 확인한다,
클라이언트에서는 URL을 입력하고 HIT MISS 결과를 확인한다.
bye를 통해서 연결을 끊는다.

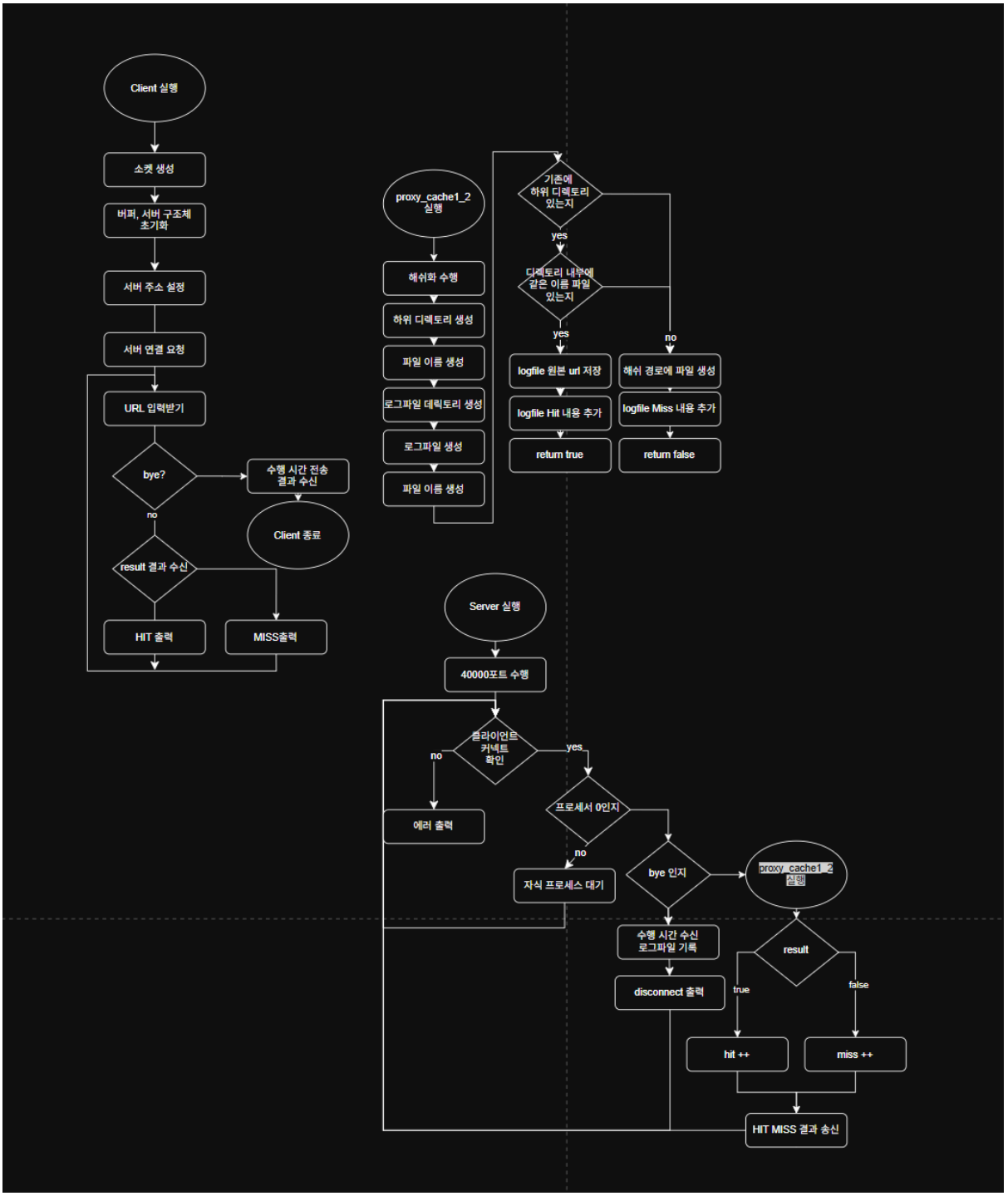


3번의 miss로 3개의 파일이 해쉬화되어 생성된 모습이다.

```
1 [Miss] ServerPID : 2678 | www.naver.com-[2025/05/01, 13:31:47]
2 [Hit] ServerPID : 2680 | fed/818da7395e30442b1dcf45c9b6669d1c0ff6b-
  [2025/05/01, 13:31:53]
3 [Hit]www.naver.com
4 [Miss] ServerPID : 2680 | www.google.com-[2025/05/01, 13:32:02]
5 [Miss] ServerPID : 2678 | www.nate.com-[2025/05/01, 13:32:13]
6 [Terminated] ServerPID : 2680 | run time : 0 sec. #request hit : 1, miss : 1
7 [Terminated] ServerPID : 2678 | run time : 0 sec. #request hit : 0, miss : 2
```

각각의 PID넘버와 HIT MISS TERMINATED의 결과가 기록되어 있다.

Algorithm – Flow Chart



Pseudo Code

Server:
홈 디렉토리 경로 생성
로그파일 디렉토리를 만들고 로그파일 경로를 설정

소켓을 생성
서버 주소 정보를 초기화

소켓을 주소에 바인딩
클라이언트 연결을 대기하기 위해 listen()을 호출
SIGCHLD 시그널 핸들러를 등록한다 (좀비 프로세스 방지)

무한 반복:
  클라이언트 연결을 accept()
  fork()를 통해 자식 프로세스를 생성

  자식 프로세스에서는:
    hit, miss 카운터를 0으로 초기화
    반복:
      클라이언트로부터 URL 또는 "bye" 메시지를 수신

      "bye"인 경우:
        클라이언트로부터 실행 시간(runtime)을 수신
        로그파일에 종료 메시지와 hit/miss 횟수를 기록
        반복 종료

      그 외의 경우:
        proxy_cache1_2() 함수로 캐시 여부를 확인한다
        결과를 클라이언트에 전송
        hit 또는 miss 카운터를 증가

    클라이언트 연결을 닫고 자식 프로세스 종료

  부모 프로세스는 클라이언트 소켓을 닫고 다음 연결 대기

Client:

소켓을 생성
서버 주소 정보를 설정한다 (IP, 포트 등)
서버에 연결을 시도
"input url >"를 출력

반복:
  사용자 입력을 읽고 개행 문자를 제거
  입력값을 서버에 전송

  입력값이 "bye"라면:
    현재 시간을 측정하여 실행 시간을 계산
    실행 시간을 서버에 전송
    반복 종료

  서버로부터 캐시 여부 결과(result)를 1바이트 수신

  수신된 결과에 따라:

true → "HIT" 출력
false → "MISS" 출력

"input url >"를 다시 출력

소켓을 닫고 프로그램 종료

고찰

이번 과제에서 서버와 클라이언트의 역활을 나누고 이전 과제에서 구현한 fork를 통해서 여러 클라이언트에게 서비스를 제공하는 모습을 구현해보았습니다. fork의 자식 프로세서에서 변한 변수는 메인에서는 변동이 없는점을 이용해 hit miss를 계산할 수 있었습니다.

구현 과정에서 echo를 통한 소통이 있었는데 TCP에서 write결과가 클라언트에 read부분에서 중첩되어 hit miss + echo를 보내 proxy 1_2의 결과와 리턴되는 출력 결과가 다른 상황을 경험했습니다. 이후 echo를 제거하여 hit miss만 전송하여 올바른 서버 수행을 할 수 있었습니다.

서버와 클라이언트간의 소통이 정확하게 제한되어야 한다는 사실을 알게 되었습니다.

Reference

2025-1_SPLab_proxy_Assginment2-1