시스템프로그래밍 과제 보고서

Proxy 2-2

수업 명: 시스템프로그래밍 월5수6

담당 교수: 김태석 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

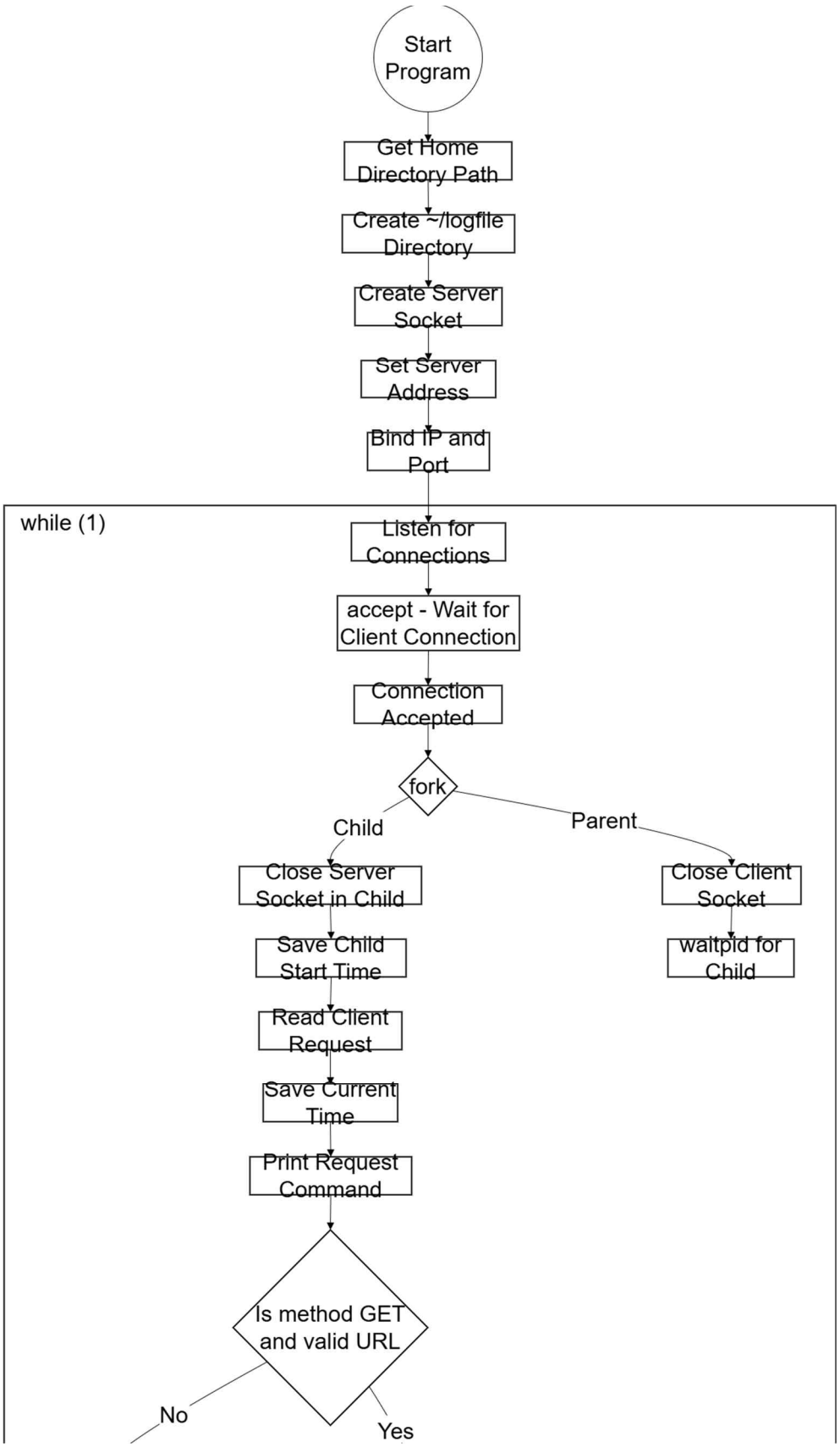
이름: 최현진

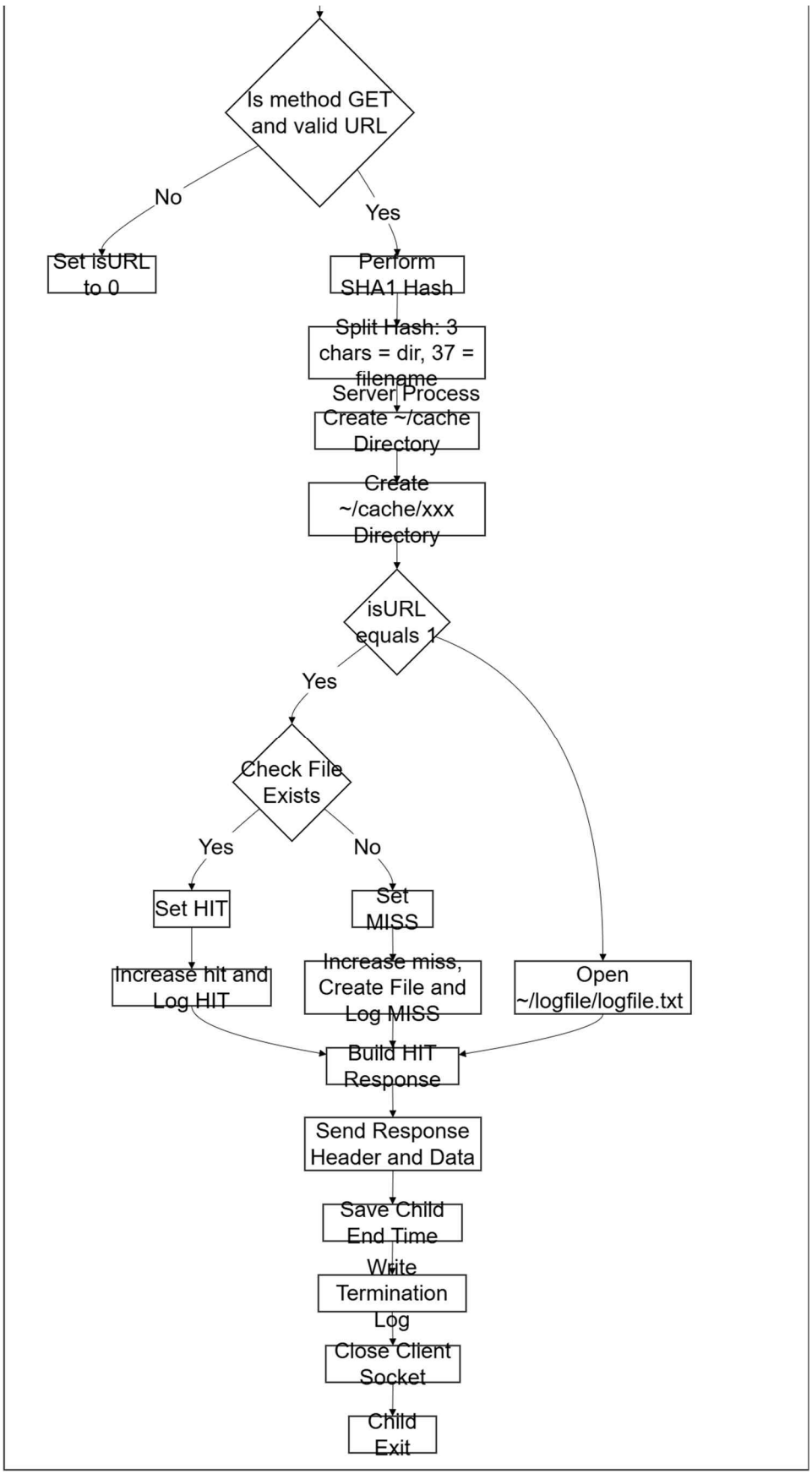
제출일: 2025.05.08

Introduction

이번 Proxy 2-2 과제는 이전 Proxy 과제에서 구현한 기능을 확장하여, 클라이언트 요청을 처리할 수 있는 멀티 프로세스 기반의 프록시 서버를 구현하는 실습 과제이다. 프록시 서버는 클라이언트(웹 브라우저)로부터 URL 요청을 수신하고, 각 요청에 대해 출력 후 자식 프로세스를 fork하여 독립적으로 처리한다. 자식 프로세스는 SHA1 해시를 기반으로 캐시 디렉토리 및 파일을 생성하고, 해당 캐시의 존재 여부에 따라 HIT 또는 MISS를 판별한다. 판별 결과는 클라이언트에 응답으로 전송되며, 모든 요청 기록은 콘솔에 기록된다. 자식 프로세스 종료 시 실행 시간과 HIT/MISS 요청 통계를 로그로 남긴다. 본 과제를 통해 시스템 프로그래밍의 핵심인 프로세스 제어, 네트워크 통신, 파일 입출력, 시그널 제어와 프록시 서버 구조에 대해 종합적으로 학습할 수 있다.

Flow Chart


Start Program
Get Home Directory Path
Create ~/logfile Directory
Create Server Socket
Set Server Address
Bind IP and Port
while (1)
Listen for Connections
accept - Wait for Client Connection
Connection Accepted
fork
Child
Parent
Close Server Socket in Child
Close Client Socket
Save Child Start Time
waitpid for Child
Read Client Request
Save Current Time
Print Request Command
Is method GET and valid URL
No
Yes


Is method GET and valid URL
No
Yes
Set isURL to 0
Perform SHA1 Hash
Split Hash: 3 chars = dir, 37 = filename
Server Process Create ~/cache Directory
Create ~/cache/xxx Directory
isURL equals 1
Yes
Check File Exists
Yes
No
Set HIT
Set MISS
Increase hit and Log HIT
Increase miss, Create File and Log MISS
Open ~/logfile/logfile.txt
Build HIT Response
Send Response Header and Data
Save Child End Time
Write Termination Log
Close Client Socket
Child Exit

Pseudo code

main:

```
프로그램 시작
    홈 디렉토리 경로 얻기
    ~/logfile 디렉토리 생성

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

    무한 반복:
        accept()로 클라이언트 연결 수락
        연결 성공 메시지 출력

        fork()로 자식 프로세스 생성
        자식이면:
            서버 소켓 닫기
            자식 시작 시간 저장

            클라이언트 요청 받기
            현재 시간 저장
            요청 cmd 출력

            method가 GET이 아니거나 URL이 없거나 html/css/txt/ico 파일이면 isURL = 0

            SHA1 해시 수행
            앞 3자리 → 디렉토리 이름
            뒤 37자리 → 파일 이름

            ~/cache 디렉토리 생성
            ~/cache/xxx 디렉토리 생성

            파일 존재 여부 확인 (isURL일 때만):
                존재 → HIT
                없으면 → MISS
```

~/logfile 디렉토리로 이동 후 logfile.txt 열기
flock으로 락 획득

HIT이면:
hit++
클라이언트에 HIT 응답 전송
로그에 HIT 기록

MISS이면:
miss++
파일 생성
클라이언트에 MISS 응답 전송
로그에 MISS 기록

응답 헤더+본문 전송

자식 종료 시간 저장
자식 종료 로그 기록
연결 종료 메시지 출력
클라이언트 소켓 닫기
자식 종료

부모면:
클라이언트 소켓 닫기
waitpid()로 좀비 자식 수거

서버 소켓 닫고 프로그램 종료

결과 화면

1. proxy 2-2

1. make

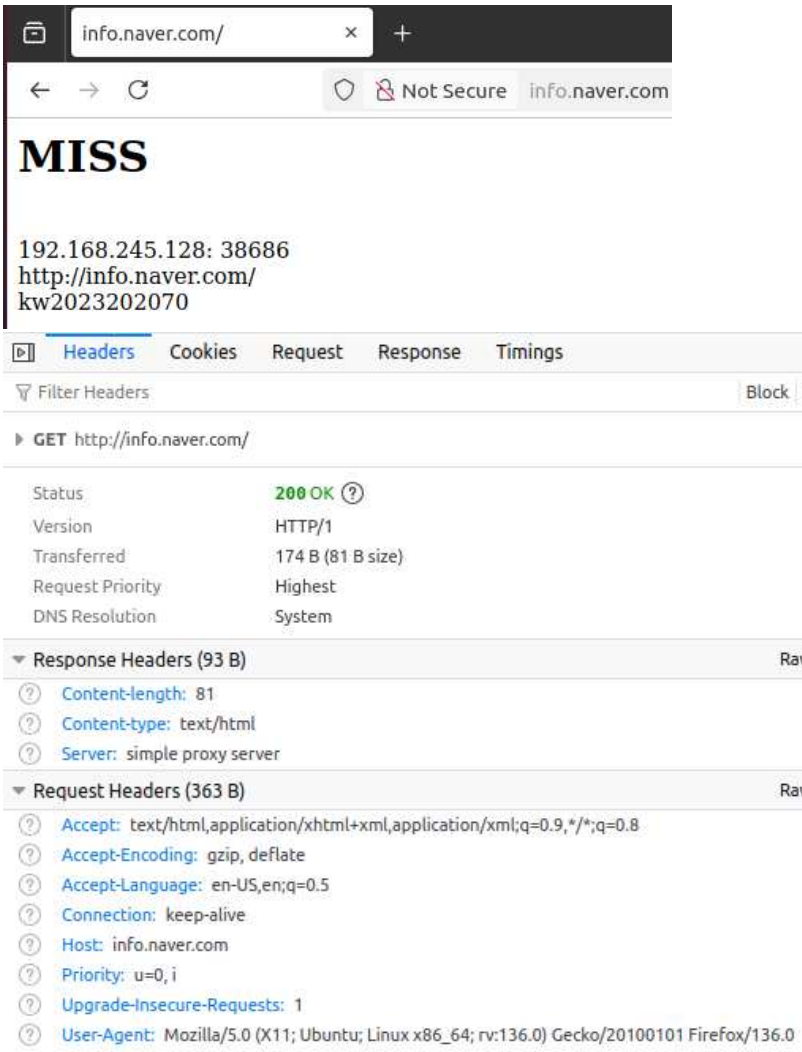
```
kw2023202070@ubuntu:~$ ls
Desktop  Documents  Downloads  Makefile  Music  Pictures  proxy_cache.c  Public  Templates  Videos
kw2023202070@ubuntu:~$ make
gcc proxy_cache.c -o proxy_cache -lcrypto
kw2023202070@ubuntu:~$ ls
Desktop  Documents  Downloads  Makefile  Music  Pictures  proxy_cache  proxy_cache.c  Public  Templates  Video
kw2023202070@ubuntu:~$
```

$ ls: Makefile과 소스 코드를 작성하였다.

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

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

2. operation

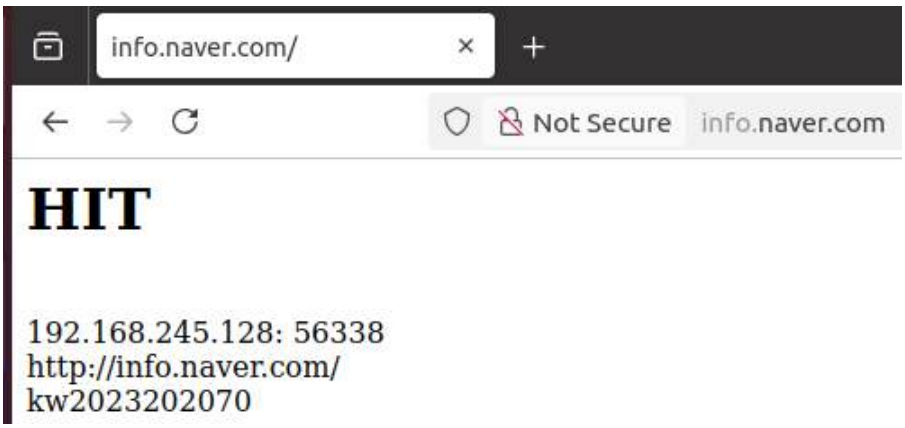


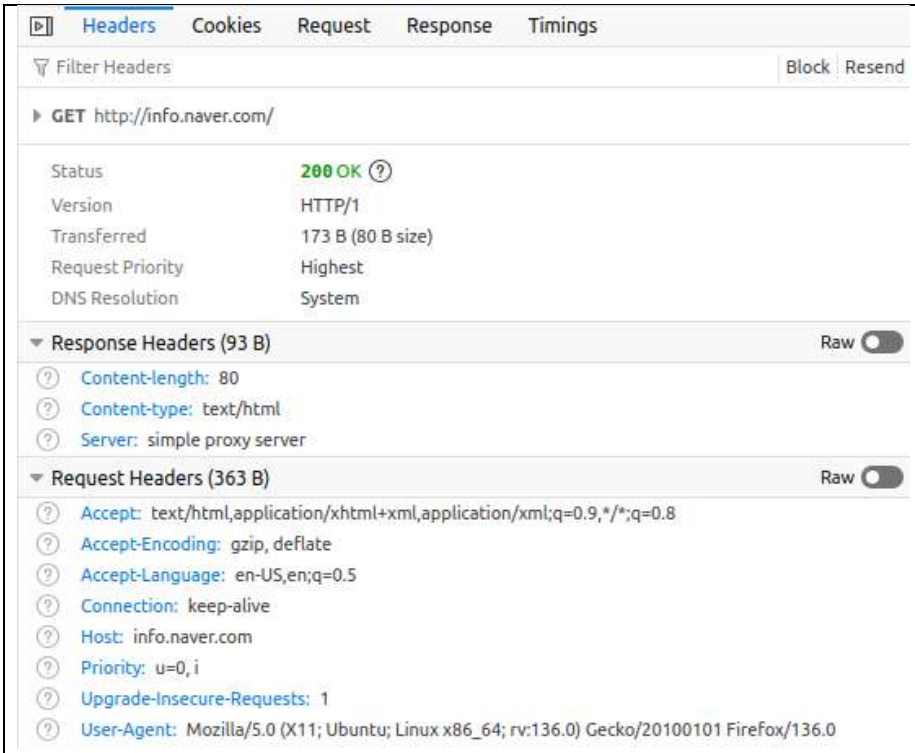
http://info.naver.com/ 입력 후, 프록시 서버에서 브라우저로 전송된 MISS 응답 데이터와 content-length, content-type, server를 지정해서 생성한 헤더를 확인했다.

```
[192.168.245.128 : 38686] client was connected
================================================
Request from [192.168.245.128 : 38686]
GET http://info.naver.com/ HTTP/1.1
Host: info.naver.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u=0, i

================================================
[192.168.245.128 : 38686] client was disconnected
[192.168.245.128 : 38698] client was connected
================================================
Request from [192.168.245.128 : 38698]
GET http://info.naver.com/favicon.ico HTTP/1.1
Host: info.naver.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://info.naver.com/
Priority: u=6

================================================
[192.168.245.128 : 38698] client was disconnected
```

콘솔에는 들어온 모든 요청을 기록한다. 실제로는 아이콘 요청 또한 들어와서, 38686, 38698 포트에 해당하는 두 브라우저 요청이 프록시 서버로 들어왔지만, GET http://info.naver.com/ 요청의 직접 입력한 url 부분만을 추출하여 캐시를 생성한다.

해당 결과 사진의 경우 이외에도 파이어폭스 자동 요청이나 브라우저의 자동 재요청이 들어오는 경우가 있어, CONNECT 메소드 요청과 .txt, .html 요청 등을 처리해야 한다.

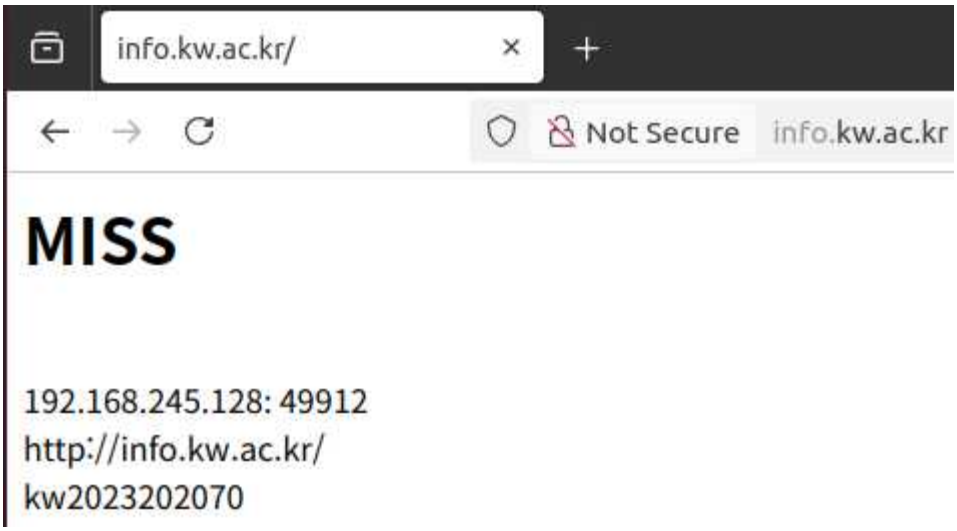


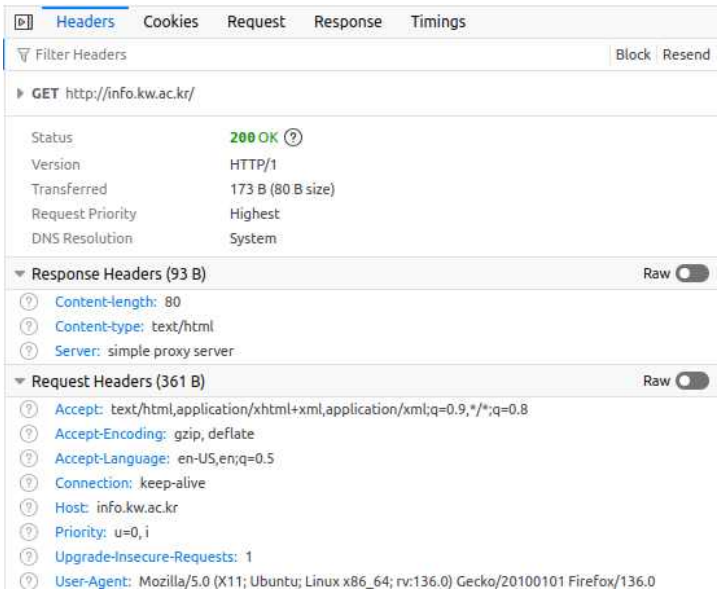


한 번 더 http://info.naver.com/ 입력 시 HIT 응답 데이터와 헤더를 확인했다.

```
[192.168.245.128 : 56338] client was connected
================================================
Request from [192.168.245.128 : 56338]
GET http://info.naver.com/ HTTP/1.1
Host: info.naver.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u=0, i

================================================
[192.168.245.128 : 56338] client was disconnected
```

콘솔에는 프록시 서버가 받은 56338 포트번호 요청이 출력됐다.

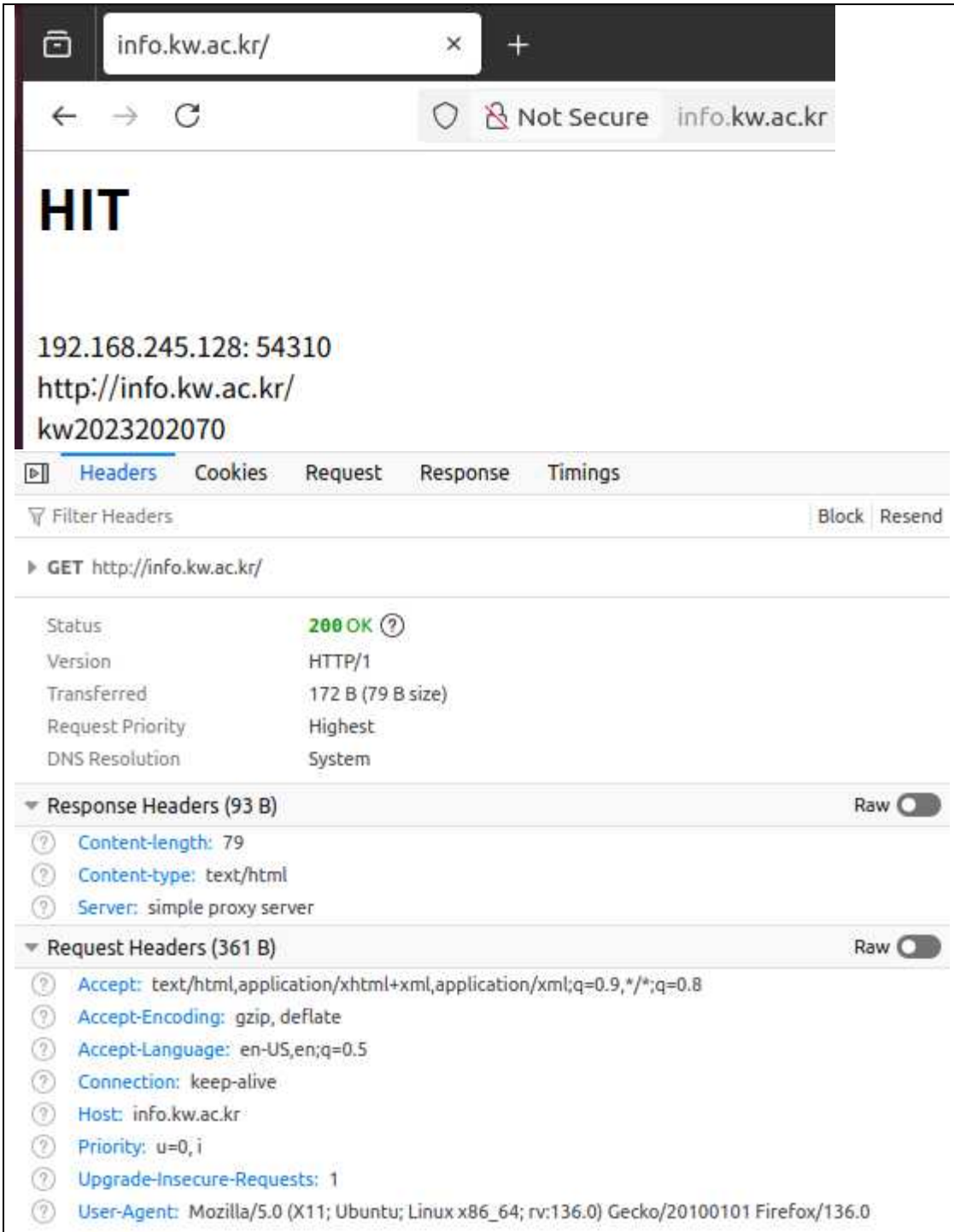




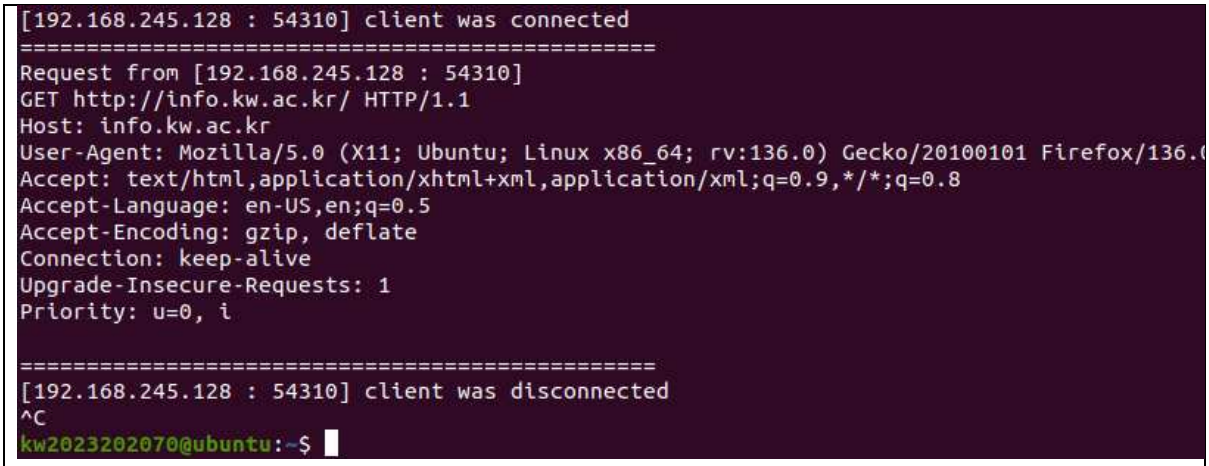
이번에는 http://info.kw.ac.kr/ 입력 후, 프록시 서버에서 브라우저로 전송된 MISS 응답 데이터와 헤더를 확인했다

```
[192.168.245.128 : 49912] client was connected
================================================
Request from [192.168.245.128 : 49912]
GET http://info.kw.ac.kr/ HTTP/1.1
Host: info.kw.ac.kr
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Priority: u=0, i

================================================
[192.168.245.128 : 49912] client was disconnected
[192.168.245.128 : 49924] client was connected
================================================
Request from [192.168.245.128 : 49924]
GET http://info.kw.ac.kr/favicon.ico HTTP/1.1
Host: info.kw.ac.kr
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://info.kw.ac.kr/
Priority: u=6

================================================
[192.168.245.128 : 49924] client was disconnected
```

마찬가지로 콘솔에는 들어온 모든 요청을 기록한다. 실제로는 아이콘 요청 /favicon.ico 또한 들어와서, 49912, 49924 포트에 해당하는 두 브라우저 요청이 프록시 서버로 들어왔지만, GET http://info.kw.ac.kr/ 요청의 직접 입력한 url 부분만을 추출하여 캐시를 생성한다.

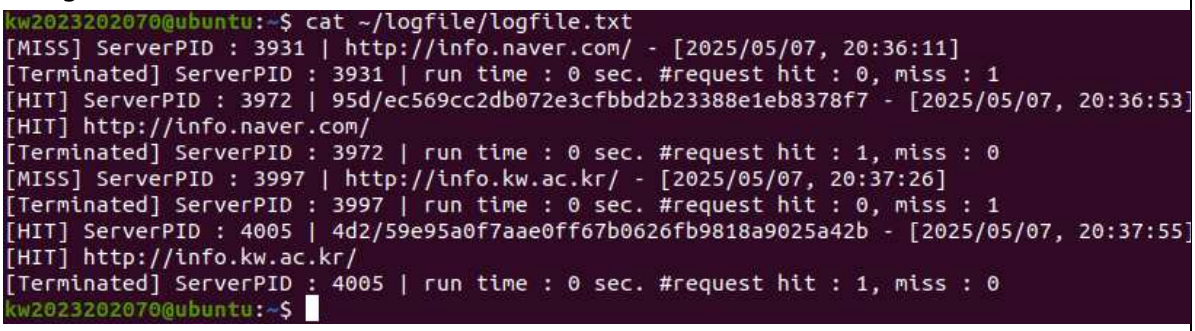


한 번 더 http://info.kw.ac.kr/ 입력 시 HIT 응답 데이터와 헤더를 확인했다.



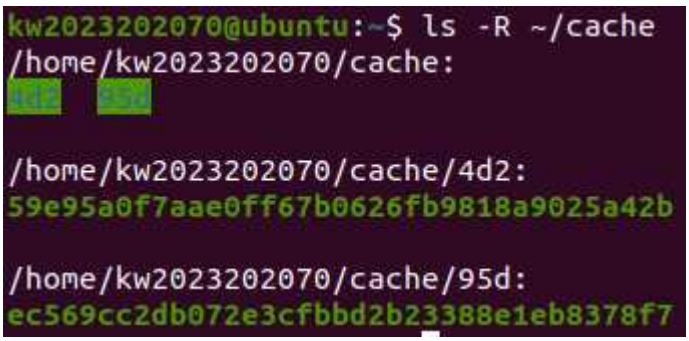
콘솔에는 프록시 서버가 받은 54310 포트번호 요청이 출력됐다. 이후 ctrl+c를 입력하여 프로그램을 종료했다.

3. Log file



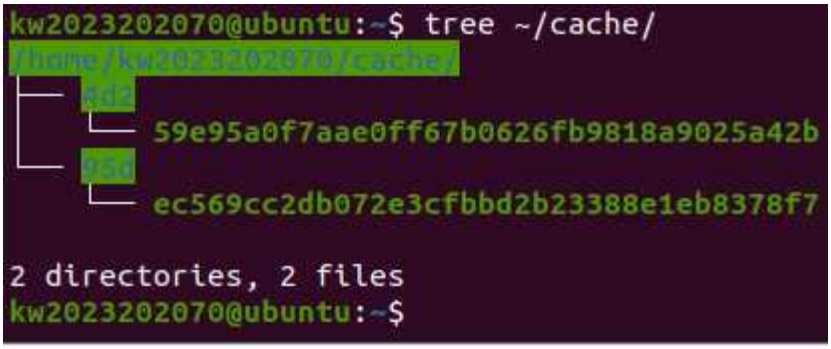
$ cat ~/logfile/logfile.txt: logfile.txt을 출력한 결과 miss 동작 처리가 되어 url과 url 입력 당시의 시간, 처리한 자식 프로세스 id가 저장되었고, hit 동작 처리가 되어 생성된 디렉토리와 파일과 시간 정보, url, 처리한 자식 프로세스 id가 저장된 것을 확인했다. 또한 각 자식 프로세스 당 하나의 브라우저 요청을 처리하기 때문에 자식 프로세스 종료 로그의 실행 시간은 0초이고, hit 또는 miss 횟수 1번이 저장된 것을 확인했다. operation 결과 사진에서 설명했듯이, 최대한 예외 처리 하여 입력한 url만 hit/miss 판단한 결과 총 hit 2번, miss 2번이 수행됐다.

4. cache directory



$ 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.

콘솔에 출력된 바와 같이 다양한 브라우저 요청이 들어오지만, 예외 처리를 통해 실제로 입력된 http://info.naver.com/과 http://info.kw.ac.kr/ 두 url에 대해서만 해싱하고 캐시한 결과를 나타낸다.

고찰

이번 Proxy 2-2 과제를 수행하면서 멀티 프로세스를 이용한 동시성 처리에 대해 실습해볼 수 있었다. 기존 Proxy 1-2 과제에서 구현했던 기능들(SHA1 해시를 이용한 캐시 구조, HIT/MISS 판별 방식 등)을 그대로 활용하면서도, 새로운 프로세스를 fork()로 생성하고 이를 waitpid()로 수거하는 구조를 직접 구현하며, 병렬 처리의 흐름과 그에 따른 자원 관리 방식까지 함께 이해할 수 있었다.

특히 이번 과제에서 먼저 헷갈린 부분은 로그 파일 처리 방식이었다. 이전 과제들의 일관성을 유지하되 강제 종료 종료 로그는 기록하지 않도록, 프로세스 실행 시간을 계산하고 HIT/MISS 수를 기록하는 로그 출력을 Proxy 2-1 방식을 그대로 따랐다. 동시성 환경에서의 파일 스트림 처리 및 버퍼링 문제에 대해 보다 명확하게 이해할 수 있었다.

또한 응답을 구성할 때 HTTP 헤더와 데이터의 구조를 올바르게 구성하는 것이 중요하다는 점도 새롭게 알게 되었다. Content-length를 명확히 지정하지 않으면 웹 브라우저에서 응답을 정상적으로 처리하지 못하기 때문에 브라우저의 다양한 요청을 그냥 무시하고 exit하면 네트워크 오류가 발생하였다. 실제 통신 시 프로토콜 형식이 얼마나 엄격하게 지켜져야 하는지 체감할 수 있었다. 이런 것은 f12의 네트워크 탭을 통해 직접 서버-클라이언트 간 응답과 요청을 확인하며 해결해 나갔다.

브라우저 요청이 예상보다 많이 들어오는 부분에서도 혼란이 있었다. 제안서에 나온 예시에서는 단일 요청만을 가정한 것처럼 보였지만, 실제 구현을 하다 보니 favicon, txt 등 다양한 요청이 함께 들어와 이들을 전부 처리해야 하는지 의문이 생겼다. 이 부분은 조교님께 묻고답하기 게시판의 질문을 통해 해결했으며, 특정 확장자 또는 메소드를 가지는 요청을 캐시 동작 처리 대상에서 제외하도록 조건을 추가하여 문제를 해결하였다.

이번 구현은 클라이언트와 프록시 서버 간의 통신만을 담당하는 점이 중요하다. 다음 Proxy 2-3 과제에서는 웹 서버까지 추가되어 프록시 서버가 클라이언트와 서버의 역할을 모두 수행해야 하는 구조로 확장될 예정이다. 따라서 현재 구현에서는 응답을 프록시 서버에서 직접 생성하여 웹 브라우저에 보내는 형식이다. 이번 과제를 통해 프록시 서버의 역할과 구조에 대한 기초를 단단히 다질 수 있었고, 다음 단계로 넘어가기 위한 준비 과정으로서도 큰 의미가 있었다.

Reference

시스템프로그래밍 이론 및 실습 자료 참고하였습니다.