수업 명 : 시스템프로그래밍

과제 이름 : Proxy 2-4

학 과: 컴퓨터정보공학부

담당 교수님: 김태석 교수님

분 반: 월5, 수6

학 번: 2023202043

성 명: 최은준

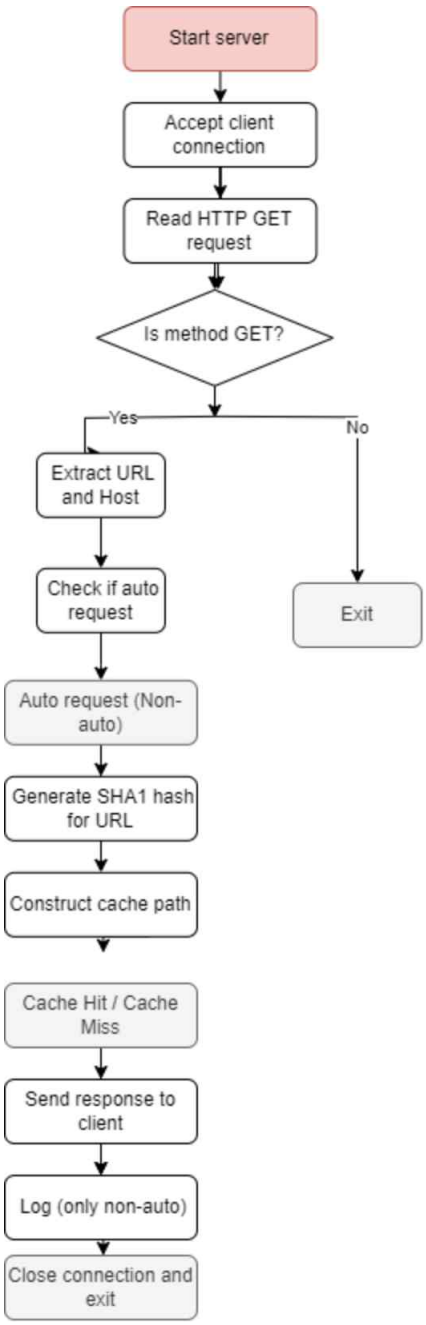
0. Introduction

이 과제에서 웹 브라우저를 클라이언트로 사용하는 프록시 서버를 구현하였다. 이때 프록시 서버는 클라이언트의 HTTP 요청을 수신하고, 해당 요청이 이전에 처리된 적이 있는지 캐시 디렉토리를 통해 판단한다. 요청한 URL이 캐시에 존재하면 HIT, 존재하지 않으면 MISS로 판단하여 그에 맞는 응답을 브라우저로 전송한다. 이 프록시 서버는 SHA1 해시 함수를 이용하여 캐시 파일명을 생성하고, 요청 URL 중 자동으로 발생하는 요청(favicon.ico, js, css 등)은 캐싱은 하되 로그에는 기록하지 않도록 처리한다.

또한, SIGALRM을 이용해 타임아웃(응답 없음)을 처리하고, SIGINT(Ctrl+C)를 통해 서버 종료 시 총 처리 요청 수와 실행 시간을 로그에 남긴다.

1. Flow Chart

1-1. proxy server #2-4 코드 작성 순서도



위 코드 순서도의 동작 방식은 다음과 같다.

1. 서버 소켓을 생성하여 포트 39999에 바인딩하고 리스닝 시작

2. 웹 브라우저로부터 HTTP GET 요청을 수신하면, URL과 Host 정보를 추출한다.

3. 요청된 URL에 대해 SHA1 해시를 생성하여 캐시 경로를 결정한다.

4. 캐시가 존재하면(HIT), 해당 파일을 읽어 클라이언트에 전송하고 로그를 기록한다.

5. 캐시가 없다면(MISS), 원 서버에 요청하여 응답을 받고 캐시에 저장한 뒤 클라이언트에 전송한다.

6. favicon.ico나 기타 자동 요청의 경우 캐시는 하지만 로그에는 기록하지 않는다.

7. SIGALRM으로 타임아웃을 처리하며, SIGINT가 발생하면 서버 종료 시 실행 시간 및 총 처리 개수를 로그에 남긴다.

2. Pseudo code

2-1. proxy client #2-4 알고리즘

```
Function main():
    Record server start time
    Register signal handlers: SIGCHLD, SIGALRM, SIGINT

    Create TCP server socket
    Set socket options for address reuse
    Bind socket to PORT 39999
    Start listening for connections

    While true:
        Accept client connection
        If accept fails:
            Continue
        Fork child process
        If child process:
```

```
            Close server socket
            Call handle_client()
        Else if parent:
            Close client socket
            Increment total process count
        Else:
            Print fork error
    Close server socket
    Return 0
End Function
```

```
Function handle_client(clnt_sock, clnt_addr):
    Set alarm for timeout (15 sec)
    Read HTTP request from client
    Cancel alarm

    Parse method and URL
    If method is not GET:
        Close socket and exit

    Extract Host header
    Check if request is auto-generated (is_auto)

    Generate SHA1 hash from URL
    Extract dir and file name from hash
    Create cache directory and subdirectory

    Build full cache file path
    If cache file exists:
        Open file and send to client
        If not auto-request:
            Write HIT log
    Else:
        Resolve hostname via gethostbyname()
        Create socket to origin server
        Connect to origin server

        Set alarm for timeout
        Format and send GET request
```

```
        Open cache file for writing

        While reading response from origin server:
            Write to client
            Write to cache file

        Close all descriptors
        Cancel alarm
        If not auto-request:
            Write MISS log

    Close client socket
    Exit process
End Function
```

```
Function sigint_handler(signo):
    Calculate runtime using current time and start_time
    Open logfile in append mode
    Write termination summary:
        - run time
        - number of sub processes
    Close logfile
    Print same summary to terminal
    Exit program
End Function
```

```
Function sigchld_handler(signo):
    While zombie children exist:
        Reap with waitpid (non-blocking)
End Function
```

```
Function sigalrm_handler(signo):
    Print timeout message to stderr
    Exit program
End Function
```

```
Function error_handling(message):
    Print error using perror
    Exit with failure status
```

End Function

Function sha1_hash(input, output):
    Perform SHA1 hash on input string
    Convert hash bytes to hexadecimal string
    Null-terminate output
    Return output
End Function

Function is_cache_hit(path):
    If file exists at path:
        Return true
    Else:
        Return false
End Function

Function write_log(status, url, dir, file, is_auto):
    If is_auto is true:
        Return (do not log)

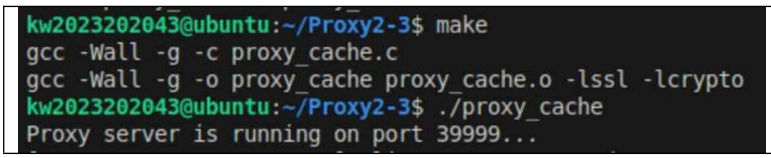
    Create log directory if not exists
    Open logfile in append mode
    Get current time

    If status is HIT:
        Write [HIT]dir/file-[time] to log
        Write [HIT]url to log
    Else:
        Write [MISS]url-[time] to log
    Close logfile
End Function

Function is_auto_request(url):
    If url contains any of:
        - favicon.ico
        - .css
        - .js
        - .png
        - .jpg

```
        - .gif
        - cdn-cgi
        - detectportal.firefox.com
        - fonts.googleapis.com
        - bootstrap
        - jquery
        - .woff, .ttf
        Return true
    Else:
        Return false
End Function
```

3. 결과화면



코드 작성 후 make 한 뒤 proxy_cache가 생성된 것을 확인할 수 있다.

06b 내 07eeaf733d9447b68feda5fd8639c9069167c

캐시 상태는 cache/<디렉토리>/<SHA1 파일>에 저장된다. MISS일 경우 응답 본문이 캐시에 저장되고, 클라이언트에 전송되고, HIT일 경우 캐시 파일의 내용을 클라이언트에 그대로 전송한다.

사용자가 접속한 URL 목록은 다음과 같다.

1. http://neverssl.com/

2. http://neverssl.com/

3. http://www.columbia.edu/~fdc/sample.html

4. http://www.catb.org/jargon/

5. http://www.catb.org/jargon/

6. http://textfiles.com

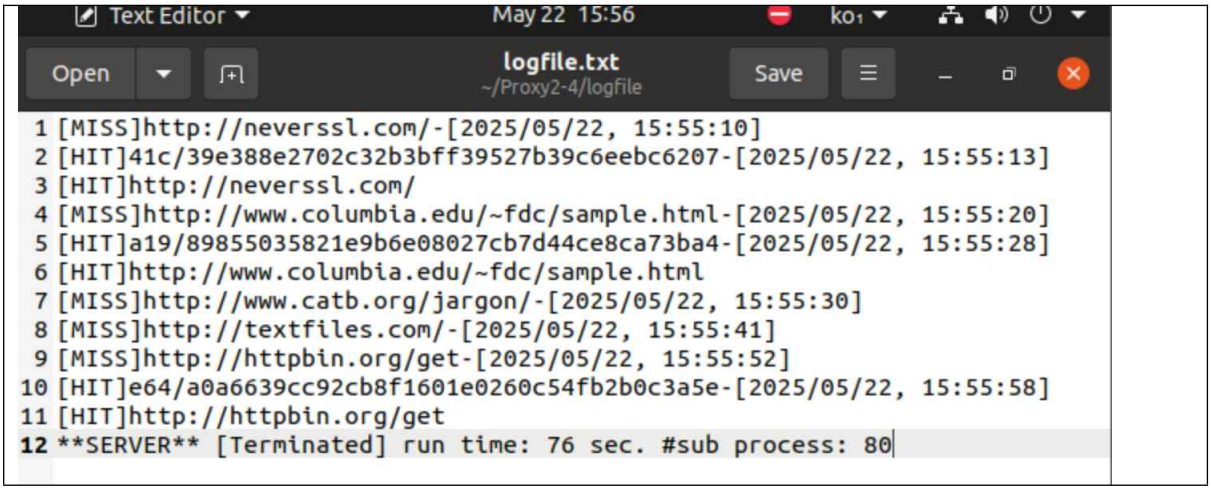
7. http://textfiles.com

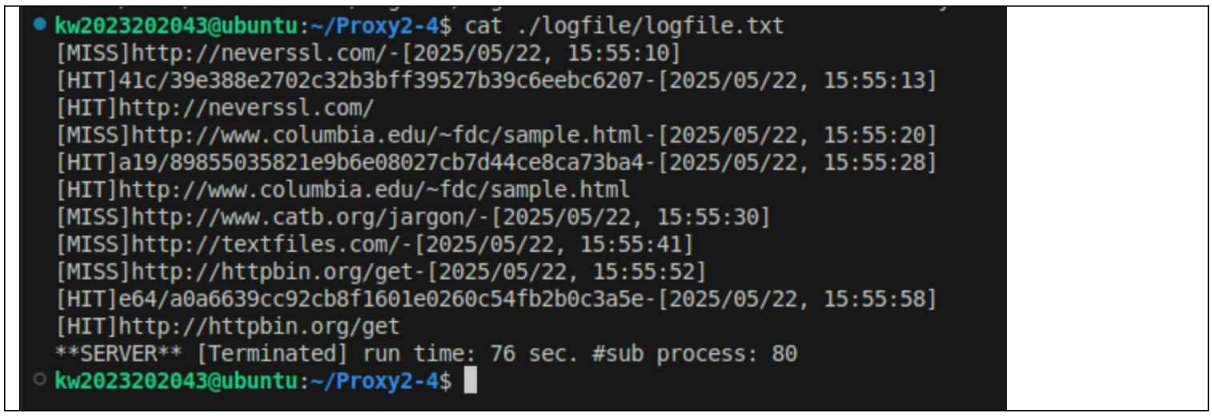
8. http://httpbin.org/get

9. http://httpbin.org/get

10. http://www.columbia.edu/~fdc/sample.html

1	http://neverssl.com/	MISS	첫 요청이므로 캐시 없음 → MISS 및 로그 기록
2	http://neverssl.com/	HIT	1번 요청 후 캐시됨 → 캐시 HIT
3	http://www.columbia.edu/~fdc/sample.html	MISS	첫 요청이므로 캐시 없음 → MISS 및 로그 기록
4	http://www.catb.org/jargon/	MISS	첫 요청이므로 캐시 없음 → MISS 및 로그 기록
5	http://www.catb.org/jargon/	HIT	4번 요청 후 캐시됨 → 캐시 HIT
6	http://textfiles.com	MISS	첫 요청이므로 캐시 없음 → MISS 및 로그 기록
7	http://textfiles.com	HIT	6번 요청 후 캐시됨 → 캐시 HIT
8	http://httpbin.org/get	MISS	첫 요청이므로 캐시 없음 → MISS 및 로그 기록
9	http://httpbin.org/get	HIT	8번 요청 후 캐시됨 → 캐시 HIT
10	http://www.columbia.edu/~fdc/sample.html	HIT	3번 요청 후 캐시됨 → 캐시 HIT





각 URL에 대해 첫 번째 요청은 [MISS]로 기록되며 그 이후 동일 URL은 [HIT] 디렉토리/파일명-[시간]형식으로 기록된다. 자동 요청(예: favicon.ico 등)은 기록되지 않으며 마지막 줄에 서버 종료 로그가 작성된다.

4. 고찰

이번 2-4과제를 통해 프로세스 포크 및 신호 처리(Signal Handling), 파일 입출력, TCP 소켓 프로그래밍에 대해 실습할 수 있었다. 특히, 자동 요청을 필터링하여 로그에서 제외하는 방식은 웹 요청의 본질과 구조를 이해하는 데 도움이 되었고, SIGINT를 통한 서버 종료 로그 기록은 서버 프로그램의 상태 추적을 이해할 수 있었다. 캐시 처리와 로깅 방식의 세분화, 시그널(SIGALRM, SIGINT 등) 처리, 멀티 프로세스 기반 요청 분기등에 대해 구현한 것이 매우 의미 있었다. SIGINT 발생 시 서버의 총 실행 시간과 처리된 클라이언트 수를 기록하는 기능을 구현하면서, 프로그램 종료 시점의 정리(clean-up) 로직 또한 중요한 것을 알 수 있었으며, 이번 과제는 단순한 소켓 프로그래밍을 넘어서, 실질적인 서버 구조와 동작의 전반적인 흐름을 다루며 시스템 전반에 대한 이해도를 높일 수 있었다.