수업 명 : 시스템프로그래밍

과제 이름 : Proxy 3-1

학 과: 컴퓨터정보공학부

담당 교수님: 김태석 교수님

분 반: 월5, 수6

학 번: 2023202043

성 명: 최은준

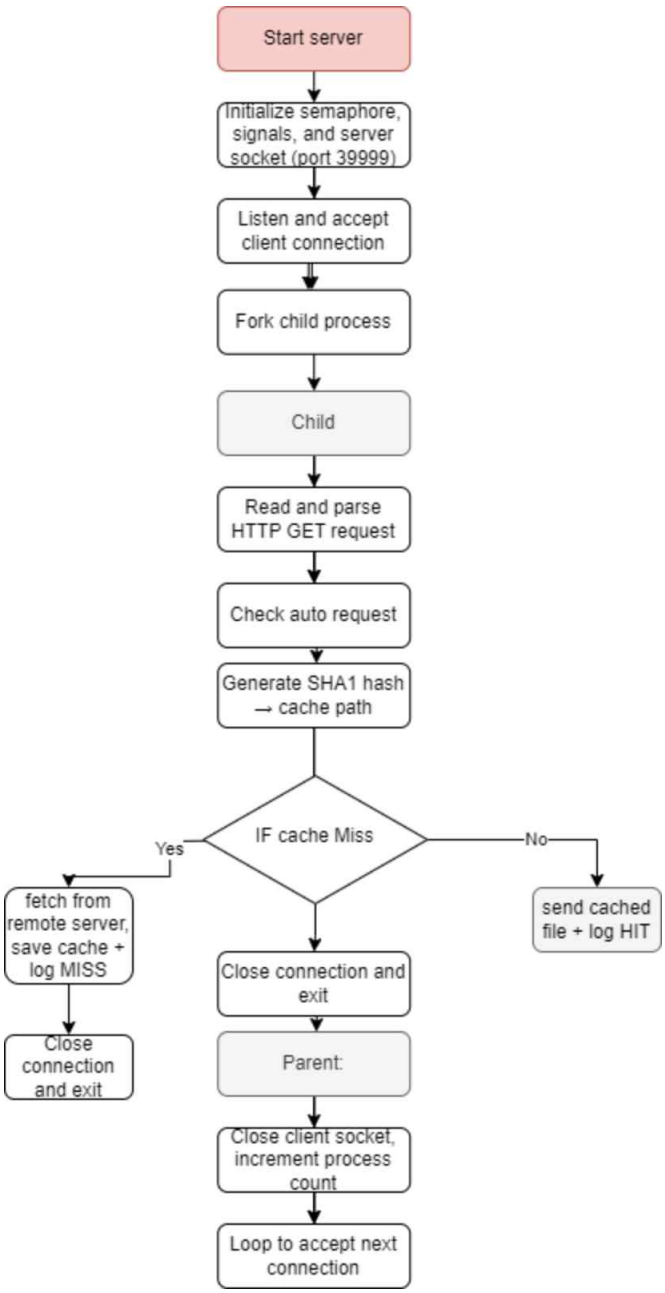
0. Introduction

이 과제는 다중 클라이언트 요청을 동시에 처리할 수 있는 동시성 프록시 서버를 구현하는 것을 목표로 한다. 서버는 HTTP GET 요청을 받아 요청 URL의 SHA1 해시값을 생성하여 캐시 파일을 고유하게 관리하며, 캐시가 존재하면 이를 클라이언트에 즉시 제공하여 응답 속도를 높이고 네트워크 부하를 줄인다. 캐시가 없거나 유효하지 않은 경우에는 원격 웹 서버에 요청을 전달하여 데이터를 받아오고 이를 캐시에 저장한다.

서버는 여러 클라이언트를 병렬 처리하며, 동시에 여러 프로세스가 로그 파일에 접근할 때 발생할 수 있는 경쟁 조건을 방지하기 위해 System V 세마포어를 사용하여 로그 기록의 동기화를 보장한다. 시그널 핸들링(SIGCHLD, SIGALRM, SIGINT)을 통해 자식 프로세스의 좀비 방지, 타임아웃 처리, 서버 종료 시 런타임 및 처리한 프로세스 수 등을 기록한다.

1. Flow Chart

1-1. proxy server #3-1 코드 작성 순서도



위 코드 순서도의 동작 방식은 다음과 같다.

1.서버 소켓을 생성하고 포트 39999에 바인딩한 후, 클라이언트 연결을 대기한다.

2. 클라이언트로부터 HTTP GET 요청이 들어오면, 요청 내용을 읽고 HTTP 메서드, URL, Host 헤더를 파싱한다.

3. 요청 URL에 대해 SHA1 해시를 계산하여, 캐시 파일의 저장 경로를 결정한다.

4.요청이 자동 생성 요청(예: favicon, CSS, JS 등)인지 판별한다.

5. 캐시 디렉터리와 파일이 존재하고 파일 크기가 0보다 크면 캐시 HIT로 판단한다.

5-1.. 캐시 HIT 시 캐시 파일을 클라이언트로 전송한다.

5-2. HIT 로그를 세마포어를 이용해 동기화하여 기록한다.

5-3. 이미 HIT 로그가 존재하면 중복 로그를 남기지 않는다.

6. 캐시가 없으면 원격 서버에 접속하여 HTTP 요청을 보내고 응답을 읽는다.

6-1. 응답을 클라이언트에 전달하는 동시에 캐시 파일로 저장한다.

6-2. MISS 로그를 기록한다.

7. 자동 요청인 경우에는 캐시는 하되 로그는 남기지 않는다.

8. SIGALRM으로 타임아웃 처리하여 일정 시간 내 응답이 없으면 프로세스를 종료한다.

9. SIGCHLD 핸들러로 자식 프로세스가 종료될 때 좀비 프로세스가 남지 않도록 한다.

10. SIGINT (Ctrl+C) 시 서버가 종료되며, 총 서버 실행 시간과 처리한 자식 프로세스 개수를 로그에 기록한다.

11. 서버는 다중 프로세스 구조로 각 클라이언트 요청 시마다 fork() 하여 병렬 처리한다.

12. 로그 파일 접근은 System V 세마포어를 이용해 임계 구역을 보호한다.

2. Pseudo code

2-1. proxy client #3-1 알고리즘

```
Function init_semaphore():
    Create System V semaphore with key = PORT
    Initialize semaphore value to 1
End Function
```

```
Function p():
    Perform semaphore wait (decrement)
End Function
```

```
Function v():
    Perform semaphore signal (increment)
End Function
```

```
Function is_auto_request(url):
    If url contains any of [favicon.ico, .css, .js, .png, .jpg, .gif, cdn-cgi, bootstrap, jquery, fonts.googleapis.com, detectportal.firefox.com, orchestrate/chl_page, .woff, .ttf]:
        Return 1
    Else:
        Return 0
End Function
```

```
Function sigalrm_handler(signo):
    Print "No Response" error
    Exit process with code 1
End Function
```

```
Function sigchld_handler(signo):
    While any child process has exited:
        Reap child to prevent zombie
End Function
```

```
Function sigint_handler(signo):
    Calculate server run time
    Open logfile in append mode
    Write "**SERVER** [Terminated]" with runtime and total processed count
    Close logfile
    Print termination message
    Exit server
End Function
```

```
Function error_handling(message):
    Print system error message and exit
```

```
End Function
```

```
Function sha1_hash(input, output):
    Compute SHA1 hash of input string
    Convert hash to hexadecimal string
    Store in output buffer
    Return output
End Function
```

```
Function is_cache_hit(path):
    If file at path exists and size > 0:
        Return 1
    Else:
        Return 0
End Function
```

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

    Print PID waiting message
    p()   // wait semaphore
    Print PID critical zone enter message
    Sleep 3 seconds (simulation)

    Create "logfile" directory if it doesn't exist
    Open logfile in append mode
    If open failed:
        Print PID exit message
        v()   // signal semaphore
        Return

    If status is HIT:
        Search logfile for existing HIT log for the same info
        If found:
            Close logfile
            Print PID exit message
            v()   // signal semaphore
            Return

    Get current timestamp
```

```
    If status is HIT:
        Write "[HIT]dir/file-[timestamp]" to logfile
        Write "[HIT]info" to logfile
    Else:
        Write "[MISS]info-[timestamp]" to logfile

    Close logfile
    Print PID exit message
    v()   // signal semaphore
End Function
```

```
Function handle_client(clnt_sock, clnt_addr):
    Set alarm 15 seconds
    Read HTTP request from clnt_sock into buf
    Cancel alarm
    If read failed or zero bytes:
        Close clnt_sock and exit

    Parse HTTP method and URL from buf
    If method is not GET:
        Close clnt_sock and exit

    Parse Host header from buf
    If no Host:
        Close clnt_sock and exit

    Determine if request is auto via is_auto_request(url)

    Compute SHA1 hash of URL
    Extract first 3 chars as dir_name
    Create cache directories CACHE_DIR and CACHE_DIR/dir_name

    Compose full cache file path CACHE_DIR/dir_name/hash[3..end]

    If cache hit (file exists and size > 0) and file is not temporary (.tmp):
        Open cache file and send contents to client socket
        Call write_log("HIT", url, dir_name, hash+3, is_auto)
    Else:
        Resolve host via gethostbyname
        If fail, close clnt_sock and exit
```

```
        Create socket to remote server (port 80)
        Connect to remote server
        Set alarm 15 seconds

        Send HTTP GET request to server with Host header
        Open cache file for write (truncate)
        While reading from remote server:
            Write data to clnt_sock (client)
            Write data to cache file
        Close cache file and server socket
        Cancel alarm

        Call write_log("MISS", url, dir_name, hash+3, is_auto)

    Close clnt_sock
    Exit child process
End Function
```

```
Function main():
    init_semaphore()
    Record start_time
    Setup signal handlers for SIGCHLD, SIGALRM, SIGINT

    Create server socket
    Bind to port 39999 on all interfaces
    Listen on socket

    While true:
        Accept client connection
        Fork child process
        If child:
            Close server socket
            handle_client(clnt_sock, clnt_addr)
        Else if parent:
            Close client socket
            Increment total_proc
        Else:
            Print fork error
```

```
    Close server socket
    Return 0
End Function
```

3. 결과화면

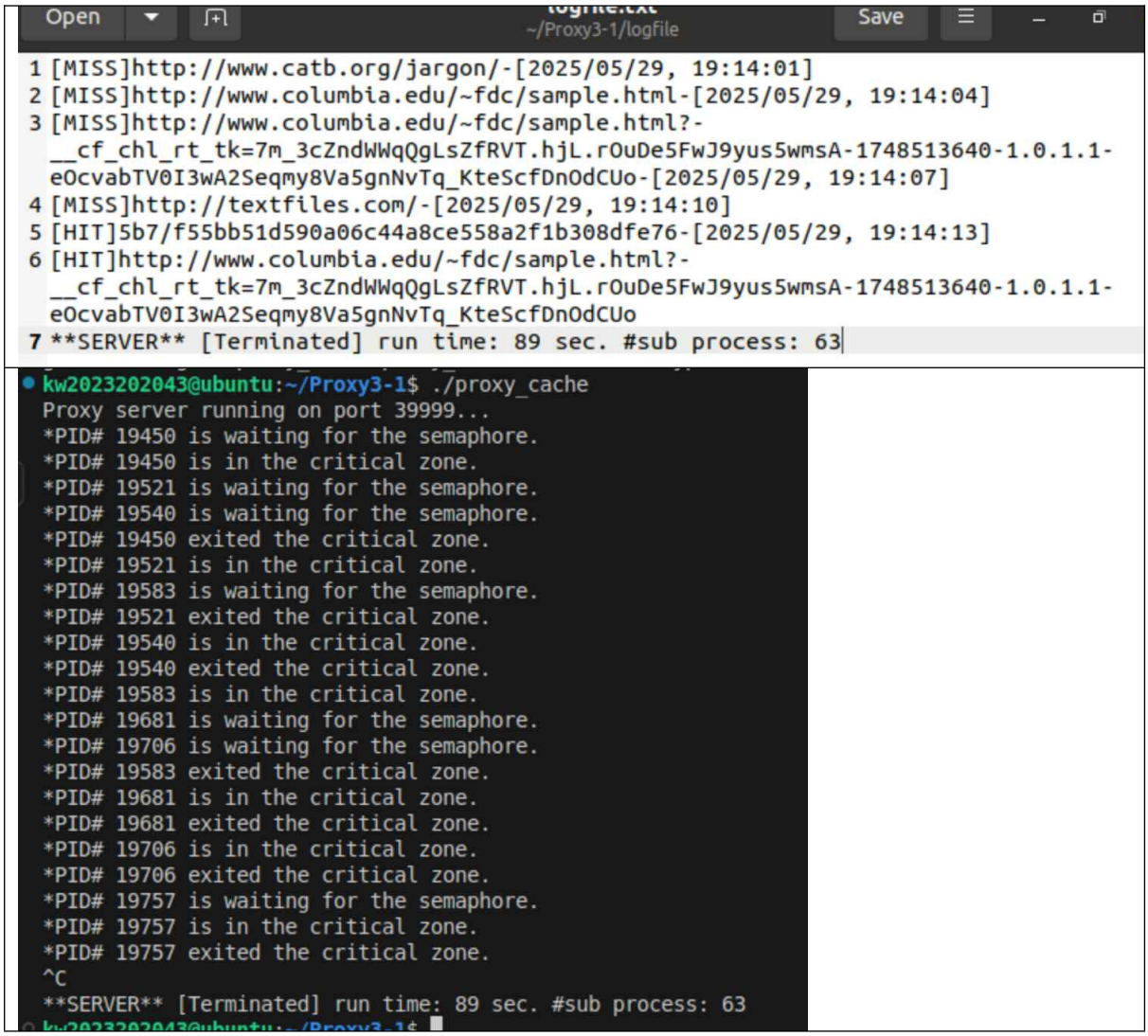
```
kw2023202043@ubuntu:~/Proxy2-3$ make
gcc -Wall -g -c proxy_cache.c
gcc -Wall -g -o proxy_cache proxy_cache.o -lssl -lcrypto
kw2023202043@ubuntu:~/Proxy2-3$ ./proxy_cache
Proxy server is running on port 39999...
```

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

Open | logfile.txt ~/Proxy3-1/logfile | Save

```
1 [MISS]http://www.catb.org/jargon/-[2025/05/28, 16:49:31]
2 [MISS]http://www.columbia.edu/~fdc/sample.html-[2025/05/28, 16:49:42]
3 [HIT]44a/9a68c0ca86342fead55c8dd129f0f183fa96a-[2025/05/28, 16:49:50]
4 [HIT]http://www.catb.org/jargon/
5 [HIT]a19/89855035821e9b6e08027cb7d44ce8ca73ba4-[2025/05/28, 16:49:58]
6 [HIT]http://www.columbia.edu/~fdc/sample.html
7 [MISS]http://textfiles.com/-[2025/05/28, 16:50:36]
8 **SERVER** [Terminated] run time: 100 sec. #sub process: 76
```

```
kw2023202043@ubuntu:~/Proxy3-1$ ./proxy_cache
*PID# 10767 is waiting for the semaphore.
*PID# 10767 is in the critical zone.
*PID# 10767 exited the critical zone.
*PID# 10907 is waiting for the semaphore.
*PID# 10907 is in the critical zone.
*PID# 10907 exited the critical zone.
*PID# 10969 is waiting for the semaphore.
*PID# 10969 is in the critical zone.
*PID# 10969 exited the critical zone.
*PID# 11096 is waiting for the semaphore.
*PID# 11096 is in the critical zone.
*PID# 11096 exited the critical zone.
*PID# 11224 is waiting for the semaphore.
*PID# 11224 is in the critical zone.
*PID# 11224 exited the critical zone.
*PID# 11285 is waiting for the semaphore.
*PID# 11285 is in the critical zone.
*PID# 11285 exited the critical zone.
*PID# 11331 is waiting for the semaphore.
*PID# 11331 is in the critical zone.
*PID# 11340 is waiting for the semaphore.
*PID# 11331 exited the critical zone.
*PID# 11340 is in the critical zone.
*PID# 11340 exited the critical zone.
^C
**SERVER** [Terminated] run time: 100 sec. #sub process: 76
kw2023202043@ubuntu:~/Proxy3-1$
```



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. 타임아웃 발생 시 (SIGALRM), ========== No Response ==========가 출력된다.

클라이언트가 접속하면, 서버는 새로운 프로세스를 fork해서 요청을 병렬 처리한다. 메인 프로세스는 계속해서 다른 연결을 받아들일 수 있으며 자식 프로세스는 클라이언트로부터 HTTP GET 요청을 받아 URL과 Host를 파싱한다. 요청이 자동생성된 리소스(예: favicon, CSS 등)인지 검사하여, 자동 요청이라면 로그 기록을 하지 않고 빠르게 처리한다.

로그 파일 접근은 System V 세마포어로 동기화하여 여러 프로세스가 동시에 로그를 수정하는 충돌을 방지한다. 로그는 HIT/MISS 상태와 요청 URL, 타임스탬프를 기록하며, HIT는 중복 기록을 막아 불필요한 로그 중복을 줄인다.

4. 고찰

이번 3-1과제를 통해 동시성을 갖는 서버 프로그램에서 자원 공유 문제와 프로세스 관리를 어떻게 해야 하는지 깊이 이해할 수 있었다. 특히, 여러 프로세스가 동일한 로그 파일에 동시에 접근할 때 발생할 수 있는 데이터 손상이나 로그 충돌 문제를 세마포어로 해결하는 부분에 대해 학습함으로써 운영체제의 프로세스 간 동기화 기법에 대해 알 수 있었다.

또한, 자동 요청과 사용자 요청을 구분하여 로그를 필터링하는 작업으로 불필요한 로그 생성을 줄이고 성능을 최적화하는 것을 알게 되었다.

시그널 처리 부분에서 타임아웃과 자식 프로세스 정리, 그리고 종료 시 서버 상태 로그 기록 등을 구현하며 안정적인 서버 운영에 필요한 요소들을 경험할 수 있었다. 전반적으로 네트워크 소켓 프로그래밍, 프로세스 관리, 그리고 동기화 메커니즘을 통합하여 완성도 높은 시스템 프로그래밍 기술을 익히는 좋은 기회였다.