# 수업 명 : 시스템프로그래밍

# 과제 이름 : Proxy 3-2


학 과: 컴퓨터정보공학부

담당 교수님: 김태석 교수님

분 반: 월5, 수6

학 번: 2023202043

성 명: 최은준

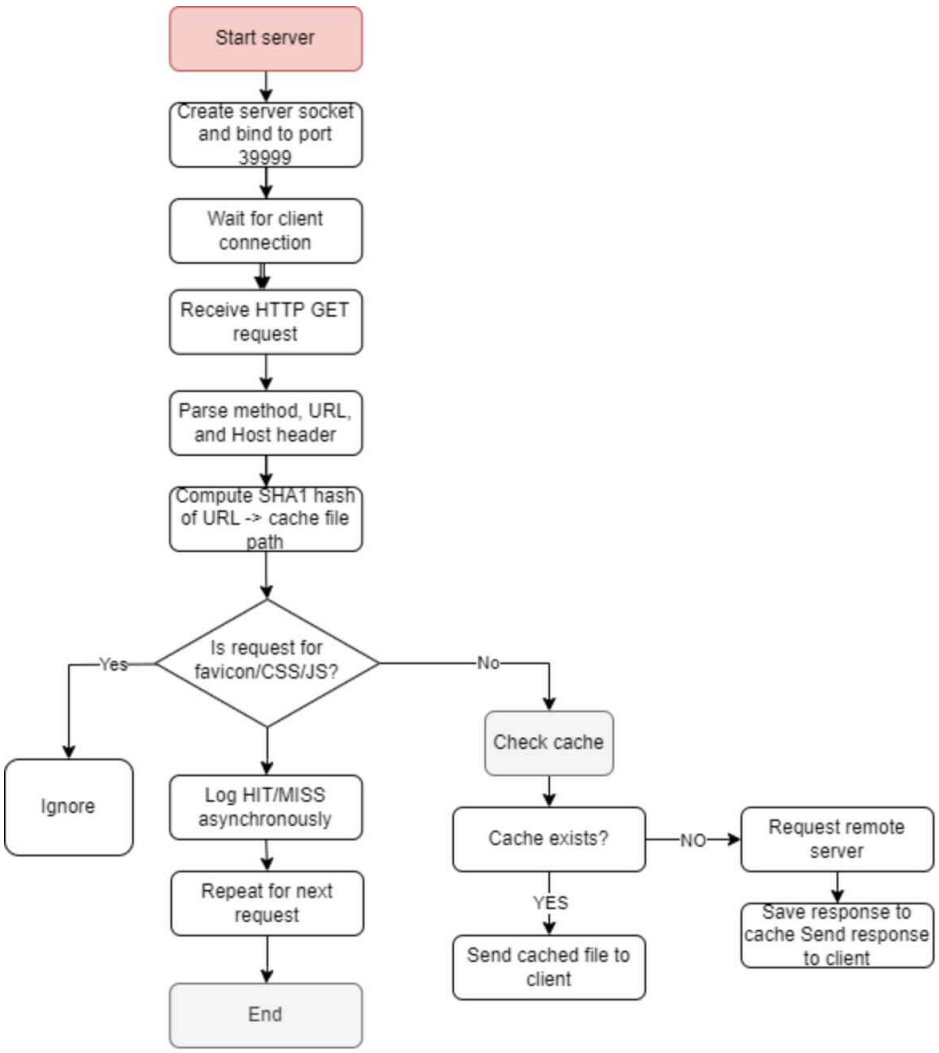
# 0. Introduction

이 과제에서는 다중 프로세스 기반의 동시성 프록시 서버를 구현한다. 클라이언트의 HTTP GET 요청을 처리하며, 요청된 웹 페이지를 SHA1 해시를 이용하여 캐시 디렉터리에 저장하고, 캐시된 파일이 존재하면 이를 클라이언트에 제공함으로써 네트워크 트래픽과 응답 시간을 줄이는 기능을 포함한다. 캐시 여부는 파일의 존재 및 크기를 통해 판단하며, 캐시된 파일이 완전하지 않은 경우에는 원격 서버로부터 새로 데이터를 받아 저장한다.

또한, 여러 프로세스가 동시에 로그 파일에 접근할 때 발생할 수 있는 상태를 방지하기 위해 세마포어를 사용하여 로그 기록에 대한 동기화 처리를 구현한다. 서버는 SIGINT 시그널을 받아 종료될 때 서버의 총 실행 시간과 처리한 자식 프로세스 수를 로그에 남기며, SIGALRM을 이용해 일정 시간 이상 응답이 없으면 자식 프로세스를 종료하여 데드락이나 좀비 프로세스 생성을 방지하는 등 안정적인 동작 환경을 구축한다.

# 1. Flow Chart

## 1-1. proxy server #3-2 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 요청을 수신하면, 요청 메시지를 읽고 HTTP 메서드, URL, Host 헤더를 파싱한다.

3. 요청 URL에 대해 SHA1 해시를 계산하여 캐시 파일 경로를 생성한다.

4. 요청이 favicon, CSS, JS 등 자동 생성 요청인지 판단하여 처리 여부를 결정한다.

5. 캐시된 파일이 있으면 해당 파일을 클라이언트에게 전달하며 HIT로 기록한다.

6. 캐시가 없으면 원격 서버에 요청을 보내 응답을 받아 캐시에 저장 후 클라이언트에게 전달하며 MISS로 기록한다.

7. 각 요청 및 응답 로그를 별도 쓰레드에서 파일에 기록하고, 쓰레드 종료 시 로그를 출력한다.

8. 서버는 여러 클라이언트 요청을 반복적으로 처리하며, 종료 시 적절히 자원을 정리한다.

# 2. Pseudo code

## 2-1. proxy client #3-2 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, .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):
    end_time = current time
    runtime = end_time - server_start_time
    open logfile in append mode
    write "**SERVER** [Terminated] run time and total subprocess count" to logfile
    close logfile
    print the same message to terminal
    send SIGTERM to all child processes
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_request is true:
    Return
  Print "*PID# <pid> is waiting for semaphore ..."
  Call p()to wait on semaphore
  Print "*PID# <pid> entered critical section."
  If "logfile" directory does not exist:
    Create "logfile" directory with mode 0700
  Open "logfile/proxy_cache.log" for appending
  If open fails:
    Print error message
    Call v()to release semaphore
    Return
  If status is "HIT":
    Open same logfile for reading
    If open fails:
        Print error message
        Close write log file
        Call v()
        Return
    For each line in the log file:
        If line contains "[HIT]" and the same url_info:
            Set is_duplicate to true
            Break loop
    Close read log file
    If is_duplicate is true:
        Close write log file
        Call v()
        Return
  Get current time string formatted as "[yyyy-MM-dd HH:mm:ss]"
  Write formatted log line:
    "[timestamp] [status] [URL] [dir_name/file_name]"
  Close write log file
  Call v()
  Print "*PID# <pid> exited critical section."
End Function
Function log_thread_func(arg):
  Detach this thread using pthread_detach
```

```
    Print "*TID# <tid> is created."
    Cast argto log_args pointer
    If casting fails or fields are invalid:
        Print error and return NULL
    Call write_log()with arguments from log_args
    Free memory allocated for log_args
    Print "*TID# <tid> is exited."
    Return NULL
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
```

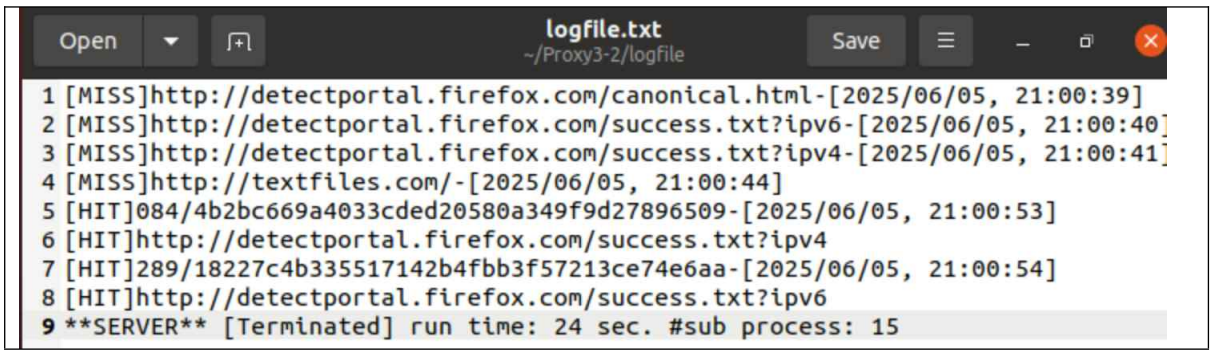
```
Function main():
    init_semaphore()
    Record start_time
    Setup signal handlers for SIGCHLD, SIGALRM, SIGINT

    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
            Print "*PID# ... is created."
            handle_client(clnt_sock, clnt_addr)
        Else if parent:
            Close client socket
            Increment total_proc
        Else:
            Print fork error
```

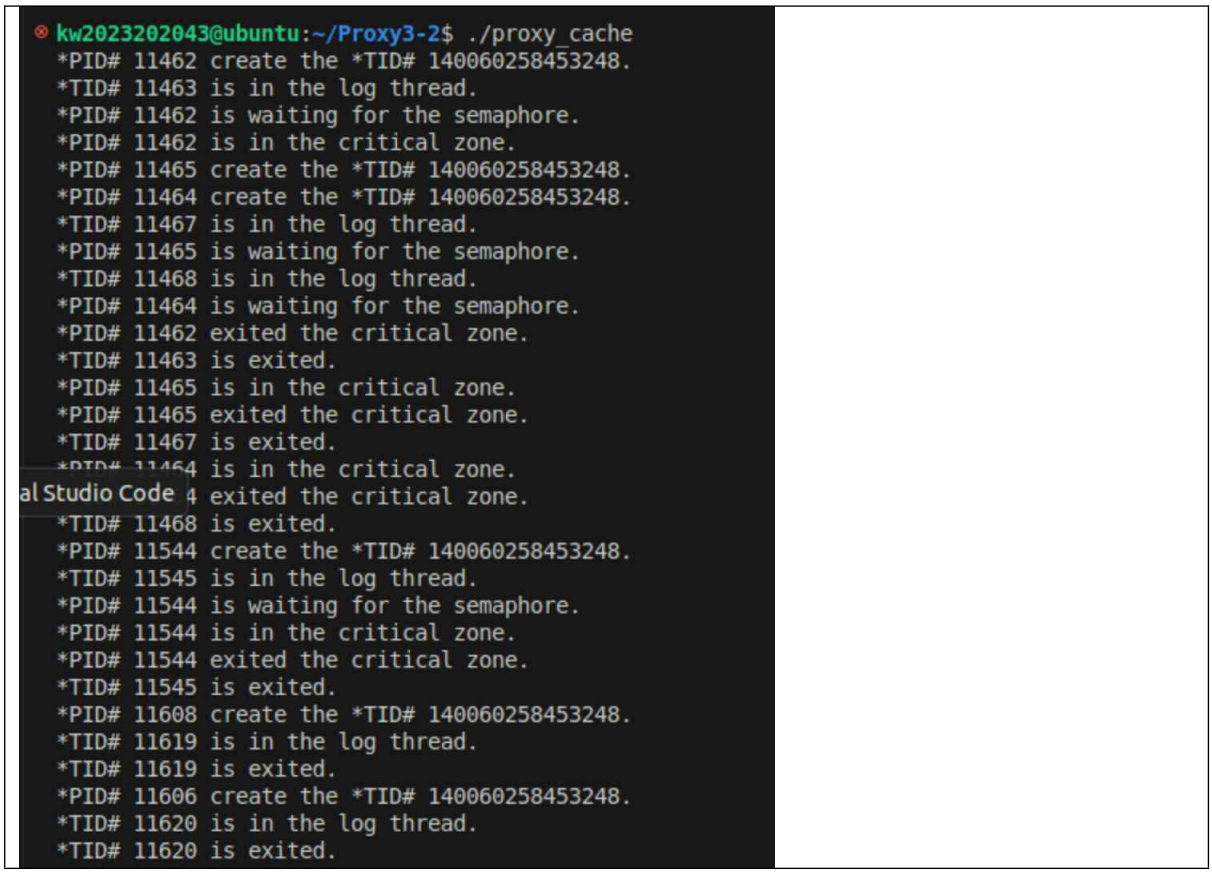
```
    Close server socket
    Return 0
End Function
```

# 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.






각 URL에 대한 첫 번째 요청은 캐시에 존재하지 않기 때문에 [MISS]로 로그에 기록된다. 동일한 URL에 대해 두 번째 이후 요청은 캐시된 결과를 사용하므로 [HIT]으로 처리되며, 로그에 기록된다. 자동 요청(ex: favicon.ico, .css, .js등)은 자동 요청 필터링을 통해 탐지되며, 로그에 기록되지 않는다. 서버 종료 시, 로그 파일에는 종료 메시지가 기록된다. 클라이언트가 연결 후 일정 시간 응답하지 않으면 SIGALRM 시그널에 의해 타임아웃 처리된다.

클라이언트가 연결을 시도하면, 메인 프로세스는 fork()를 호출하여 자식 프로세스를 생성한다. 자식 프로세스는 클라이언트와 통신을 담당하며,

클라이언트의 HTTP GET 요청수신

요청에서 URL과 Host 정보 파싱

요청이 자동 요청인지 검사하고, 해당되면 빠르게 처리하며 로그를 남기지 않음

자동 요청이 아니라면 캐시 확인 → HIT 또는 MISS 판단

필요한 경우 서버에 요청하여 데이터를 받아 캐시에 저장

write_log()를 통해 요청 결과를 로그에 기록을 수행한다.

메인 프로세스는 클라이언트 소켓을 닫고 다음 요청을 계속 받아들일 준비를 한다. 이 구조 덕분

에 다중 클라이언트 요청을 병렬로 처리할 수 있다.

로그 파일 접근은 System V 세마포어를 이용한 동기화 방식으로 구현되어 있고,이는 다수의 자식 프로세스가 동시에 로그 파일을 접근할 때 충돌을 방지하기 위한다. P()와 V()연산으로 임계 구역(critical section)을 보호하여 HIT/MISS 기록 중 중복 쓰기, 깨진 로그 등의 문제를 방지하며 [HIT] 상태인 경우에도 이미 기록된 URL이라면 중복 기록을 피하여 불필요한 로그 누적을 줄인다.

# 4. 고찰

이번 3-2과제를 통해 동시성 프로그래밍과 프로세스 간 자원 공유 및 동기화의 중요성을 이해할 수 있었다. 여러 프로세스가 동시에 파일에 접근할 때 발생하는 경합 문제를 세마포어로 해결하는 방법을 알게 되었다. 로그 파일과 같이 공유 자원에 대한 일관성 유지가 필수적인 경우 세마포어의 필요성과 활용법을 직접 구현해보면서 익힐 수 있었다.

SHA1 해시를 이용한 캐시 파일명 생성은 URL을 고유하게 식별하고 관리하는 데 효과적이며 임시 파일과 완전한 캐시 파일을 구분하여 HIT 여부를 판단하는 로직은 캐시의 신뢰성을 높이는 중요한 요소임을 알게 되었다.

이번 과제는 네트워크 프로그래밍, 프로세스 관리, 동기화 기법을 종합적으로 활용하는 좋은 경험이 되었으며, 시스템 프로그래밍 전반에 대한 이해를 높이는 데 크게 기여하였다.