시스템프로그래밍 과제 보고서

Proxy 2-3

수업 명: 시스템프로그래밍 월5수6

담당 교수: 김태석 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

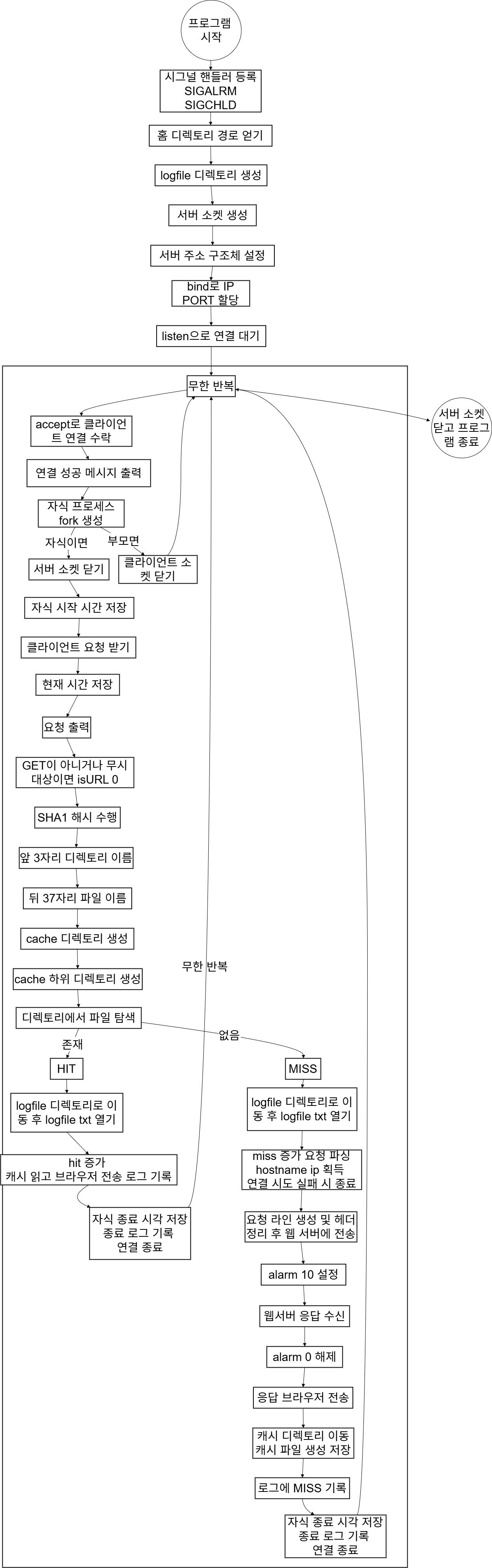
이름: 최현진

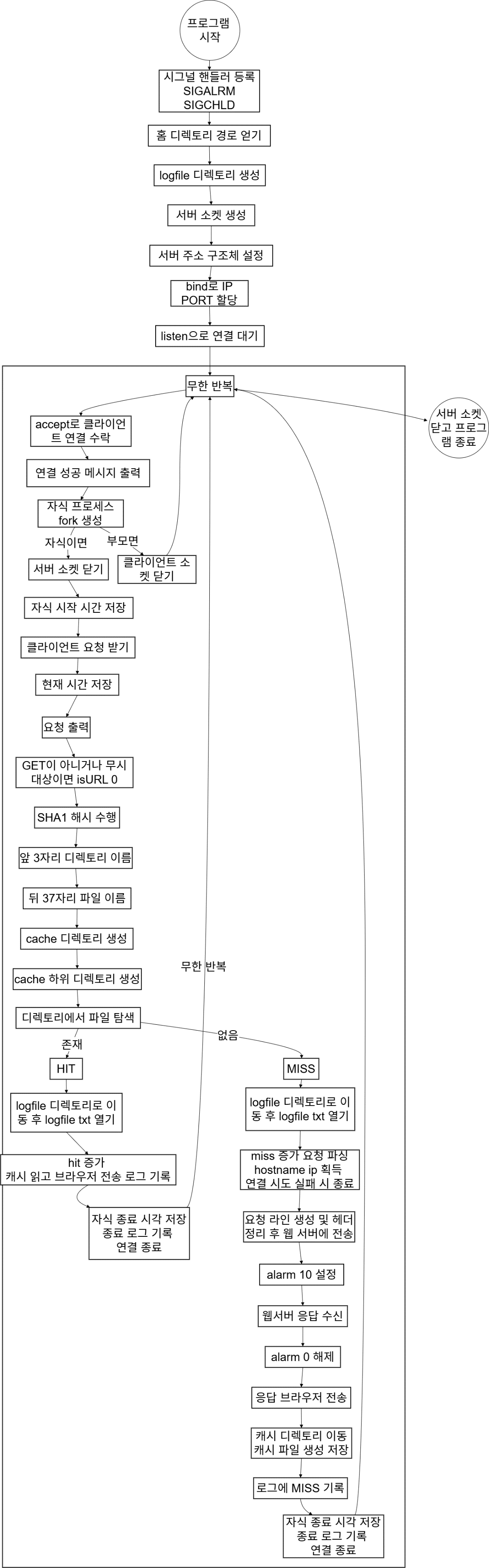
제출일: 2025.05.15

Introduction

이번 Proxy 2-3 과제는 이전 Proxy 2-2 과제에서 구현한 기능을 기반으로, 오리진 서버(웹 서버)와의 통신 기능을 포함한 프록시 서버를 구현하는 실습 과제이다. 프록시 서버는 웹 브라우저로부터 HTTP 요청을 수신하고, 해당 요청을 기반으로 HIT 또는 MISS를 판별한다. MISS인 경우, 오리진 서버에 직접 HTTP 요청을 전송하고, 응답을 수신하여 브라우저에 전달한 뒤 캐시에 저장한다. 응답이 10초 이내에 수신되지 않을 경우, SIGALRM 시그널을 통해 응답 없음 메시지를 출력하고 자식 프로세스를 종료한다. 자식 프로세스의 종료 시에는 SIGCHLD 시그널을 통해 좀비 프로세스를 정리한다. 모든 요청 결과(HIT/MISS)와 자식 프로세스의 종료 시점 정보를 로그 파일에 기록한다. 본 과제를 통해 시스템 프로그래밍의 주요 주제인 멀티프로세스, 네트워크 통신, 시그널 제어, 파일 입출력에 대해 통합적으로 학습할 수 있다.

Flow Chart





Pseudo code

|  |
| --- |
| main: |
| 프로그램 시작  시그널 핸들러 등록(SIGALRM, SIGCHLD)  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 출력  method가 GET이 아니거나 URL이 없거나 ico/css/txt 파일이면 isURL = 0  SHA1 해시 수행  앞 3자리 → 캐시 디렉토리 이름  뒤 37자리 → 캐시 파일 이름  ~/cache 디렉토리 생성  ~/cache/xxx 디렉토리 생성  isURL일 경우:  디렉토리에서 파일 탐색  존재 → HIT  없으면 → MISS  ~/logfile 디렉토리로 이동  logfile.txt 열기  HIT이면:  hit++  캐시 디렉토리로 이동 후 캐시 파일 열기  파일 읽어 브라우저로 전송  로그에 HIT 기록  MISS이면:  miss++  요청에서 hostname, port, path 추출  hostname으로 오리진 서버 IP 얻기  소켓 생성 및 오리진 서버에 연결 시도  실패 시 종료  HTTP/1.0 요청 라인 생성  브라우저 요청 헤더들 정제하여 재조합  오리진 서버에 요청 전송  alarm(10) 설정  오리진 서버로부터 응답 수신  alarm(0)으로 해제  응답을 브라우저로 전송  ~/cache/xxx 디렉토리로 이동 후 캐시 파일 생성  응답 내용을 캐시 파일에 저장  로그에 MISS 기록  자식 종료 시간 저장  종료 로그 기록  연결 종료 메시지 출력  클라이언트 소켓 닫기  자식 종료  부모면:  클라이언트 소켓 닫기  서버 소켓 닫고 프로그램 종료 |

결과 화면

1. proxy 2-2

|  |
| --- |
| 1. make |
| $ 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. no response 동작 확인    <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> 입력후 바로 가상머신의 입력을 끊었다.    터미널 출력 확인 결과 child process는 browser request의 url을 확인하고 MISS임을 판단했다. 이후 web server로부터 response를 받기 위해 origin server와 연결했다. 연결 후 response를 read()를 통해 받아오는 동안, 인터넷 연결이 끊겼기에 no response가 출력되고 child process가 종료됐다.    MISS이지만 response를 10초 동안 받지 못하여 child process가 종료됐고 생성할 캐시 파일 data가 없으므로, log는 기록되지 않았다. |

|  |
| --- |
| 3. MISS, HIT 동작 확인  <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> 을 브라우저 상에 입력했다.      58580 client가 요청을 보내오고 child process는 MISS임을 판단했다. 이후 58580 client가 origin web server로 연결하고 read()를 통해 받은 응답 버퍼를 cmd에 출력해봤다. (너무 길어서 중간에 잘랐다.) 해당 응답은 캐시 파일에 저장된다. 이후 web server와의 연결이 끊기고 child process가 종료됐다.    중간에 위와 같은 favicon.ico 요청이 많이 들어오지만, HIT/MISS 판단에서 제외된다. 따라서 log나 캐시되지 않고, client 연결과 child process가 종료된다.    <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> 을 브라우저 상에 재입력했다.  55158 client가 요청을 보내오고 child process는 HIT임을 판단했다. 이후 child process는 캐시 파일에 저장되어 있는 response를 client(browser)로 보낸다. client 연결과 child process가 종료된다.    MISS일 때 browser에 web server가 보내온 응답과 같은 화면이 HIT일 때도 잘 출력된다.    <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> url에 대한 한 child process에서 MISS 1번, 또다른 child process에서 HIT 1번일 때의 로그 파일 출력 결과를 확인했다. |

|  |
| --- |
| 4. cache directory    $ ls -R ~/cache: ls 명령어를 –R 옵션을 통해 재귀적으로 실행하여 ~/cache 디렉토리와 그 하위 디렉토리까지 출력하여 캐시 디렉토리와 파일이 생성됨을 확인했다.    $ tree ~/cache/: ~/cache 구조 확인 결과, SHA1 해시된 url의 앞 3글자를 이름으로 하여 디렉토리가 생성되었다. 그 디렉토리의 하위에는 나머지 37글자 이름으로 파일이 생성되었다.  cmd에 출력된 바와 같이 다양한 브라우저 요청이 들어오지만, 예외 처리를 통해 실제로 입력된 <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html> url에 대해서만 해싱하고 캐시한 결과를 나타낸다.      캐시 파일을 열어 MISS일 때 web server로부터의 response가 저장된 것을 확인했다. |

고찰

이번 Proxy 2-3 과제를 수행하면서, 시그널 처리와 HTTP 요청 구조에 대한 실질적인 이해를 얻을 수 있었다. 기존 Proxy 2-2 과제까지는 프록시서버와 클라이언트와의 통신에 집중했다면, 이번 구현에서는 프록시 서버가 오리진 웹 서버와도 직접 통신해야 하므로 클라이언트와 서버의 역할을 모두 수행해야 했다.

우선, 이번 과제에서 새롭게 적용한 기능 중 하나는 SIGALRM과 SIGCHLD 시그널 핸들러였다. 이론 시간에 배운 내용을 실제로 코드에 적용해보며, 비동기 이벤트 기반의 흐름 제어에 대해 더 구체적으로 체감할 수 있었다. 특히 오리진 서버로 요청을 보낸 뒤 일정 시간 안에 응답이 도착하지 않으면 자식 프로세스를 종료시키는 구조를 alarm()과 SIGALRM 시그널로 구현하면서, 단순한 프로세스 포크를 넘어서 시그널 기반의 비동기 제어 방식을 익힐 수 있었다.

또한, 응답 처리 과정에서 read()와 write()에 사용하는 버퍼의 크기가 충분하지 않으면 브라우저 화면에서 응답이 깨지거나, 콘솔 출력에 쓰레기 값이 포함되는 문제가 발생했다. 처음에는 응답이 잘못 왔다고 생각했지만, 응답 자체는 정상이었고 내가 설정한 버퍼가 Content-Length보다 작았기 때문에 발생한 문제였다. 이를 해결하기 위해 기존에 사용하던 BUF\_SIZE \* 4에서 BUF\_SIZE \* 10 수준으로 버퍼 크기를 확장했고, 이로 인해 응답 데이터가 잘리는 문제를 해결할 수 있었다. 실제 응답 길이를 정확히 계산하고 그에 맞는 크기를 확보하는 것으로 발전하고 싶다.

가장 어려웠던 부분은 오리진 서버에 HTTP 요청을 보낼 때의 요청 구성 방식이었다. 처음에는 브라우저로부터 받은 요청을 그대로 오리진 서버에 전송했지만, 응답이 오지 않거나 408 Timeout이 나는 현상이 반복되었다. 구글링을 통해 다양한 자료를 찾아보면서 HTTP/1.0 형식으로 요청 라인을 다시 구성해야 한다는 것을 알게 되었고, 그에 따라 요청 라인을 GET /path HTTP/1.0 형식으로 수정하고 필수적인 헤더를 포함시키는 방식으로 코드를 변경했다. 단순한 요청 전달이 아니라, 서버가 인식할 수 있는 형식으로 정확히 재조립하는 작업이 필요하다는 점에서, HTTP 프로토콜의 다시 한 번 학습할 수 있었다.

Reference

<https://kimjingyu.tistory.com/entry/proxy-%EC%84%9C%EB%B2%84-%EB%A7%8C%EB%93%A4%EA%B8%B0> – proxy server 만들기

시스템프로그래밍 이론 및 실습 자료 참고하였습니다.