시스템프로그래밍
Proxy #3-2

담당교수 : 최상호 교수님(목4)

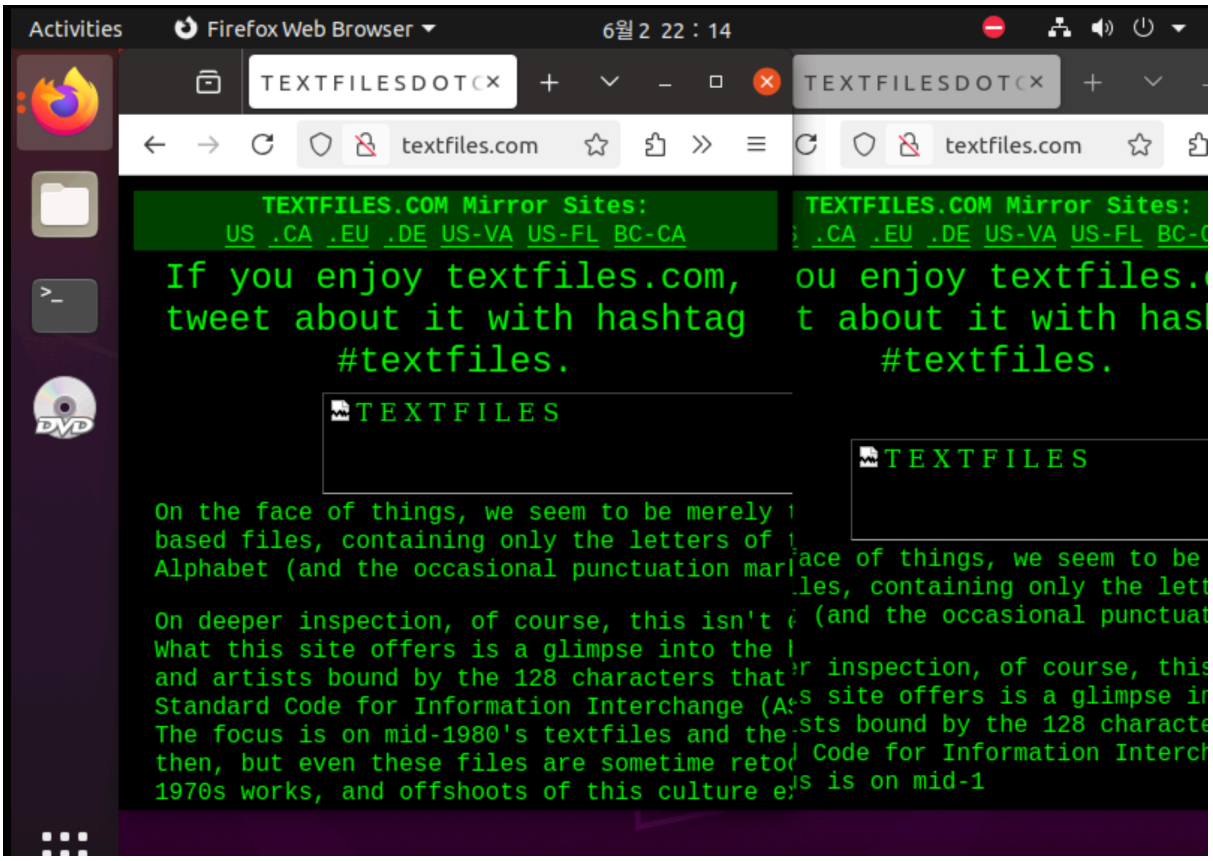
2021202003 강준우

Proxy #3-2

Introduction

이번 프로젝트는 기존의 프로잭트에서 연장하여 다중 프로세스 환경에서의 임계 구역 접근 시 로그를 기록하는 스레드를 추가하였습니다.
기존 3-1 과제의 3단계 기능은 유지하되, 프로세스가 생성한 스레드의 에서 크리티컬 섹션에서 로그 내용을 추가 기록하는 모습을 구현하였습니다.
또한 어떤 자식 프로세스가 해당 기능을 수행하고 있는지 확인할 수 있도록 합니다.

결과화면



해당 2개의 파이어폭스 창에서 같은 페이지에 접근하려는 모습입니다

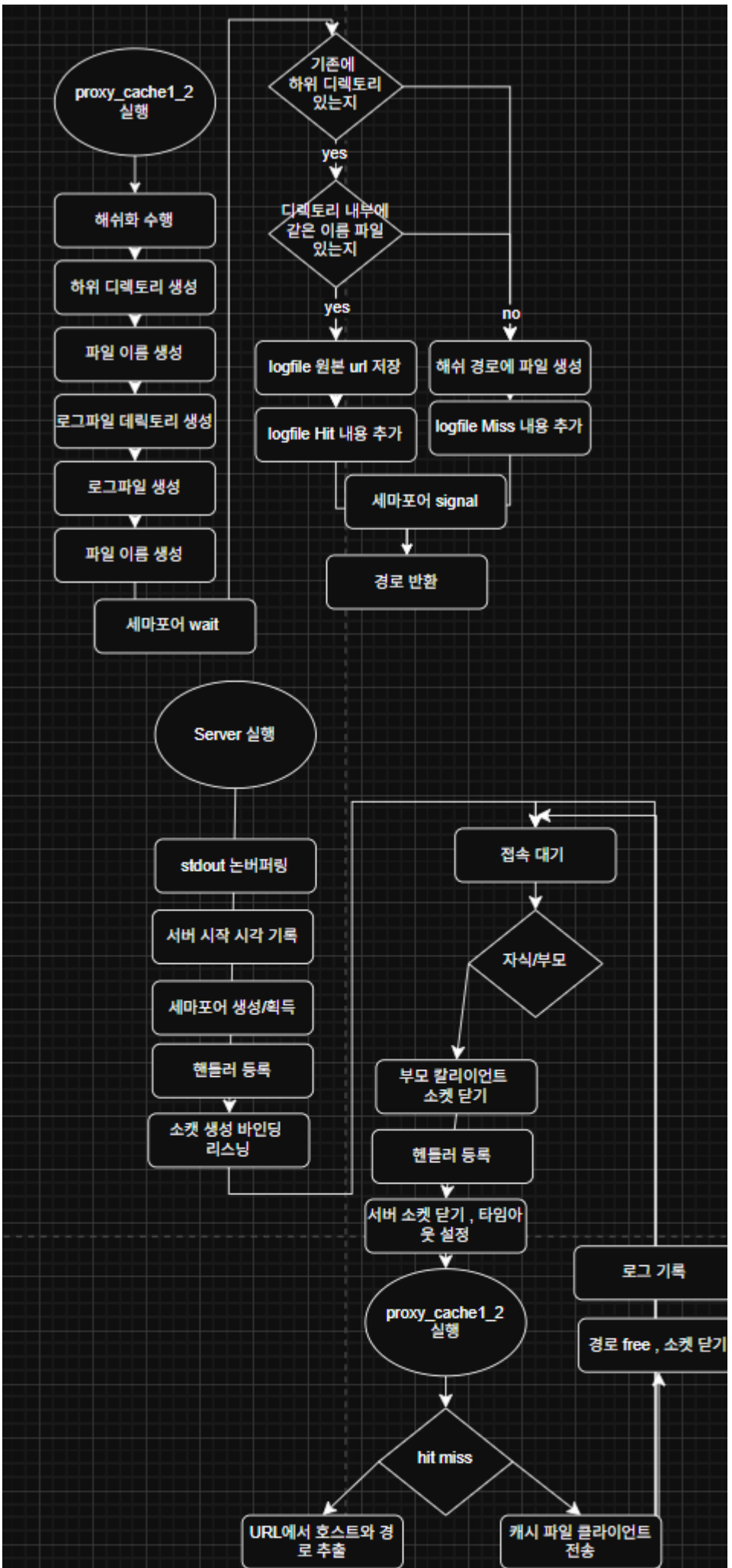
```
kw2021202003@ubuntu:~/work/Proxy3-2_B_2021202003_강준우$ ./proxy_cache
*PID# 7027 is waiting for the semaphore.
*PID# 7027 is in the critical zone.
*PID# 7027 create the *TID# 140549983930112.
*TID# 140549983930112 is exited.
*PID# 7027 exited the critical zone.
*PID# 7144 is waiting for the semaphore.
*PID# 7144 is in the critical zone.
*PID# 7144 create the *TID# 140549983930112.
*TID# 140549983930112 is exited.
*PID# 7144 exited the critical zone.
```

터미널에서는 각 자식 프로세스는 임계 구역에 진입할 때마다 로그 작성을 위한 스레드를 생성하고, 작업 후 해당 스레드는 종료됩니다. 이 터미널 출력은 프로세스의 세마포어 대기, 임계 구역 진입·이탈, 스레드 생성·종료 과정을 순차적으로 보여줍니다.

[MISS]565-[73643331b10dd1a32919b722a5982e85e6662]
[HIT]565/73643331b10dd1a32919b722a5982e85e6662-[2025/06/02, 22:13:28]
[HIT]http://textfiles.com/
SERVER [Terminated] run time: 148 sec. #sub process: 0
로그 파일에서는 다음과 같은 내용을 반환합니다

kw2021202003@ubuntu:~/cache$ tree
.
└── 565
    └── 73643331b10dd1a32919b722a5982e85e6662
또한 해당 경로도 하나만 생성된것을 볼 수 있습니다.

플로우차트는 다음과 같습니다.


proxy_cache1_2 실행
해쉬화 수행
하위 디렉토리 생성
파일 이름 생성
로그파일 데릭토리 생성
로그파일 생성
파일 이름 생성
세마포어 wait
기존에 하위 디렉토리 있는지
yes
디렉토리 내부에 같은 이름 파일 있는지
yes
no
logfile 원본 url 저장
logfile Hit 내용 추가
해쉬 경로에 파일 생성
logfile Miss 내용 추가
세마포어 signal
경로 반환
Server 실행
stdout 논버퍼링
서버 시작 시각 기록
세마포어 생성/획득
핸들러 등록
소캣 생성 바인딩 리스닝
접속 대기
자식/부모
부모 칼리이언트 소켓 닫기
핸들러 등록
서버 소켓 닫기 , 타임아웃 설정
proxy_cache1_2 실행
hit miss
URL에서 호스트와 경로 추출
캐시 파일 클라이언트 전송
로그 기록
경로 free , 소켓 닫기

psudo code

[main 함수 시작]
- stdout 버퍼링 비활성화 (setbuf)
- 서버 시작 시간 기록
- 세마포어 생성 또는 기존 것 가져오기
- SIGINT, SIGCHLD 시그널 핸들러 등록
- 소켓 생성 → 바인딩 → 리스닝

[클라이언트 접속 대기 반복]
- 클라이언트 접속 수락 (accept)
- 자식 프로세스 fork
  ㄴ 부모: 클라이언트 소켓 닫고 반복 계속
  ㄴ 자식:
    - 서버 소켓 닫기
    - SIGALRM 타이머 핸들러 설정
    - 자식 프로세스 수 증가
    - 클라이언트 요청 수신 (read)
    - HTTP method / URL / protocol 파싱
    - is_main_url()로 메인 요청 여부 확인
    - proxy_cache1_2(url, &hit) 호출

    - HIT일 경우:
      - 캐시 파일 열어서 내용 클라이언트로 전송

    - MISS일 경우:
      - get_host_from_url(), get_path_from_url()
      - 웹 서버와 소켓 연결 후 요청 전송
      - 응답 받아 캐시 파일에 저장 및 클라이언트로 전송

    - 리소스 정리: host, path, file free
    - 클라이언트 소켓 close
    - 자식 프로세스 종료

[proxy_cache1_2 함수]
- 홈 디렉토리 경로 얻기
- 캐시 디렉토리, 로그 디렉토리 생성
- URL SHA1 해시 수행 → hash 값 생성
- 서브디렉토리 이름 추출 (앞 3자리)
- 캐시 파일 경로 생성
- 해당 디렉토리 존재 여부 및 파일 존재 여부 검사
  → 있으면 HIT, 없으면 MISS

[임계구역 진입]
- 세마포어 wait(P) → *PID waiting 메시지 출력
- *PID in critical zone 메시지 출력

[로그 기록 스레드 준비]
- 시간 정보 timestamp(ts) 생성
- log_param 구조체 생성 (logpath, found, sub, hash3, ts, url)
- pthread_create()로 스레드 생성
  → *PID create the *TID 출력
- pthread_join()으로 스레드 종료 대기
  → *TID is exited 출력

[임계구역 종료]
- *PID exited the critical zone 출력
- 세마포어 signal(V)

- 결과적으로 HIT 여부 반환 및 파일 경로 반환

고찰

세마포어를 통해서 구역을 제어하는 방법을 구현해보았습니다.
이전에는 자식프로세스에서 바로 수행하는 방식이었지만 이번 과제에선느 동시 접근에 따른 레이스 컨디션에 대한 상황이 보였고 이에 따른 세마포어를 구현하였지만 수행할 때 여러 사이트를 연달아서 눌러서 수행햐는 부분에 대해서 통제 제어가 완벽하지 않는 생각이 들기도 합니다.
그동안의 함수 구현 및 서버 연결 딜레이 계산 등의 수행들을 하며 시스템 프로그래밍에 대한 전반적인 수행 능력이 커졌다고 생각합니다.