시스템프로그래밍 과제 보고서

Proxy 1-3

수업 명: 시스템프로그래밍 월5수6

담당 교수: 김태석 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

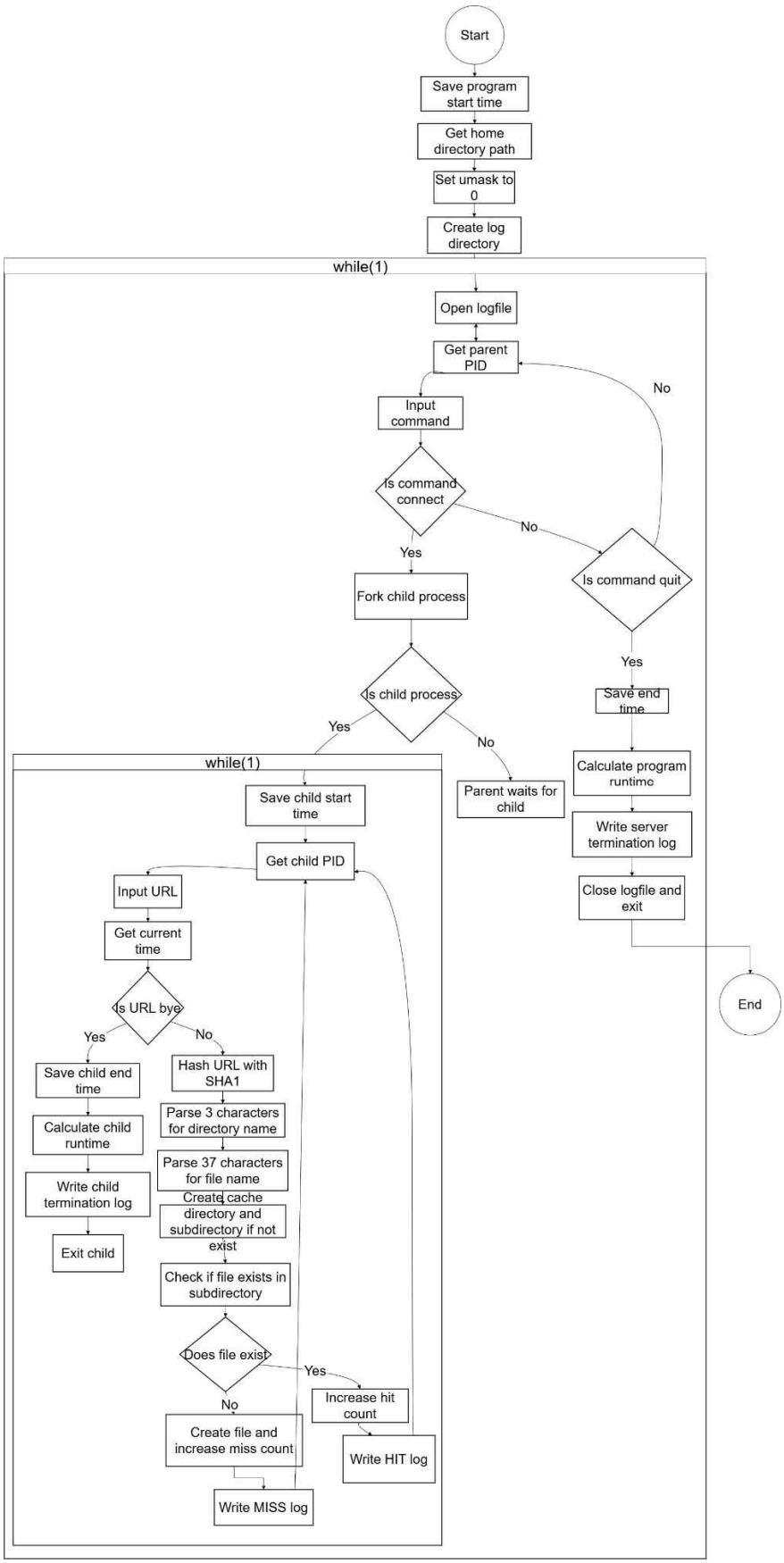
이름: 최현진

제출일: 2025.04.14

Introduction

이번 Proxy 1-3 과제는 Proxy 1-2에서 구현한 기능에, 동시성 서버 기능을 추가하는 시스템 프로그래밍 실습 과제이다. 과제는 먼저 메인 프로세스에서 명령어를 입력 받고, 명령어가 connect이면, 새로운 서브 프로세스를 생성한다. 해당 서브 프로세스는 Proxy 1-2에서 구현한 기능을 수행하는데, SHA1 해시를 통해 캐시 디렉토리 및 파일을 생성하고, 캐시 파일의 존재 여부를 판단하여 HIT 또는 MISS를 결정하고 로그 파일에 작성한다. bye 가 입력되면, 서브 프로세스는 종료되고 해당 종료 로그를 기록한 후 메인 프로세스로 돌아간다. 명령어로 quit이 입력되면, 메인 프로세스가 종료되어 실행 시간과 생성된 자식 프로세스 수의 합이 기록된다. 이번 과제를 통해 fork()와 wait(), exit()과 같은 함수를 활용하여 프로세스 생성과 종료, 파일 시스템 조작과 같은 시스템 프로그래밍을 실습할 수 있다.

Flow Chart


Start
Save program start time
Get home directory path
Set umask to 0
Create log directory
while(1)
Open logfile
Get parent PID
Input command
Is command connect
No
Yes
Fork child process
Is command quit
No
Yes
Is child process
Yes
No
Save end time
Calculate program runtime
Write server termination log
Close logfile and exit
End
Parent waits for child
while(1)
Save child start time
Get child PID
Input URL
Get current time
Is URL bye
Yes
No
Save child end time
Calculate child runtime
Write child termination log
Exit child
Hash URL with SHA1
Parse 3 characters for directory name
Parse 37 characters for file name
Create cache directory and subdirectory if not exist
Check if file exists in subdirectory
Does file exist
Yes
No
Increase hit count
Write HIT log
Create file and increase miss count
Write MISS log

Pseudo code

main:

```
프로그램 시작 시간 저장
home 디렉토리 경로 가져오기
umask 0 설정
log 디렉토리(~/logfile) 생성
로그 파일(logfile.txt) 열기

무한 반복
    부모 프로세스 PID 불러오기
    명령어 입력 받기 (cmd)

    명령어가 "connect"면
        자식 프로세스 생성 (fork)
        자식 프로세스면 (pid == 0)
            자식 시작 시간 저장

            무한 반복
                자식 PID 불러오기
                URL 입력 받기
                입력 시간 저장

                입력이 "bye"면
                    자식 종료 시간 저장
                    실행 시간 계산
                    자식 종료 로그 작성
                    자식 종료

                입력된 URL을 SHA1 해싱
                해시 앞 3글자 추출 → 캐시 디렉토리 이름
                나머지 37글자 추출 → 캐시 파일 이름

                루트 캐시 디렉토리(~/cache) 생성
                서브 디렉토리(~/cache/xxx) 생성
                해당 디렉토리로 이동
```

캐시 파일 이름으로 존재 여부 학인
일치하는 파일이 있으면 → HIT
없으면 → MISS

MISS인 경우
캐시 파일 생성
miss++

로그 디렉토리로 이동
로그 파일에 HIT/MISS 로그 작성

부모 프로세스면 (else)
자식 종료 대기

명령어가 "quit"이면
프로그램 종료 시간 저장
실행 시간 계산
서버 종료 로그 작성
로그 파일 닫기
프로그램 종료

결과 화면

1. proxy 1-3

1. Input

```
kw2023202070@ubuntu:~/proxy$ ls
Makefile  proxy_cache.c
kw2023202070@ubuntu:~/proxy$ make
gcc proxy_cache.c -o proxy_cache -lcrypto
kw2023202070@ubuntu:~/proxy$ ls
Makefile  proxy_cache  proxy_cache.c
kw2023202070@ubuntu:~/proxy$ ./proxy_cache
[3319]input CMD> connect
[3322]input URL> www.kw.ac.kr
[3322]input URL> www.google.com
[3322]input URL> bye
[3319]input CMD> connect
[3325]input URL> www.kw.ac.kr
[3325]input URL> www.naver.com
[3325]input URL> bye
[3319]input CMD> quit
```

$ ls: Makefile과 소스 코드(proxy_cache.c)를 작성하였다.

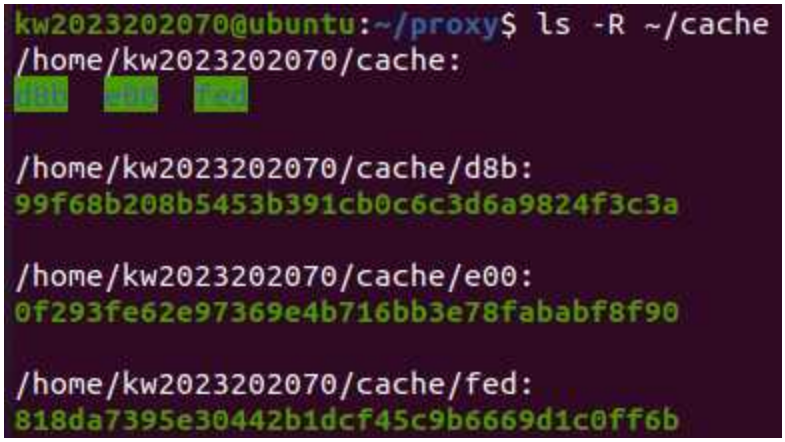
$ make: make를 실행하여 컴파일을 수행하였고, gcc 컴파일 명령어가 정상적으로 실행되었다.

$ ls: 컴파일 결과, 실행 파일 proxy_cache가 생성된 것을 확인했다.

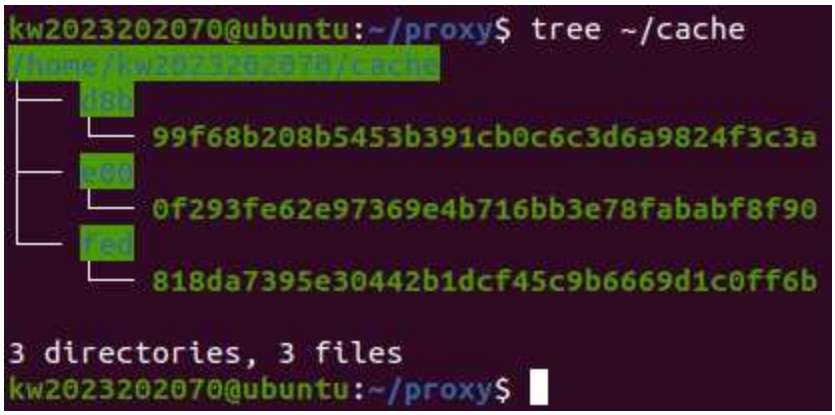
$ ./proxy_cache: 실행 파일 실행한 결과를 확인했다.

메인 프로세스에서 connect가 입력되어 새로운 프로세스 pid=3322를 생성하고, 해당 프로세스 종료까지 대기한다. 이후 두 url를 입력하고 bye를 입력하여 3322 서브 프로세스를 종료했다. 3319 메인 프로세스로 돌아와 다시 connect를 입력하여 새로운 서브 프로세스 pid=3325를 생성했다. Hit url 하나와 miss url 하나를 입력하고 bye를 입력하여 3325 서브 프로세스를 종료했다. 대기하던 3319 메인 프로세스를 quit를 입력하여 종료했다.

2. Cache Directory



$ ls -R ~/cache: ls 명령어를 –R 옵션을 통해 재귀적으로 실행하여 ~/cache 디렉토리와 그 하위 디렉토리까지 출력하여 캐시 디렉토리와 파일들이 생성됨을 확인했다.



$ tree ~/cache/: ~/cache 구조 확인 결과, SHA1 해시된 url의 앞 3글자를 이름으로 하여 디렉토리가 생성되었다. 그 디렉토리의 하위에는 나머지 37글자 이름으로 파일이 3개 생성되었다.

3. Log file

```
kw2023202070@ubuntu:~/proxy$ cat ~/logfile/logfile.txt
[Miss]www.kw.ac.kr-[2025/04/14, 05:45:56]
[Miss]www.google.com-[2025/04/14, 05:46:00]
[Terminated] run time: 9 sec. #request hit : 0, miss : 2
[Hit]e00/0f293fe62e97369e4b716bb3e78fababf8f90-[2025/04/14, 05:46:07]
[Hit]www.kw.ac.kr
[Miss]www.naver.com-[2025/04/14, 05:46:10]
[Terminated] run time: 9 sec. #request hit : 1, miss : 1
**SERVER** [Terminated] run time: 25 sec. #sub process: 2
```

$ cat ~/logfile/logfile.txt: logfile.txt을 출력한 결과를 확인했다.

메인 프로세스에서 connect 명령어로 처음 생성한 서브 프로세스의 동작으로, miss 동작 처리가 되어 url과 시간 정보가 저장되었다. 이후 해당 프로세스가 종료되어 프로세스 실행시간 및 요청 결과 (hit/miss 횟수)가 저장된 것을 확인했다.

다음 생성한 서브 프로세스 동작으로, hit 동작 처리가 되어 생성된 디렉토리와 파일과 시간 정보, url이 저장된 것을 확인했다. miss 로그또한 작성되었다. 이후 해당 프로세스의 종료 로그도 작성되었다.

대기하던 메인 프로세스가 quit 명령어로 종료되어 총 프로그램 실행 시간과 생성된 총 서브 프로세스 개수가 로그에 기록되었다.

고찰

이번 과제를 수행하며 처음으로 여러 프로세스가 동시에 동작하는 구조를 직접 구현해보는 경험을 했다. 특히 fork() 함수를 통해 두 가지 반환값을 다루며, 부모와 자식 프로세스를 분리하여 각각 다른 작업을 수행하게 하는 작업을 이해하며 이론 시간에 배운 수업 내용을 충분히 복습하는 시간이 필요했다.

특히 wait() 함수를 사용하여 부모 프로세스가 자식 프로세스의 종료를 기다리는 타이밍을 조절하는 부분, 그리고 각각의 프로세스가 작성해야 하는 로그를 처리하는 과정이 까다로웠다. 따라서 각각 프로세스의 역할을 수행하면서 필요한 time(), getpid(), exit() 등의 함수 사용법에 익숙해질 수 있었다.

해당 프로세스 관련 함수를 이해하고 사용해보는 것을 제외하면 Proxy 1-2 과제까지의 내용을 그대로 활용하는 부분이라 수월하게 진행 할 수 있었다.

과제를 완성하면서 앞으로 더 복잡한 서버 구조나 프로세스 구현을 해 보는 것에 기대가 되었다.

Reference

시스템프로그래밍 이론 및 실습 자료 참고하였습니다.