시스템 프로그래밍 실습 Report

실습 제목: Proxy #3-2

Logging using threads

실습일자: 2022년 5월 23일 (월)

제출일자: 2022년 6월 08일 (수)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 목요일 7,8교시

학 번: 2018202065

성 명: 박 철 준

- Introduction

1. 제목

Proxy 3-2

logging using threads.

2. 목표

- 과거 3-2과제 부분에서 logfile을 기록하는 thread를 추가하여한다.
- 즉 3-1의 critical section 내에서 log를 기록하는 thread를 추가하는 것이다.
- 3-1에서 구현했던 메시지는 그대로 유지하여야 한다.
- 동시에 여러 프로세스가 접근했을 때를 시뮬레이션하여 서버의 터미널에 상태 출력하여야 한다.
- critical sectopm 접근부만 TID관련 정보를 추가하여야 한다.
- 즉 어떤 자식 프로세스가 스레드를 생성하였는가를 출력해야하며
- 양식은 다음과 같다.(*PID# getpid() create the *TID# phtread_self().)
- 스레드가 종료될 시 (*TID# pthread_self() is exited.)
- 캐시 파일의 락 경쟁 문제는 고려하지 않는다.

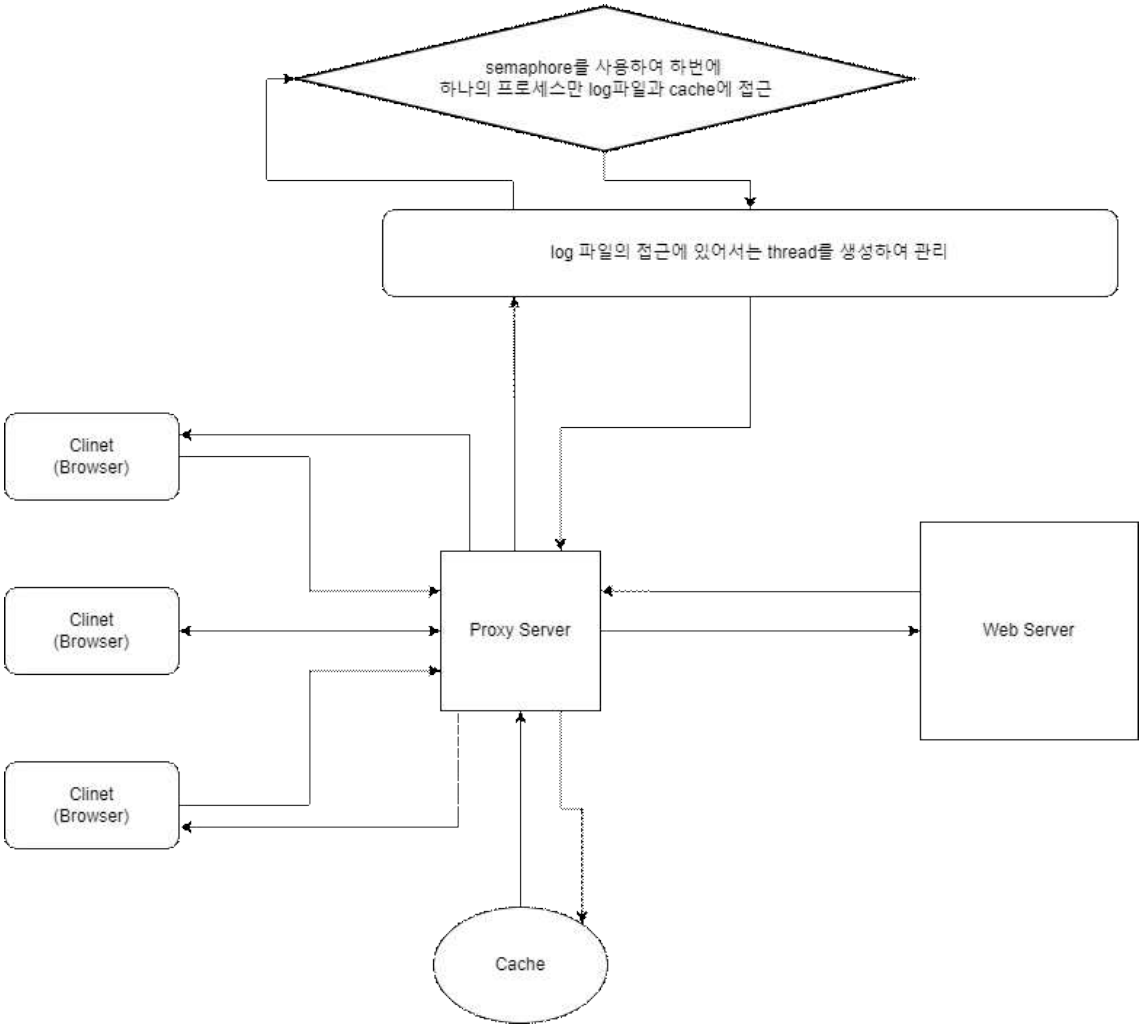
3. 실험 방법

- 시뮬레이션 시나리오를 작성해서 실험하여야 한다.
- Critical section 내에 sleep() 등을 사용하면 동시 접근하는 경우를 볼 수 있다.
- 터미널 출력 화면에 출력하여야 하며 터미널 양식은 아래와 같다.
- *PID# getpid() is waiting for the semaphore.
- *PID# getpid() is in the critical zone.
- *PID# getpid() create the *TID# phtread_self().
- *TID# pthread_self() is exited.
- *PID# getpid() exited the critical zone.

● 실습 구현

-Flow Chart

세마포어를 이용하여 하나의 프로세스만이 Critical section을 지정하여 공유자원인 log파일과 cache파일에 접근 할 수 있는 흐름도를 가졌다.



-Pseudo code

이번 과제의 Pesudo code는 이전 과제에서 설명한 것은 제외하고 구현한 스레드 함수에 넣을 인자들을 저장할 구조체, 스레드 함수인 thread_function(), 로그파일에 hit or miss와 종료문을 입력하는 함수인 write_log_about_hit_or_miss 함수 그리고 수정된 main함수에 대해 설명하고자 한다. 가장 먼저 스레드 함수에 넣을 인자들을 저장할 구조체 passing_thread_data_struct이다.

- passing_thread_data_struct 구조체

```
typedef struct passing_thread_data_struct
{
        //쓰레드 함수에서 사용할 변수들을 구성하는 구조체이며
        //이는 서브 프로세스에서 구조체 변수를 선언하고 저장하여
        //쓰레드 함수에서 쓰인다.
        char passing_thread_data_hash_passing_url[BUFFSIZE]; // URL을 저장하는 변수
        int passing_thread_data_subprocess_misscount; // miss개수를 저장하는 변수
        int passing_thread_data_subprocess_hitcount ; // hit 개수를 저장하는 변수
        int passing_thread_data_hit_or_miss_check ; // hit인지 miss인지 판단한 데이터를 저장할 변수
        int passing_thread_data_result; // 종료시간을 저장할 변수
}passing_thread_data_struct;// 예명으로 사용한다.
```

- thread_fuction() 함수

```
void* thread_function(void* passing_thread_data){
    //스레드함수이며
    //인자는 void형 포인터 자료형 이는 구조체 변수의 void*형으로 타입 케스트를 통해
    //인자로 넘겨줄 것이다.
    printf("*PID# %ld create the *TID# %ld.₩n", (long)getpid(), (long)pthread_self());
    // 인자를 사용하기위해 다시 타입 케스트
    passing_thread_data_struct* passing_thread_data_in_thread_function
    =(passing_thread_data_struct*)passing_thread_data;
    //Hit or miss를 저장할 함수 호출
     write_log_about_hit_or_miss(
     passing_thread_data_in_thread_function->passing_thread_data_hash_passing_url,
     passing_thread_data_in_thread_function->passing_thread_data_hit_or_miss_check);
     // 서브 프로세스의 종료문을 출력하기위한 함수 호출
     bye_browser_subprocess(
     passing_thread_data_in_thread_function->passing_thread_data_subprocess_hitcount,
     passing_thread_data_in_thread_function->passing_thread_data_subprocess_misscount,
     passing_thread_data_in_thread_function->passing_thread_data_result);
     printf("*TID# %ld is exited.₩n", (long)pthread_self());
}
```

- write_log_about_hit_or_miss() 함수

```
void write_log_about_hit_or_miss(char * host_url, int hit_or_miss_check)
{
	//로그파일에 hit or miss를 입력하기 위한 함수
	//인자로 호스트 유알엘과 힛 혹은 미스 개수를 입력받는다
	호스트 유알엘을 해쉬함수를 사용하여 해쉬된 유알엘로 변경
	해쉬된 유알엘을 3글자를 제외하여 저장하고 나머지를 따로 저장함

	if (hit)
	{
		로그파일에  hit을 형식에 맞게 입력
	}
	else if (hit_or_miss_check == 0)
	{
		로그파일에 miss를 형식에 맞게 입력
	}
}
```

-main 함수
기존과제 3-1에서는 크리티컬 섹션 부분을
1. hit miss를 판별하여 로그파일에 작성하는 부분
2. 캐시파일에 접근하여 브라우저에 데이터를 전달하는 부분
3. 클라이언트의 종료로 인한 로그파일에 종료문 작성
이렇게 3개의 부분으로 나눠서 설계하였지만 크리티컬 섹션이 많아지면 프로그램 동작에 있어 악영향을 미칠 수 있고 3개의 영역을 잘 생각해보면 하나로 합칠 수 있기 때문에 3개를 하나의 크리티컬 섹션으로 합쳤다. 그래서 다음과 같이 쉐도 코드를 가진다
코드가 긴 관계로 바뀐 부분만을 설명하겠다.

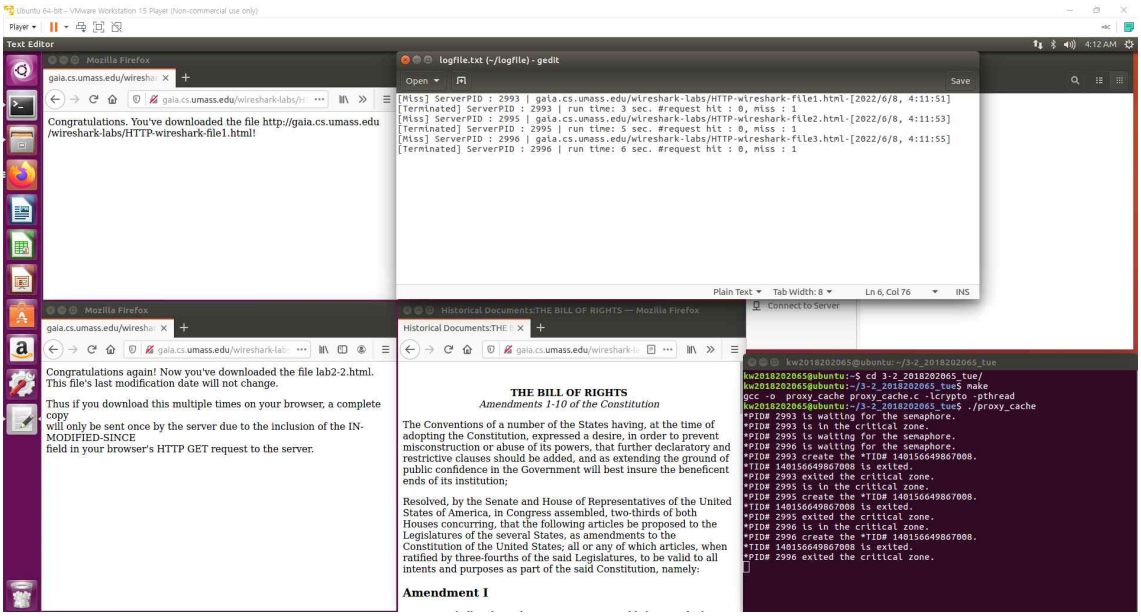
```
main()
{
        pthread_t thread_id;//스레드 아이디를 생성한다.
        int err; //스레드 생성에 실패하면 에러를 저장하기 위해 생성
        void* thread_return; //스레드 함수의 리턴값을 저장할 변수

        1. hit miss 판별(따로 로그파일에 작성하지 않고 디렉터리 및 캐시파일 만 생성함)

        if(hit or miss라면 작동 방식이 같기 때문에 같이 설명하겠다.)
        {
                p(semid)
                2. 캐시에서 데이터 사용 후 전달
                err=pthread_create(&thread_id,NULL,
                thread_function,(void*)&passing_thread_data);
                //쓰레드 아이디 변수에 쓰레드 생성후 해당 아이디 담기
                if (err != 0)
                {
                        printf("pthread_create() error.₩n");
                }
                3. 로그파일에 hit or miss 밑 종료문 작성
                v(semid)
        }
}
```

이렇게 앞서 정의된 세마포어의 크리티컬 영역에 쓰레드를 생성하여 로그파일의 접근을 하며 한 번에 한 프로세스만이 접근하고 사용할 수 있도록 하였다.

● 실습 결과

-실습 방법 (시나리오)
크리티컬 존에 대한 순서를 다음과 같이 나열 한다.
1번 : 캐시에 접근하여 브라우저에 캐시 데이터를 전송하는 크리티컬 존(Hit, Miss의 경우를 통합해서 지칭하겠다.)
이후 로그파일에 Hit or Miss 내용과 브라우저에 대한 서브 프로세스의 종료문을 기록한다.
이후 아래의 웹사이트에 대해
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html
거의 동시에 차례로 접속하게 되면
가장 먼저의 PID 2993이 크리티컬 존에 들어가게 되고 이에 따라 같은 일을 하는 크리티컬 존에 대해 2995와 2996은 웨이팅 상태가 된다. 그리고 2993이 크리티컬 내부에 있으면서 2993 프로세스에서 생성된 쓰레드 아이디를 출력하고 그 다음으로는 해당 쓰레드를 종료하고 2993은 크리티컬 섹션을 빠져나간다. 이후 대기하던 2995가 크리티컬 영역으로 들어와 2993번처럼 프로세스에서 생성된 쓰레드 아이디를 출력하고 그 다음으로 해당 쓰레드를 종료하고 크리티컬 섹션을 빠져나간다. 이후 마지막으로 대기하던 2996이 크리티컬 영역으로 들어와 2995번처럼 프로세스에서 생성된 쓰레드 아이디를 출력하고 다음으로 해당 쓰레드를 종료하고 크리티컬 섹션을 빠져나간다. 이에 대한 실행이
올바른 결과가 나오게 되는지 터미널과 로그파일의 기록정보를 확인하여 판단한다.

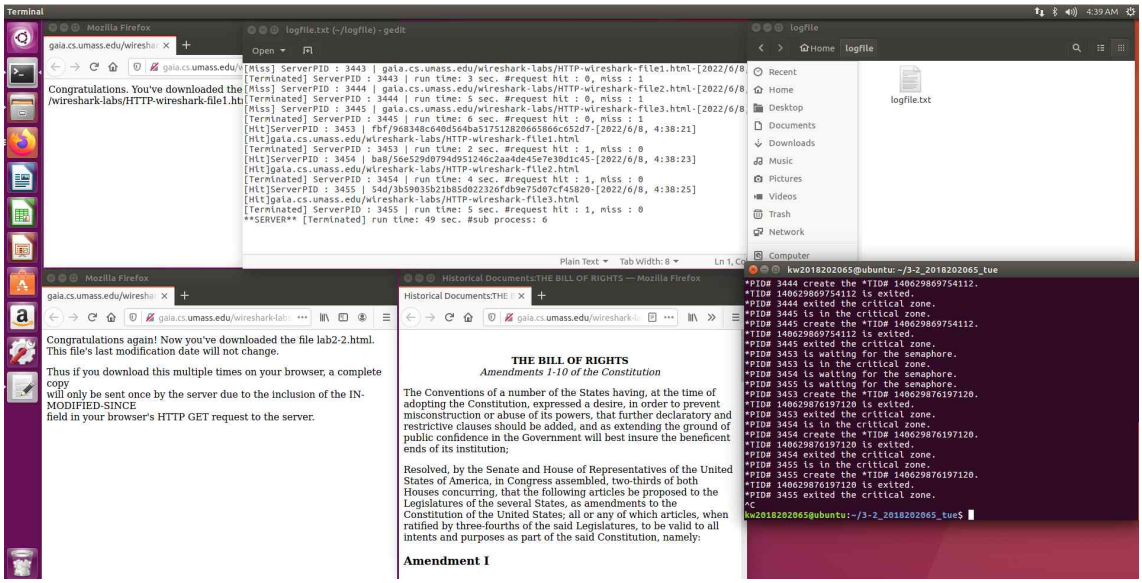
-결과 화면



위 시나리오와 정확히 일치 하게 터미널에 출력됨을 알 수 있다. 1개의 프로세스가 1개의 크리티컬 존에 들어가는 모습을 터미널을 통해 알 수 있다. 또한 이때 크리티컬 존에서 해당하

는 쓰레드 영역을 생성하는 것을 알 수 있다. 또한 동시접근을 제한하여 순서에 맞게 공유 메모리에 접근하는 것을 터미널을 통해 알 수 있다.

Hit일 경우도 문제 없이 잘 작동한다. (아래 사진 참조)



● 고찰

이번 과제를 진행하면서 발생한 이슈는 다음과 같다. 또한 이것에 대한 고찰을 같이 서술하고자 한다.

1. 이번 과제를 구현하면서 기존 3-1과제에서 나눠 두었던 크리티컬 섹션을 하나로 합친 이유

먼저 3-1과제를 진행하면서 프로세스간 공유 자원 접근 제어를 가능케 하여 로그파일의 동시접근을 제어하였지만 가독성 측면에서 크리티컬 섹션이 여러개 있으면 프로세스가 동작간에 여러번 크리티컬 섹션에 들어갈 것이고 이것에 대한 순서는 물론 정해져 있지만 가독성 측면에도 안 좋고 프로세스의 흐름을 제어하기에도 불편함이 따른다 그래서 크리티컬 영역을 하나로 만들고 한프로세스가 한번 크리티컬 영역에 들어가기 때문에 터미널 상의 출력문의 가독성이나 로그파일의 가독성을 더 좋게 업그레이드 할 수 있었다.

2. 부모 프로세스의 스레드에 관한 이슈

과제를 진행하면서 pthread_ctreate() 함수와 pthread_join 함수를 사용함에 있어 main함수에서 pthread_self()함수를 사용하여 현재 쓰레드의 아이디를 확인 한 결과 main함수에서 실행한 pthread_self() 함수의 결과가 쓰레드 함수에서 실행한 것과는 서로 다른 것을 알게 되었다 즉 메인 함수를 실행하는 쓰레드는 메인 쓰레드로써 부모 프로세스와 자식 프로세스에서도 같은 것을 볼 수 있었다.

3. Hit인 경우와 Miss인 경우 생성한 쓰레드 아이디값이 서로 다름에 관한 이슈

먼저 Miss인 경우 동시입력받은 프로세스에 대한 생성한 쓰레드 아이디 값은 같은 것을 알 수 있었지만 Hit인 경우가 다음으로 입력받는 다면 해당 프로세스에 대한 생성한 쓰레드 아이디 값이 Miss인 경우와 다른 것을 알 수 있다. 서로 다른 쓰레드 아이디를 생성해서 공유자원 사용에 있어 동기화 측면에 오류를 줄 수 있겠지만 이번 과제는 세마포어 내부에 로그파일을 저장하는 쓰레드 생성임으로 크리티컬 섹션내에서 실행되기 때문에 동기화 적인 측면의 문제는 발생하지 않는다. 또한 만약 세마포어를 사용하지 않고 동기화 문제를 해결하기 위해서는 뮤텍스 방법을 사용하는 것도 좋을 것 같다.