System Programming

Assignment#3-2

09_Proxy 3-2

Professor	목 3 4 황호영 교수님
Department	컴퓨터공학과
Student ID	2012722028
Name	장 한 별
Date	2018. 06. 07

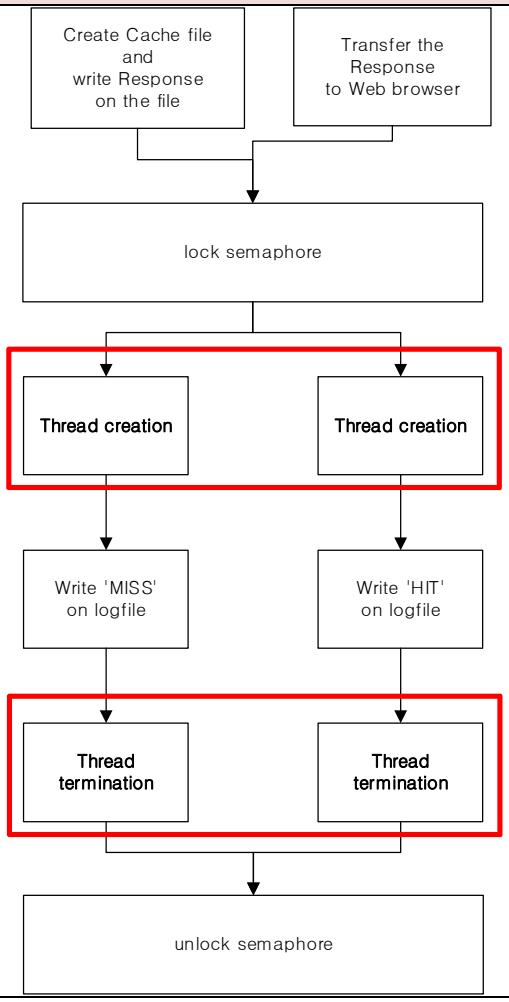
Introduction.

시스템 프로그래밍 강의 시간에 배운 proxy server 를 구현하는 것을 목표로 한다.

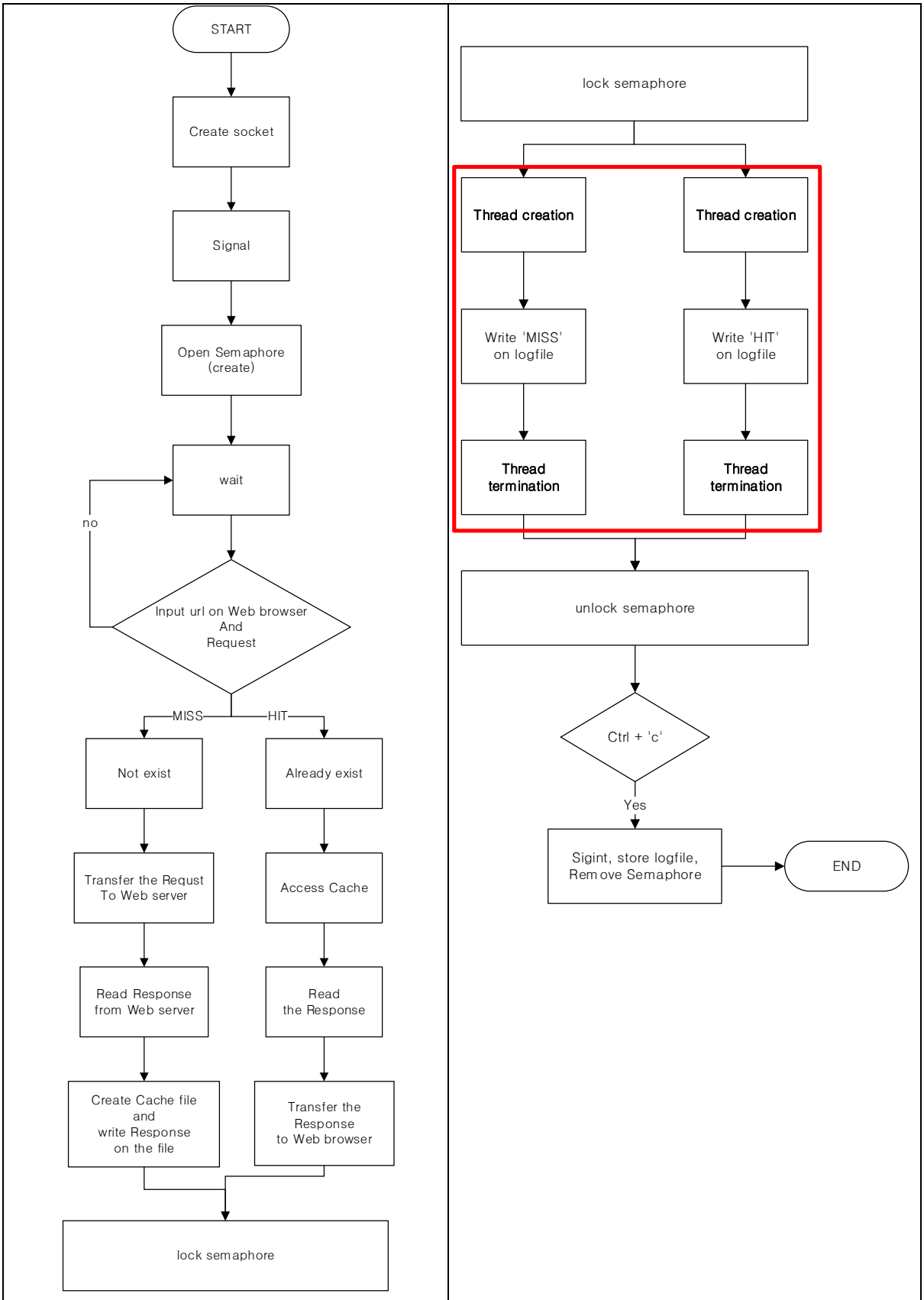
이번 과제는 logging using thread 이다. Thread는 특정 process 내에서 실행되는 하나의 흐름을 나타내는 단위인데, 이번 과제에서 thread 의 역할은 HIT, MISS를 판별 후, 저장해야 할 내용을 logfile 에 써넣는 것이다. Thread 관련 함수들은 process 관련 함수들과 아주 유사한데, 이 차이를 확실하게 구분하여 이번 과제를 구현하도록 한다.

Flow chart.

proxy_cache



이번 과제에 추가된 내용은 위 그림과 같다. HIT, MISS 를 판별해 그 내용을 logfile 에 저장하는 역할을 각 thread가 수행하게된다. 이를 위해 HIT, MISS 가 판별되면 각각의 위치에서 thread를 생성하고, HIT이나 MISS 를 logfile에 써넣고, thread를 종료하게 된다.



이번 과제 의 전체 flow chart 를 한눈에 보면 위 그림과 같다. 빨간색 네모는 이번 과제에 새로 추가된 내용이다.

Pseudo code.

proxy_cache

```
Int main(void)
{
    Socket();
    Setsockpot();
    Bind(socket);
    Listen(socket, 5);
    Signal(SIGCHLD);
    Signal(SIGALRM);
    Signal(SIGINT);

    Sem_open();

    While(1)
    {
        Accept();
        Read(client);

        Printf("%s", Request );
        Get url;

        Assignment #1-2;
        If(MISS)
        {
            Socket(web);
            Connect(web);
            Write(web, request);
            Alarm(10);
            While(read(web, buf))
            {
                Write(client, buf);
              Alarm(0);
            }
            Close(web);

            Fprintf(fp, response);
            Fclose(fp);

            Sem_wait( );
            Sprintf(logfile, "MISS");
            Pthread_create(&tid, NULL, thr_fn, logfile);
            Pthread_join( );
            Sem_post( );
```

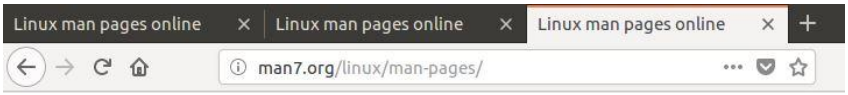
```
        }
        Else
        {
            Read(cache, response);
            Write(client, response);

            Sem_wait( );
            Sprintf(logfile, "HIT");
            Pthread_create(&tid, NULL, thr_fn, logfile);
            Pthread_join( );
            Sem_wait( );
        }
        Close(client);
    }
    Close(socket);
    Sem_unlink( );
    Return 0;
}

Void* thr_fn(void*)
{
    fprintf(logfile);
    pthread_exit( );
}
```

위 그림은 proxy_cache 의 전체 pseudo code 이다. Thread를 이용해서 logfile에 HIT, MISS 내용을 기록해야한다. Thread 를 생성하는 함수는 pthread_create( ) 이다. 이 함수를 사용해서 thread를 생성할 때, 하나의 함수를 인자로 같이 넣어줘야 하는데 이 함수가 새로 생성된 thread가 수행해야할 함수이다. 이 부분에 logfile을 기록하는 내용을 넣어주면 된다. 그 후 thread를 종료하는 함수인 pthread_exit( ) 를 사용하여 해당 thread를 종료시킨다. Exit( ) 는 전체가 모두 종료되기 때문에 pthread_exit( )함수를 이용하여 해당 thread 하나만을 종료시키도록 한다.

위 과정을 하는 동안 main thread는 새로운 thread 가 종료 되길 기다려야하는 데, 이 함수는 pthread_join( ) 이다. Process 관련 함수 중 waitpid( )와 유사하다.

Result.

3-2.(1)



기존의 main thread 와 새로 생성된 thread 들이 잘 동작하는지 확인하기 위해 3개의 filefox를 실행시켜 같은 url('man7.org/linux/man-pages')을 입력한다.

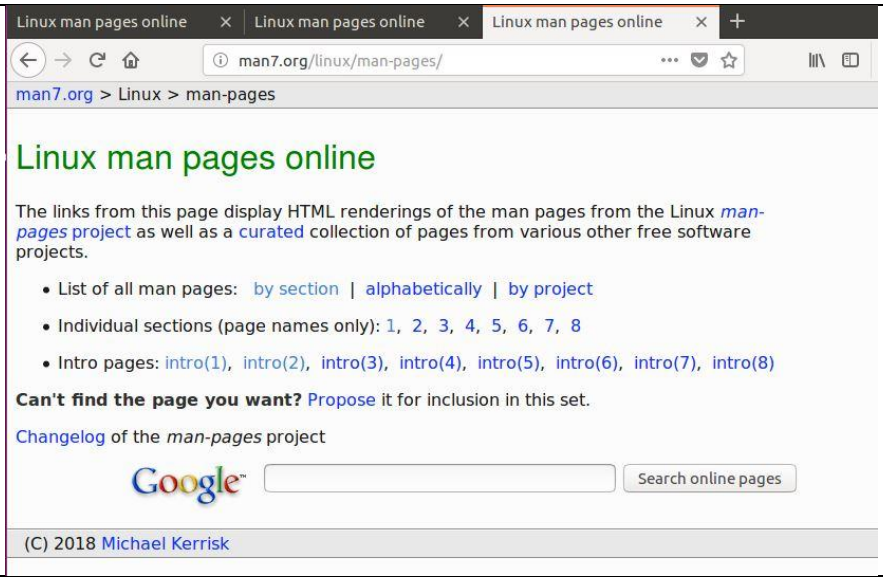
```
2012722028@sslab-desktop:~$ ./proxy_cache
*PID# 19734 is waiting for the semaphore.
*PID# 19734 is in the critical zone.
*PID# 19734 create the *TID# 140379426506496.
*TID# 140379426506496 is exited.
*PID# 19767 is waiting for the semaphore.
*PID# 19768 is waiting for the semaphore.
*PID# 19785 is waiting for the semaphore.
*PID# 19734 exited the critical zone.
*PID# 19767 is in the critical zone.
*PID# 19767 create the *TID# 140379426506496.
*TID# 140379426506496 is exited.
*PID# 19769 is waiting for the semaphore.
*PID# 19767 exited the critical zone.
*PID# 19768 is in the critical zone.
*PID# 19768 exited the critical zone.
*PID# 19785 is in the critical zone.
*PID# 19785 create the *TID# 140379426506496.
*TID# 140379426506496 is exited.
^C2012722028@sslab-desktop:~$ ls
```

Process 가 thread를 생성했다는 것을 확인하기 위해, 위 그림과 같이 출력했다. 빨간색 네모를 살펴보면, 첫 번째 thread 가 수행되고, 동작을 하고 나서 종료되었다는 것을 확인할 수 있다. 마찬가지로, 2번째, 3번째 thread도 역시 정상적으로 생성, 동작, 종료되었다는 것을 확인할 수 있다. 이때, thread 의 동작은 logfile에 HIT나 MISS 정보를 기록하는 것인데 이 수행이 정상적으로 작동했는지 해야한다.

3-2.(2)

```
2012722028@sslab-desktop:~$ cat ~/logfile/logfile.txt
[Miss]man7.org/linux/man-pages-[2018/6/7, 19:51:47]
[Hit]f3a/fe9a64ddd001d5bff149d718dab6cb68a43f9-[2018/6/7, 19:51:50]
[Hit]man7.org/linux/man-pages
[Hit]f3a/fe9a64ddd001d5bff149d718dab6cb68a43f9-[2018/6/7, 19:51:51]
[Hit]man7.org/linux/man-pages
**SERVER** [Terminated] run time: 21 sec. #sub process: 8
2012722028@sslab-desktop:~$
```

cat 명령어를 이용해서 logfile.txt 에 기록된 내용을 terminal에 출력하도록 한다. 3번의 같은 url 입력은 이론적으로 첫번째 입력: MISS, 두번째 입력: HIT, 세번째 입력: HIT 이다. 위 그림을 통해 thread가 정상적으로 동작했음을 확인할 수 있다.



위 그림을 통해, 웹 페이지 역시 올바르게 출력되었음을 확인할 수 있다.

Conclusion.

이번 과제는 thread를 이용해서 logfile 를 작성하는 것이다. Pthread_create( ) 함수를 이용해서 thread를 생성하고, pthread_exit( ) 함수를 이용해서 해당 thread를 종료하고, 그 동안 main thread는 pthread_join( ) 함수를 이용해서 새로운 thread가 종료되기를 기다린다. 위 과정들은 마치 process 를 생성하고, 어떤 일을 수행하고, 기다리고, 종료되는 과정과 매우 비슷하다. 이미 process 관련 함수들을 이전 과제들을 통해 다뤄봤기때문에 이번 과제는 어렵지않게 구현할 수 있었다.

Thread 관련 함수를 사용하는 것은 어렵지 않았지만, thread를 생성할 시, 그와 동시에 호출 하는 함수 thr_fn 을 이용하는 것이 까다로웠다. 이 함수는 void* 형으로 인자를 전달받고, void*형으로 반환하는 함수이다. 여기서 어려웠던 점은, 생성된 새로운 thread 에서 logfile을 기록해야 하는데 logfile에 기록할 내용을 void* 형을 인자로 받는 이 함수로 어떻게 넘기는 지가 문제였다. 최한솔 조교님의 도움을 받아 struct 구조체를 넘기는 첫번째 방법과 main 함수 내에서 logfile 을 기록해야 할 내용을 string 으로 만들어 넘기는 두번째 방법을 가르쳐주셔서 필자는 후자를 선택했다. 그 후 thr_fn 함수 내에서 void* 로 넘어온 logfile을 담고 있는 변수를 char* 형으로 형 변환시켜 logfile.txt 에 기록하도록 구현함으로 이번 과제를 마무리할 수 있었다. String을 만드는 과정에서 이미 sprintf( ) 함수를 여러 번 사용해봤음에도 불구하고, 이 함수가 떠오르지 않아 strcat( ) 함수로 이어 붙이는 미련했던 때도 있었지만, 코드가 너무 길어져 수정할 방법이 없을까 하다가 sprintf( )함수 가 떠올라서 곧바로 수정하기도 했다.

한 학기 동안 구현한 proxy server 는 혼자의 힘으로 라면 절대 완성할 수 없었던 것 같다. 그동안 도움 받았던 학우들과 조교님들에게 감사의 인사를 전하며 이 글을 마치도록 한다.