시스템 프로그래밍(화요일)

최상호 교수님

Proxy 3-2

컴퓨터정보공학부

2018202076

이연걸

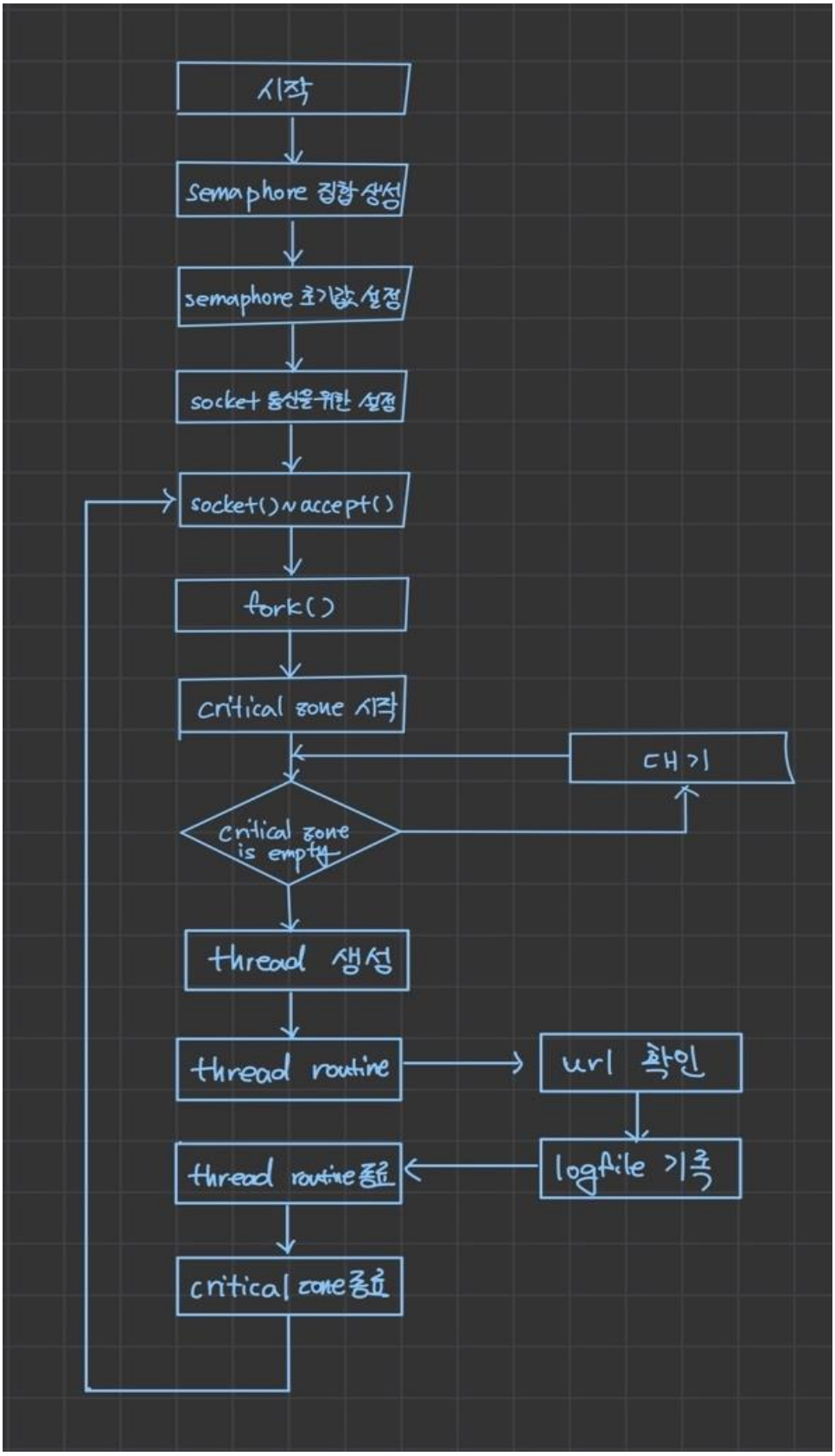
- Introduction

  이번 과제는 logfile에 관한 동시접근을 제어하는 것에 이어 logfile의 작성을 단순 critical zone에서 처리하는 것이 아닌, 다른 흐름, 즉 Thread를 새로 만들어 해당 Thread에서 logfile 작성을 하는 것을 목적으로 한다.

  프로그램의 흐름은 지난 과제와 동일 하지만 process가 새로 생성될 때마다 생성된 process에서 추가로 thread가 하나 더 필요하다. 이를 위해 필요한 것이 POSIX 운영체제를 위한 pthread.h 헤더파일이다. POSIX는 Portable Operation System Interface의 줄임말으로 서로 다른 UNIX OS의 공통 API를 정리한 Application interface이다. 해당 과제가 진행되는 환경은 Ubuntu16.04 이므로 사용 가능하다.

  헤더파일을 추가한 후 코드 작성은 간단하다. Thread를 추가하는 함수와 thread의 동작이 종료될 때까지 기다리는 함수 등을 사용해 과제에서 요구하는 동작을 처리하면 된다.

- Flow Chart



- Pseudo Code

  시작

Semaphore 집합 생성

Semaphore 초기값 설정

Socket 통신용 옵션들 설정

Socket()

Bind()

Listen()

While

Accept()

Fork()

Critical zone 시작

If Critical Zone is full

대기

Else

Thread 생성

T thread routine 시작

url 확인

logfile 기록

cache 파일 기록

endif

if thread routine is Not Finished

대기

Endif

Critical zone 종료

Endwhile

종료

- 결과 화면

HIT MISS 판별과 로그파일 기록을 한 개의 함수(check_url)로 묶었기 때문에 thread routine은 다음과 같다.

```
void *thread_routine(void *arg)
{                                       // thread routine to record log files
    t_hit = check_url(arg, http_req); // write logfile in thread routine
    return (void *)&t_hit;
}
```

결과화면은 다음과 같다.

```
kw2018202076@ubuntu:~/workspace/KW-Univ/시스템 프로그래밍/Proxy 3-2$ ./proxy_cache
*PID# 3490 is waiting for the semaphore.
*PID# 3490 is in the critical zone.
*PID# 3490 create the *TID# 69703424.
*TID# 69703424 is exited.
*PID# 3490 exited the critical zone.
*PID# 3510 is waiting for the semaphore.
*PID# 3510 is in the critical zone.
*PID# 3510 create the *TID# 69703424.
*TID# 69703424 is exited.
*PID# 3510 exited the critical zone.
^Ckw2018202076@ubuntu:~/workspace/KW-Univ/시스템 프로그래밍/Proxy 3-2$
```

- 고찰

처음 과제를 진행할 때는 단순 Thread를 추가하면 끝나는 일로 알고 있었다. 하지만 thread에서 해당 동작을 처리하기 위해 해당 로직이 담긴 함수를 thread routine에 넣어준다면 전달해야 하는 인자가 너무 많아져서 pthread_create 함수에 한 번에 전달할 수 없었다.

이를 위해 처음에는 구조체를 사용했다. 구조체에 hit, miss의 counter와 url, http request 정보를 담아 구조체 포인터를 사용해 전달했다. 동작은 확인하였지만 코드 수가 너무 많아져 가독성이 심각하게 떨어졌다. 이를 해결하기 위해 전역변수를 사용하였는데 전역변수의 사용은 최대한 자제해야 하는 것이 맞지만 간단한 프로그램에서 코드 수를 줄이는데 사용하는 것은 이상이 없을 것으로 판단하고 진행하였다. 전역 변수로 전달한 변수는 http request정보와 hit miss 판별을 위한 flag이다. 이렇게 13줄 정도의 코드를 줄일 수 있었다.

또한 지금까지 사용했던 코드를 전역변수를 사용해 더 쉽게 고칠 수 있었다. Child process와 parent process는 Data 영역을 공유하기 때문에 child process에서 실행된 함수를 parent 함수에서 다시 실행해 동작을 확인하는 대신 Data 영역에 저장되는 전역변수를 사용해서 코드를 더 직관적으로 개선할 수 있었다.