3-1

금3,4

과목명	시스템프로그래밍
담당교수	김태석 교수님
학과	컴퓨터정보공학부
학년	3학년
학번	2019202009
이름	서여지
제출일	21.05.25(화)

KWANGWOON UNIVERSITY

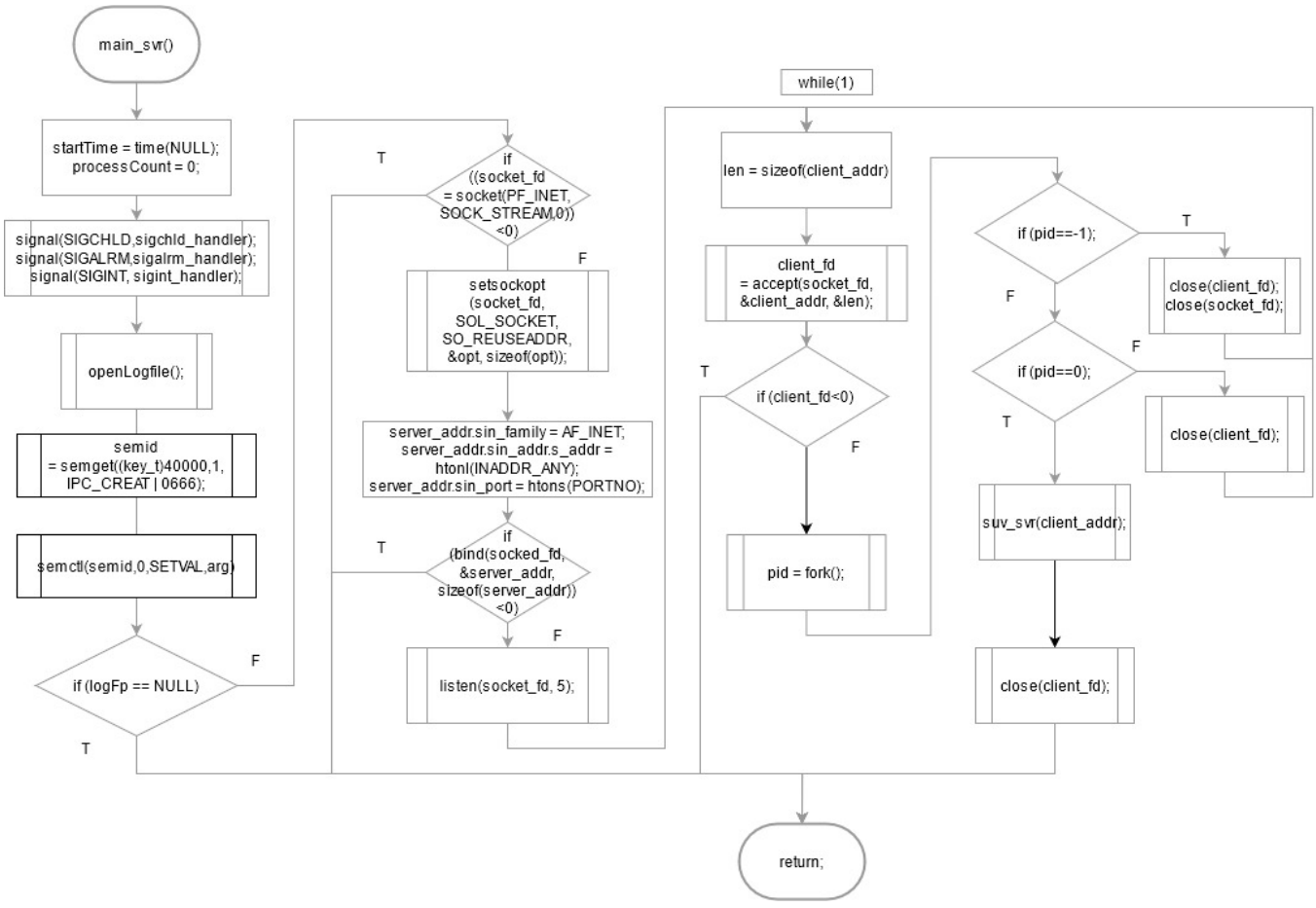
1. Introduction //과제 소개 – 5줄 이하 background 제외

이번 과제는 2-3-2 과제에서 추가한 logfile작성 과정을 critical section으로 지정하여, 한 번에 하나의 프로세스에서만 접근 가능하도록 제한하는 기능을 추가하는 과제이다. semaphore를 이용하여 접근을 제어하는 p함수와 v함수를 추가하여 해당 기능을 구현한다.

2. Flow Chart //코드 작성 순서도

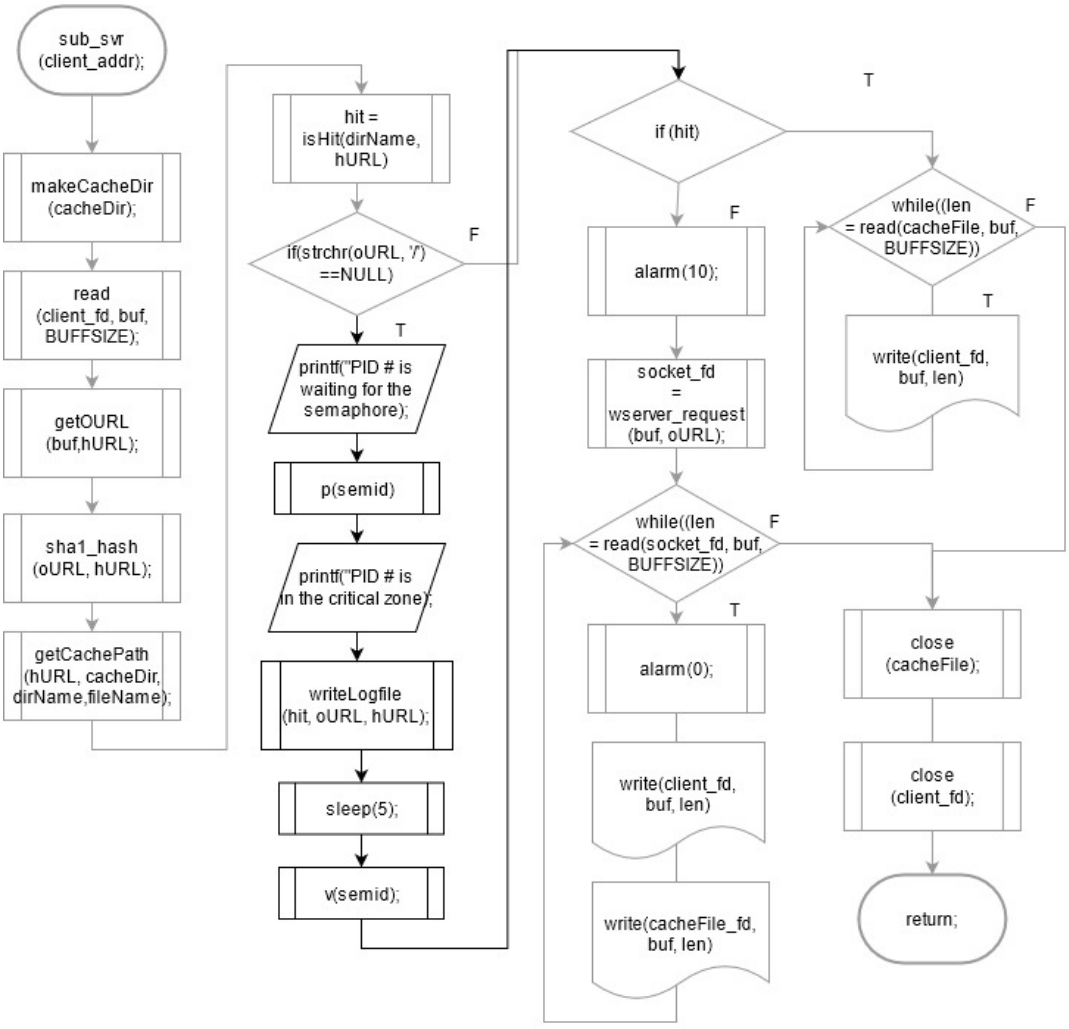
(1) main_svr

이전 프로젝트의 내용에서 semaphore를 생성하는 부분이 추가되었으며, 소켓 연결에 관한 출력 부분이 삭제되었다.



(2) sub_svr

writeLogfile 함수를 critical section으로 하여 semaphore를 사용하기 위한 p함수와 v함수가 추가되었다. 관련 터미널 출력이 추가되었고, client에서 받은 request message의 터미널 출력이 삭제되었다.



3. Pseudo code //알고리즘

```
void main_svr(void) {
      startTime = time(NULL);
      processCount = 0;
      signal(SIGCHLD, sigchld_handler);
      signal(SIGALRM), sigalrm_handler);
      signal(SIGINT, sigint_handler);

      openLogfile();
      if(logfile open error) return;      //can't open logfile

      semid = semget((key_t)40000,1,IPC_CREAT | 0666);
      if(semget error) return;

      arg.val = 1;
      semctl(semid, 0, SETVAL, arg);
      if(semctl error) return;

      socket(PF_INET, SOCK_STREAM);
```

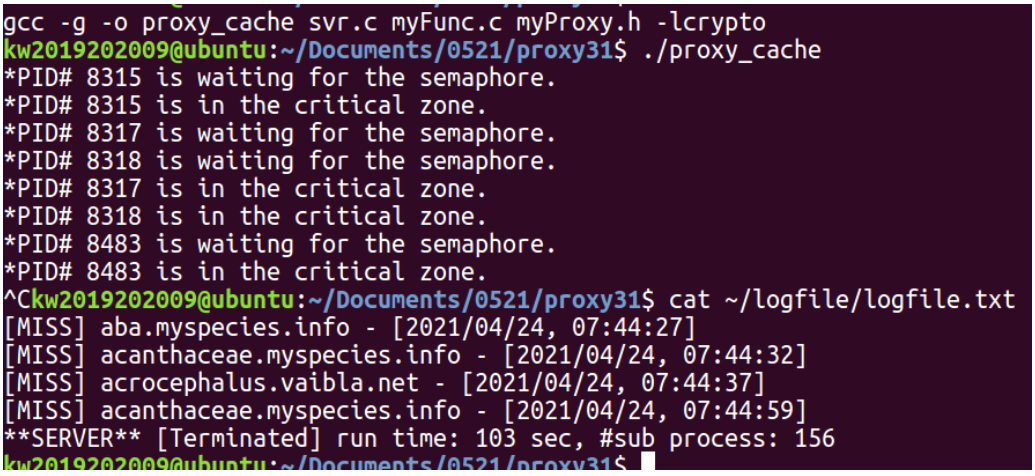
```
    if (socket error ) return;      // can't open socket
    setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    server_addr = 0;
    server_addr.sin_family = AF_INET;
    srever_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    srever_addr.sin_port = htons(PORTNO);

    bind(socket_fd, &server_addr, sizeof(server_addr))
    if( bind error ) return;      // can't bind address

    listen(socket_fd, 5);

    while(true){
        client_addr = 0;
        len = sizeof(client_addr);
        client_fd = accept(socket_fd, &server_addr, &len)
        if(client_fd <0)      return;      // accept err
        processCount++;
        pid = fork();
        if (fork err) {
            close(client_fd);
            close (socket_fd);
            continue;
        }
        if (sub process) {
            sub_svr(client_addr);
            close(client_Fd);
            return;
        }
        close(client_fd);      //main process
    }
    close(socket_fd);
    return;
}
```

```
void sub_svr(struct sockaddr_in   client_addr) {
    makeCacheDir(cacheDir);

    read(client_fd, buf, BUFFSIZE);
    strcpy(tmp, buf);
    getOURL(tmp, oURL);
    sha1_hash(oURL, hURL);
    getCachePath(hURL, cacheDir, dirName, fileName);

    hit = isHit(dirName, hURL);
    if(oURL is webserver address) {
        printf("*PID# is waiting for the semaphore₩n");
        p(semid);
        printf("*PID# is in the critical zone₩n");
        writeLogfile(hit, oURL, hURL);
        sleep(5);      // to check semaphore
        v(semid);
```

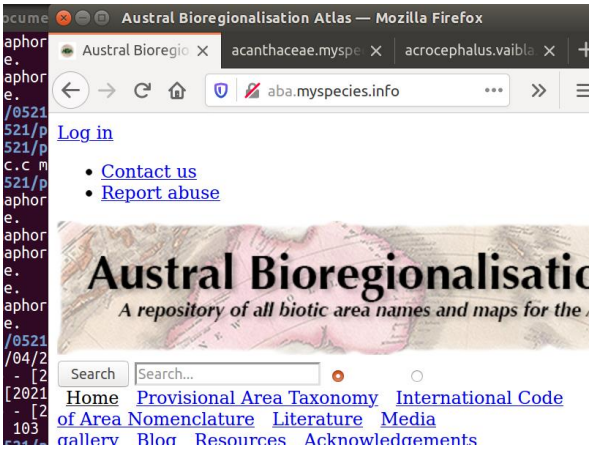
```
    }

    if(hit) {
        cacheFile = open(fileName, "r");
        while((read(cacheFile, buf, BUFFSIZE))>0)
            write(client_fd, buf, len);
        close(cacheFile);
    }
    else {
        alarm(10);
        socket_fd = wserver_requset(buf, oURL);
        cacheFile = open(fileName, "w");
        while((len = read(socket_fd, buf, sizeof(buf)){
            alarm(0);
            write(client_fd, buf, len);
            write(cacheFile, buf, len);
            buf = 0;
        }
        close(socket_fd);
        close(cacheFile);
    }
    close(client_fd);
    return;
}
```

4. 결과화면 //수행한 내용을 캡쳐 및 설명

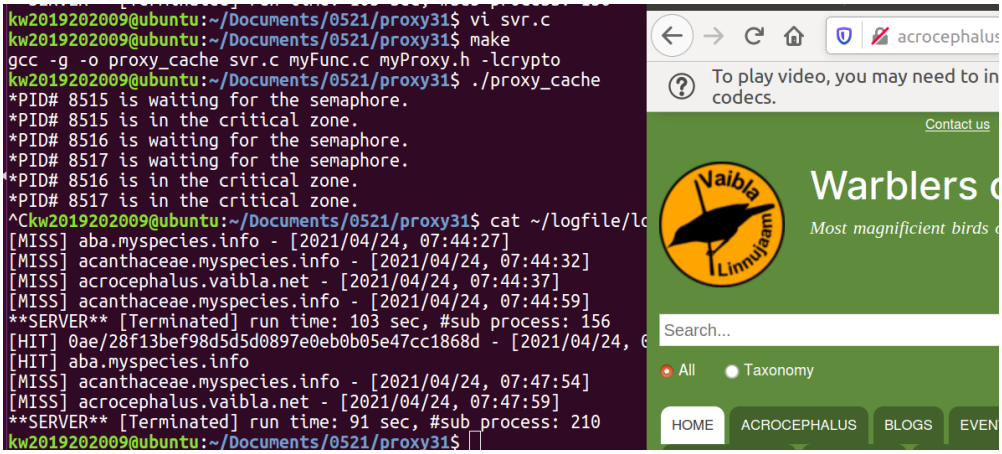
firefox에서 여러 개의 탭을 열어 첫 번째 자식 프로세스가 critical section의 sleep에서 벗어나기 전에 여러 개의 자식 프로세스를 생성한다. 터미널 출력 결과를 확인한다.



3개의 사이트에 한 번에 접속하고, 웹 브라우저의 화면이 갱신된 이후에 두 번째 사이트에 추가로 접속한 결과 위와 같이 출력되었다. 로그 파일에서 처음 3개 번의 요청에 대해 sleep으로 지정한 5초 간격으로 쓰기가 이루어졌음을 확인할 수 있다. 이 실험에서 네 번째 요청이 miss인 이유는 두 번째 요청이 response message를 받지 못하고 timeout되었기 때문이다.



페이지의 로딩이 느려지거나, timeout되는 경우가 발생하였다.



alarm의 설정 시간을 늘려 3개의 페이지를 불러오는 실행을 했을 때, 모든 페이지가 정상적으로 출력되었다.

5. 결론 및 고찰

이번 과제는 semaphore를 이용하여 sub process간의 공유데이터인 logfile을 기록하는 부분을 critical section으로 지정하는 과제였다. 실습강의 자료를 통해 과제를 구현하는 것에 큰 어려움은 없었다.

그러나 위의 실험 결과에서 alarm의 시간이 10초일 때, 시간 내에 response message를 받아오지 못해서 timeout되는 상황이 발생하였다. 이전 과제를 진행하며 한 번에 하나의 request에 대해서 webserver에 접속한 반면, 이번 과제에서는 여러 개의 request를 동시에 처리하는 상황을 가정하며 새로운 문제를 발견할 수 있었다. 동시에 여러 process에서 request 혹은 response message를 웹서버와 주고받기 때문에 각각의 속도가 느려져서 발생한 문제인 것 같다. 이 문제는 alarm의 시간을 늘리는 것으로 해결할 수 있었다.