시스템프로그래밍실습 보고서 #3-2

실험제목: Synchronize Shared Resource

제출일자: 2015년 06월 12일 (금)

학 과: 컴퓨터공학과

담당교수: 박철수 교수님

실습분반: 목요일 3, 4

학 번: 2010720019

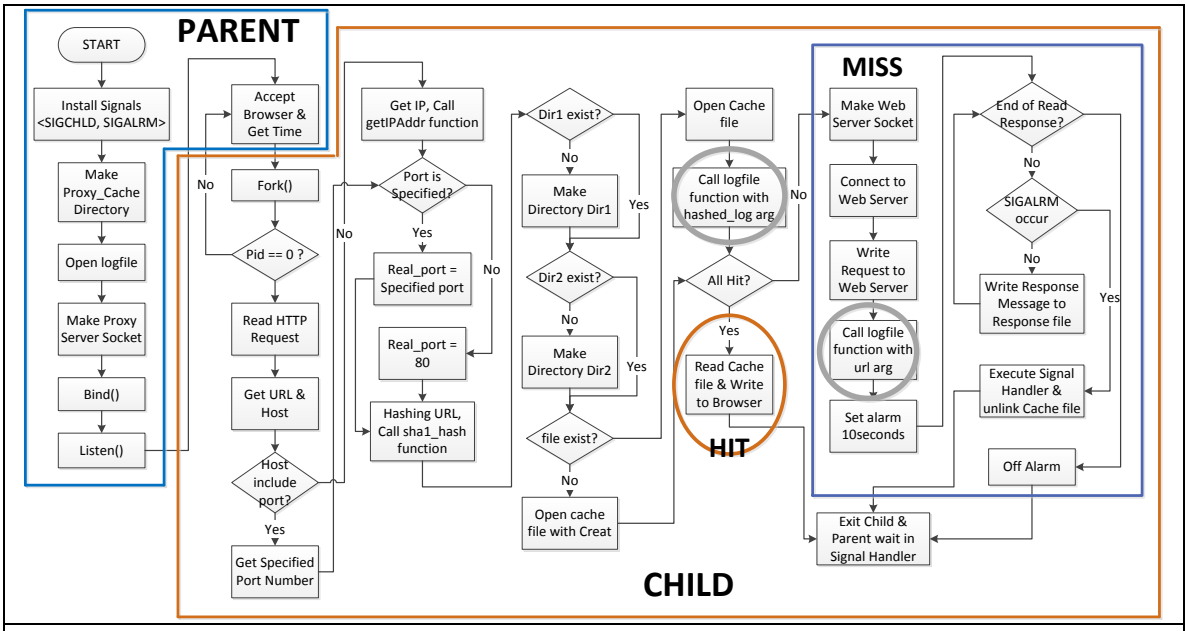
성 명: 박완배

1. Introduction

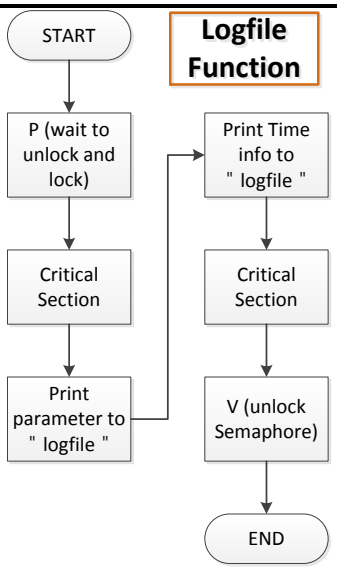
A. 과제 소개

이번 실습은 Semaphore에 대한 이해를 하고 적용하는 실습에 해당한다. Proxy Server의 기능 중, Request된 URL에 대해서 Hit인 경우와 Miss인 경우에 따라서 Hashed URL 혹은 URL을 logfile에 시간과 함께 기록을 하는데 각 fork된 Process되는 접근 시간이 겹치게 된다. 따라서 이러한 Shared Resource에 접근을 동기화 하기 위해서 Semaphore를 이용한다.

2. Algorithm



Proxy Server Algorithm



Logfile Function Algorithm with Semaphore

Main Proxy Server Algorithm에서 변경된 부분은 Logfile을 바로 열어서 기록 하는 것이 아닌,

Logfile 함수를 호출해서 Semaphore에 의해서 하나의 Process만 기록을 하도록 하였다. 변경된 부분을 동그라미로 표시하였다. 또 한, Logfile function에서는 semid와 log할 값을 받아와서, P 연산으로 unlock이 될 때 까지 기다리다가, unlock이 되면, Critical Section으로 들어가서 해당 Code를 수행한다. Code는 Logfile에 기록을 하는 것이다. 이 후, V 연산으로 lock을 unlock으로 바꾸면서 종료하면, 다른 Process가 P 연산을 하면서 기다리다가 들어 가게 된다.

3. Pseudo code

• Proxy main

```
////////////Global Variables////////////
int browser_fd, web_fd, response_fd;     //전역변수 선언을 통해서, signal handler함수에서 child종료시, 같이 종료 될 수 있게 해준다.
char* file;    //to unlink file in signal handler
time_t now;  //시간 정보를 logfile(Shared Resource)에 넘겨주기 위해서 전역변수로 선언
int main()
{

   set union semun arg;  //semun union 변수 선언 내부에 int val, struct semid_ds* buf, int* array가 포함된다.
   semget(40077,1 semaphore, CREAT);  //새로운 Semaphore를 만드는데, 1개 이며, key값은 자신의 Port번호 40077이다.
   arg's meber variable val would be 1;
   semctl(semid with arg to SETVAL);    //semaphore를 control하는데, flag를 SETVAL로 주면서, semnum을 arg의 member 변수 val의 값으로 설정

   install SIGALRM to act myhandler function;    //SIGALRM과 SIGCHLD에 대해서 signal발생시 동작 install
   install SIGCHILD to act myhandler function;

   getHome Directory;
   Make proxy_cache pathname;

   use opendir and find proxy_cache file;
   if(Not find proxy_cache)
      make directory of proxy_cache pathname in home directory;

   file open with make logfile with concantanous write;
   close logfile;
```

```
proxy_fd=socket(address_family, TCP); //Make Socket & Get Socket Descripter of Server;

proxysocket.family = AF_INET; //Address family
proxysocket.address = host to network byte order(128.134.52.60); //host order byte를 Network order byte로 변환
proxysocket.port = host to network byte order(Portnumber); //Set port number of mine (40077)

bind(proxy_fd, proxysocket, sizeof(proxysocket)); //만든 Socket에 Address를 연결시킨다.
listen(proxy_fd, integer); //accept할 준비가 되었으며, queue 양은 integer만큼 설정

while(1)
{       ///////Parent는 fork후에 Accept로 돌아와서 Block된다.
    browser_fd = accept(proxy_fd, browsersocket, length of browsersocket); //client(Browser)의 연결요청을 accept한다.

    Get Current Time;

    if((pid=fork())<0)    //Make Child Process
        fail to fork;
    else if(pid==0)    //Child Process
    {
        memset('buf' to be zero);    //buf array를 0으로 초기화 해서 사용
        host = NULL;    //hostname정보를 가리킬 host pointer가 NULL을 가리키도록 한다.
        memset('request' to be zero);    //request array를 0으로 초기화해서 사용
        sprint(proc, getpid() to decimal);    //Process ID를 Terminal에 Write하기위해서 변환

        read from browser, buf, Bufsize; //browser의 Request를 읽어온다.
        string copy(request, buf);  //request를 복사
        Get URL and Store in variable;
        get host & IPaddress(getIPAddr function) // host를 구하고 getIPAddr function을 이용해서 IP를 구한다.
        if(port number is include in Host)
            token port number and store in variable;

        write(Request to Terminal) //Request 정보를 Terminal에 출력
        write(host&IP to Terminal) //host와 IP를 Terminal에 출력
```

```
        if(Not Specified Port number)
            Real Portnumber = 80;
        else
            Real Portnumber = stringtoshort(Specified Port number);  //문자열숫자를 십진수 숫자로 변환 Unsigned short 형태로 변환

        Hashing URL call sha1_hash function;
        copy Hashed URL to use log;

        'dir1' = make pathname of 'proxy_cache/X' formant, 'X' mean 0~f 16진수
        readdir(opendir(home));
        for(readdir(opendir(home)) ; NULL ; readdir(opendir(home)))
        {
            find same file name with dir1's file name;
            if(exist)
            {
                Hit++;
                break;
            }
        }
        if(not Exist)
          mkdir('dir1', with drwxrwxrwx permission);
        closedir(opendir(home));

        opendir(dir1);
        'dir2' = make pathname of 'proxy_cache/'dir1'/X' formant, 'X' mean 0~f 16진수
        readdir(opendir(dir1));
        for(readdir(opendir(dir1)) ; NULL ; readdir(opendir(dir1)))
        {
          find same file name with dir2's file name;
          if(exist)
          {
            Hit++;
            break;
          }
        }
        if(not Exist)
          mkdir('dir2', with drwxrwxrwx permission);
        closedir(opendir(dir1));
```

```
opendir(dir2);
'file' = make pathname of 'proxy_cache/'dir1'/'dir2'/XXX... formant, 'XXX....' mean 0~f
16진수 38글자
readdir(opendir(dir2));
for(readdir(opendir(dir2)) ; NULL ; readdir(opendir(dir2)))
{
    find same file name with 'file"s file name;
    if(exist)
    {
      Hit++;
      break;
    }
}
if(not Exist)
  cache=open('file', with drwxrwxrwx permission);
else
{
  cache=open(file, Read Write only);
  //logfile 함수 호출에, semaphore ID와 출력할 Hashed_url을 넘겨준다. (Hit Case)
  call logfile function with argument, semid & hashed_log;

}
closedir(opendir(dir2));

if(Hit Case)
{
    while(read cache file is exist)
    {
        write to Browser from read cache;
        memory set the buffer;
    }
}
else      //Miss Case
{
    web_fd = socket(address_family, TCP);   //web server와 통신할 socket을 만든다.
    web_address = getIPAddr & RealPort(80 port(HTTP) or Specified Port);
    connect(web_fd, web_address );   //Web server에 연결 요청
    umask set 0;
```

```
            write(Request to Web Server);    //Request를 Web Server로 전송

            //logfile 함수 호출에, semaphore ID와 출력할 URL을 넘겨준다. (Miss Case)
            call logfile function with argument, semid & url;

            alarm to 10seconds;    //Alarm 시간을 10초로 설정
            while(read(received from Web Server to buf)>0)
            {
                write(buf value to browser client);
                write(buf value to cache file);
                if(write error in cache file)
                    unlink the cache file;    //remove와 같은 의미, 제대로 된 file이 아니기 때문에 삭제 한다.
                memory set buf to zero;
            }
            alarm delete;    //10초내에 Response를 받은 경우 Alarm을 해제 한다.

            close(Browser);   //Brower의 socket을 닫는다.
            close(web);   //Web Server socket을 닫는다.
            close(Cache);   //response file descripter를 닫는다.
            exit Child;   //Child를 종료한다. Generate SIGCHLD
        }
    }

    semctl(delete Semaphore with IPC_RMID flag);   //semaphore를 삭제한다.
    close(proxy);   //Proxy Server의 socket을 닫는다.
    return 0;
}
```

• sha1_hash

```
char* sha1_hash(char* input_url)
{
    Call SHA1 function with parameter;
    //hasing is input_url, and the length of hasing(input_url), 20byte size character array
    for(0 to until 20byte size array is done)
      sprint(hashed_hex[2byte] = hashed_160bits[1byte]를 2자리 hexadecimal로 바꾼값)

    //hased된 값에 디렉토리 경로마다 '/'를 삽입하는 과정 (수정됨)
    dynamic allocation hashed_url with size is '43';
```

```
    memory set hashed_url;
    insert hashed_hex[0] and / to hashed_url;
    insert hashed_hex[1] and / to hashed_url;
    insert rest of hashed_hex to hashed_url;

    return hashed_url;
}
```

• getHomeDir

```
char* getHomeDir()
{
    declare struct passwd type pointer;
    get passwd user id with getuid() parameter;
    return member variable 'pw_dir';
}
```

• getIPAddr

```
char* getIPAddr(char* addr)
{
    struct hostent* hent;   //hostent struct type의 변수
    char* haddr;    //host address 즉, IP를 가리킬 포인터

    hent = gethostbyname(addr);   //host name을 인자로 넣어서 해당 정보의 구조체를 저장
    haddr = inet_ntoa(hent->h_addr_list[0]);   //구조체 내의 IP정보를 inet_ntoa를 이용해서 32bit Address를 Dotted Decimal로 변환
    return haddr;   //host IP Address를 반환
}
```

• myhandler

```
void myhandler(int signo)
{
    switch(signo)
    {
        case SIGCHLD :    //Child가 종료될 때 발생하는 Signal
            pid would be wait(0) return;    //종료되는 Child에 대해서 wait을 통해 Resource 환원
            print PID Child END!;    //해당 Pid Child의 종료를 출력해준다.
            break;
        case SIGALRM :    //Alarm을 설정한 시간이 다되면 발생하는 Signal
            print No Response;
            close(Browser_fd, Web_fd, Cache_fd);    //사용한 file or socket Descriptor close
            unlink Cache file;    //Cache file 을 삭제한다.
```

```
            exit child;     //child 종료
            break;
    }
}
```

• stringtoshort

```
unsigned short stringtoshort(char* port)
{
    Get port's string length;
    Calculate digit by length;
    invert character to decimal by Minus 48 which ASCII value;
    return result;
}
```

• P operation

```
void v(int semid)
{
      declare sembuf struct;
      member variable sem_num would be zero;
      member variable sem_op would be -1;       //V operation sem_op값은 -1이다.
      member variable sem_flg would be SEM_UNDO;    //프로세스 종료시, semaphore 제거를 특별하게 하지 않아도 자동으로 해제 가능
      semop(do semaphore operation of struct of one operation);
}
```

• V operation

```
void p(int semid)
{
      declare sembuf struct;
      member variable sem_num would be zero;
      member variable sem_op would be 1;    //P operation sem_op값은 1이다.
      member variable sem_flg would be SEM_UNDO;
      semop(do semaphore operation of struct of one operation);


}
```

• logfile

```
void logfile(int semid, char* log)
{
      p operation for semaphore ID;     //wait until unlock semaphore than lock
      //////Critical Section
      open file name is "logfile" with a+ option;     //logfile에 argument로 받아온 log와 시간정보 출력
```

```
        print argument log to "logfile";
        transform time struct;
        print time information to "logfile";
        close file;
        //////Critical Section
        v operation for semaphore ID;     //turn into unlock
}
```

4. Result screen capture

• logfile

```
2010720019@sslab-splab-15:~$ ls
a.out  examples.desktop  logfile  proxy  proxy_cache  proxy_server.c
2010720019@sslab-splab-15:~$ cat logfile
========================================
3418 process is using logfile
http://www.naver.com/-[2015/6/12, 21:36:33]
3418 process is returning logfile
========================================
3419 process is using logfile
http://img.naver.net/static/newsstand/up/2014/0715/366.gif-[2015/6/12, 21:36:33]
3419 process is returning logfile
========================================
3421 process is using logfile
http://nv1.ad.naver.com/adshow?unit=002AN&nrefreshx=1-[2015/6/12, 21:36:33]
3421 process is returning logfile
========================================
3423 process is using logfile
http://nv.ad.naver.com/adshow?unit=002AT&da_dom_id=ad_timesquare-[2015/6/12, 21:36:33]
3423 process is returning logfile
========================================
3422 process is using logfile
http://my.naver.com/?20150612213608-[2015/6/12, 21:36:33]
3422 process is returning logfile
========================================
========================================
3427 process is using logfile
http://nv2.ad.naver.com/adshow?unit=002AP&nrefreshx=1-[2015/6/12, 21:36:33]
3427 process is returning logfile
========================================
3428 process is using logfile
http://castbox.shopping.naver.com/shopAdBox.nhn-[2015/6/12, 21:36:33]
3428 process is returning logfile
========================================
3429 process is using logfile
http://nv1.adcreative.naver.net/ad3/1087/1087173/20150609021921-QXF2KLkk.png-[2015/6/12, 21:36:33]
3429 process is returning logfile
========================================
3430 process is using logfile
http://nv2.adcreative.naver.net/ad3/1090/1090627/20150612064920-REZkGo5l.swf-[2015/6/12, 21:36:34]
3430 process is returning logfile
========================================
3463 process is using logfile
8/1/1521d171e1066027bbe67c1ca3f2e08e41c8d4-[2015/6/12, 21:36:43]
3463 process is returning logfile
========================================
3464 process is using logfile
5/d/3b6d74544a6be5b3db2627b96fb986e1ab8d10-[2015/6/12, 21:36:43]
3464 process is returning logfile
========================================
```

```
3468 process is using logfile
6/5/4a1a11df238f4c025340aec6c0ccdca4d2c531-[2015/6/12, 21:36:44]
3468 process is returning logfile
========================================
3469 process is using logfile
c/a/13f2ed944e3c345753f076788bce3cc2407c10-[2015/6/12, 21:36:44]
3469 process is returning logfile
========================================
3470 process is using logfile
2/6/30f45a36831bdc5fcffd508228997beeb6b8da-[2015/6/12, 21:36:44]
3470 process is returning logfile
========================================
3471 process is using logfile
3/0/20ce3bbe6897fe6522bc977219931530bcc2b9-[2015/6/12, 21:36:44]
3471 process is returning logfile
========================================
3472 process is using logfile
7/8/2ac117232cae78ed5ea71fa980d351dc2bbd94-[2015/6/12, 21:36:44]
3472 process is returning logfile
========================================
3473 process is using logfile
http://mtag19.midas-i.com/mrps-tag?cp=2519&me=1405&sp=23564&an=55984&tp=imj-[2015/6/12, 21:36:44]
3473 process is returning logfile
========================================
3474 process is using logfile
http://www.naver.com/include/newsstand/press_info.json-[2015/6/12, 21:36:44]
3474 process is returning logfile
2010720019@sslab-splab-15:~$
```

위의 결과 화면은, www.naver.com URL에 대한 Request를 2번 실행했을 때의 logfile의 화면이다. 처음 HTTP Request를 보냈을 때, logfile에 URL의 정보와 시간이 출력되며, 밑에 부분에는 Hit으로 인해, Hashed URL이 시간과 출력되는 것을 볼 수 있다. 또한, ProcessID process is using logfile or returning logfile과 같은 부분으로 Critical Section이 나타나게끔 출력을 해서 결과 화면을 출력 하였다. 하나의 임계구역의 시작과 끝에서 다른 Process ID는 해당 Code를 실행하지 못한다는 것을 알 수 있다. 해당 출력은 결과화면에서 설명을 하기 위해서 나타낸 것이기 때문에, 실제 Code를 작성해서 제출할 때는 주석으로 처리하였다.

5. Reference

• Advanced Programming in the UNIX Environment/ 리처드 스티븐스, 스티븐 레이고 지음

6. Conclusion

이번 실습은 Shared Resource에 대해서, Semaphore를 이용하여 Synchronize를 하는 실습이었다. 각 각의 HTTP Request에 대해서 응답을 하고, logfile에 해당 URL을 쓰거나, Hashed URL을 써서, Miss인지 Hit인지 판단하며, 해당 HTTP Request가 언제 입력되었는지, 시간정보와 출력을 해야 하는데, 각 HTTP Request는 Fork를 통해서 Child Process들이 일을 수행하므로, 언제 어떤 HTTP Request에 대한 작업이 먼저 종료 될 지 알 수 없다. 따라서, Logfile에 접근을 할 때, 여러 Process가 접근하는 경우가 생기게 되므로 Logfile이 바로, Shared Resource인 것이다. 이러한 Shared Resource에 대한 접근을 하나의 Process로만 제한하기 위해서 Semaphore를 사용해서 문제를 해결하는데, 이를 Synchronized되었다고 표현을 한다. Semaphore는 크게 4가지로 구성되어 있는데, 먼저, Semaphore를 만드는 연산이 존재한다. Semaphore를 만들기 위해서는 Key 값이 필요하며, Semaphore의 개수도 설정해 주어야 한다.

이후, Shared Resource에 대해서 각 process들은 일을 처리해야 하는데, 이 Shared Resource에 접근하기 전에, P연산을 한다. P연산은 Shared Resource에 대한 code 즉, Critical Section을 수행하기 전에, 이미 사용하는 Process가 있다면, P연산에 의해서 block된다. 이는 Semaphore가 Unlock될 때까지 P연산을 수행하는데, 이전에 사용하던 Process가 Critical Section을 나와서, V연산으로 Semaphore를 Unlock 하면, 다른 Process가 Critical Section으로 들어가면서 lock을 하는 것이다. 그렇게 되면, 다른 Process는 해당 Critical Section을 실행하지 못하고 다시 P 연산에서 block이 되면서 기다린다. 이러한 Semaphore방법을 통해서 Shared Resource인 Logfile에 접근하는 Process를 제한하는데 우리는 1개의 Process만 접근하도록 설정한 것이다. 이제까지 처음 학기를 시작하면서부터, Proxy Cache, Proxy Server 그리고 Semaphore를 이용한 logfile Synchronize까지 실습을 진행했는데, 모두 개념적으로 이해는 했지만, Code를 직접 작성해서 구현하기까지는 그리 쉽지 않았다. 또한, Unix 환경에서의 Coding과 debugging은 이전 2학년 때까지 해온 visual studio에서와 같이 간단하지 않았고, 무엇보다 System Call을 이용한다는 차이점도 있었다. Proxy Server에 대한 개념을 배우고 이를 직접 구현하면서, Proxy Server동작 원리를 배우고, Cache에는 어떻게 저장되는지, Client와 Server는 어떻게 HTTP Request에 대한 HTTP Response를 하는지, 하나의 Shared Resource 접근을 동기화 하는지에 대해서 많은 것을 배우는 의미 있는 실습이었다.