
System Programming

Assignment#1-3 Multiple Processing



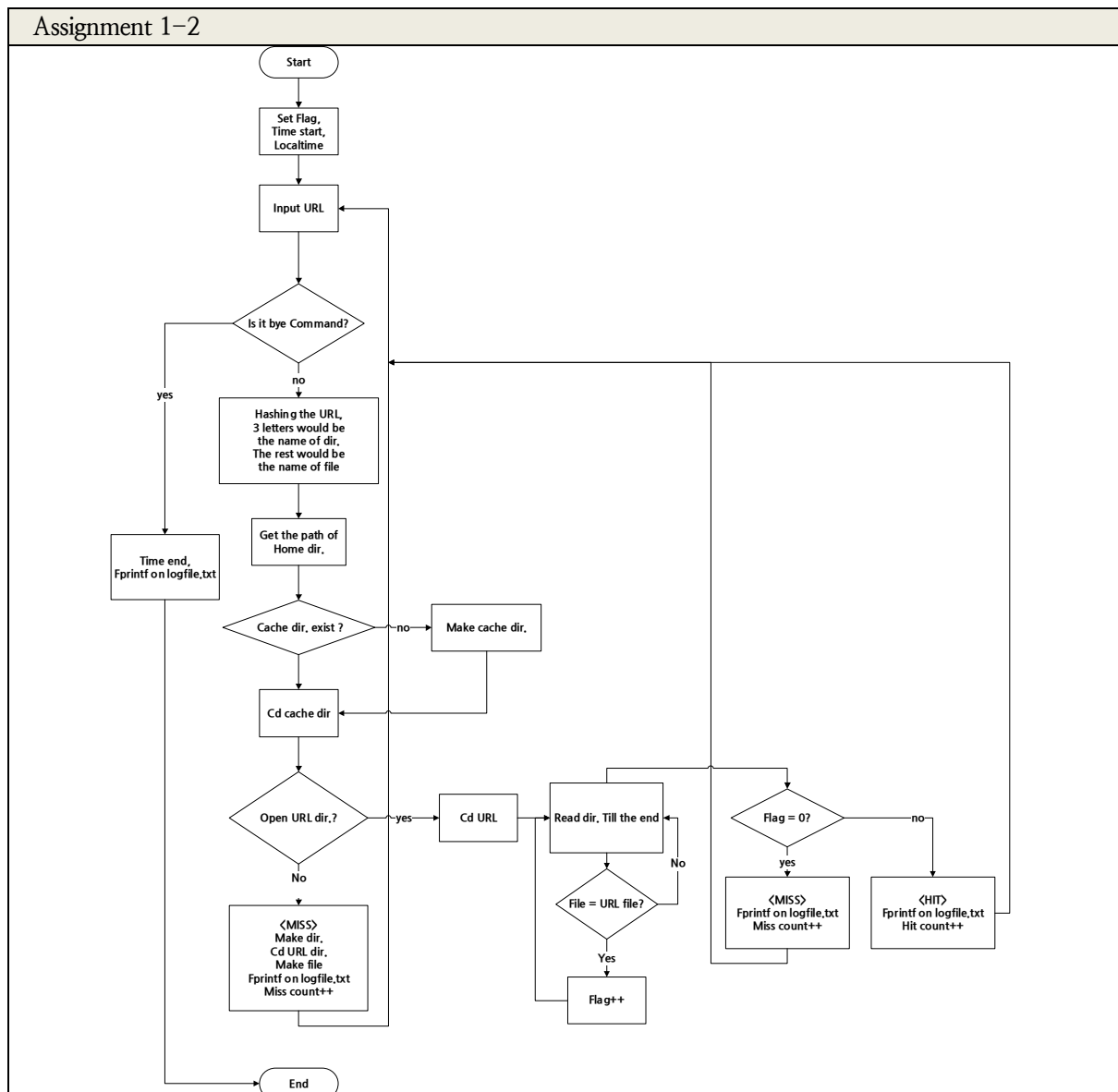
| | |
|------------|---------------|
| Professor | 목 3 4 황호영 교수님 |
| Department | 컴퓨터공학과 |
| Student ID | 2012722028 |
| Name | 장 한 별 |
| Date | 2018. 04. 13 |

Introduction.

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

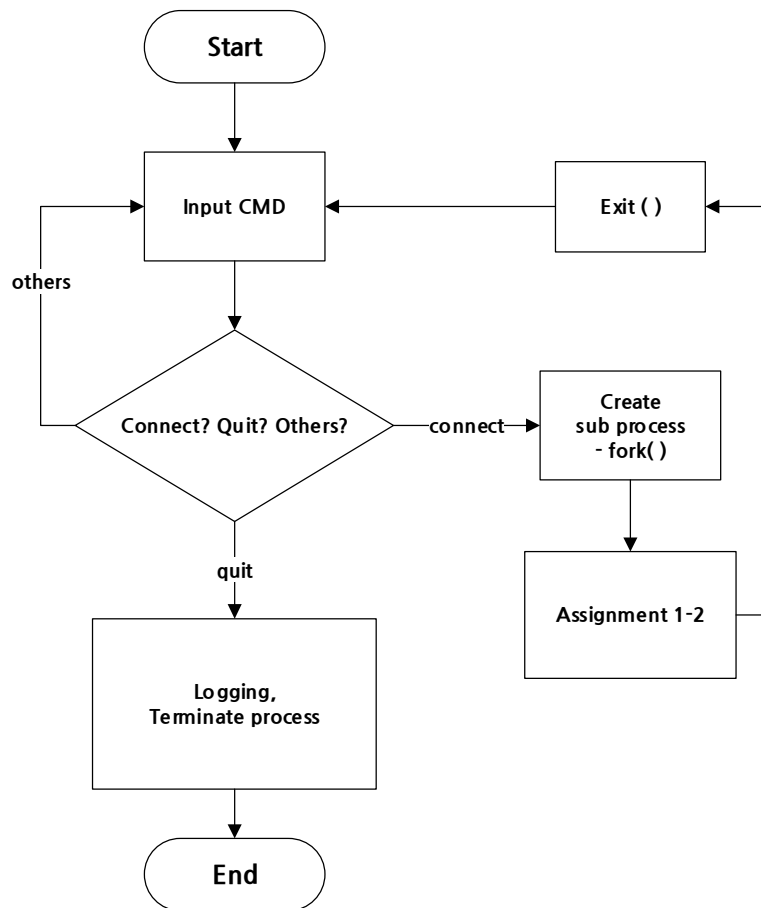
이번 과제는 새로운 프로세스를 생성하고 관리하는 프로그램을 구현하는데 목적이 있다. 'connect'를 입력 시, 새로운 프로세스(Sub process)를 생성(fork 함수 이용)하고, 생성된 프로세스에서 Assignment 1-2 까지 구현한 내용이 동작된다. 'quit'을 입력 시 프로세스를 종료한다. 프로세스와 생성과 관리하는 부분에 있어 어떤 함수를 어떻게 응용해야 하는지 명확하게 파악해야한다.

Flow chart.



위 그림은 지난 과제 전체의 Flow chart 이다. 이번 과제에서 fork() 함수 호출 시, sub process 가 생성되고, 그 sub process 에서 진행되는 flow chart 이다. 편의를 위해 위 과정을 Assignment 1-2 라고 하겠다.

Main



위 그림은 이번 과제의 Main 함수에 대한 Flow chart 이다. CMD 를 입력 받아 'connect'시 함수 fork() 를 이용해 sub process(child process)를 생성한다. 이 새로 생성된 process 에서는 Assignment 1-2 의 작업이 이루어지고 'bye' 명령어를 입력 시 그 process 가 종료되는 exit ()를 호출 시켜 종료 시킨다. 이 과정에 기존에 있던 process(parent process)는 wait 상태이고, 'quit' 을 입력 시, process 를 완전히 종료시킨다. (logfile.txt 에 작업내용을 형식에 맞게 logging 후) 물론 connect 는 한 작업 시 여러 번 입력할 수 있으며, 'connect' 나 'quit'이 아닌 다른 명령어나 문장이 입력될 경우 메시지와 함께 다시 CMD를 입력하도록 구현했다.

Pseudo code.

Assignment 1-2

```
Int main(int argc, char* argc[])
{
    Time start;
    While(1)
    {
        Set Flag;
        Localtime;

        Input URL;
        If(input == "bye")
        {
            Time end;
            Fprintf(logfile, "Terminated", runtime, hit, miss);
            break;
        }
        Hashing;
        String copy(directory name, hashed URL, 3);
        String copy(file name, heshed URL +3);
        Make directory (cache);
        Cd dir.(cache);

        If(directory name==NULL)
        {
            Make directory(Hashed URL);
            Change directory(Hashed URL);
            Make file(Hashed URL);
            Fprintf(logfile, "Miss", input, year/mon/mday/hour/min/sec);
            Miss count++;
        }
        Else
        {
            Cd dir.(cache);
            While( read dir. )
            {
                If(same filename , hashed filename)
                {
                    Flag++;
                }
            }
            If( flag==0)
            {
                Fprintf(logfile, "Miss", input, year/mon/mday/hour/min/sec);
                Miss count++;
            }
            Else
            {
                Fprintf(logfile, "Hit", dir/file, year/mon/mday/hour/min/sec);
                Hit count++;
            }
        }
    }
}
```

위 그림은 지난 과제의 pseudo code 이다 이번 과제에서 fork() 함수 호출 시, sub process 가 생성되고, 그 sub process 에서 진행되는 pseudo code 이다. 편의를 위해 위 과정을 Assignment 1-2 라고 하겠다.

main

```

Int main ()
{
    While(1)
    {
        Printf("input CMD");
        Scanf("%s", cmd);

        If( cmd == "quit")
        {
            Fprintf(logfile);
            Exit();
        }
        Else if( cmd == "connect")
        {
            Fork()
            If(pid = 0)
                Assignment 1-2 ;
            Exit();
            Else if( pid >0)
                Wait();
        }
        Else
        {
            Printf("input connect or quit");
        }
    }
}

```

위 그림은 이번 과제의 Pseudo code 이다. CMD 를 입력 받아 'connect'시 함수 fork() 를 이용해 sub process(child process)를 생성한다. 이 새로 생성된 process 에서는 Assignment 1-2 의 작업이 이루어지고 'bye' 명령어를 입력 시 그 process 가 종료되는 exit ()를 호출 시켜 종료 시킨다. 이 과정에 기존에 있던 process(parent process)는 wait 상태이고, 'quit' 을 입력 시, process 를 완전히 종료시킨다. (logfile.txt 에 작업내용을 형식에 맞게 logging 후) 물론 connect 는 한 작업 시 여러 번 입력할 수 있으며, 'connect' 나 'quit'이 아닌 다른 명령어나 문장이 입력될 경우 메시지와 함께 다시 CMD를 입력하도록 구현했다.

Result.

1-3.(1)

```
hanbyeol@ubuntu:~/proxy_server$ ./proxy_cache
[2674]input CMD> connect
[2675]input URL> a
[2675]input URL> b
[2675]input URL> c
[2675]input URL> www.naver.com
[2675]input URL> r
[2675]input URL> stu
[2675]input URL> a
[2675]input URL> bye
[2674]input CMD> quit
hanbyeol@ubuntu:~/proxy_server$
```

위 그림은 이번 과제를 구현한 코드를 make 후 정상적으로 실행 시킨 화면이다. CMD 에 connect 를 입력하면 fork() 함수를 호출하여 child process 가 생성되고, 앞에 출력되는 pid 가 그렇게 생성된 child process 의 pid 값으로 바뀐 것을 확인할 수 있다. Assignment 1-2 과정을 수행한 후 exit() 로 child process 를 종료시켰더니 wait 상태였던 parent process 로 돌아와 기존 process 의 pid로 바뀐 것을 확인할 수 있다. quit 입력시켜 process(parent process)를 종료시킨다.

1-3.(2)

```
hanbyeol@ubuntu:~/proxy_server$ tree ~/cache
/home/hanbyeol/.cache
├── 4dc
│   └── 50fc3bc007be011b5445f3f79298b9eeb51b7
│       └── 7c9ec434ed06502767136789763ec11d2c4b7
├── 54f
│   └── 516841ba77a5b4648de2cd0dfcb30ea46dbb4
├── 86f
│   └── 7e437faa5a7fce15d1ddcb9eaeaea377667b8
├── 89d
│   └── 71f5ee7c92d6dc9e92ffdad17b8bd49418f98
└── fcd
    └── 818da7395e30442b1dcf45c9b6669d1c0ff6b
```

위 명령어를 입력해 cache directory에 입력한 URL 이 정상적으로 hash 되어 directory 와 file 들을 만들어 저장되어있음을 확인할 수 있다.

1-3.(3)

```
hanbyeol@ubuntu:~/proxy_server$ cat ~/logfile/logfile.txt
[Miss]a-[2018/4/13, 04:39:27]
[Miss]b-[2018/4/13, 04:39:28]
[Miss]c-[2018/4/13, 04:39:28]
[Miss]www.naver.com-[2018/4/13, 04:39:29]
[Miss]r-[2018/4/13, 04:39:31]
[Miss]stu-[2018/4/13, 04:39:31]
[Hit]86f/7e437faa5a7fce15d1ddcb9eaeaea377667b8-[2018/4/13, 04:39:34]
[Hit]a
[Terminated] run time: 10 sec. #request hit: 1. miss: 6
**SERVER** [Terminated] run time: 27 sec. #sub process: 1
```

1-3.(1) 에서 quit 으로 process 를 종료시, logfile.txt 를 생성하는데 위 화면은 그 txt 파일을 출력한 화면이다. Parent process 의 총 run time, 생성된 sub process 의 개수를 확인할 수 있다. 올바르게 출력되었음을 확인할 수 있다.

1-3.(4)

```
hanbyeol@ubuntu:~/proxy_server$ ./proxy_cache
[2690]input CMD> connect
[2691]input URL> a
[2691]input URL> b
[2691]input URL> c
[2691]input URL> d
[2691]input URL> bye
[2690]input CMD> connect
[2692]input URL> www.naver.com
[2692]input URL> www.google.com
[2692]input URL> www.kw.ac.kr
[2692]input URL> bye
[2690]input CMD> connect
[2693]input URL> www.kw.ac.kr
[2693]input URL> bye
[2690]input CMD> connect
[2694]input URL> bye
[2690]input CMD> quit
```

위 그림은 1-3.(1)과 달리 한 번의 작업 수행 안에 connect 를 여러 번 입력시킨 결과 화면이다. 새로 생성된 child process 의 수행 종료 후 exit () 함수로 종료 시켜 parent process 로 돌아와 다시 connect 시켜 또 다른 child process를 생성, 종료를 반복한다. 몇 번을 반복해도 다시 parent process 로 돌아온다는 것을 pid 를 확인하면 정확하게 알 수 있다. 마지막 부분은 connect 한 후 다른 URL 을 입력시키지 않고 오직 bye 만 입력 했을 때이다. 이 내용은 1-3.(6)에서 다루도록 한다.

1-3.(5)

```
hanbyeol@ubuntu:~/proxy_server$ tree ~/cache
/home/hanbyeol/cache
├── 3c3
│   └── 63836cf4e1666669a25da280a1865c2d2874
├── 4dc
│   └── 50fc3bc007be011b5445f3f79298b9eeb51b7
│       └── 7c9ec434ed06502767136789763ec11d2c4b7
├── 44a
│   └── 516841ba77a5b4648de2cd0dfcb30ea46dbb4
├── 46f
│   └── 7e437faa5a7fce15d1ddcb9eaeaea377667b8
├── 48b
│   └── 99f68b208b5453b391cb0c6c3d6a9824f3c3a
├── 48b
│   └── 0f293fe62e97369e4b716bb3e78fababf8f90
├── 49d
│   └── 71f5ee7c92d6dc9e92ffdad17b8bd49418f98
└── 7c3
    └── 818da7395e30442b1dcf45c9b6669d1c0ff6b


8 directories, 9 files
```

위 명령어를 입력시켜 역시 directory 와 file 들이 올바르게 생성되었음을 확인할 수 있다.

1-3.(6)

```
hanbyeol@ubuntu:~/proxy_server$ cat ~/logfile/logfile.txt
[Miss]a-[2018/4/13, 04:39:27]
[Miss]b-[2018/4/13, 04:39:28]
[Miss]c-[2018/4/13, 04:39:28]
[Miss]www.naver.com-[2018/4/13, 04:39:29]
[Miss]r-[2018/4/13, 04:39:31]
[Miss]stu-[2018/4/13, 04:39:31]
[Hit]86f/7e437faa5a7fce15d1ddcb9eaeaea377667b8-[2018/4/13, 04:39:34]
[Hit]a
[Terminated] run time: 10 sec. #request hit: 1, miss: 6
**SERVER** [Terminated] run time: 27 sec. #sub process: 1
```


| | | |
|--|--|--|
| | <pre> [Hit]86f/7e437faa5a7fce15d1ddcb9eaaea377667b8-[2018/4/13, 04:41:07] [Hit]a [Hit]e9d/71f5ee7c92d6dc9e92ffdad17b8bd49418f98-[2018/4/13, 04:41:08] [Hit]b [Hit]84a/516841ba77a5b4648de2cd0dfcb30ea46dbb4-[2018/4/13, 04:41:09] [Hit]c [Miss]d-[2018/4/13, 04:41:10] [Terminated] run time: 10 sec. #request hit: 3, miss: 1 [Hit]fed/818da7395e30442b1dcf45c9b6669d1c0ff6b-[2018/4/13, 04:41:19] [Hit]www.naver.com [Miss]www.google.com-[2018/4/13, 04:41:20] [Miss]www.kw.ac.kr-[2018/4/13, 04:41:25] [Terminated] run time: 12 sec. #request hit: 1, miss: 2 [Hit]e00/0f293fe62e97369e4b716bb3e78fababf8f90-[2018/4/13, 04:41:35] [Hit]www.kw.ac.kr [Terminated] run time: 14 sec. #request hit: 1, miss: 0 [Terminated] run time: 4 sec. #request hit: 0, miss: 0 **SERVER** [Terminated] run time: 57 sec. #sub process: 4 hanbyeol@ubuntu:~/proxy_server\$ </pre> | |
| | <p>위 화면은 1-3.(4) 가 수행된 후 저장된 logfile.txt 를 출력한 결과 화면이다. Connect 를 총 4번 입력 시켜 4개의 각각 다른 child process 를 순서대로 생성하고 종료했음을 확인할 수 있다. 1-3.(4) 작업 중 마지막 부분에서 connet 과정이 새로운 process 를 생성하는 것(fork)이므로, connect 후 bye 만 입력 시킨 후 quit 을 입력해도 sub process 가 생성되고 종료되는 과정을 거쳤기 때문에 총 4개의 sub process 를 생성했다는 것을 알 수 있다. 위 그림을 통해 올바르게 출력되었음을 확인 할 수 있다.</p> | |

| | |
|---------|--|
| 1-3.(7) | |
| |  A terminal window with a dark background. The prompt is 'hanbyeol@ubuntu:~/proxy_server\$'. The command './proxy_cache' has been executed. The output shows '[2715]input CMD> abcd' followed by 'please input \'connect\' or \'quit\'' on the next line. The output text is highlighted with a red rectangular box. |
| | <p>만약 input CMD 에 connect 나 quit 명령어가 아닌 다른 명령어나 문장을 입력시켰을 경우, 'please input 'connect' or 'quit' ' 를 출력하고, 다시 CMD 를 입력 받도록 구현했다.</p> |

Conclusion.

이번 과제는 여러 개의 process 를 생성하고, 종료(관리) 하는 프로그램을 구현했다. Process 를 생성할 때는 fork() 함수를, process 를 종료 시킬 때는 exit() 함수를 호출 시켜 과제 format 에 적절히 응용하는 것이 이번 과제의 핵심이었다.

이번 과제는 지난 Assignment 1-1, 1-2 와는 달리 전반적인 process 에 대한 이해도가 많이 요구되어 시작 부분에서 바로 감이 오지않았다. 특히 fork() 함수 의 대표적인 특징인 한 번 호출에 2개의 return 값이 반환된다는 점인데 하나는 0, 그리고 다른 하나는 child process 의 process id 이다. 이 부분이 잘 이해가 가지않았고, child process 를 처리 했음에도 다른 return 값을 처리해 주지 못해 많은 에러를 발생시켰다.

Exit() 함수를 호출 시켜 child process를 종료시킬 때 parent process 는 child process 실행되고 종료 될 때 까지 기다려야한다. 이때의 함수는 wait() 이며, waitpid() 는 pid 를 지정할 수 있고, 옵션 값을 넣어 줄 수 있어 wait() 함수 보다 더 구체적인 활용이 가능하다. Pid 는 child process 의 pid 를 받아와 입력시켰다. 그렇게 child process 가 종료될 때 까지 wait 할 수 있는 코드를 구현할 수 있었다.