

# System Programming

## ( Assignment4-1 )

|      |                  |
|------|------------------|
| 과 목  | 시스템프로그래밍실습       |
| 담당교수 | 이기훈 교수님          |
| 학 과  | 컴퓨터공학과           |
| 학 번  | 2010720149       |
| 성 명  | 이동현              |
| 날 짜  | 2016. 05. 27 (금) |

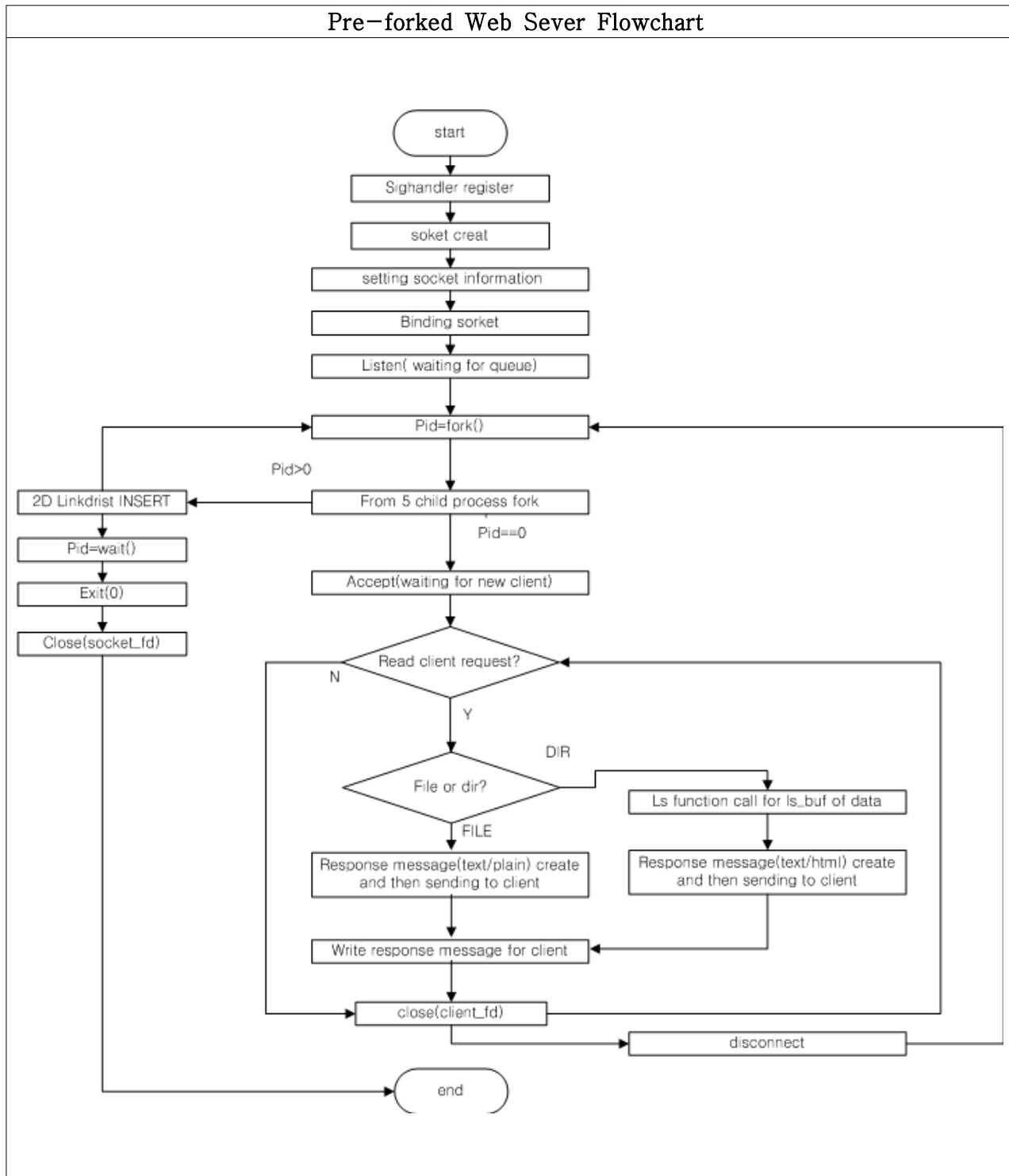


## A. Introduction

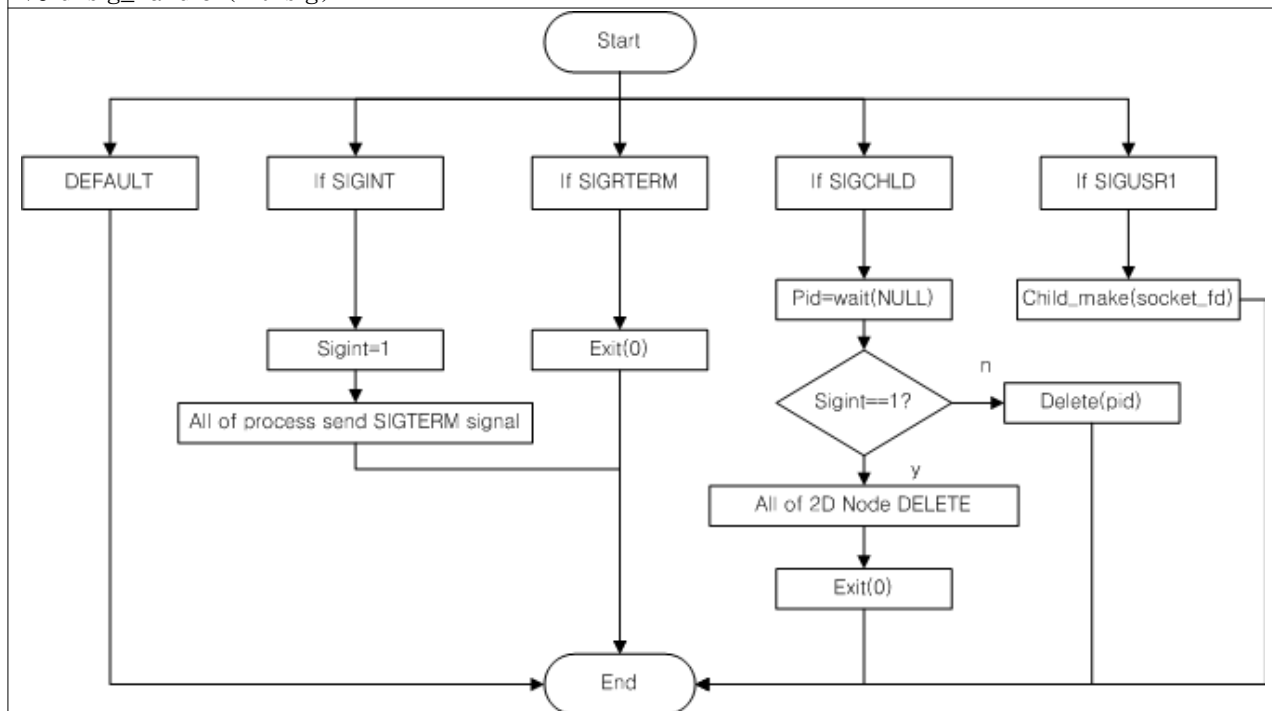
### ♣ Pre-forked Web Sever ♣

이번 과제는 정해진 개수의 child process를 복제하여, 각 process가 server와 연결할 수 있도록 각각 기능을 수행함으로써 서비스가 가능하다. 이것은 request 도착 전에 미리 만들어놓은 프로세스로 도착후에 프로세스를 fork하는 낭비를 줄일 수 있다.

## B. Flowchart



```
void sig_handler(int sig)
```



### C. Pseudo code

#### Pre-forked Web Sever Pseudo code

```

int main(){
    signal(SIGCHLD, sig_handler);
    signal(SIGINT, sig_handler);
    signal(SIGTERM, sig_handler);
    signal(SIGUSR1, sig_handler);
    initializing buf, arv
    time struct
    socket_fd <- socket(PF_INET, SOCK_STREAM, 0)
    opt <- 1;
    bzero((char*)&server_addr, sizeof(server_addr));
    server_addr.sin_family <- AF_INET;
    server_addr.sin_addr.s_addr <- htonl(INADDR_ANY);
    server_addr.sin_port <- htons(PORTNO);
    setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
    bind(socket_fd, (struct sockaddr*)&server_addr, sizeof(server_addr));
    listen(socket_fd, 5);
    maxNchildren <- 5;
    for(i=0; i<maxNchildren; i++){
        child_make(socket_fd);
        sleep(1);
    }
    pause
}

int child_make(int sockfd){
    pid=fork();

```

```

    if(pid>0)
        insert to 2D
        return 0
    else if(pid==0)
        child_main(socketfd)
}
void child_main(int sockfd){
    while(1){
        initializing buf, response, argv
        len <- sizeof(client_addr)
        client_fd <- accept(sockfd, (struct sockaddr*)&client_addr, &len);
        cur_time_time[idx]<-current time set
        client_num++
        fp<- fopen("accessible usr", "r")
        while(!feof(fp))
            fscanf(fp, "%s", ip_buf)
            if(fnmatch(ip_buf, inet_ntoa(client_addr.sin_addr), 0)==0)
                flag<-1;
        if flag==0
            No access Response Message create and write to client
            continue;
        while(len_out <- read(client_fd, buf, BUFFSIZE)>0){
            argc <-2;
            Dflag, Fflag <-0;
            if(Dflag==1)
                ls(argc, argv, ls_buf);
                strcpy(response, "HTTP/1.1 200 OK\r\nAccept-Ranges: bytes\r\nConnection:
close\r\n");
                strcat(response, "Content-Length: 100000\r\n");
                strcat(response, "Content-Type: text/html\r\n");
                strcat(response, "\r\n");
                strcat(response, ls_buf);
            else if(Fflag==1)
                strcpy(response, "HTTP/1.1 200 OK\r\nAccept-Ranges: bytes\r\nConnection:
close\r\n");
                strcat(response, "Content-Length: 100000\r\n");
                strcat(response, "Content-Type: text/plain\r\n");
                strcat(response, "\r\n");
                strcat(response, ls_buf);
            write(client_fd, response, 100000);
            initializing buf, response
        }
        allocation delete for argv
        kill(getppid(), SIGUSR1);
    }
}

```

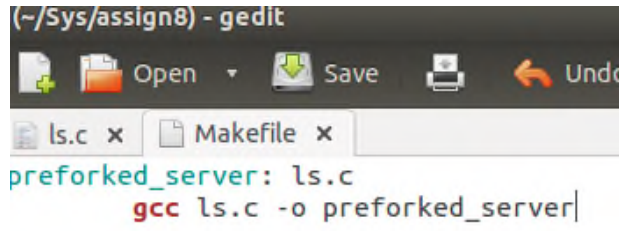
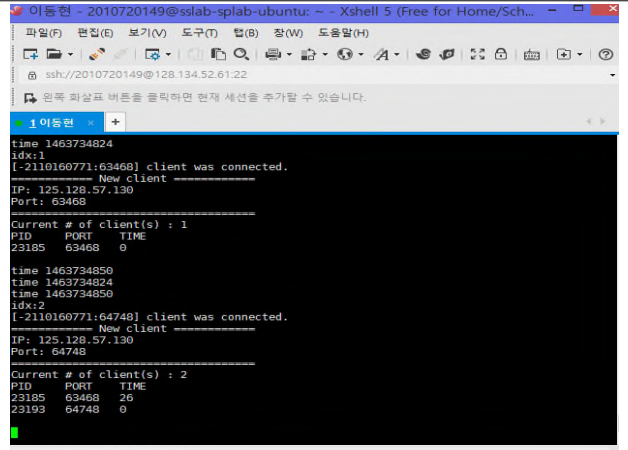
```
        close(socket_fd);
        exit(0);
    }
}

void sig_handler(int sig)
{
    if sig == SIGINT
        sigint=1;
        all of process send SIGTERM signal
    else if sig== SIGTERM
        exit(0)
    else if sig == SIGCHLD
        pid <- wait(NULL)
        if(sigint == 1)
            all of 2D DELETE
            exit(0)
        else
            delete(pid);
    else if sig == SIGUSR1
        child_make(socket_fd);
    else
        another signal exe
        return;
}
```

## D. Reference

- 강의자료 ' 2016-1\_SPLab\_12\_Pre-fork+Web+Server\_v2

# E. Conclusion

| ♣ 조건 ♣  | Makefile  |
|---|---|
| <ul style="list-style-type: none"> <li>- 포트는 #define으로 구현</li> <li>- pre-forked 방식으로 server 구현</li> <li>- signal programming 기법 사용</li> <li>- parent process에서 child process를 관리</li> <li>- 터미널에 간단한 log 기록 출력</li> </ul> |   |
| My port number  | My shell  |
| 40051   |  |
| my ifconfig ip and port   |   |
| http://192.168.111.131:40051/   |   |

## ♣ 결과 화면 1 ♣

```

dong@ubuntu:~/Sys/assign8$ ./preforked_server
[Thu May 26 11:08:01 2016] Server is started.
[Thu May 26 11:08:01 2016] Socket is created. Ip: 127.0.1.1, Port: 29596
[Thu May 26 11:08:01 2016] 9282 process is forked.
[Thu May 26 11:08:02 2016] 9283 process is forked.
[Thu May 26 11:08:03 2016] 9284 process is forked.
[Thu May 26 11:08:04 2016] 9285 process is forked.
[Thu May 26 11:08:05 2016] 9286 process is forked.

===== New client =====
[Thu May 26 11:08:41 2016]
IP: 192.168.111.1
Port: 30665
PID: 9282

=====
===== Disconnected client =====
[Thu May 26 11:08:57 2016]
IP: 192.168.111.1
Port: 30665
PID: 9282

=====
[Thu May 26 11:08:57 2016] 9282 process is terminated.
[Thu May 26 11:08:57 2016] 9288 process is forked.

[Thu May 26 11:09:25 2016] 9288 process is terminated.
[Thu May 26 11:09:25 2016] 9286 process is terminated.
[Thu May 26 11:09:25 2016] 9285 process is terminated.
[Thu May 26 11:09:25 2016] 9284 process is terminated.
[Thu May 26 11:09:25 2016] 9283 process is terminated.
[Thu May 26 11:09:25 2016] server is terminated.
dong@ubuntu:~/Sys/assign8$

```

실행파일 실행 결과 Server가 시작이되고 Socket을 생성한 후 Server ip와 port 값을 확인 할 수 있으며 5개의 process가 순서대로 fork가 되었음을 알 수 있다

클라이언트 요청 후에 새로 클라이언트 연결이 되고 종료 시 해제가 되었음을 확인 할 수 있다 클라이언트 종료 시 해당 프로세스는 종료가 되고 새로운 프로세스가 fork가 되어이한다 새로운 할당된 프로세스는 링크드리스의 phead로 추가된다

^c 컨트롤+C 입력시 모두 종료하게 되는데 phead부터 종료가 되며 순서에 맞게 종료가 되고 출력되었음을 확인 할 수 있다

## ♣ 결과 화면 2 ♣

```

dong@ubuntu:~/Sys/assign8$ ./pretorked_server
[Thu May 26 11:13:08 2016] Server is started.
[Thu May 26 11:13:08 2016] Socket is created. Ip: 127.0.1.1, Port: 29596
[Thu May 26 11:13:08 2016] 9292 process is forked.
[Thu May 26 11:13:09 2016] 9293 process is forked.
[Thu May 26 11:13:10 2016] 9294 process is forked.
[Thu May 26 11:13:11 2016] 9295 process is forked.
[Thu May 26 11:13:12 2016] 9296 process is forked.
===== New client =====
[Thu May 26 11:14:05 2016]
IP: 192.168.111.1
Port: 53705
PID: 9292
=====
===== New client =====
[Thu May 26 11:14:10 2016]
IP: 192.168.111.1
Port: 54729
PID: 9293
=====
===== New client =====
[Thu May 26 11:14:12 2016]
IP: 192.168.111.1
Port: 55497
PID: 9294
=====
===== New client =====
[Thu May 26 11:14:14 2016]
IP: 192.168.111.1
Port: 56777
PID: 9295
=====

```

```

===== New client =====
[Thu May 26 11:14:16 2016]
IP: 192.168.111.1
Port: 57033
PID: 9296
=====
===== Disconnected client =====
[Thu May 26 11:14:27 2016]
IP: 192.168.111.1
Port: 53705
PID: 9292
=====
[Thu May 26 11:14:27 2016] 9298 process is forked.
[Thu May 26 11:14:27 2016] 9292 process is terminated.
===== Disconnected client =====
[Thu May 26 11:14:31 2016]
IP: 192.168.111.1
Port: 54729
PID: 9293
=====
[Thu May 26 11:14:31 2016] 9293 process is terminated.
[Thu May 26 11:14:31 2016] 9299 process is forked.
^C
[Thu May 26 11:14:32 2016] 9294 process is terminated.
[Thu May 26 11:14:32 2016] 9299 process is terminated.
[Thu May 26 11:14:32 2016] 9298 process is terminated.
[Thu May 26 11:14:32 2016] 9296 process is terminated.
[Thu May 26 11:14:32 2016] 9295 process is terminated.
[Thu May 26 11:14:32 2016] server is terminated.
dong@ubuntu:~/Sys/assign8$

```

위의 결과는 결과 화면 1의 설명과 큰 차이는 없다. 다만, 미리 생성된 5개의 프로세스에 각각 모두 순서대로 client가 연결되었음을 확인 할 수 있고, 클라이언트 연결을 두 개를 끊은 후 9292와 9293의 프로세스가 종료되었음을 확인 할 수 있고, 새로운 프로세스가 fork가 된다. 컨트롤+C를 통해 링크드리스트의 pHead부터 모든 프로세스를 종료시킨다. 종료 시 9298과 9299의 프로세스가 추가되었음을 확인 할 수 있다.

## ♣ 고찰 ♣

이번 과제는 저번 과제를 충분히 이해했다면 어렵지 않는 과제였다. 단순히 미리 정해둔 개수의 child 프로세스를 생성하여, 클라이언트 연결에 수행할 수 있도록 하는 것이 핵심이다. 굳이 이 유가 궁금했는데, 이것은 클라이언트 request시 fork하여 생성하는 낭비를 줄일 수 있는 역할을 할 수 있다고 한다. 내 생각엔, 과제에서 정해둔 서버요청만 가능하도록 해서 그렇지만 실제 서버에 접속하는 클라이언트의 수는 예측할 수 없기 때문에, 배운 부분에서까지는 이전 과제처럼 요청 시 fork하여 생성을 하는게 맞는 것이 아닌가 생각이 든다.

코드를 구현하면서 추가적인 부분은 fork시 생성되는 프로세스의 terminal에 확인을 하기 위해 sleep 함수를 통해 시간차로 인터페이스에 맞게 출력을 하였고, connect와 disconnect시 해당 프로세스를 확인하기 위해 pid의 id도 같이 출력하도록 하였다.

이번과제는 급작스럽게 과제의 내용이 수정이 되면서 시간은 부족해졌지만, 더욱 쉬워진 난이도가 되게 되었다. 수정되기 전 parent와 child간의 fork의 특성상 공유자원을 할 수 없기 때문에 서로 데이터를 주고 받기에 쉽지 않았다. sighandler로 최대한 해보려고했지만, 한계가 존재했고 마침 과제가 수정이 되어 쉽게 과제를 이어나갈 수 있었다. 수정된 과제는 서버 종료 시 발생하는 signal들과 sighandler의 signal flow 완벽히 이해한다면 매우 쉽게 구현이 가능했다. parent는 child를 종료 시 까지 기다리고 있고 child process 정보를 링크드리스트로 관리하고 있고 child는 프로세스를 복제하여 클라이언트와 연결 통신을 한다. 갑작스런 서버 종료 시(컨트롤+c), sigint가 발생을 하고 그 것은 parent와 child에게 신호를 보내주는데, child process는 무시를 하고, parent는 그 것을 인지하고 child process를 모두 종료시키기위하여 생성 된 자식 프로세스를 sigterm을 신호를 보내 모두 종료하도록 한다. 그러면 sigchild 신호가 발생하게 되고, 자식 프로세스를 정리한 후 parent process를 종료하도록 하여야만 좀비 프로세스가 발생하지 않고 정상적인 종료를 마치게 된다. 이 flow와 맞게 sighandler를 잘 구성한다면, 어렵지 않게 할 수 있었다. 그리고 server ip는 server\_addr 구조체에서 가져오지 못하므로 HOST의 정보를 얻는 함수 gethostbyname의 함수를 이용을 해야하면, 인자가 다소 복잡하기 때문에 잘 확인해 볼 필요가 있다.

앞으로 두 번의 과제가 남았는데, 다음 과제는 세마포어랑 스레드에 관하여 공유자원을 서로 공유하며, 겹쳐서 사용하지 않도록하여 매우 구성도가 높은 과제를 구현할 예정이다. 어렵과 난관에 봉착할 것 같지만, 이와 같이 이해를 중심으로 코드를 구현하여 나가도록 공부를 더 해야할 필요가 있다고 생각이 든다.