

# System Programming

## ( Assignment4-3 )

과 목	시스템프로그래밍실습
담당교수	이기훈 교수님
학 과	컴퓨터공학과
학 번	2010720149
성 명	이동현
날 짜	2016. 06. 10 (금)

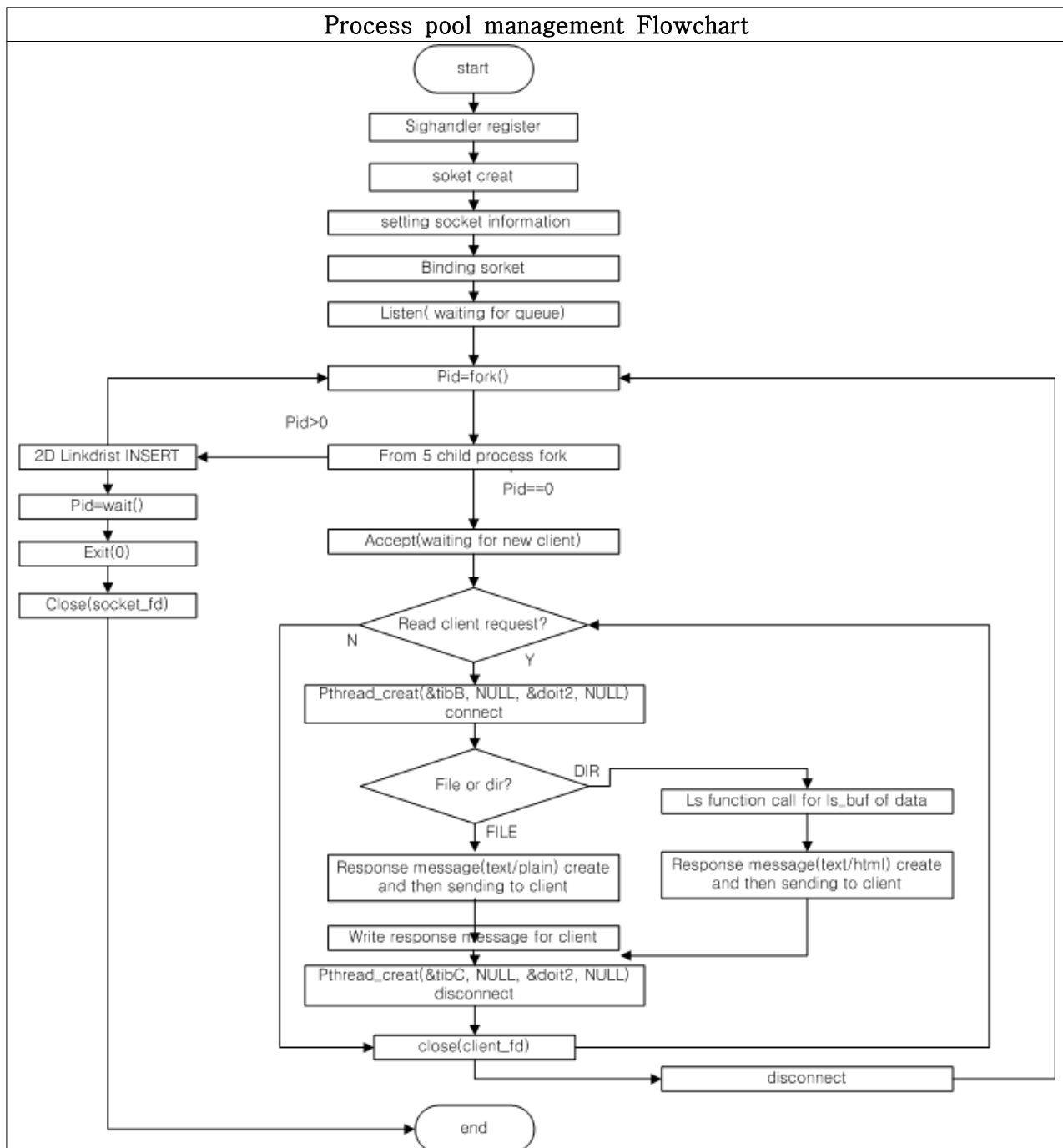


## A. Introduction

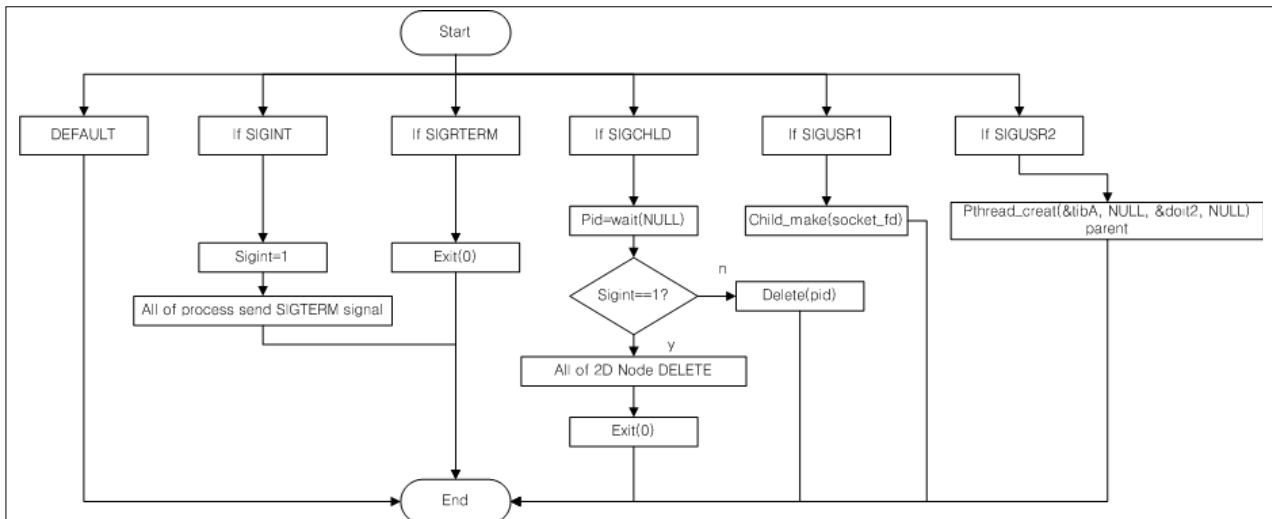
### ♣ Mutual Exclusion ♣

이번 과제는 ipc통신을 통해 프로세스간의 통신이 이루어지는 thread를 이용한 shared memory를 이용을 하였고, race-condition을 피하기 위해 log file 기록을 semaphore의 방법으로 해결을 하여 주어진 환경 변수에 맞게 process의 상태를 관리하여 서버와 클라이언트 통신이 이루어 질 수 있도록 한다.

## B. Flowchart



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



### C. Pseudo code

#### Process pool management 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, 10);
    maxNchildren <- 5;
    for(i=0; i<maxNchildren; i++){
        IdleServerCount++;
        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 <- sizeofI(client_addr)
        client_fd <- accept(sockfd, (struct sockaddr*)&client_addr, &len);
        pthread_create(&tidB, NULL, &doit2, NULL);
        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
        }
        pthread_join(tidB, NULL);
        close(client_fd);
        pthread_create(&tidC, NULL, &doit3, NULL);
        kill(getppid(), SIGUSR2);
        pthread_join(tidC, NULL);
    }
}

```

```
    }  
}  
  
void *doit1(void *vptr)  
{  
    if shm_id = shmget((key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)) == -1  
        error  
    if shm_addr = shmat( shm_id, (void*)0, 0))== (void*)-1)  
        error  
    wait(mysem)  
    sleep(1);  
    if strstr((char*)shm_addr, ",1")!=NULL  
        IdleServerCount--;  
        strcpy(prev, (char*)shm_addr);  
        token<--strtok(prev, ",");  
        child_pid<--atoi(token);  
        func status_switch(child_pid, 1);  
        if MinSpareServers>IdleServerCount  
            if MaxChild>ProcessNumber  
                if IdleServerCount==MinSpareServers  
                    kill(getpid(), SIGUSR1);  
  
        printf<--"[s] IdleServerCount: %d\n", t_buf, IdleServerCount;  
  
    if strstr((char*)shm_addr, ",0")!=NULL  
        status_switch(child_pid, 0);  
        IdleServerCount++;  
        printf<--"[s] IdleServerCount: %d\n", t_buf, IdleServerCount;  
        If MaxSpareServers<IdleServerCount){  
            if IdleServerCount==MaxSpareServers)  
                kill(child_pid, SIGTERM);  
    post(mysem)  
    sleep(1);  
    return NULL;  
}  
  
void *doit2(void *vptr)  
{  
    if shm_id = shmget((key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)) == -1  
        error  
    if shm_addr = shmat( shm_id, (void*)0, 0))== (void*)-1)  
        error  
    wait(mysem)  
    sprintf<--(char*)shm_addr, "%d,%d", getpid(), 1;  
    post(mysem)  
    sleep(1);
```

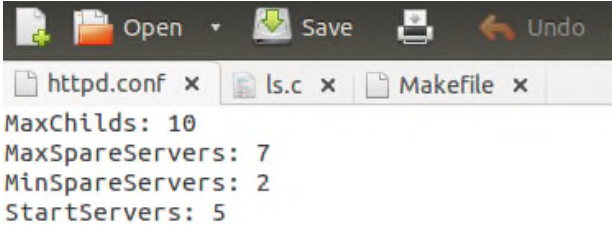
```
    return NULL;
}
void *doit3(void *vptr)
{
    if shm_id = shmget((key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)) == -1
        error
    if shm_addr = shmat( shm_id, (void*)0, 0))== (void*)-1)
        error
    wait(mysem)
    sprintf<-(char*)shm_addr, "%d,%d", getpid(), ;
    post(mysem)
    sleep(1);
    return NULL;
}
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 if sig == SIGUSR2
        sleep(1);
        pthread_create(&tidA, NULL, &doit1, NULL);

    else
        another signal exe
        return;
}
```

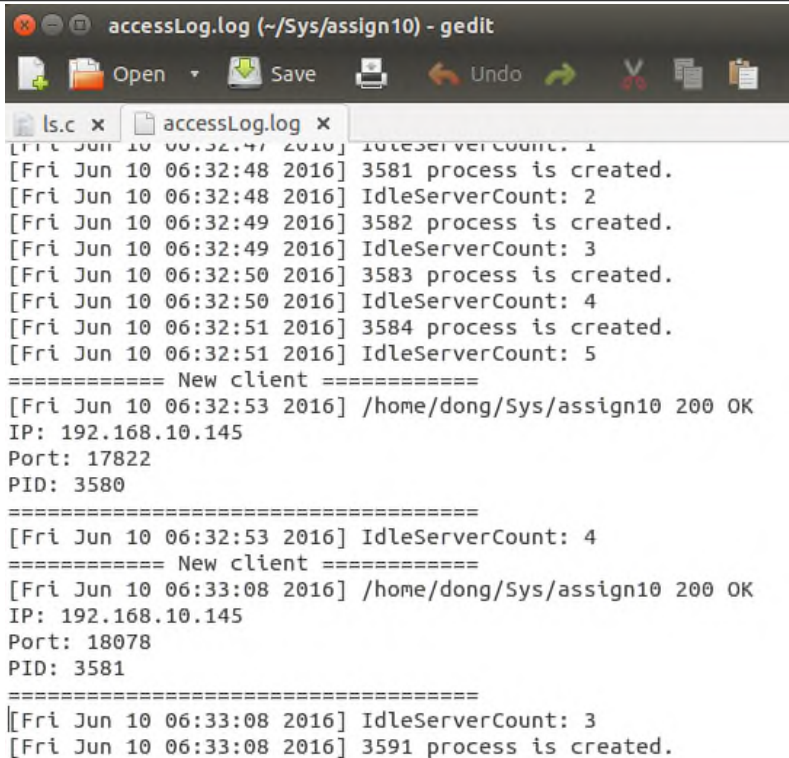
## D. Reference

- 강의자료 ' 2016-1\_SPLab\_14\_log\_file\_v2

## E. Conclusion

Makefile	
	
My port number	httpd.conf
40051	

## ♣ 결과 화면 1 ♣



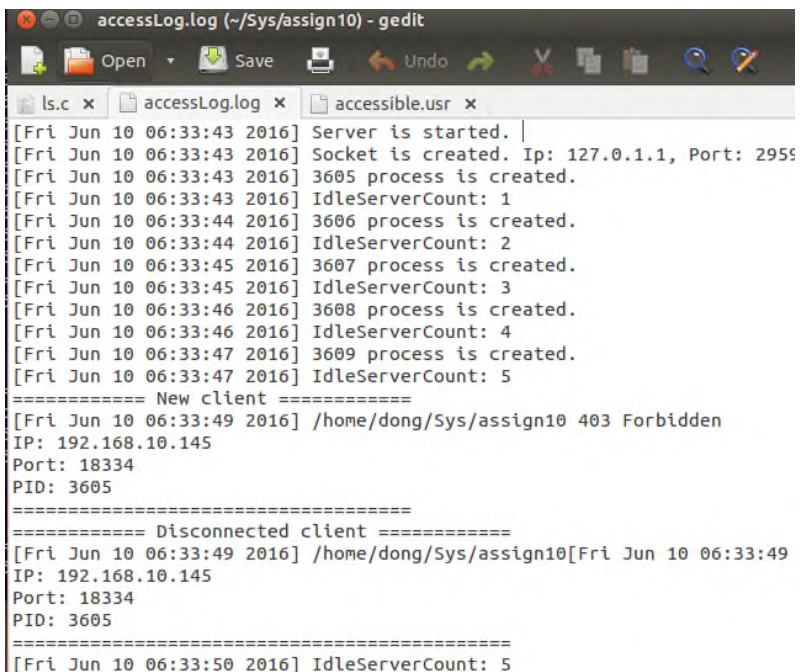
```

accessLog.log (~/.Sys/assign10) - gedit
ls.c x accessLog.log x
[Fri Jun 10 06:32:47 2016] IdleServerCount: 1
[Fri Jun 10 06:32:48 2016] 3581 process is created.
[Fri Jun 10 06:32:48 2016] IdleServerCount: 2
[Fri Jun 10 06:32:49 2016] 3582 process is created.
[Fri Jun 10 06:32:49 2016] IdleServerCount: 3
[Fri Jun 10 06:32:50 2016] 3583 process is created.
[Fri Jun 10 06:32:50 2016] IdleServerCount: 4
[Fri Jun 10 06:32:51 2016] 3584 process is created.
[Fri Jun 10 06:32:51 2016] IdleServerCount: 5
===== New client =====
[Fri Jun 10 06:32:53 2016] /home/dong/Sys/assign10 200 OK
IP: 192.168.10.145
Port: 17822
PID: 3580
=====
[Fri Jun 10 06:32:53 2016] IdleServerCount: 4
===== New client =====
[Fri Jun 10 06:33:08 2016] /home/dong/Sys/assign10 200 OK
IP: 192.168.10.145
Port: 18078
PID: 3581
=====
[Fri Jun 10 06:33:08 2016] IdleServerCount: 3
[Fri Jun 10 06:33:08 2016] 3591 process is created.

```

다음은 accessLog.log에 기록한 사항이다.  
터미널 콘솔창에 쓰여진 것과 동일 하다.  
이 경우는 access가 가능한 경우이다.

## ♣ 결과 화면 2 ♣



```

accessLog.log (~/.Sys/assign10) - gedit
ls.c x accessLog.log x accessible.usr x
[Fri Jun 10 06:33:43 2016] Server is started. |
[Fri Jun 10 06:33:43 2016] Socket is created. Ip: 127.0.1.1, Port: 2955
[Fri Jun 10 06:33:43 2016] 3605 process is created.
[Fri Jun 10 06:33:43 2016] IdleServerCount: 1
[Fri Jun 10 06:33:44 2016] 3606 process is created.
[Fri Jun 10 06:33:44 2016] IdleServerCount: 2
[Fri Jun 10 06:33:45 2016] 3607 process is created.
[Fri Jun 10 06:33:45 2016] IdleServerCount: 3
[Fri Jun 10 06:33:46 2016] 3608 process is created.
[Fri Jun 10 06:33:46 2016] IdleServerCount: 4
[Fri Jun 10 06:33:47 2016] 3609 process is created.
[Fri Jun 10 06:33:47 2016] IdleServerCount: 5
===== New client =====
[Fri Jun 10 06:33:49 2016] /home/dong/Sys/assign10 403 Forbidden
IP: 192.168.10.145
Port: 18334
PID: 3605
=====
===== Disconnected client =====
[Fri Jun 10 06:33:49 2016] /home/dong/Sys/assign10[Fri Jun 10 06:33:49
IP: 192.168.10.145
Port: 18334
PID: 3605
=====
[Fri Jun 10 06:33:50 2016] IdleServerCount: 5

```

다음은 accessLog.log에 기록한 사항이다.  
터미널 콘솔창에 쓰여진 것과 동일 하다.  
이 경우는 access가 불가능 한 경우이다.



## ♣ 고찰 ♣

이번 과제는 저번 과제에 이어서 스레드와 세마포어 기법을 통한 race condition을 피하기 위하여 공유메모리를 이용해서 프로세스간의 통신이 가능하도록 구현하는 server program을 구현하였다. 그 사이에 log파일에 기록을 할 때에는 세마포어 내에서 race condition을 피해야한다. 우선 생소한 개념을 이해하기위해 이 키워드들의 중점으로 공부를 오랜 시간을 투자하였다. 쓰는 api 문법은 강의 자료대로 나와 있지만, 각 키워드들의 기능이 무엇이고, 과제가 무엇을 요구하는지를 파악하고 구현하기 위해선 반드시 공부는 필요했다. 시험공부를 하면서 큰 도움이 되었다. 우선 스레드는 간단히 말해서 비교하자면 프로세스가 사람이 행동에서처럼 손과 발이라고 비유할 수 있겠다. 각 프로세스는 단일 스레드를 가지고 있고 이것을 수행함으로써 프로세스의 기능하는데, 스레드를 추가 생성하면 멀티 스레드로 한 프로세스 내에서 동시 수행이 가능하도록 한다. 한 프로세스의 내에서 스레드는 스택을 제외한 나머지 자원을 공유하는 큰 특징을 갖고 있다. 이것은 fork()로 통한 프로세스의 생성과 매우 상반된 특징을 갖는다. fork()를 통한 프로세스를 생성하면 자식프로세스는 부모프로세스의 자원을 그대로 복제를 하지만, 그 이후 자원을 공유 할 수 없기 때문에 서로간의 독립적인 행동을 하게 된다. log 파일에 기록할 때 세마포어를 이용한다. 공유자원의 동시 접근을 막아주는 세마포어등 여러 가지 기법이 존재하며, 각 동기화라는 부분이 쉽게 생각할 문제는 아니기 때문에 많은 고민과 개념의 이해 바탕으로 접근하는게 매우 중요한 과제였다. 이해한다고 공부를 했지만, 그 이상으로 어려운 개념이 존재하기 때문에 더 많은 공부가 필요로 할 것이라고 생각이 들었다.

또 이번과제의 어려움을 느꼈던 부분이 동기화에 있어서 sleep을 추가하여 사용자의 인터페이스가 가능하게 하여 출력화면을 잘 보도록 하도록 해야 했고, 리눅스 자체의 인터넷 연결이 잘 못되어 고치느라 많은 애를 먹었다. 에러 및 메모리 발생문제 또한 그렇다.

이제 마지막 과제로 시스템 프로그래밍 실습이 끝나는데, 많은 과제를 수행하면서 너무 바쁘기도 힘들기도 하였지만, 큰 프로젝트를 해낸거 같아서 기분이 좋았다.