

System Programming

(Assignment4-2)

과 목	시스템프로그래밍실습
담당교수	이기훈 교수님
학 과	컴퓨터공학과
학 번	2010720149
성 명	이동현
날 짜	2016. 06. 3 (금)

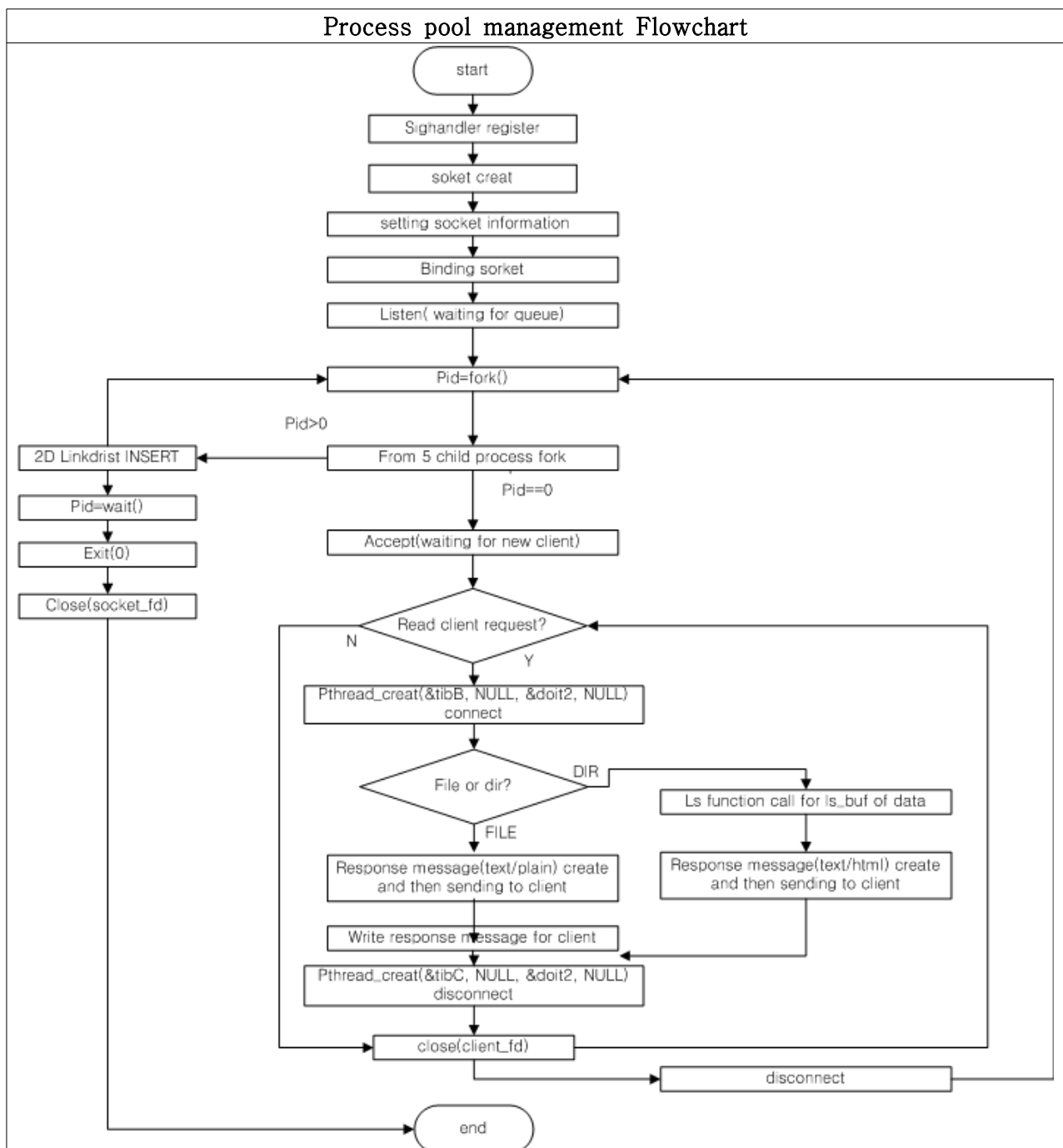


A. Introduction

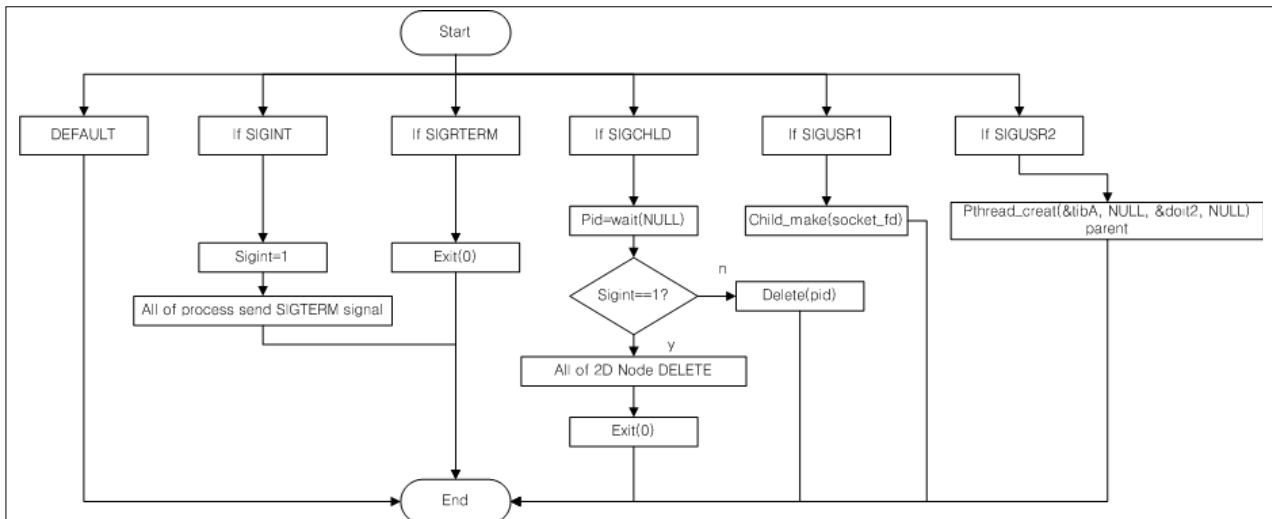
♣ Process pool management ♣

이번 과제는 ipc통신을 통해 프로세스간의 통신이 이루어지는 thread를 이용한 shared memory를 이용을 하였고, race-condition을 피하기 위해 mutex의 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  
    pthread_mutex_lock(&counter_mutex);  
    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);  
    pthread_mutex_unlock(&counter_mutex);  
    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  
    pthread_mutex_lock(&counter_mutex);  
    sprintf<--(char*)shm_addr, "%d,%d", getpid(), 1;  
    pthread_mutex_unlock(&counter_mutex);  
    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
    pthread_mutex_lock(&counter_mutex);
    sprintf<-(char*)shm_addr, "%d,%d", getpid(), ;
    pthread_mutex_unlock(&counter_mutex);
    sleep(1);
    return NULL;
}

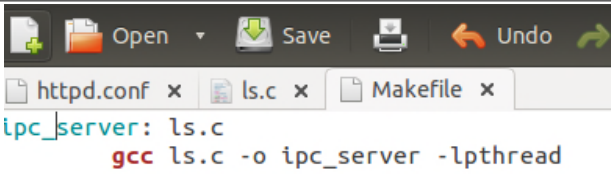
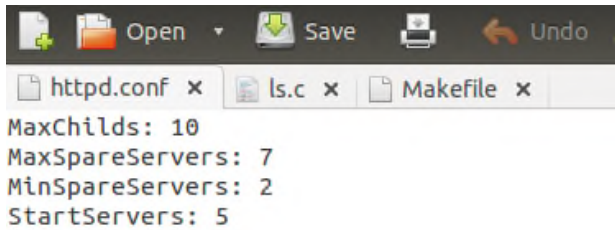
void sig_handler(int sig)
{
    if sig == SIGINT
        sigint=1;
        all of process send SIGTERM signal
    else if sig== SIGTERM
        exit(0)
    else if sig == SIGCHLD
        pid <- wait(NULL)
        if(sigint == 1)
            all of 2D DELETE
            exit(0)
        else
            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_12_Process_pool_management_v3

E. Conclusion

Makefile	
 <pre>ipc_server: ls.c gcc ls.c -o ipc_server -lpthread</pre>	
My port number	httpd.conf
40051	 <pre>MaxChilds: 10 MaxSpareServers: 7 MinSpareServers: 2 StartServers: 5</pre>

♣ 결과 화면 1 ♣

```
=====
[Thu Jun 2 08:35:51 2016] IdleServerCount: 3
[Thu Jun 2 08:35:51 2016] 3272 process is created.
[Thu Jun 2 08:35:51 2016] IdleServerCount: 4
[Thu Jun 2 08:35:52 2016] 3273 process is created.
[Thu Jun 2 08:35:52 2016] IdleServerCount: 5
===== New client =====
[Thu Jun 2 08:36:02 2016]
IP: 192.168.1.48
Port: 1496
PID: 3235
=====
[Thu Jun 2 08:36:04 2016] IdleServerCount: 4
===== New client =====
[Thu Jun 2 08:36:08 2016]
IP: 192.168.1.48
Port: 1752
PID: 3250
=====
[Thu Jun 2 08:36:10 2016] IdleServerCount: 3
[Thu Jun 2 08:36:10 2016] 3278 process is created.
[Thu Jun 2 08:36:10 2016] IdleServerCount: 4
[Thu Jun 2 08:36:10 2016] Maximum exceed! Not Create child
```

다음은 Maxchild의 조건을 반영한 결과이다. Maxchild를 10으로 잡았는데 옆 캡처를 통해 08:36:04 시간의 idle의 개수일 때의 프로세스 개수는 9개이다. 추가적인 그리고 나서 create를 한 결과 1개만 생성하여 10개를 맞춘 후 다음 은 프로세스를 생성할 수 없게 조건을 걸었다.

♣ 결과 화면 2 ♣

```

jong@ubuntu:~/Sys/assign9$ ./ipc_server
Thu Jun 2 08:29:58 2016] Server is started.
Thu Jun 2 08:29:58 2016] Socket is created. Ip: 127.0.1.1, Port: 29596
Thu Jun 2 08:29:58 2016] 3156 process is created.
Thu Jun 2 08:29:58 2016] IdleServerCount: 1
Thu Jun 2 08:29:59 2016] 3157 process is created.
Thu Jun 2 08:29:59 2016] IdleServerCount: 2
Thu Jun 2 08:30:00 2016] 3158 process is created.
Thu Jun 2 08:30:00 2016] IdleServerCount: 3
Thu Jun 2 08:30:01 2016] 3160 process is created.
Thu Jun 2 08:30:01 2016] IdleServerCount: 4
Thu Jun 2 08:30:02 2016] 3162 process is created.
Thu Jun 2 08:30:02 2016] IdleServerCount: 5
===== New client =====
[Thu Jun 2 08:30:06 2016]
IP: 192.168.1.48
Port: 63447
PID: 3156
=====
[Thu Jun 2 08:30:08 2016] IdleServerCount: 4
===== New client =====
[Thu Jun 2 08:30:10 2016]
IP: 192.168.1.48
Port: 63703
PID: 3157
=====
[Thu Jun 2 08:30:12 2016] IdleServerCount: 3
[Thu Jun 2 08:30:12 2016] 3172 process is created.
[Thu Jun 2 08:30:12 2016] IdleServerCount: 4
[Thu Jun 2 08:30:13 2016] 3174 process is created.
[Thu Jun 2 08:30:13 2016] IdleServerCount: 5
===== New client =====
[Thu Jun 2 08:30:16 2016]
IP: 192.168.1.48
Port: 63959
PID: 3158
=====
[Thu Jun 2 08:30:18 2016] IdleServerCount: 4
===== Disconnected client =====
[Thu Jun 2 08:30:26 2016]
IP: 192.168.1.48
Port: 63959
PID: 3158
=====
[Thu Jun 2 08:30:28 2016] IdleServerCount: 5
===== Disconnected client =====
[Thu Jun 2 08:30:30 2016]
IP: 192.168.1.48
Port: 63703
PID: 3157
=====
[Thu Jun 2 08:30:32 2016] IdleServerCount: 6
===== Disconnected client =====
[Thu Jun 2 08:30:33 2016]
IP: 192.168.1.48
Port: 63447
PID: 3156
=====
[Thu Jun 2 08:30:35 2016] IdleServerCount: 7
[Thu Jun 2 08:30:35 2016] 3156 process is terminated.

```

우선 ipc_server를 실행시키면 5개의 프로세스가 생성이되고 idle이 5개가 된다. 클라이언트 연결을 하면 idle 수는 줄어들고 3개가 될 경우 minimum이 4이기 때문에 start server 개수 만큼 맞춰 프로세스를 생성을 한다. 그리고 maximum idle수 6개를 idle 수가 넘을 경우 start server 개수 만큼 프로세스를 종료시킨다. 그리고 컨트롤 씨를 누를 경우 모든 프로세스와 idle의 개수가 맞춰 종료되었음을 확인 할 수 있다.


```
[Thu Jun 2 08:30:35 2016] IdleServerCount: 6
[Thu Jun 2 08:30:36 2016] 3157 process is terminated
[Thu Jun 2 08:30:36 2016] IdleServerCount: 5
^C[Thu Jun 2 08:30:43 2016] 3174 process is terminat
[Thu Jun 2 08:30:43 2016] IdleServerCount: 4
[Thu Jun 2 08:30:43 2016] 3172 process is terminated
[Thu Jun 2 08:30:43 2016] IdleServerCount: 3
[Thu Jun 2 08:30:43 2016] 3162 process is terminated
[Thu Jun 2 08:30:43 2016] IdleServerCount: 2
[Thu Jun 2 08:30:43 2016] 3160 process is terminated
[Thu Jun 2 08:30:43 2016] IdleServerCount: 1
[Thu Jun 2 08:30:43 2016] 3158 process is terminated
[Thu Jun 2 08:30:43 2016] IdleServerCount: 0
[Thu Jun 2 08:30:43 2016] server is terminated.
dong@ubuntu:~/Svs/assig95
```

♣ 고찰 ♣

이번 과제는 생소한 스레드와 뮤텝스 기법을 통한 race condition을 피하기 위하여 공유메모리를 이용해서 프로세스간의 통신이 가능하도록 구현하는 server program을 구현하였다. 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. 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. 프로세스간의 통신을 가능하게 하는 방법 중의 하나가 공유메모리이다. 공유메모리는 여러 프로세스간의 자원을 공유할 수 있는 메모리중 하나로, 이 번 과제에서 부모와 자식 간의 status의 자원을 공유를 하는 것이 핵심이다. 그 status로 인해 idle process를 구분하고 생성 및 삭제를 가능하게 하기 때문이다. 뮤텝스 기법은 공유메모리 접근이나 race condition을 막아주는 것으로 동기화의 역할을 한다. 만약 공유메모리를 하나의 스레드가 접근하고 있다면 다른 스레드의 접근을 lock을 통해 막아주고 unlock을 통해 접근을 가능하게 하여 중복해서 공유메모리를 읽고 사용하는데 문제가 생기는 race condition을 막아주는 것이다. 이처럼 뮤텝스 이외에 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. 앞서서 과제이외에 임베디드 실습을 수강하는 나는 보드를 통한 device를 구현하여 프로그램을 작성을 하는데 동시의 device를 작동하기 위해 스레드를 사용해야하기 때문에 매우 큰 도움이 될 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.

다음 과제가 이 기나긴 과제들 중의 마지막 과제이다. 매우 어렵기도 복잡하기도 했지만, 점점 과제를 할수록 점차 발전해 나간다는 생각이 들어서 점점 뿌듯했다. 다음과제도 마무리 잘하도록 준비를 잘해야겠다.