REPORT

[시스템프로그래밍 Assignment 3-2]

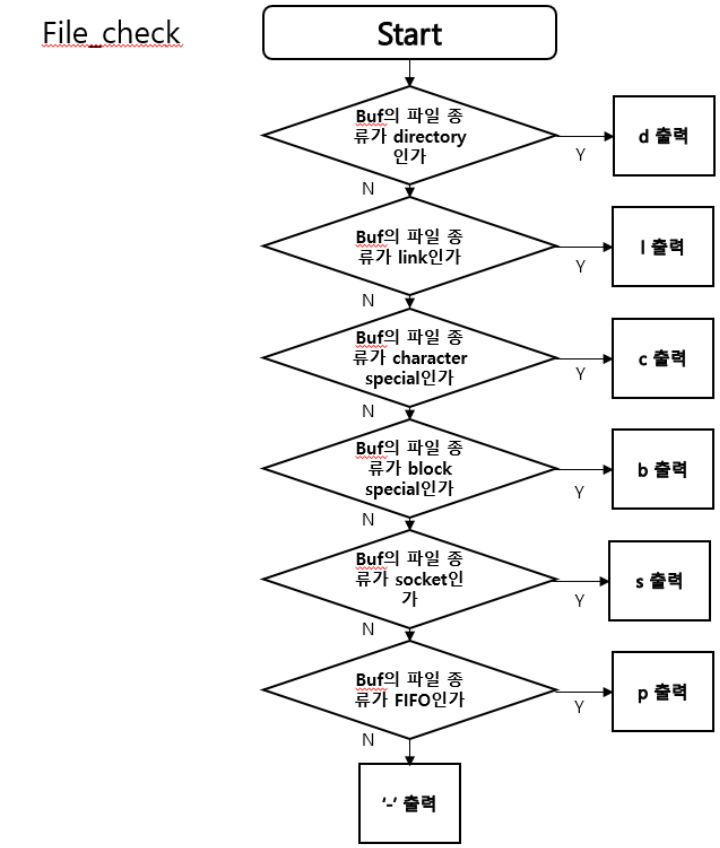
학과	컴퓨터정보공학부
학번	2018202018
이름	유승재
담당 교수	김태석 교수님
제출일	2023.05.24

1. Introduction

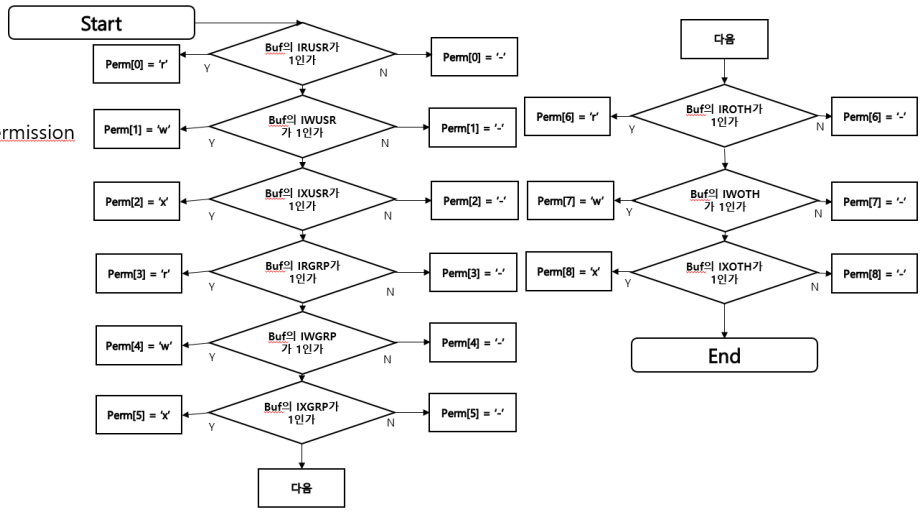
이번 과제는 다중 fork 서버를 구현하는 과정에서 미리 fork 한 프로세스를 무한정 사용하는 것이 아닌 이번 과제에서는 현재 연결되어 있지 않은 idle process 의 개수를 기준으로 새로운 process 를 생성하거나, 기존의 idle process 를 종료시켜

새로운 client 가 접속을 시도할 때마다 그에 맞춰 child process 를 생성해주어 유동적으로 연결을 해주는 서버를 작성합니다. 이번 과제에서는 또한 공유메모리, thread 와 mutex 의 개념을 이용하여 공유메모리를 이용하여 모든 process 에서 사용가능한 변수를 설정하고 이를 thread 를 생성하여 thread 내부에서 공유메모리를 control 하는 코드를 구현합니다. Thread 또한 동기화 문제를 해결하기 위해서 mutex 의 개념을 이용하여 잘못된 값이 출력되는 것을 방지하는 것 까지 이번 과제입니다.

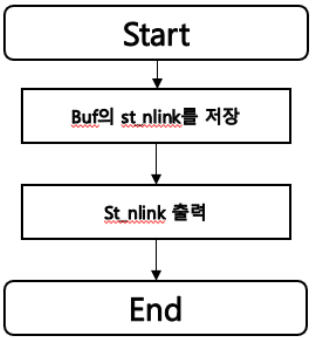
2. Flow Chart



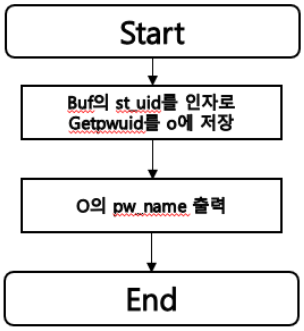
File_permission



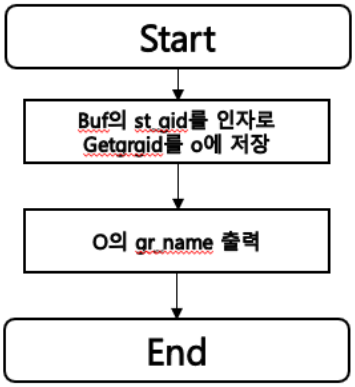
File_link



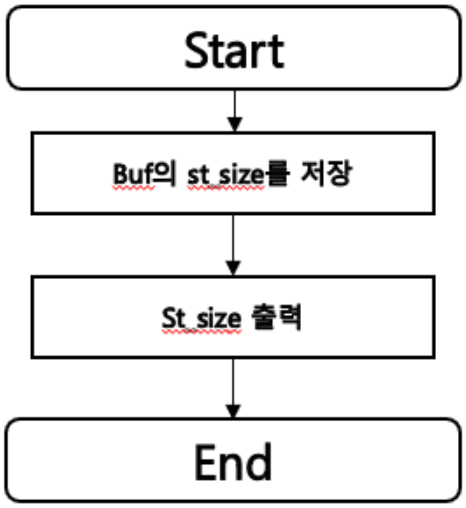
File_owner_name



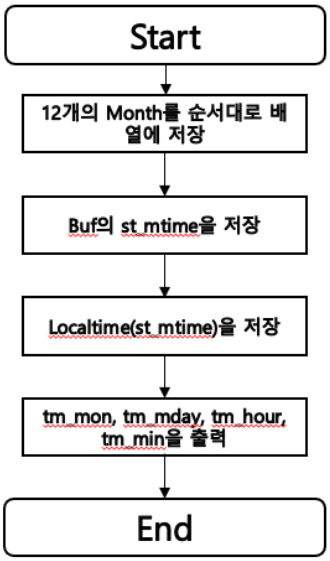
File_group_name



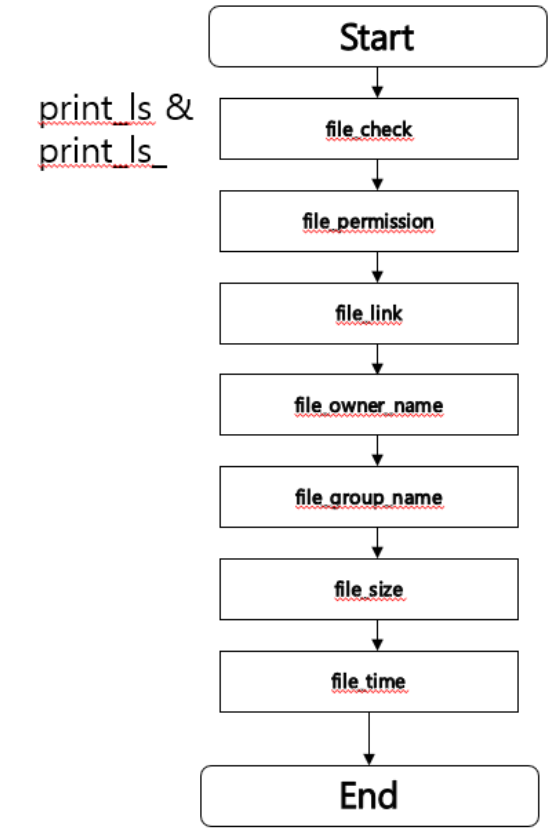
File_size



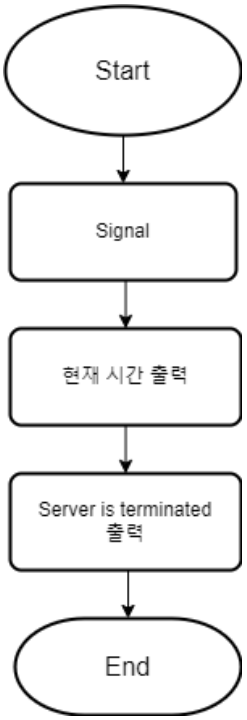
File_time



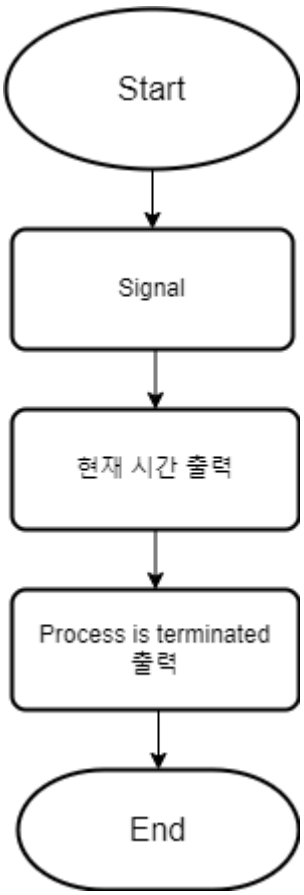
print_ls &
print_ls_



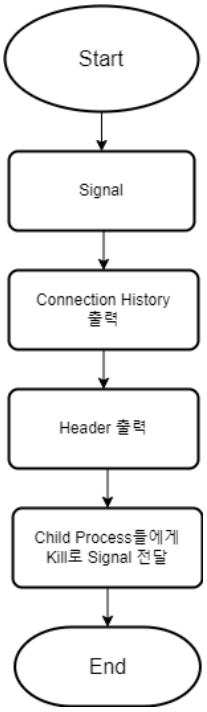
SignalHandler



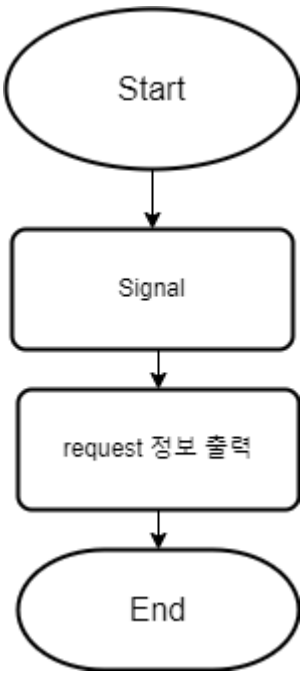
SignalHandler2



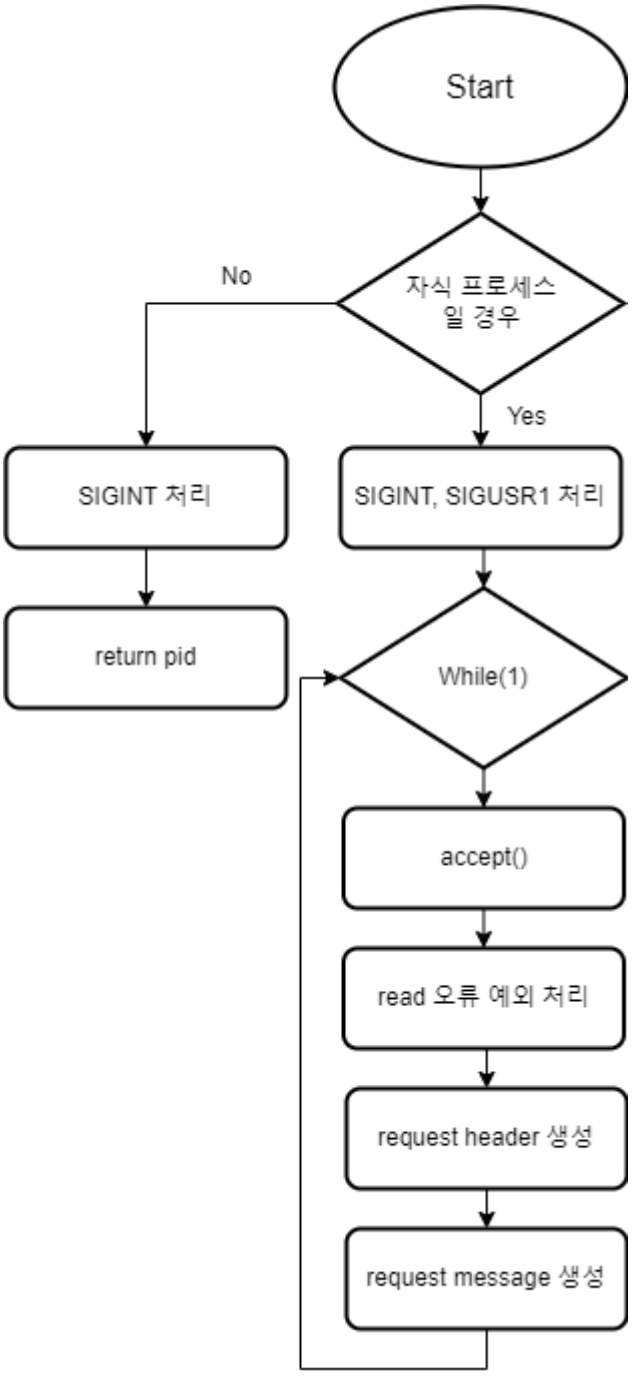
SignalHandlerChild



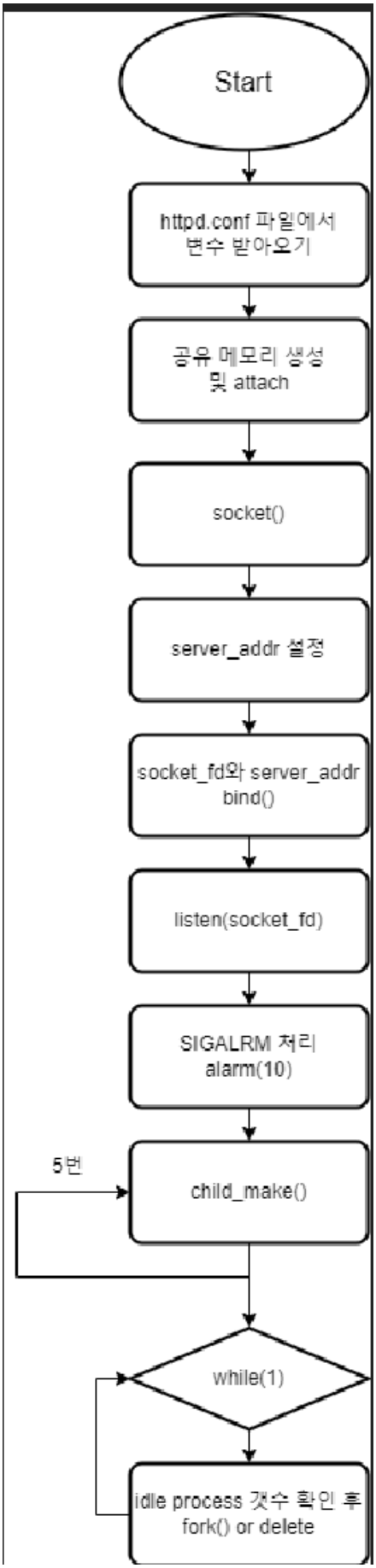
SignalHandlerParent



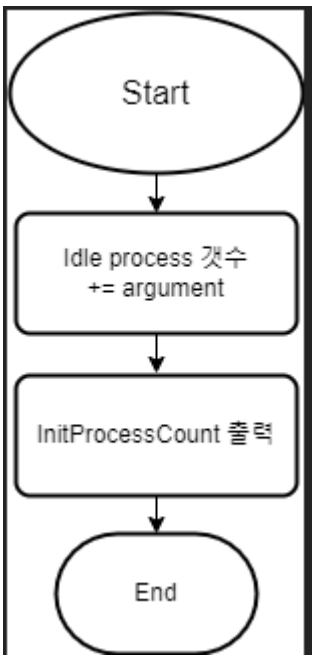
Child_make


Start
자식 프로세스
일 경우
No
Yes
SIGINT 처리
SIGINT, SIGUSR1 처리
return pid
While(1)
accept()
read 오류 예외 처리
request header 생성
request message 생성

Main


Start
httpd.conf 파일에서
변수 받아오기
공유 메모리 생성
및 attach
socket()
server_addr 설정
socket_fd와 server_addr
bind()
listen(socket_fd)
SIGALRM 처리
alarm(10)
5번
child_make()
while(1)
idle process 갯수 확인 후
fork() or delete

Threadtest



Threadtest2

3. Pseudo Code

```
void file_check(struct stat ){
   char 선언;
   table 행 시작;
   if (디렉토리){
      d 출력
   }
   else if (링크){
      l 출력
   }
   else if (character special){
      c 출력
   }
   else if (block special){
      b 출력
   }
   else if (socket){
      s 출력
   }
   else if (FIFO){
      p 출력
   }
   else if (regular){
      - 출력
   }
}
void file_permission(struct stat ){
```

Perm[10];

Stat 의 st_mode 를 가져와 권한에 맞게 perm 의 자리에 char 배치

Perm 출력

}

void file_link(struct stat){

Stat 의 st_nlink 를 가져와 link 개수 확인

Link 개수 출력

}

void file_owner_name(struct stat) {

Stat 의 getpwuid 로 st_uid 를 가져와 struct passwd 형태에 저장

Struct passwd 의 pw_name 출력

}

void file_group_name(struct stat){

Stat 의 getgrgid 로 st_gid 를 가져와 struct group 형태에 저장

Struct group 의 gr_name 출력

}

void file_size(struct stat){

Off_t 변수 0 으로 초기화;

변수에 st_size 로 대입;

St_size 출력

}

void file_time(struct stat){

12 개의 month 를 배열 크기 12 개의 문자열 배열에 1 월부터 12 월까지 저장;

Struct stat 의 st_mtime 을 localtime 함수를 이용해 struct tm 에 저장;

Struct tm 형식의 tm_mon, tm_mday, tm_hour, tm_min 출력

Table 행 끝

```
}
void print_ls(struct stat ){
    File_check, file_permission, file_link, file_owner_name, file_group_name,
    File_size, file_time 한번에 struct stat 을 넣어줘 출력
}
void SignalHandler(int Sig) {
    현재 시간 출력;
    Server is terminated 출력;
}
void SignalHandler2(int Sig) {
    현재 시간 출력;
    현재 Process is terminated 출력;
}
void SignalHandlerChild(int Sig) {
    Connection History 출력;
    출력 내용 목록 출력;
    모든 child process 에게 SIGUSR1 신호 전달;
    Alarm(10);
}
void SignalHandler(int Sig) {
    현재 시간 출력;
    Server is terminated 출력;
}
void* threadtest(void * num) {
   mutex lock;
   usleep(10000);
```

```
    인자 int 형으로 변형;
    Idle process 개수 인자만큼 더해줌;
    현재 시간 출력;
    Idle process 개수 출력;
    mutex unlock;
}
void* threadtest2(void * presenta) {
    mutex lock;
    usleep(10000);
    현재 시간 출력;
    History 공유 메모리 struct 배열에 저장;
    History 개수 증가;
    mutex unlock;
}
Pid_t child_make(int I, int socket_fd){
     Fork();
     If (부모 프로세스일 경우) {
          Signal(SIGINT, SignalHandler);
          Return pid;
     }
     Signal(SIGINT, SignalHandler2);
     Signal(SIGUSR1, SignalHandlerParent);
     Threadtest(1) 실행;
     While (1) {
     Client_fd = Accept(socket_fd);
     read(client_fd 를 읽을 경우 잘못 읽었을 때){
```

while 문 continue;

}

if (자식 프로세스일 경우){

Client_fd 를 read 해서 request message 출력;

Request message 에서 url 확인;

url 이 directory 일 경우

header_tag = text/html;

url 이 이미지파일 일 경우

header_tag = image/*

그 외일 경우

Header_tag = text/plain

Threadtest2(accept history) 실행;

if (url 이 "/" (cwd)일 경우){

cwd 의 non-hidden 파일들 배열에 저장;

파일 배열 정렬;

파일배열 출력 형식을 HTML_ls 형식에 맞추어 response_message 에 저장;

}

else if (url 이 cwd 가 아닌 경우) {

if (url 이 directory 인 경우){

url 에 위치한 파일들 전부 배열에 저장;

파일 배열 정렬;

파일 배열 출력 형식을 HTML_ls 형식에 맞추어 response_message 에 저장;

}

else if (url 이 파일인 경우){

```
            url 의 파일 정보 전부 read;
            read 한 데이터 전부 response_message 에 저장;
            response_header client_fd 에 출력;
        }
        response_message client_fd 에 출력;
        else (존재하지 않는 파일인 경우){
            404 Error 양식에 맞추어 출력;
            url 이 directory 이거나 image 파일 인 경우
            response_header 출력;
            이전에 저장한 response_message 출력;
        }
    }
}
else (부모 프로세스일 경우) {
    if (url 이 /favicon.ico 일 경우) (favicon 예외 처리) {
        continue;
    }
         if (IP 가 허가되지 않은 IP 일 경우) {
        continue;
    }

    if (abs_url 이 존재하지 않는 경로일 경우(404 Error)) {
        continue;
    }
    if (이외의 경우) {
        struct Point 에 현재 access 한 client 의 정보들 저장;
```

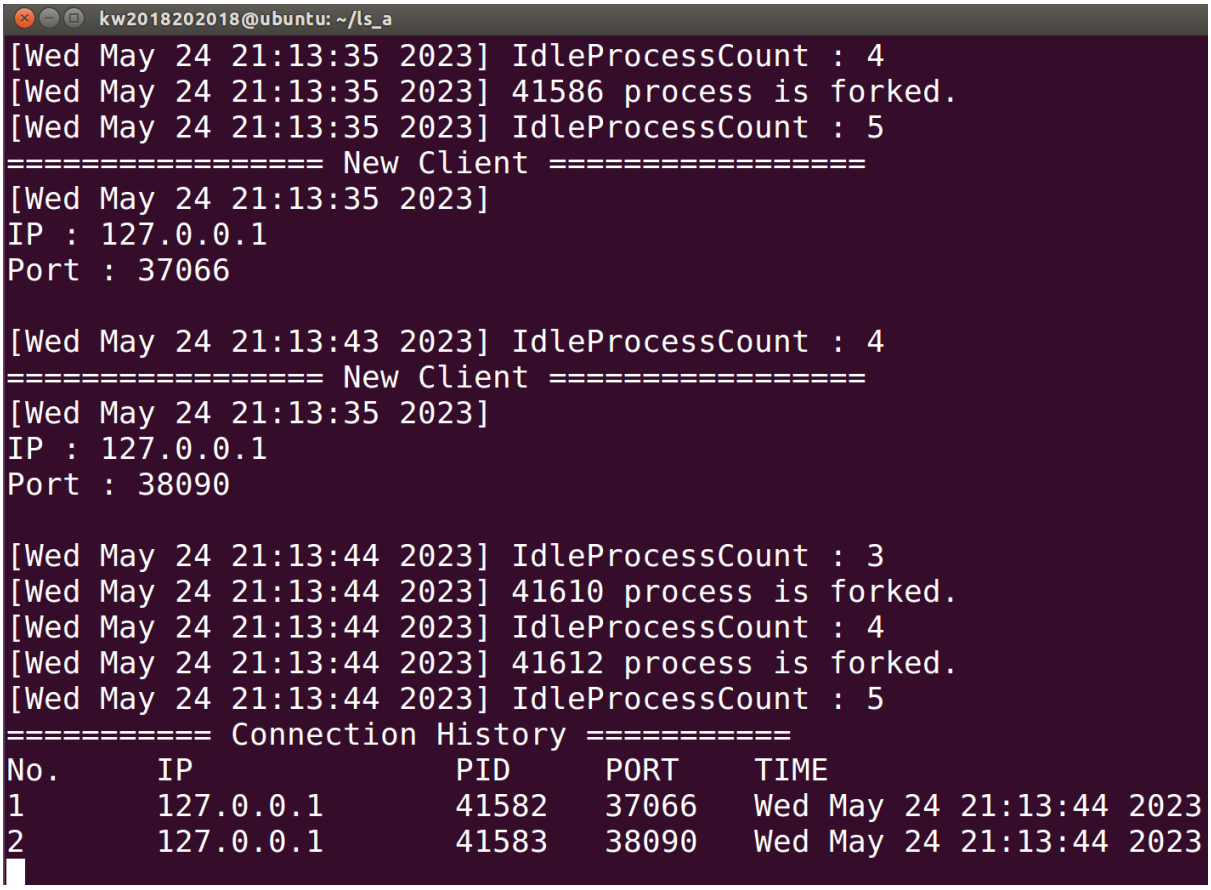
```
                이를 struct Point 배열에 저장;
                Signal Handler 로 출력;
            }
        }
        Disconnected 출력;
        Threadtest(-1) 실행;
}
int main(){
    httpd.conf 파일 읽어와서 변수에 저장;
    공유 메모리 생성;
    공유 메모리 attach;
    Socket_fd 생성;
    Setsockopt 로 socket_fd 설정;
    Struct sockaddr_in Server_addr 초기화;
    Server_addr 과 socket_fd 바인딩;
    Listen(socket_fd);
    Alarm(10);
    Pids 에 child_make 를 통해 모든 자식 process 의 pid 저장;
    Pids 출력;
    While (1)
      If (idle process 개수가 Max 보다 클경우) {
         While (idle process 개수 == 5)
            Idle process 종료;
      If (idle process 개수가 Min 보다 작을 경우) {
        While (idle process 개수 == 5)
           새로운 process 생성;
```

```
}
```

4. 결과화면

```
kw2018202018@ubuntu:~/ls_a$ ./test
[Wed May 24 20:33:15 2023] Server is started.
[Wed May 24 20:33:15 2023] 41287 process is forked.
[Wed May 24 20:33:15 2023] InitProcessCount : 1
[Wed May 24 20:33:15 2023] 41288 process is forked.
[Wed May 24 20:33:15 2023] InitProcessCount : 2
[Wed May 24 20:33:15 2023] 41289 process is forked.
[Wed May 24 20:33:15 2023] InitProcessCount : 3
[Wed May 24 20:33:15 2023] 41290 process is forked.
[Wed May 24 20:33:15 2023] InitProcessCount : 4
[Wed May 24 20:33:15 2023] 41291 process is forked.
[Wed May 24 20:33:15 2023] InitProcessCount : 5
^C[Wed May 24 20:33:15 2023] 41291 process is terminated.
[Wed May 24 20:33:15 2023] InitProcessCount : 4
[Wed May 24 20:33:15 2023] 41287 process is terminated.
[Wed May 24 20:33:15 2023] InitProcessCount : 3
[Wed May 24 20:33:15 2023] 41290 process is terminated.
[Wed May 24 20:33:15 2023] InitProcessCount : 2
[Wed May 24 20:33:15 2023] 41288 process is terminated.
[Wed May 24 20:33:15 2023] InitProcessCount : 1
[Wed May 24 20:33:15 2023] 41289 process is terminated.
[Wed May 24 20:33:15 2023] InitProcessCount : 0
[Wed May 24 20:33:16 2023] Server is terminated.
kw2018202018@ubuntu:~/ls_a$
```

기본적으로 종료할 시 현재 IdleProcessCount 를 출력해줍니다.



IdleProcessCount 가 MinIdleNum 보다 작아질 경우 기본 개수인 5 개로 다시 맞춰줍니다.

```
No.     IP              PID     PORT    TIME
1       127.0.0.1       41582   37066   Wed May 24 21:13:44 2023
2       127.0.0.1       41583   38090   Wed May 24 21:13:44 2023
=============== Disconnected client ===============
[Wed May 24 21:13:48 2023]
IP : 127.0.0.1
Port : 37066

[Wed May 24 21:13:48 2023] IdleProcessCount : 6
=============== Disconnected client ===============
[Wed May 24 21:13:49 2023]
IP : 127.0.0.1
Port : 38090

[Wed May 24 21:13:49 2023] IdleProcessCount : 7
[Wed May 24 21:13:49 2023] 41582 process is terminated.
[Wed May 24 21:13:49 2023] IdleProcessCount : 6
[Wed May 24 21:13:49 2023] 41583 process is terminated.
[Wed May 24 21:13:49 2023] IdleProcessCount : 5
=========== Connection History ===========
No.     IP              PID     PORT    TIME
1       127.0.0.1       41582   37066   Wed May 24 21:13:49 2023
2       127.0.0.1       41583   38090   Wed May 24 21:13:49 2023
=========== Connection History ===========
```

위의 연결이 종료될 경우 IdleProcessCount 가 MaxIdleNum 보다 커지게 되었고, 이로 인해 현재 IdleProcess 중 가장 오래된 2 개의 Process 를 종료합니다.

또한 Connection History 에 정보가 출력되는 것을 확인할 수 있습니다.

5. 고찰

이전까지는 다중 process 만을 다루었다면, 이번 과제는 process 뿐만 아니라 각각의 process 내부에서도 thread 를 통해 process 가 동시에 여러 줄의 코드를 수행하는 형식의 코드를 만들게 되었습니다. 또한 공유 메모리를 통해 process 간의 통신이 가능하도록 하는 개념 또한 새로 추가되었고, thread 간에 실행 중 동기화를 맞춰주기 위해서 mutex 라는 개념까지 추가하여 사용하게 되었습니다. 이렇게 여러 개의 새로운 개념을 사용하게 되어 이해하는 과정이 굉장히 오래 걸리는 과제였습니다. Thread 역시 사용함에 있어 원하지 않는 방식으로 코드가 동작하는 경우가 굉장히 많았고, 이를 제어하는 데 있어서 어려움이 많았습니다. 또한 이전에 배운 SignalHandler 와 thread 를 동시에 사용할 경우 코드가 정상적으로 작동하지 않는 경우가 다수 있었고, 이는 SignalHandler 가 빠른 시간 내에 Signal 을 처리하기 때문에 내부에서 thread 를 생성하는 것이 제한적이라는 것을 알게 되었습니다.

6. Reference