시스템프로그래밍실습

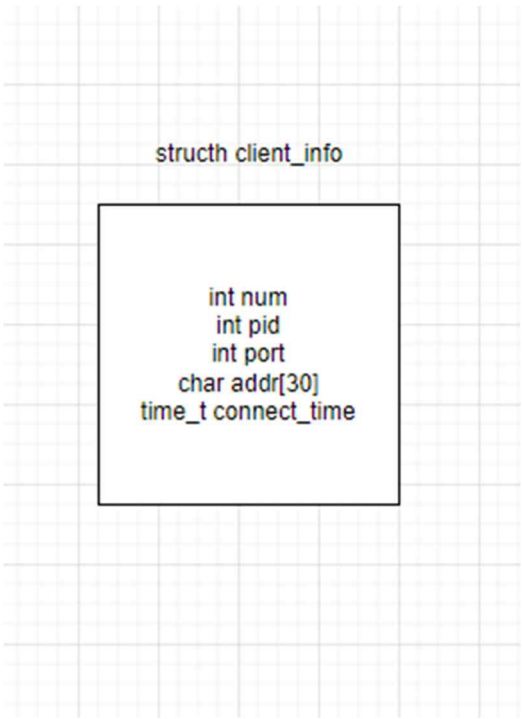
Assignment 3-2 과제

수업 명 : 시스템프로그래밍실습
과제 이름 : assignment3-2
담당 교수님 : 김태석 교수님
학 번 : 2019202005
이 름 : 남종식

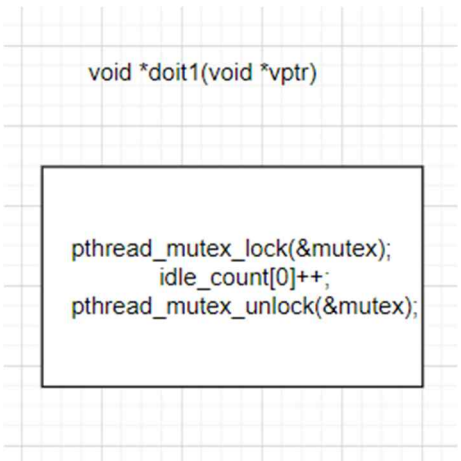
- **과제 소개**

이번 과제는 저번 과제에서 진행한 그대로 일단 fork를 미리 사용하여 자식 프로세스를 생성합니다. 이때 연결되지 않은 자식 프로세스를 공유메모리로 관리하는데 이때 이 프로세스들은 idle process라고 합니다. 각 자식 프로세스는 자신의 포트 번호를 사용하여 공유 메모리에 접근합니다. 이를 위해 Shared Memory의 Key Value를 설정합니다. 자식 프로세스는 클라이언트와의 연결 여부를 확인하고 이를 위해 Mutex를 사용하여 동기화합니다.
Idle process의 수를 공유 메모리를 통해 관리하고, 수의 변경이 발생하는 경우 부모 프로세스에서 출력합니다.
Idle process의 수가 MinIdleNum 미만이 되거나 MaxIdleNum 초과가 되는 경우, Idle process를 생성하거나 종료하여 5개를 유지합니다. 이를 위해 Mutex를 사용하여 동기화합니다. 그리고 클라이언트와 연결된 자식 프로세스의 수와 Idle process의 수의 합이 MaxChilds를 초과하지 않도록 조절합니다. 마지막으로 서버가 종료된다면 자식 프로세스 및 idle프로세스가 종료 후 공유 메모리를 제거해야 합니다. 이때, 주어진 설정 값들은 httpd.conf 파일에 저장해두고 값을 읽어와 프로그램 코드 내에서 사용해야 합니다.

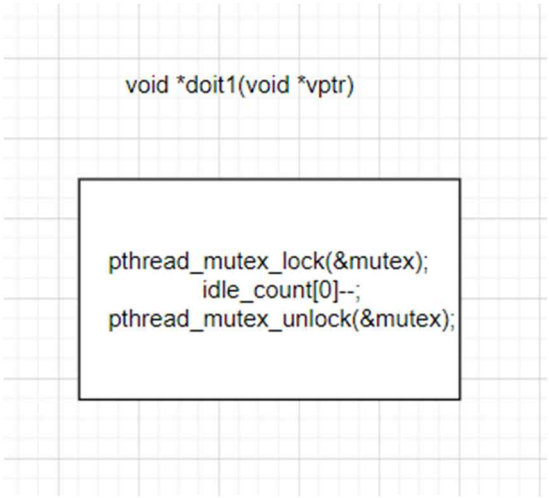
- **Flow Chart**
- **struct client_info**



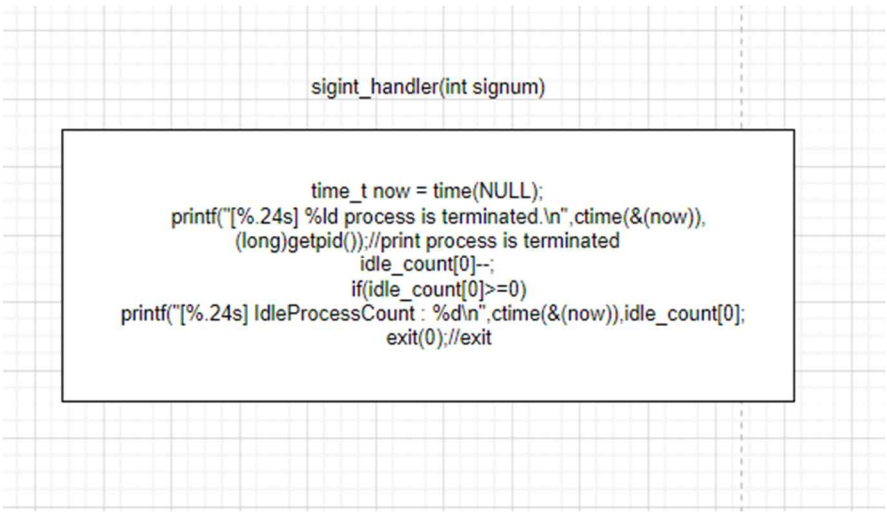
- **Void *doit1(voif*vptr)**



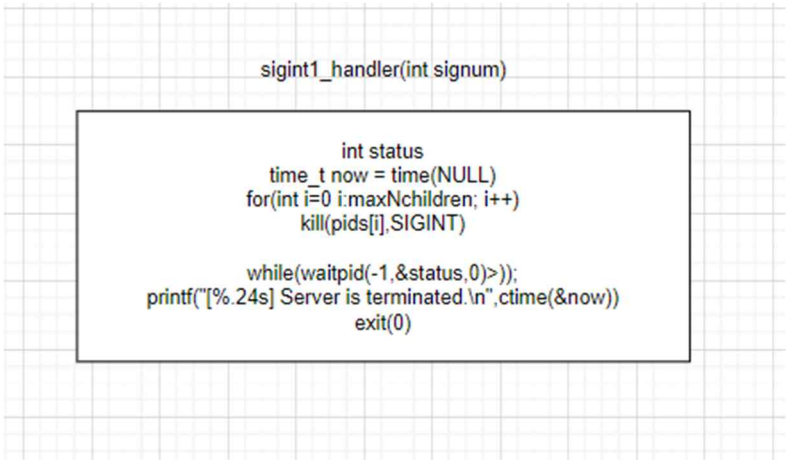
- **Void *doit1(voif*vptr)**



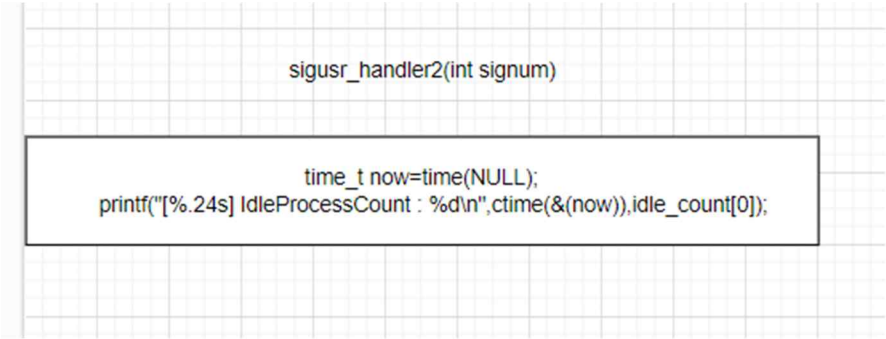
- **Sigint_handler(int signum)**



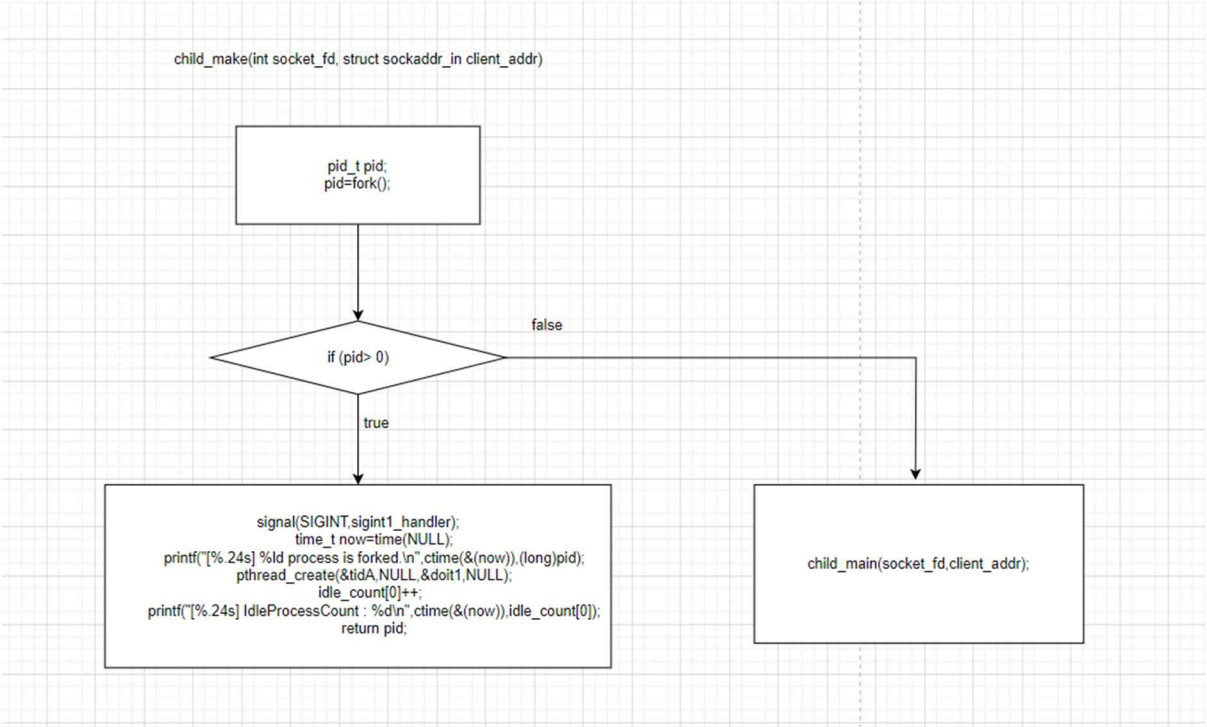
- **Sigint1_handler(int signum)**



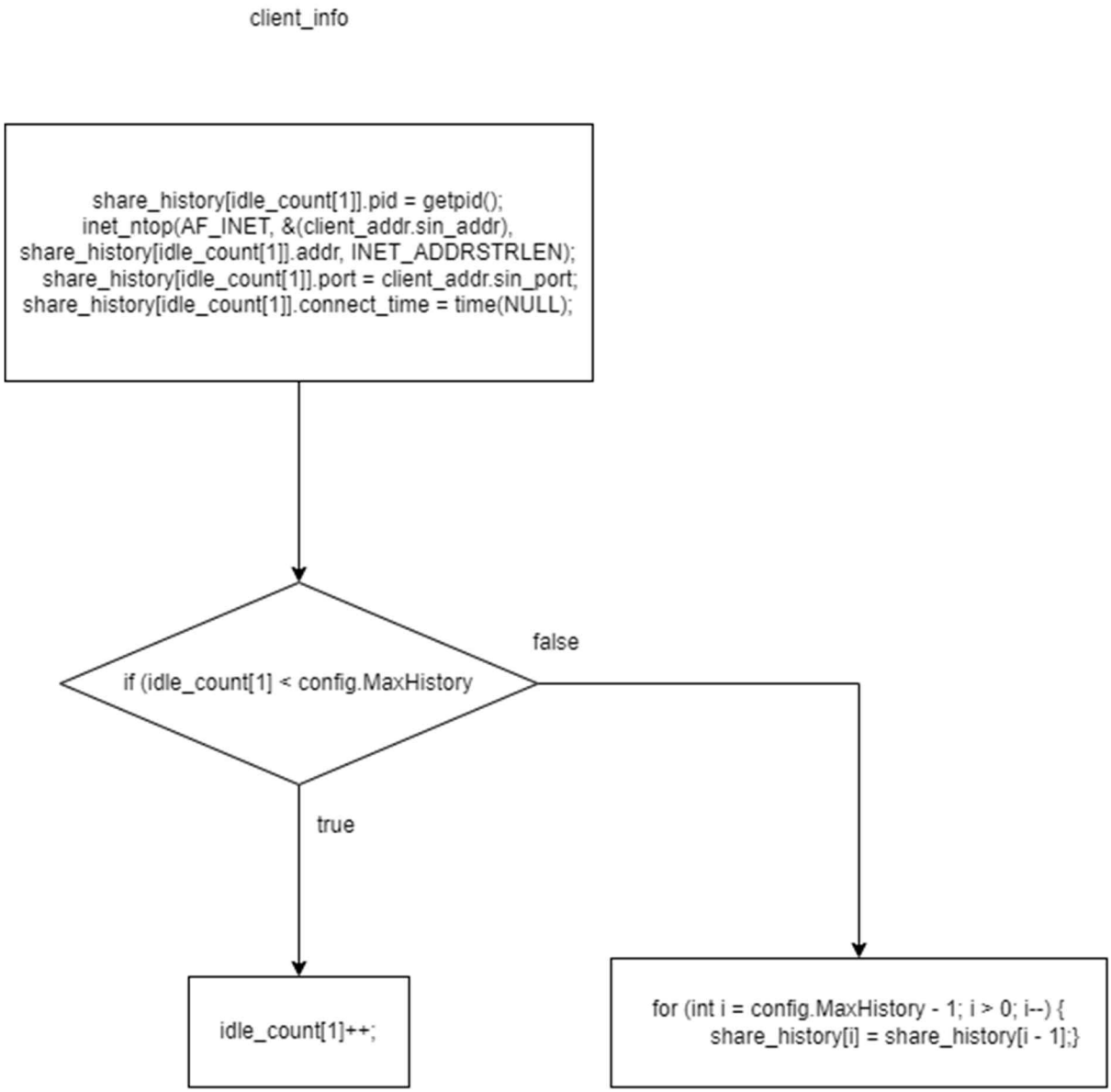
- **Sigusr_handler2(int signum)**



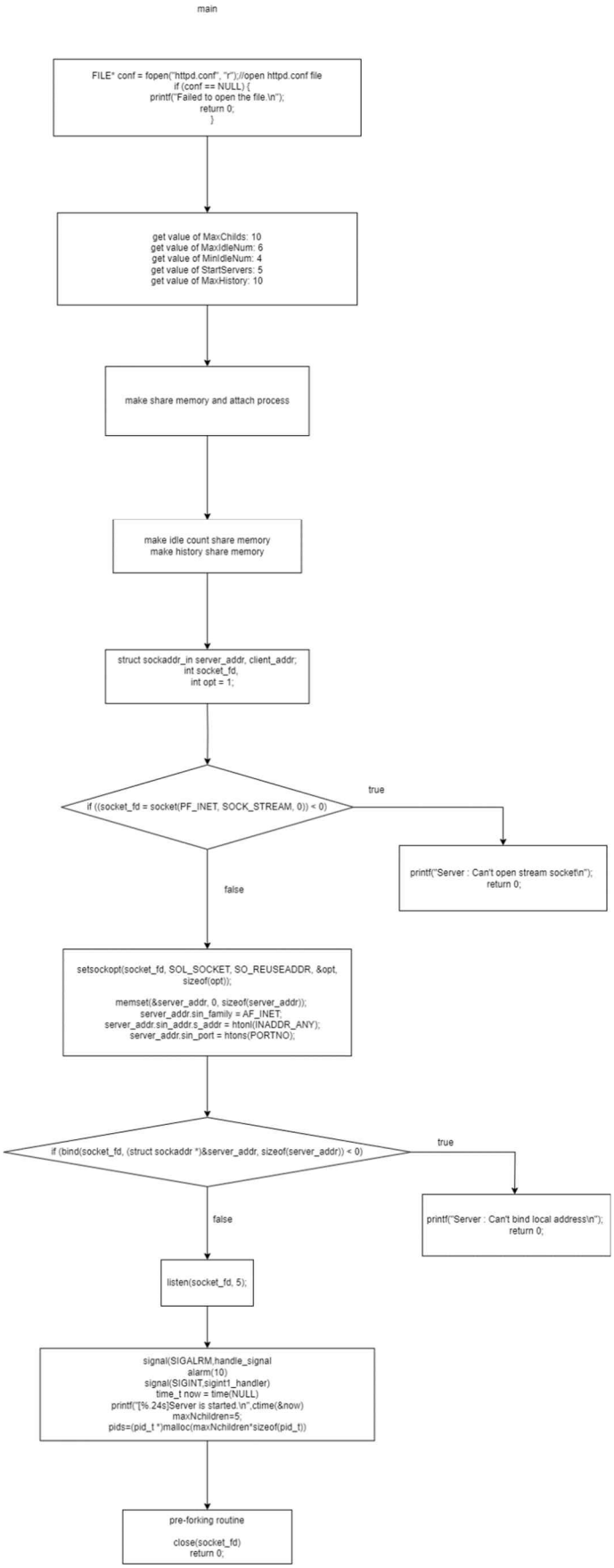
- **Child_make function**



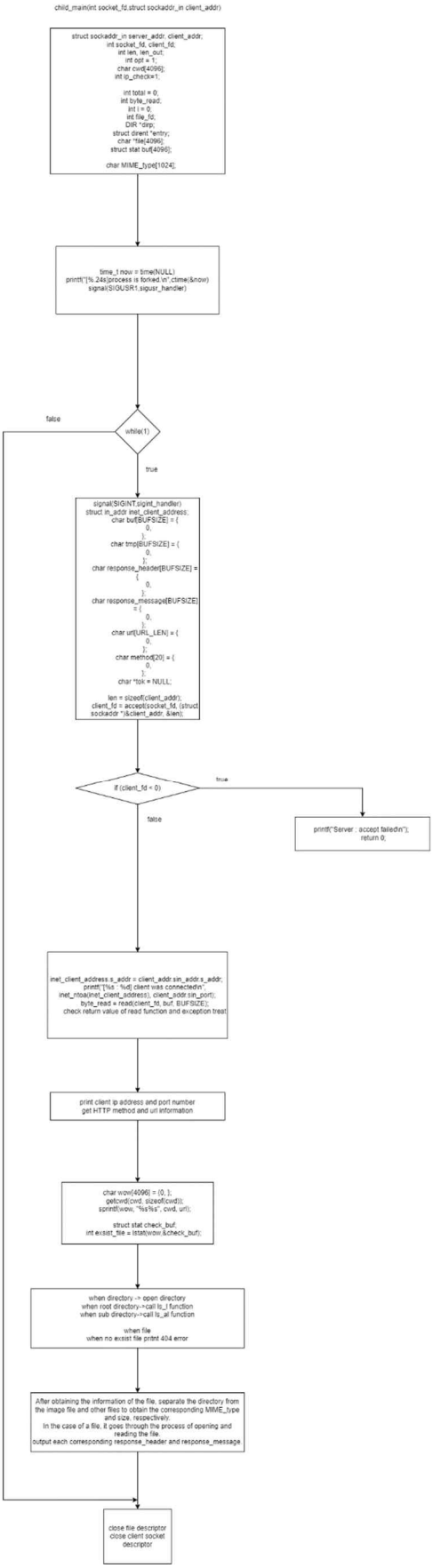
- **Client_info function**



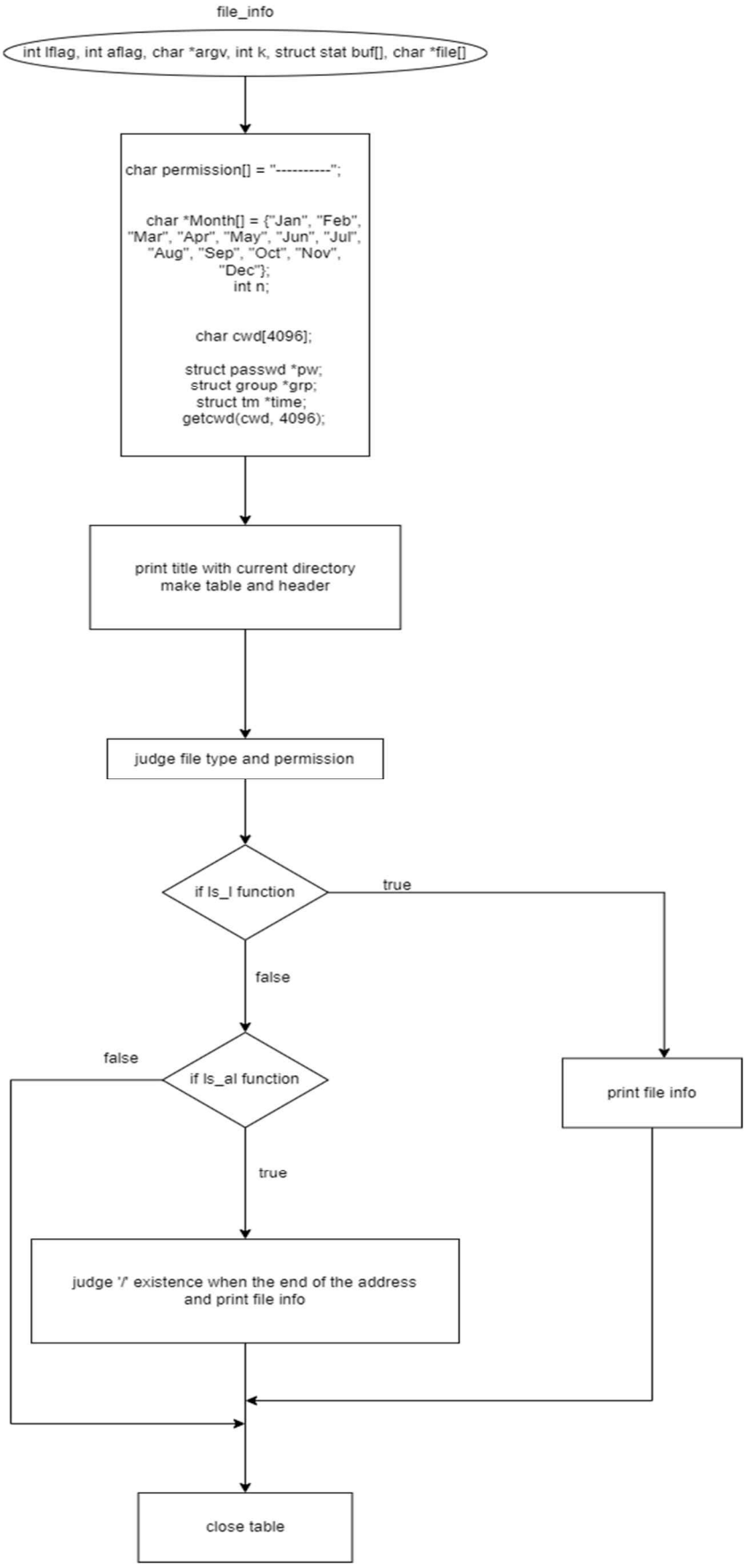
- **Main fuction**



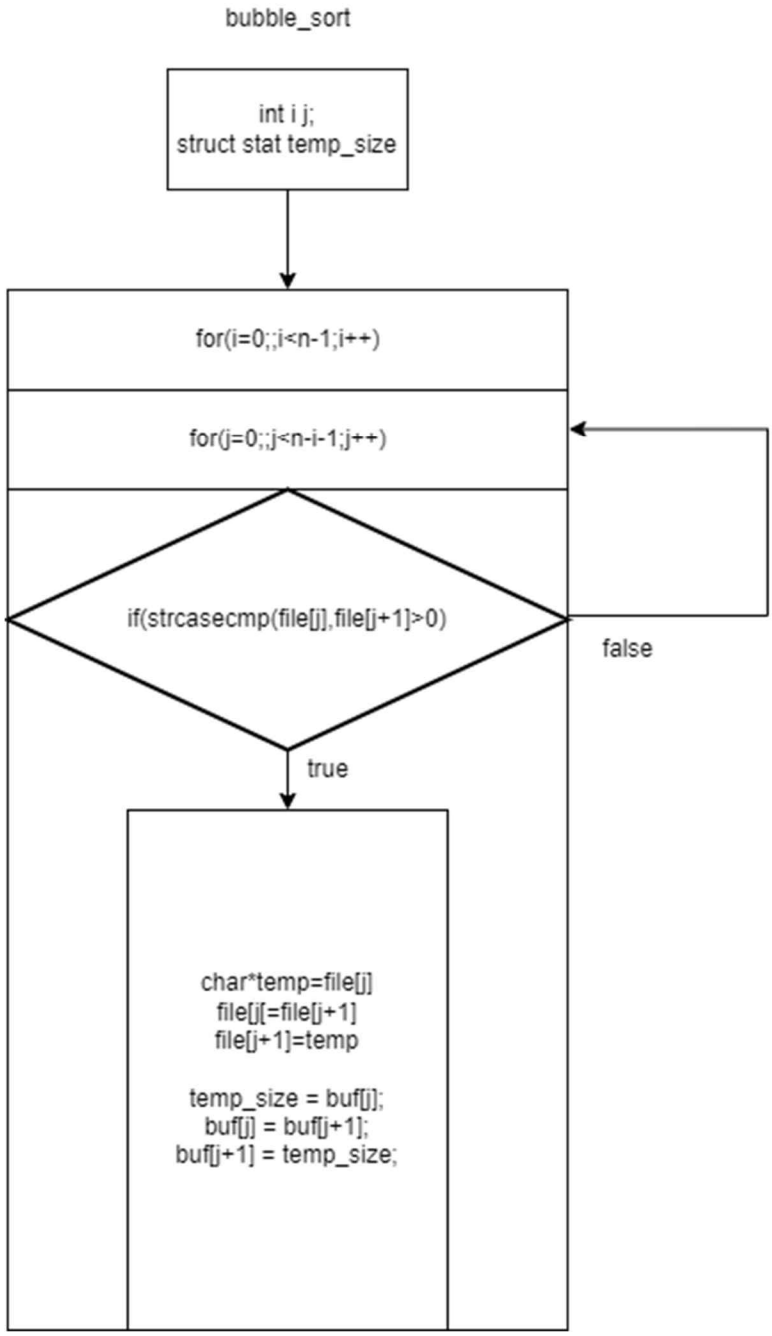
- **Child_main function**



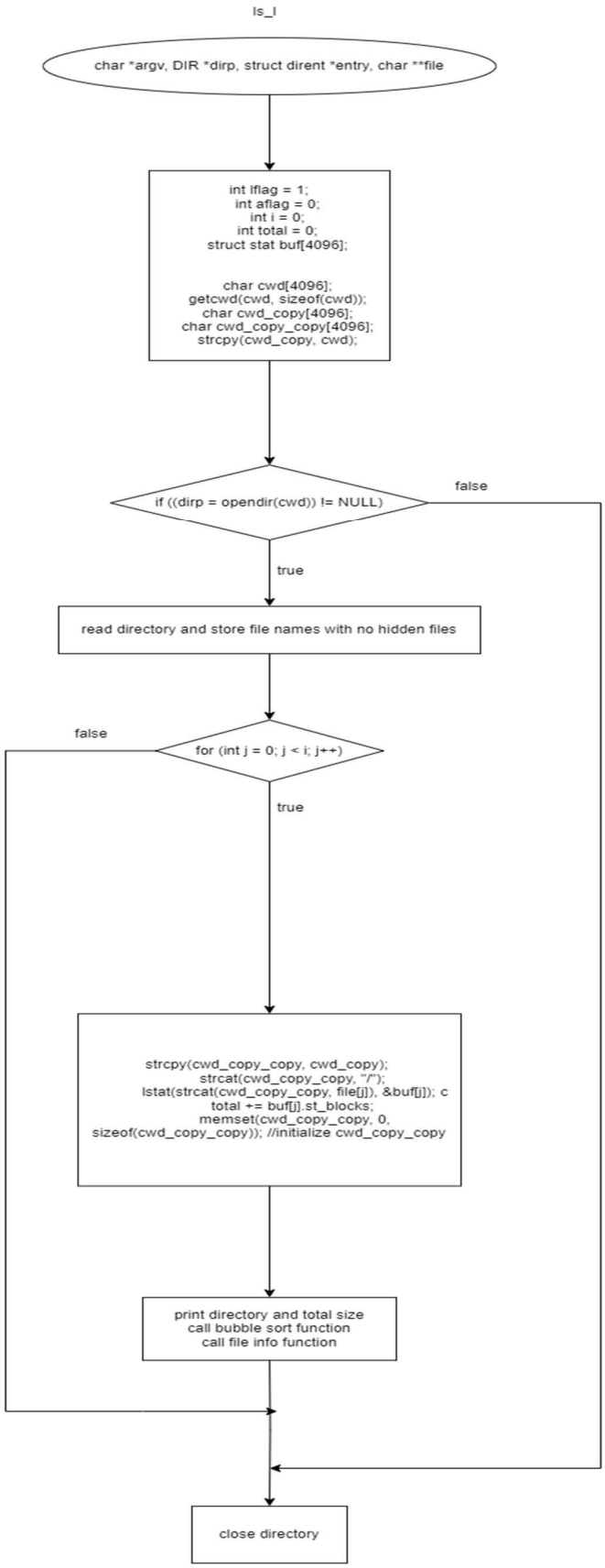
- **File_info function**



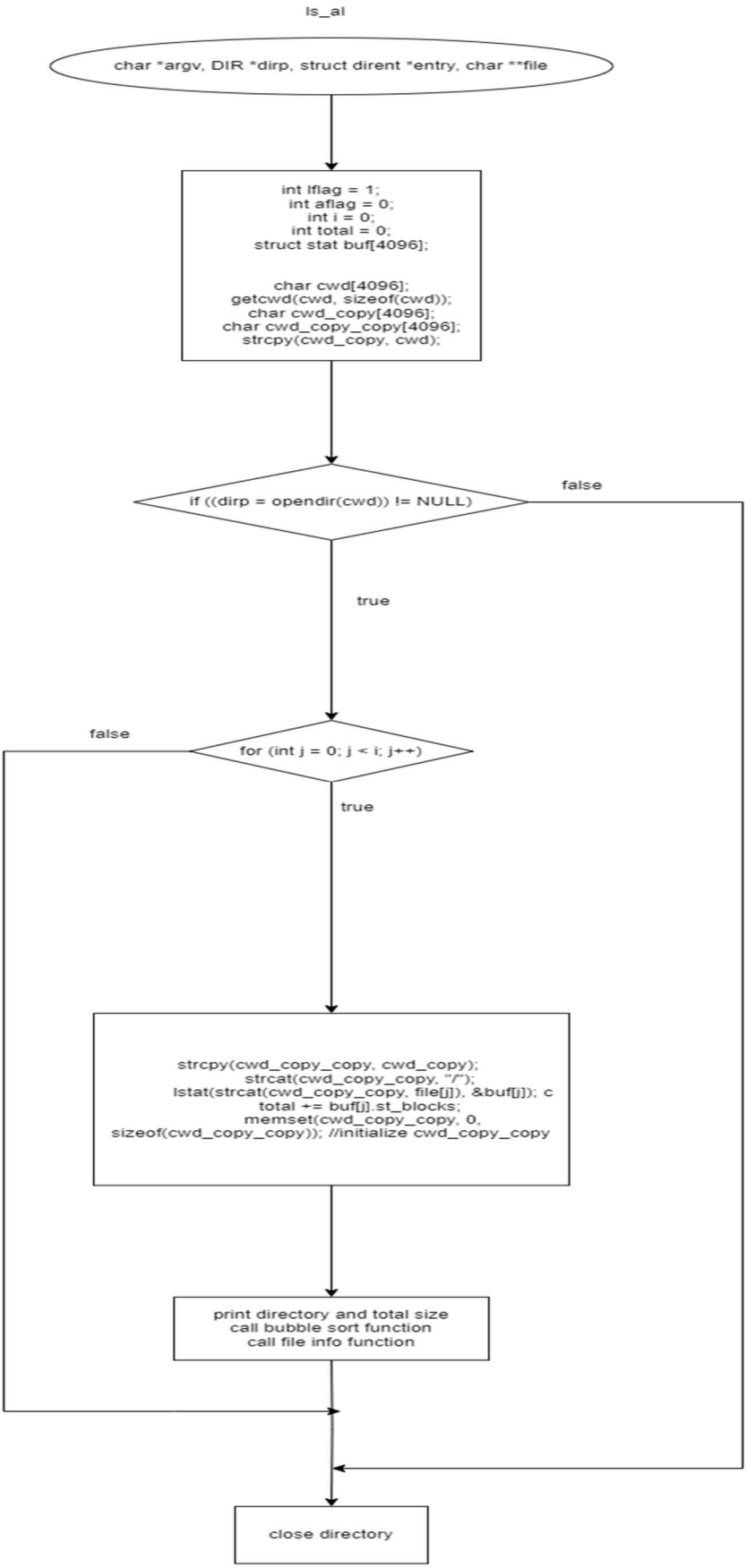
- **Bubble sort function**



- **Ls_l function**



- **Ls_al function**



- **Pseudo Code**

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <fnmatch.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <signal.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define URL_LEN 256
#define BUFSIZE 1024
#define PORTNO 40000

int shm_id
void *shm_addr
int* idle_count

int shm_id1
void *shm_addr1
```

struct client_info *share_history

pthread_t tidA
mutex initialize

define file info function
define bubble sort
define ls -l function
define ls -al function
define print_recent_clients
define handle_signal

static int MaxNchildren

static pid_t *pids

define struct client info
define struct HttpdConfig
define client_info function to store client information
define sigint_handler
define sigint1_handler
define sigusr_handler2
define *doit1(void *vptr)
define *doit2(void *vptr)

Main function
{
    Open httpd.conf file
    When get httpd.conf
    Add null string instead of ₩n
    Get value of conf
    Close httpd.conf file

    Make shared memory and attach them to process

Make idle_count shared memory and history shared memory

declare server address, client address
declare socket descriptor, opt=1

Call the socket function to create a socket file descriptor
Call setsocketopt to enable reuse of previously used port numbers
Initialize server_addr structure
Save address information for the server socket

Bind a socket with socket_fd to the IP address and port number specified in the server_addr structure
Socket wait to accept incoming connections from client sockets.

signal handler
alarm in 10 seconds
call sigint1_handler
print server is started with time

maxNchildren=5
make pids array
call child_make function
fork and call child_main function

in parent process
signal(SIGINT,sigint1_handler)
print("process is forked")
idle_count +1
print count of idle process
return pid

while(1)

```
when idle process count < minidlecount
for(int i=(idle_count[0]); i<5; i++)
    call child_make(socket_fd,client_addr);

when idle process count > maxidlecount
for(int i=(idle_count[0]); i>5; i--)
    kill(pids[b], SIGINT)
idle_count[0]—
print count of idle process

In child main function
    signal(SIGINT,sigint1_handler);
    declare pointer to directory stream and Pointer to a dirent structure
    declare file list stat structure that stores information about a file or directory

    Define variables required to process client requests
    signal(SIGUSR2,sigusr_handler2);

    while (1)
    {
        Set signal handler(sigint)

        define size of client address
        get the socket file descriptor associated with the client

        store ip address

        Make an exception when read function return 0
        Call client_info function to get client information
        print the client's IP address and port number with time
        idle_count-1
        kill(getppid(), SIGUSR2)
```

Copy HTTP method information and url information

get current working directory
get path of wow (wow=cwd+url)

Open the file to read all lines, compare it to the ip address, and proceed if the corresponding ip address exists, and if there is no ip address, print an access restriction message

get information of files

open directory
{
    when root directory
    {
        call ls -l function
    }
    when sub directory
    {
        call ls -al function
    }
}

get file info

when directory
{
    MIME_type=text/html
    get size of response message
    print response header
    write response header

```
                write response message
            }
            when image file
            {
                MIME_type=image/*
                get file size
                file open
                file read
                print response header
                write response header
                write response message
                close file descriptor
            }
            when file
            {
                MIME_type=text/plain
                get file size
                file open
                file read
                print response header
                write response header
                write response message
                close file descriptor
            }
        Sleep(5)
        print the client's IP address and port number with time
        kill(getppid(), SIGUSR2)
        close client socket descriptor
    }
}
    close socket descriptor
}
```

```
define print_recent_clients function
{
	print client information
	alarm(10)
}
define file info function about directory
{
	get current directory

	print title current directory
	make table
	make header

	for (n = 0; n < k; n++)
	{
		judge file mode and permission

		get File Owner's Information
		get the information of the file's owning group
		get the last modified time of that file
		function for Parsing Time Information

		when root directory
		{
			when directory hyperlink with blue color :file[n]
			when link file hyperlink with green color :file[n]
			  when file hyperlink with red color :file[n]
		}
		when sub directory
		{
			When the '/' is at the end of the address
			{
```

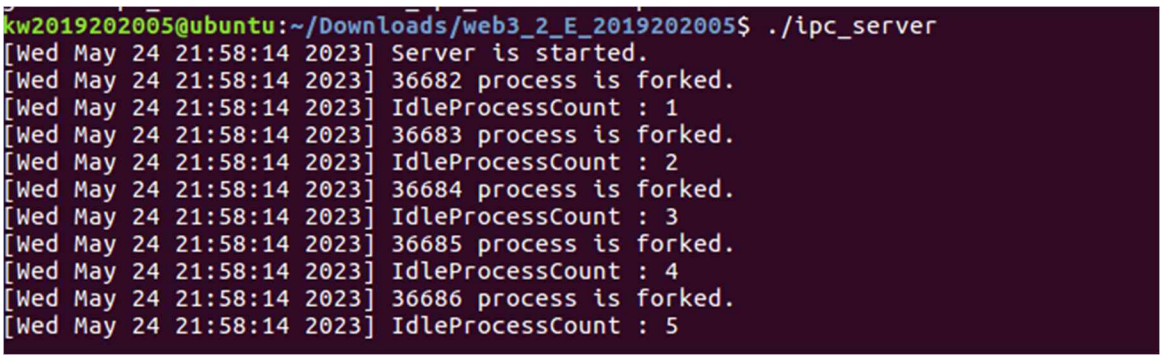
```
                    when directory hyperlink with blue color :url file[n]
                    when link file hyperlink with green color :url file[n]
                      when file hyperlink with red color :url file[n]
                }
                When there is no '/' at the end of the address
                {
                    when directory hyperlink with blue color :url/file[n]
                    when link file hyperlink with green color :url/file[n]
                      when file hyperlink with red color :url/file[n]
                }
            }
        }
        close table
}
define bubble sort function
{
        initialize value
        Bubble sort the file names alphabetically
}
define ls -l function
{
        get current directory
        get current directory+url

        when open directory
        {
            Repeat read directory
            {
                store file names without hidden files
            }

            for (int j = 0; j < i; j++)
```

```
        {
            get current directory+url+/
            get info about current directory+url+/+file
            get block size
            initialize current directory+url+/+file
        }
        print directory path & total size
        call bubble_sort function
        print file info
    }

    close directory
}
define ls -al function
{
    get current directory
    get current directory+url

    when open directory+url
    {
        Repeat read directory
        {
            store file list with hidden file
        }

        for (int j = 0; j < i; j++)
        {
            get current directory+url+/
            get info about current directory+url+/+file
            get block size
            initialize current directory+url+/+file
        }
```

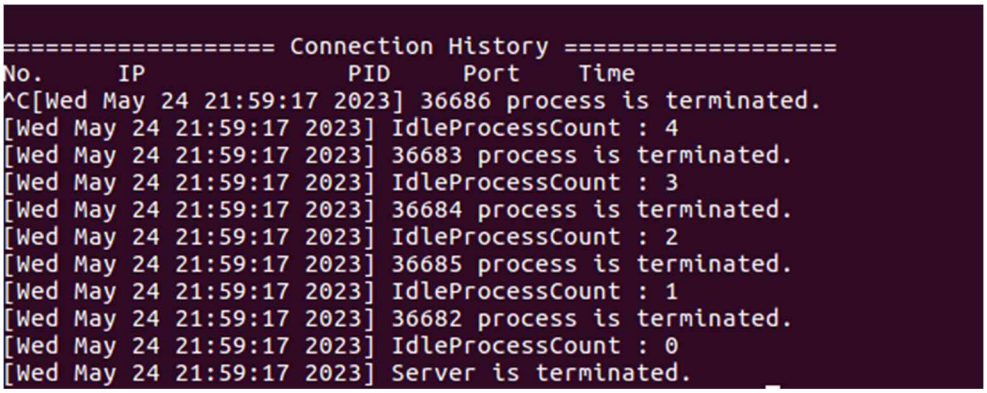
```
            print directory path & toatl size
            call bubble_sort function
            print file info
        }
        close directory
}
```

- **결과화면**

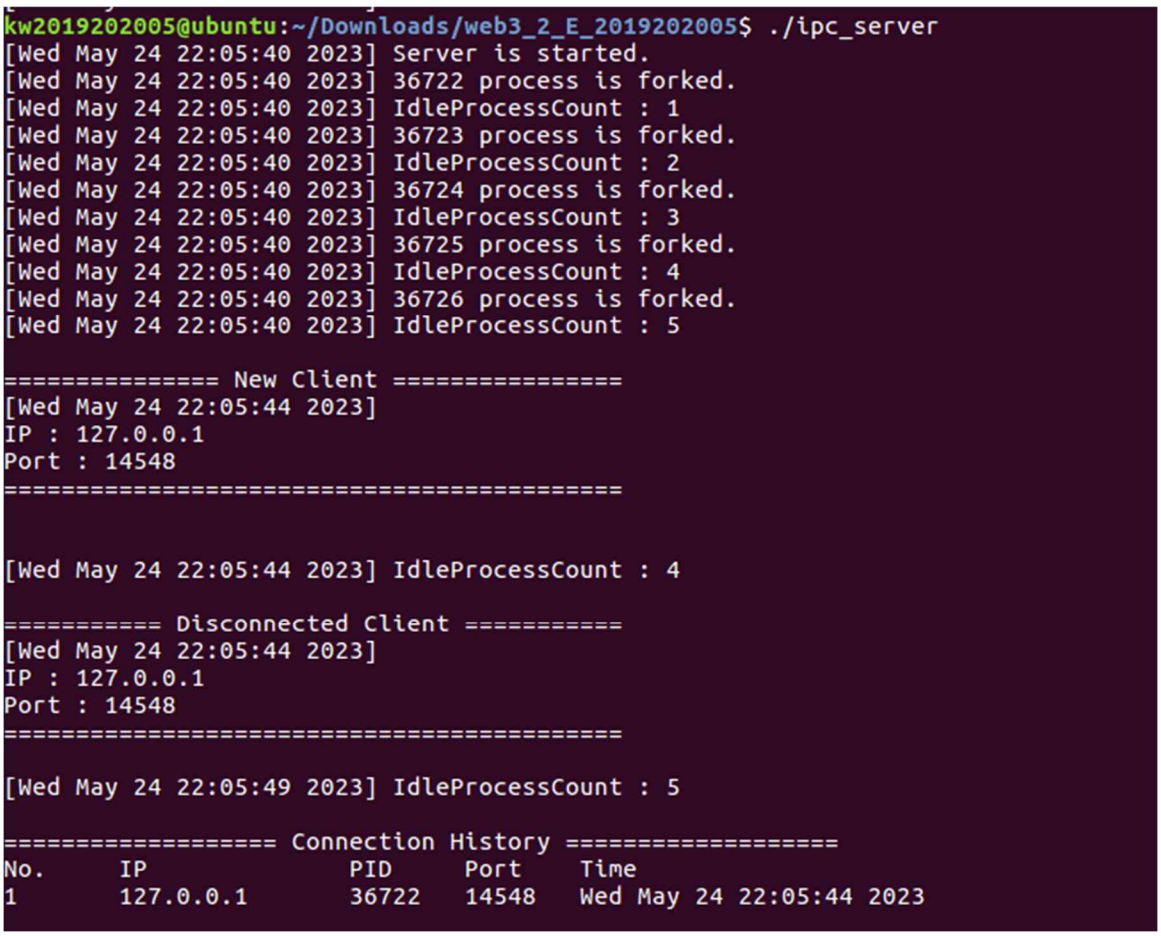
먼저 프로그램을 처음 실행했을 때 출력되는 화면입니다. 저번과제와 동일하게 미리 fork를 5번하여 자식 프로세스를 5개 만드는 모습을 확인할 수 있으며 클라이언트와 연결되지 않은 프로세스인 idle process의 수 가 fork될 때 마다 1씩 증가하는 모습을 확인할 수 있습니다.



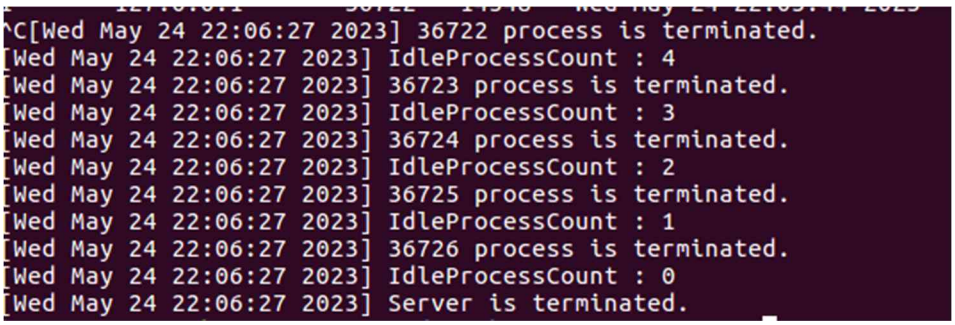
클라이언트의 연결 없이 프로그램이 종료될 때의 화면입니다. 먼저 10초 후 history가 출력되는데 연결된 클라이언트가 없어 아무것도 출력되지 않은 모습입니다. 그 후 프로세스가 종료될 때 마다 idle process의 개수가 1씩 줄어드는 모습을 확인할 수 있으며 서버가 종료되는 출력문도 확인할 수 있습니다.
Idle process의 수가 변경되는 경우 signal을 이용해 부모 프로세스에서 출력하도록 처리했습니다.

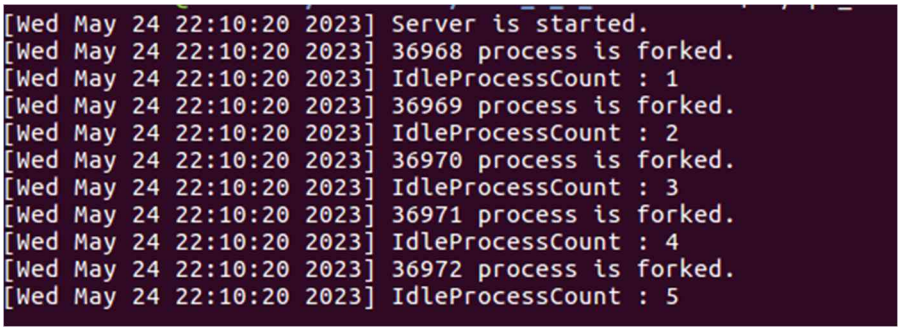


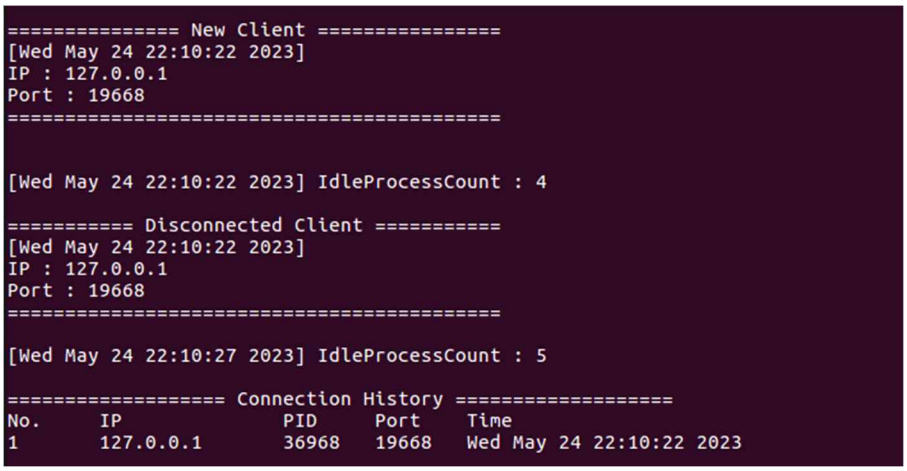
프로그램 실행 후 클라이언트가 연결된 모습을 확인할 수 있습니다. 연결이 되면 idle process의 개수가 1감소하며 클라이언트의 연결이 종료된 경우에는 다시 idle process의 개수가 1증가하는 모습을 확인할 수 있습니다. 그리고 연결된 클라이언트의 history에 이전에 연결된 클라이언트의 기록이 출력되는 모습입니다.

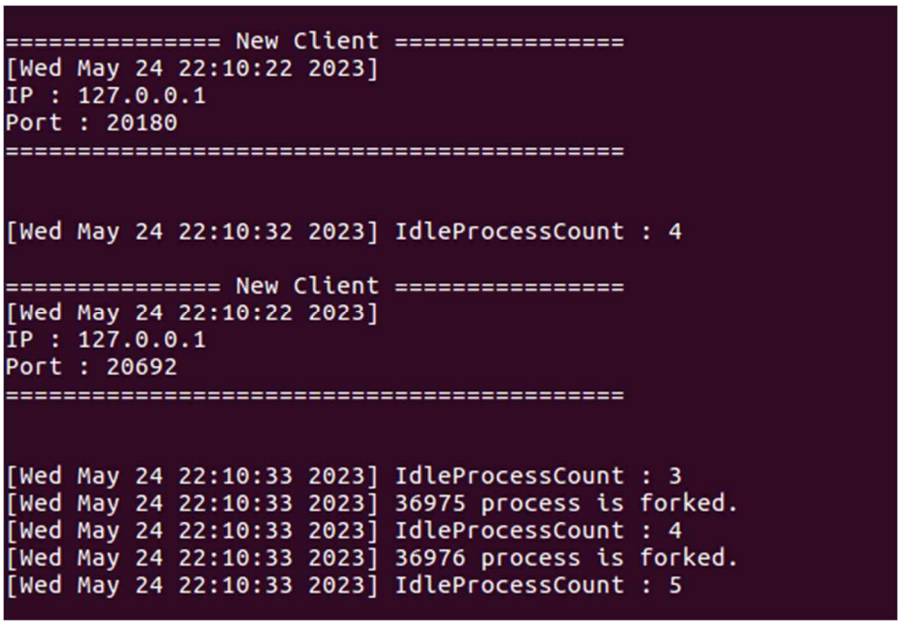


그 후 종료 또한 잘 출력되는 모습을 확인할 수 있습니다.





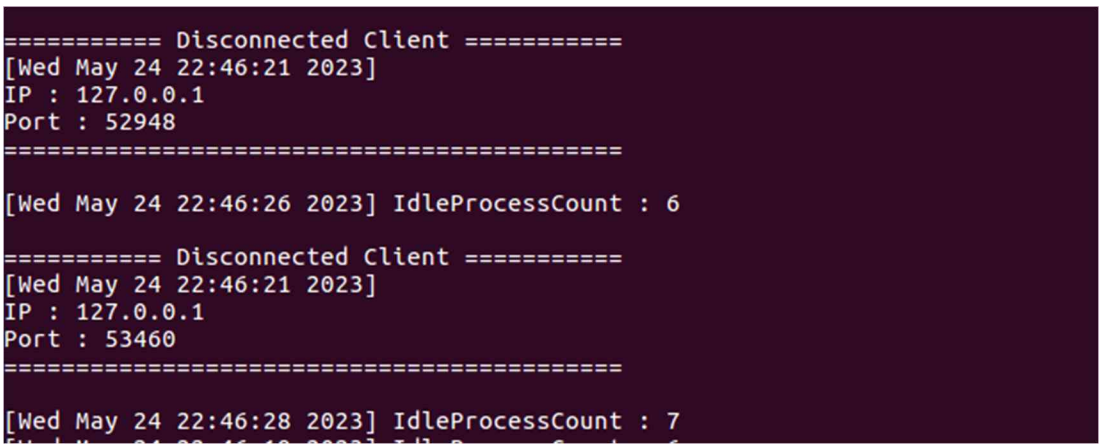


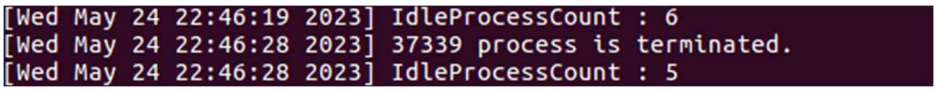


위의 두 화면은 이전에 설명한 클라이언트가 하나 연결되었을 때의 상황입니다. Idle process의 개수가 변하고 클라이언트의 연결 기록을 확인할 수 있습니다. 그 후 새로운 클라이언트가 연결된 상황이 세번째 화면입니다. 먼저 하나의

클라이언트가 연결되면 idle process의 개수가 4가 되며 이때 이 연결된 클라이언트가 연결이 끊기기 전에 다른 클라이언트가 연결되면 idle process의 개수가 하나가 더 줄어 3이 되는 모습을 확인할 수 있습니다. 현재 연결된 클라이언트가 2개이므로 연결되지 않은 클라이언트의 개수인 idleprocesscount의 개수가 3인 것을 알 수 있습니다.
하지만 과제의 조건에서 idleprocesscount가 4보다 작게 되면 다시 fork를 통해 5로 맞춰줘야 하므로 fork를 통해 idle process의 개수가 다시 5가 된 모습을 확인할 수 있습니다.

그리고 연결되었던 클라이언트가 연결이 해제되며 idle process가 늘어나는 모습이며 과제의 조건에 따라 idle process의 개수가 7을 넘어서면 프로세스의 종료를 통해 idle process의 개수를 5로 맞춰줘야 합니다.





history출력 후 프로세스들과 서버가 잘 종료되는 모습을 확인할 수 있습니다.

```
=================== Connection History ===================
No.     IP              PID     Port    Time
1       127.0.0.1       37331   53972   Wed May 24 22:46:34 2023
2       127.0.0.1       37330   53460   Wed May 24 22:46:23 2023
3       127.0.0.1       37329   52948   Wed May 24 22:46:21 2023
^C[Wed May 24 22:47:03 2023] 37341 process is terminated.
[Wed May 24 22:47:03 2023] IdleProcessCount : 4
[Wed May 24 22:47:03 2023] 37340 process is terminated.
[Wed May 24 22:47:03 2023] IdleProcessCount : 3
[Wed May 24 22:47:03 2023] 37333 process is terminated.
[Wed May 24 22:47:03 2023] IdleProcessCount : 2
[Wed May 24 22:47:03 2023] 37332 process is terminated.
[Wed May 24 22:47:03 2023] IdleProcessCount : 1
[Wed May 24 22:47:03 2023] 37331 process is terminated.
[Wed May 24 22:47:03 2023] IdleProcessCount : 0
[Wed May 24 22:47:03 2023] 37330 process is terminated.
[Wed May 24 22:47:03 2023] Server is terminated.
```

그리고 클라이언트의 연결이 여러 개 들어온 경우에는 제일 최근에 연결된 클라이언트의 정보가 number 1에 출력되게 처리했습니다.

```
=================== Connection History ===================
No.     IP              PID     Port    Time
1       127.0.0.1       37179   33492   Wed May 24 22:32:17 2023
2       127.0.0.1       37178   32980   Wed May 24 22:32:15 2023
3       127.0.0.1       37177   32468   Wed May 24 22:32:09 2023
4       127.0.0.1       37176   31956   Wed May 24 22:32:07 2023
5       127.0.0.1       37175   31444   Wed May 24 22:32:05 2023
```

고찰

이번 과제에서 공유 메모리와 스레드 등 새로운 개념에 관한 내용이 갑자기 너무 많이 나와 과제를 진행하는데 어려움이 많았습니다. 애초에 처음 접하기도 하고 이를 코드에 직접 사용해 보려고 하니까 시행착오가 많았던 것 같습니다.
처음에 httpd.conf 파일을 만들지 않고
먼저 공유 메모리를 사용하여 idle process의 수를 관리하는데 이때 공유 메모리에 접근하기 위해서는 스레드를 생성하여 이를 활용하였습니다.
이를 통해 idle process의 수를 관리하면서 MinIdleNum개 미만이 되거나 MaxIdleNum을 초과하면 자식 프로세스를 생성 및 종료하여 5개를 유지했습니다.
공유메모리를 사용하면서 여러 개의 스레드가 접근하고 idle process의 수를 수정할 때 이를 동기화하기 위해서 pthread_mutex_lock을 통해 idle process의 수를 증가 및 감소시키고 pthread_mutex_unlock을 통해 동기화를 완료했습니다.
처음에 스레드를 사용하기 위해 pthread_create함수를 호출하여 사용했는데 idle_count의 동기화 부분에 오류가 발생하여 값이 제대로 출력되지 않는 상황이 발생했습니다. Pthread_join함수 호출 없이 사용하고 있었는데 이 부분에서 오류 발생하여 값이 이상하게 출력 되는 모습을 확인할 수 있었고 함수 호출을 통해 문제를 해결할 수 있었습니다. 그리고 history를 출력할 때 계속 아무것도 출력하지 않는 오류가 있었는데 client의 정보를 공유 메모리에 담아 오지도 않고 공유 메모리에 있는 정보를 출력하고 있어 오류가 발생하는 것이었습니다.
그래서 공유 메모리의 다른 index를 참조해 한쪽 index는 idle_process의 개수를 처리하고 다른 한 부분은 client의 정보를 담아와 출력하도록 처리했습니다.
그리고 과제를 진행하면서 중간에 공유 메모리와 스레드를 사용하는 과정에서 프로그램 실행 시 우분투 자체가 종료되어 버리는 등 큰 문제가 발생하여 계속 코드를 갈아엎고 가상머신을 새로 설치하는 등 지금까지의 과제에서 제일 시행착오가 많았던 것 같습니다.
그래도 처음에 공유 메모리와 스레드의 개념이 이해가 가지 않아 많이 찾아보기도 하였으며 이를 통해 수월하게 이해할 수 있었고 위 개념을 활용해 공유 자원을 동기화하여 관리할 수 있다는 점을 알 수 있었습니다.

Reference

2023년 1학기 시스템프로그래밍 & 시스템 프로그래밍 실습 강의자료
Assignment 3-2

2023년 1학기 시스템 프로그래밍 1학기 강의자료 3. Files and directories
2023년 1학기 시스템 프로그래밍 1학기 강의자료 6. sockets
2023년 1학기 시스템 프로그래밍 실습 12주차 강의자료 Process pool management