Assignment

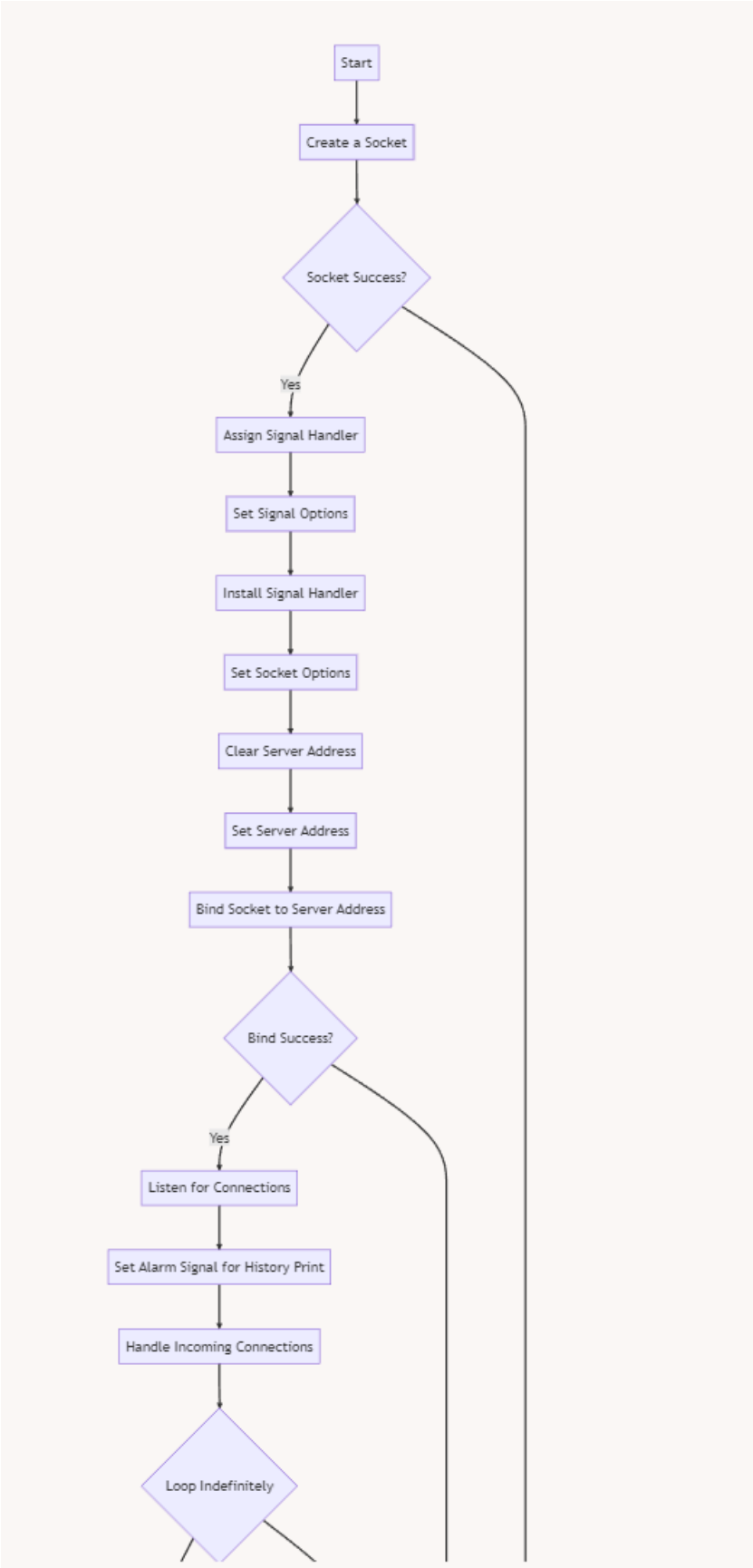
3-1

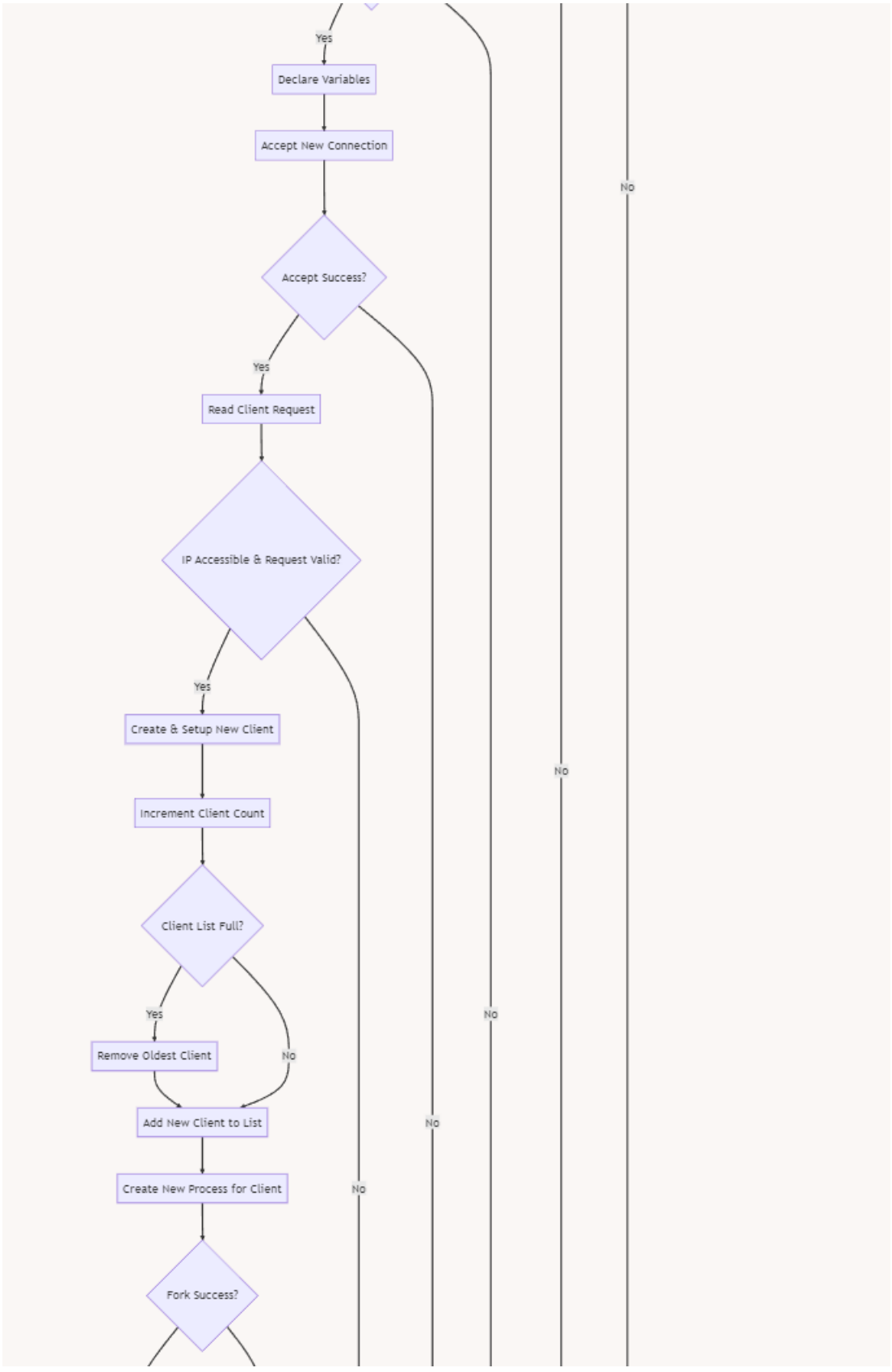
과 목 명	시스템프로그래밍실습
학 과	컴퓨터정보공학부
학 번	2020202096
성 명	우성원
교 수 님	최상호교수님
제 출 일	2023.05.16

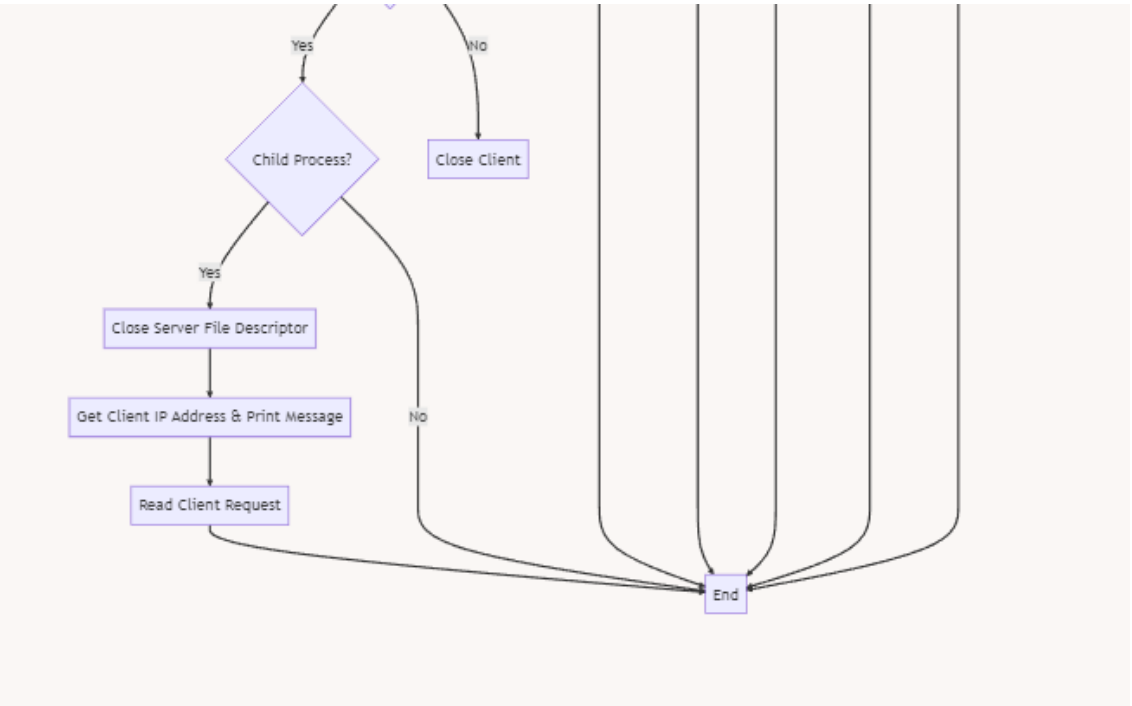
◆ Introduction

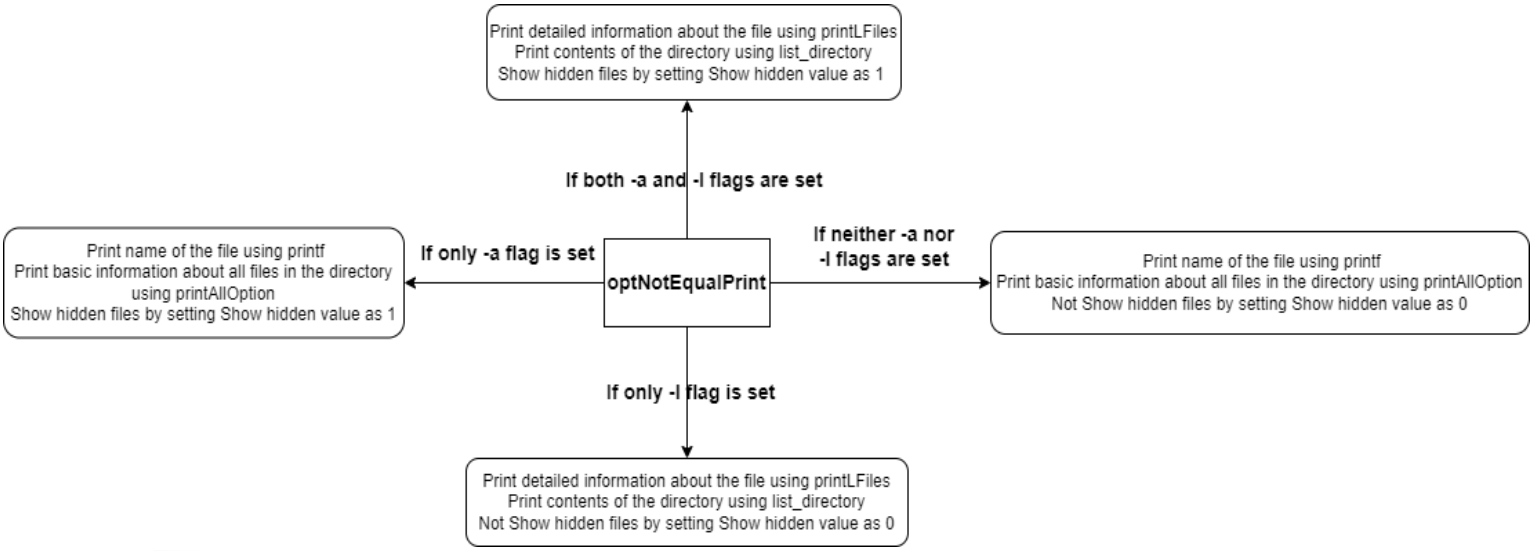
이 과제는 2-3과제에서 만든 다중 접속 지원 서버를 기반으로 pre-forked 서버를 만드는 과제이다.5개 child프로세스를 생성 및 유지해야한다.각 자식 프로세스는 10개의 클라이언트의 접속을 기록하고 10초에 한번씩 각 자식프로세스에서 연결된 history를 출력한다.클라이언트의 관리는 구조체를 통해서 구현하였으며 각 자식 프로세스마다 전역으로 선언된 클라이언트 구조체 배열을 통해 정보를 저장하고 각 자식프로세서에서 해당 정보를 alarm 시그널이 들어왔을 때 출력한다.기본적으로 pre-fork는 부모프로세스에서 서버 실행시 fork를 5번 수행하면서 동작하게끔 설정하였다.그리고 termination 및 생성시에 모든 서버 및 자식프로세스는 해당 시간과 함께 생성,소멸을 알린다.

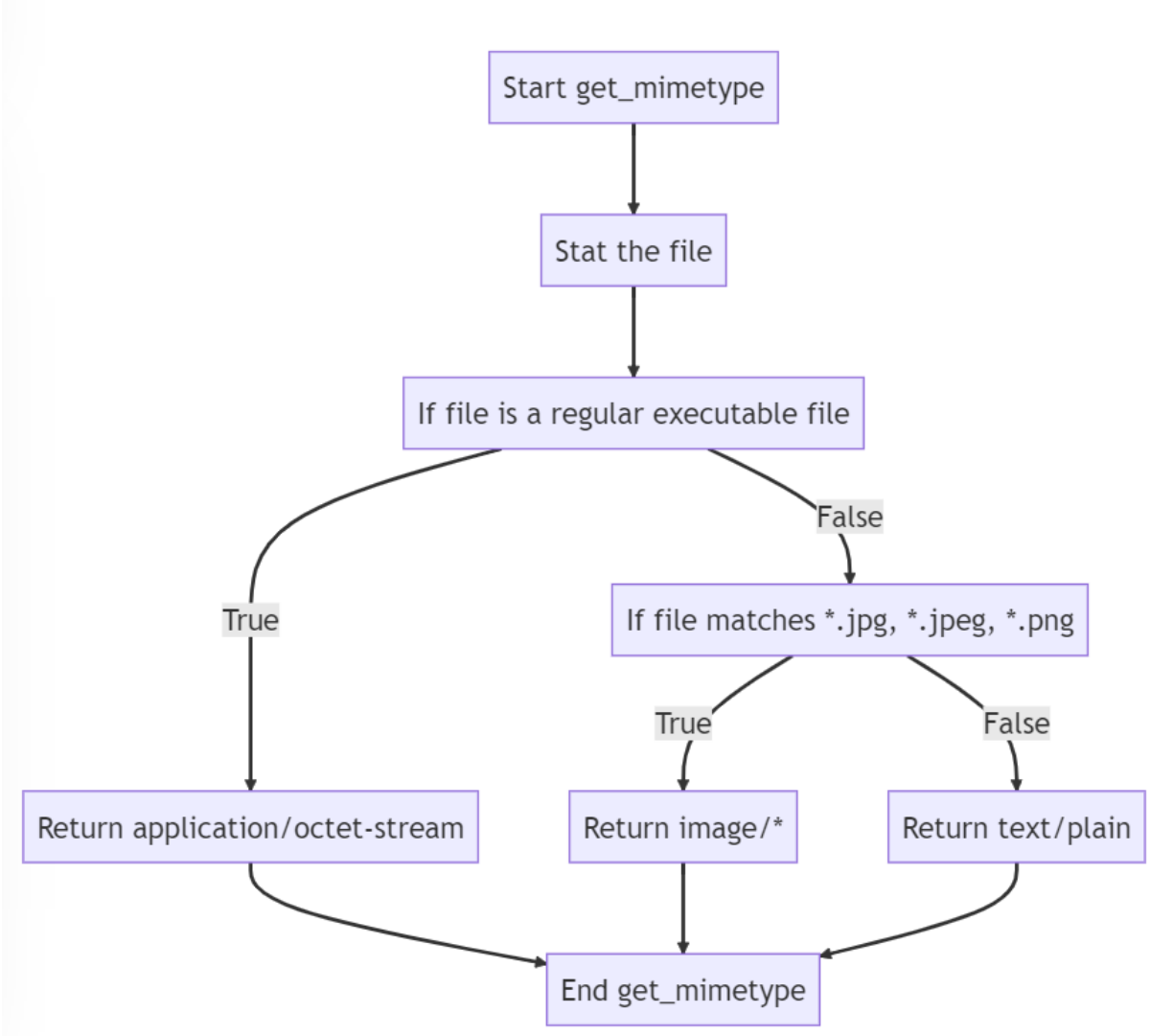
Flowchart

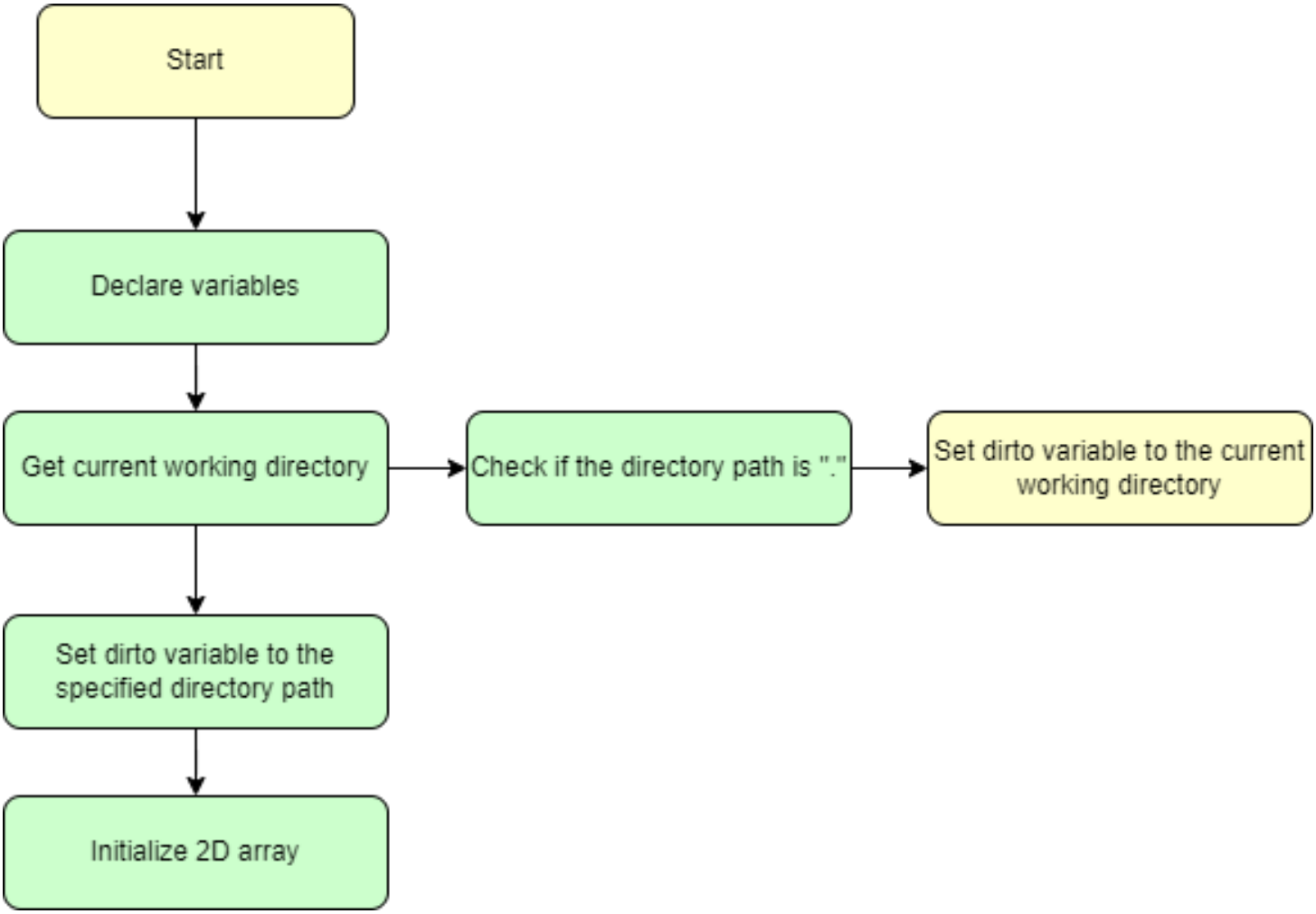

Start
Create a Socket
Socket Success?
Yes
Assign Signal Handler
Set Signal Options
Install Signal Handler
Set Socket Options
Clear Server Address
Set Server Address
Bind Socket to Server Address
Bind Success?
Yes
Listen for Connections
Set Alarm Signal for History Print
Handle Incoming Connections
Loop Indefinitely

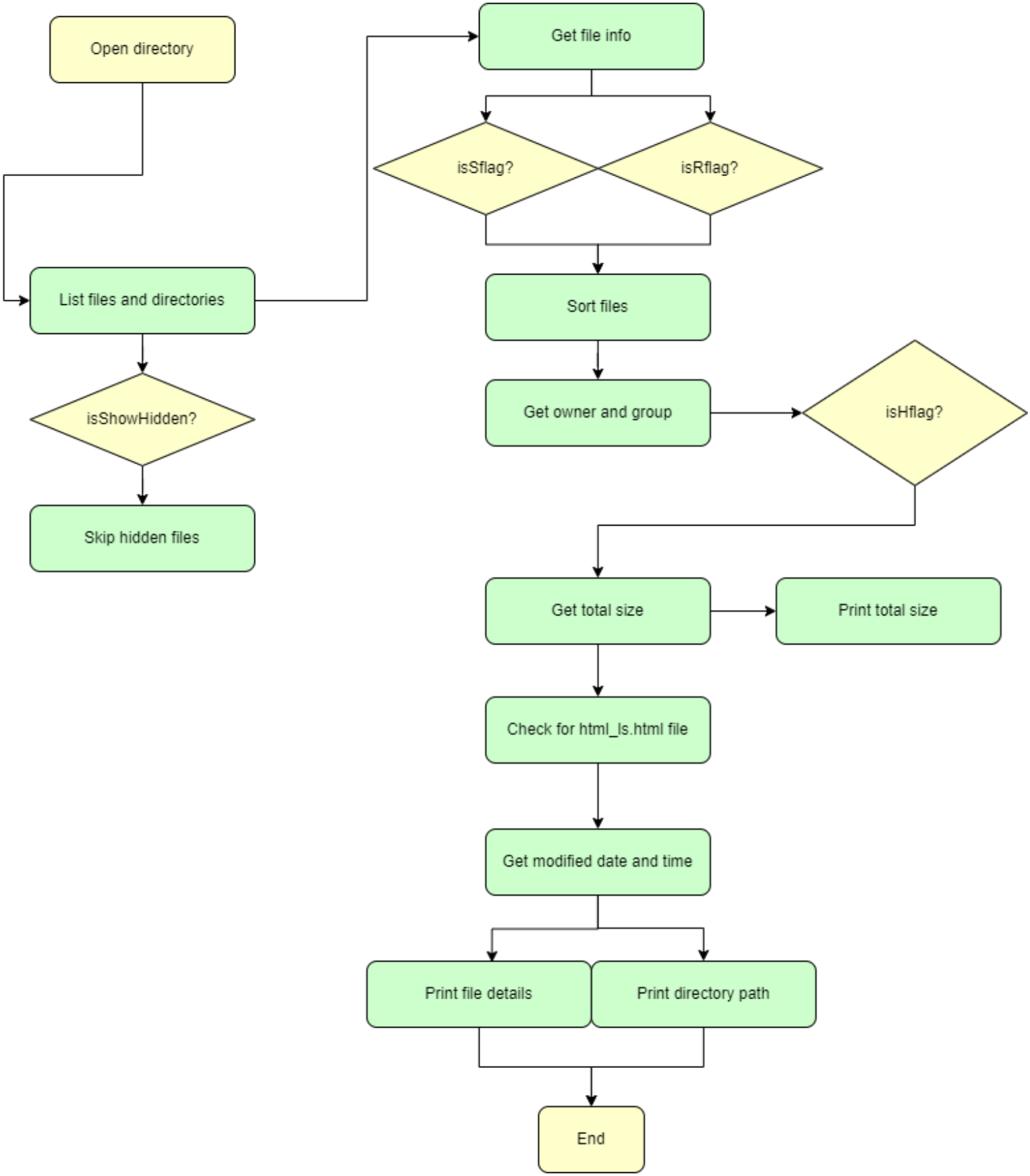

Yes
Declare Variables
Accept New Connection
Accept Success?
No
Yes
Read Client Request
IP Accessible & Request Valid?
No
Yes
Create & Setup New Client
Increment Client Count
Client List Full?
No
Yes
Remove Oldest Client
No
Add New Client to List
No
Create New Process for Client
No
Fork Success?

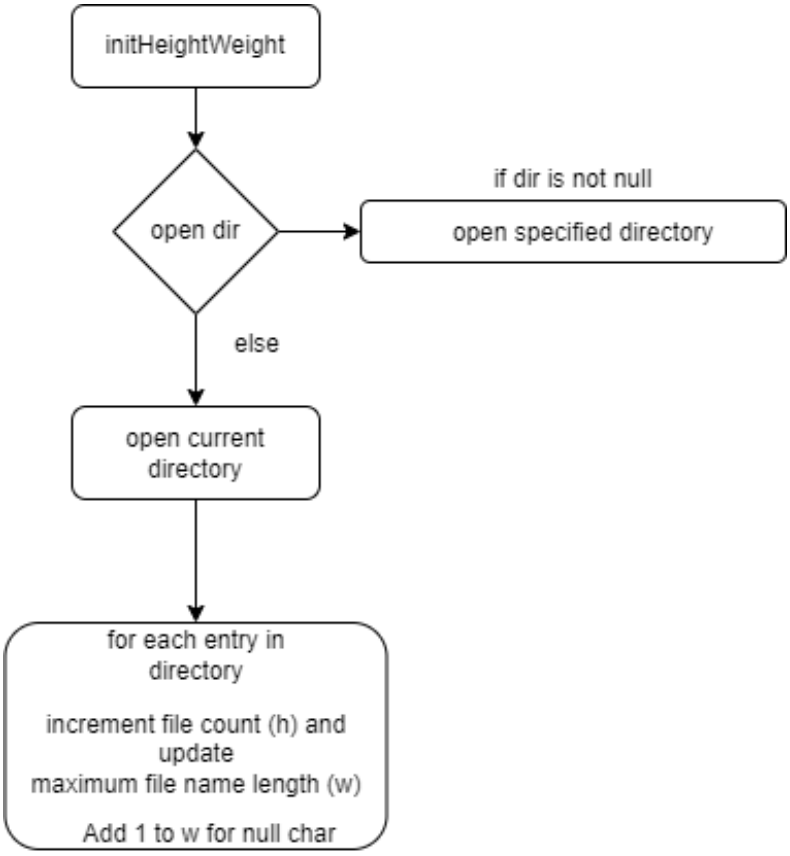

Yes
No
Child Process?
Close Client
Yes
Close Server File Descriptor
Get Client IP Address & Print Message
No
Read Client Request
End

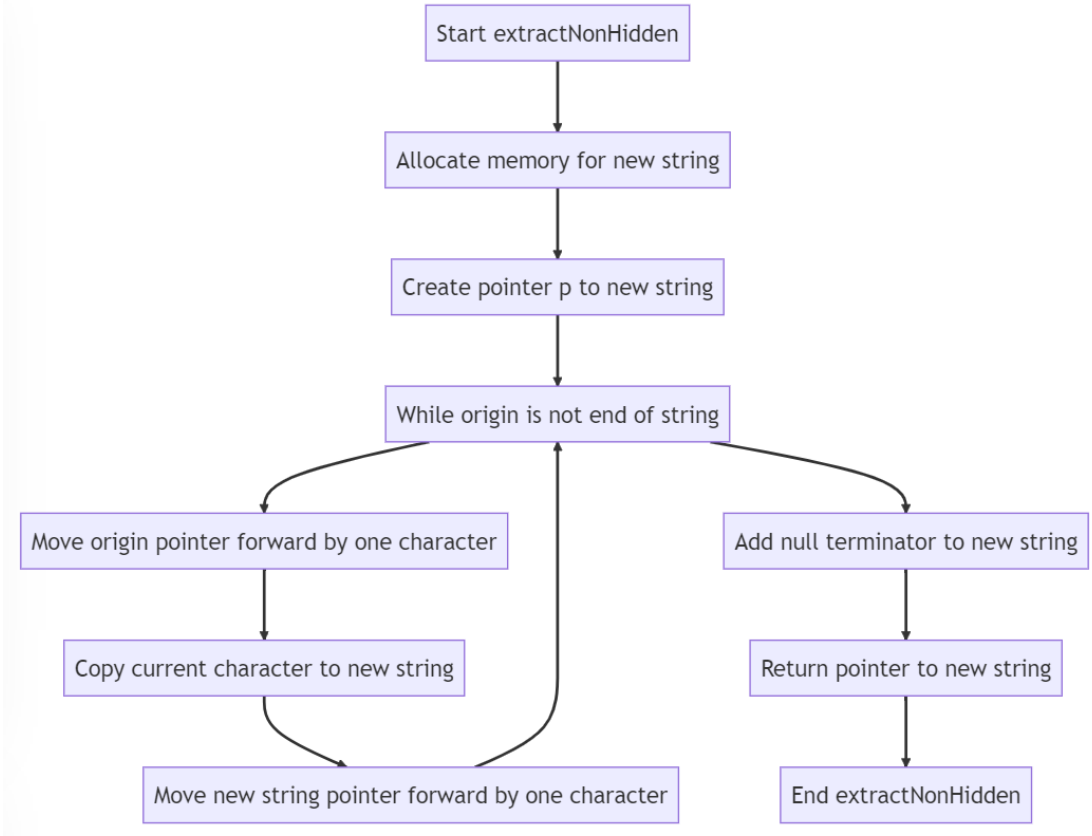

Print detailed information about the file using printLFiles
Print contents of the directory using list_directory
Show hidden files by setting Show hidden value as 1
If both -a and -l flags are set
Print name of the file using printf
Print basic information about all files in the directory using printAllOption
Show hidden files by setting Show hidden value as 1
If only -a flag is set
optNotEqualPrint
If neither -a nor -l flags are set
Print name of the file using printf
Print basic information about all files in the directory using printAllOption
Not Show hidden files by setting Show hidden value as 0
If only -l flag is set
Print detailed information about the file using printLFiles
Print contents of the directory using list_directory
Not Show hidden files by setting Show hidden value as 0

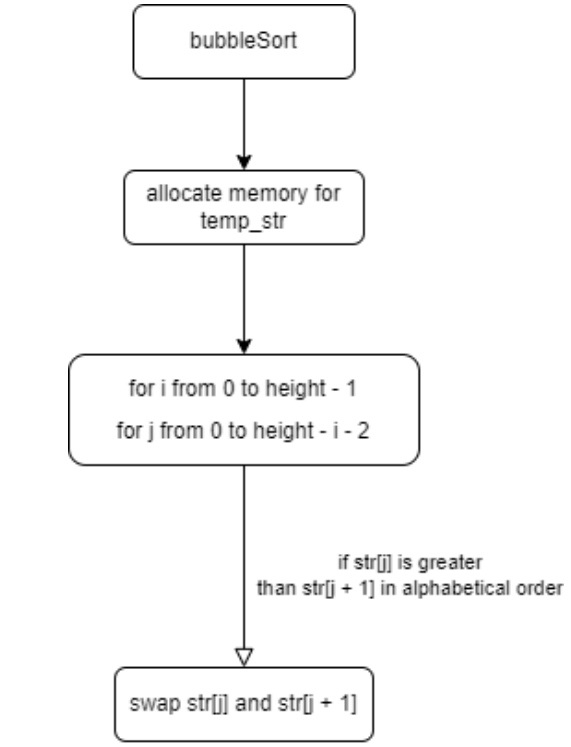

Start get_mimetype
Stat the file
If file is a regular executable file
True
False
If file matches *.jpg, *.jpeg, *.png
True
False
Return application/octet-stream
Return image/*
Return text/plain
End get_mimetype

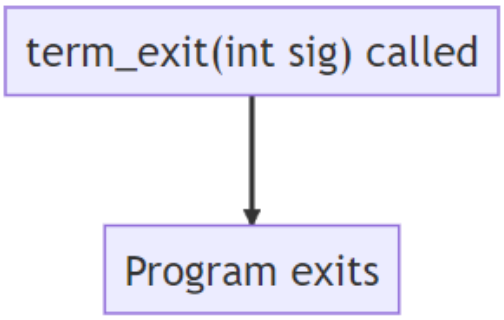

Start
Declare variables
Get current working directory
Check if the directory path is "."
Set dirto variable to the current working directory
Set dirto variable to the specified directory path
Initialize 2D array

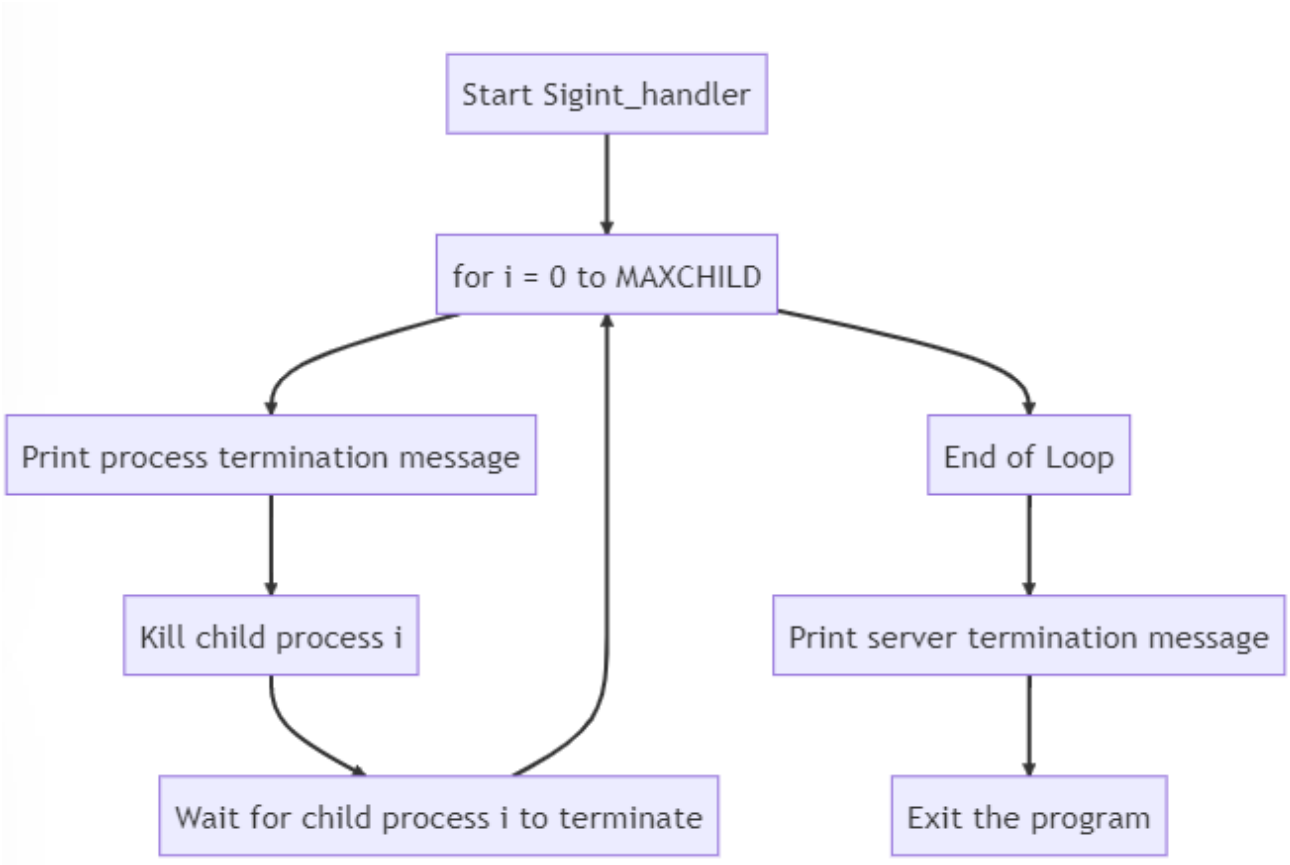

Open directory
Get file info
isSflag?
isRflag?
List files and directories
Sort files
isShowHidden?
Get owner and group
isHflag?
Skip hidden files
Get total size
Print total size
Check for html_ls.html file
Get modified date and time
Print file details
Print directory path
End

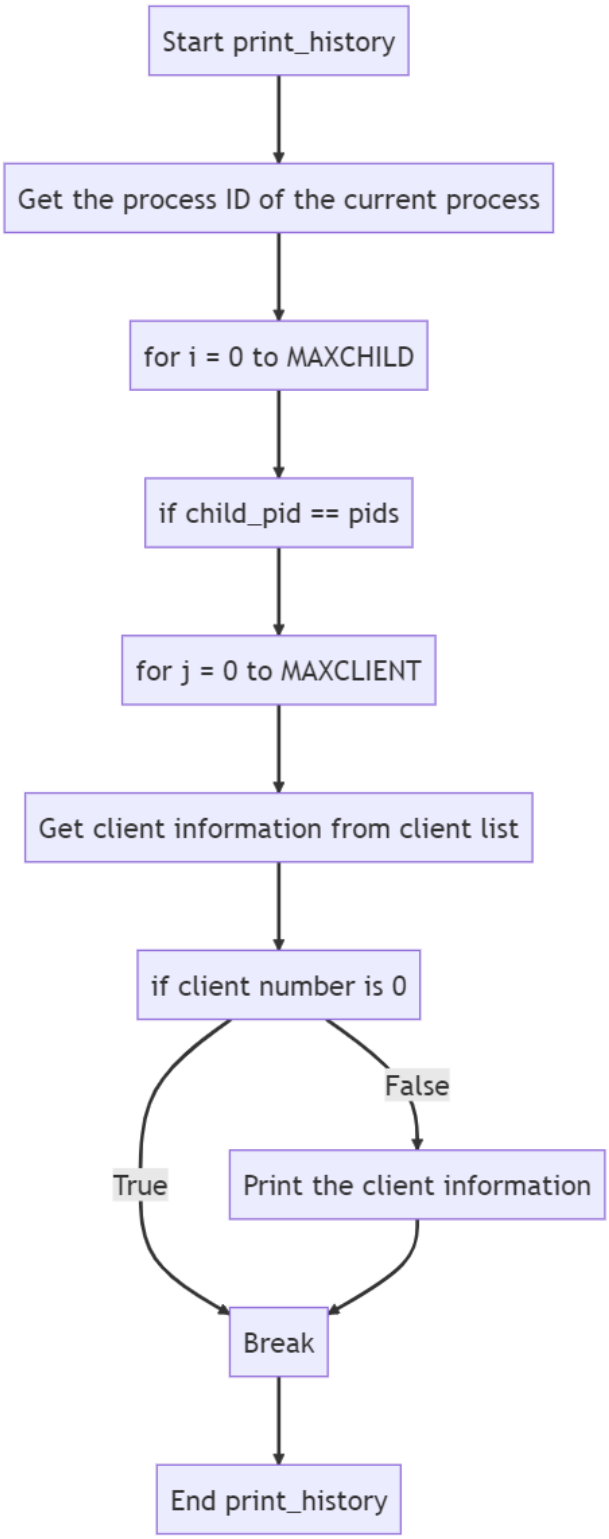

initHeightWeight
open dir
if dir is not null
open specified directory
else
open current directory
for each entry in directory
increment file count (h) and update maximum file name length (w)
Add 1 to w for null char

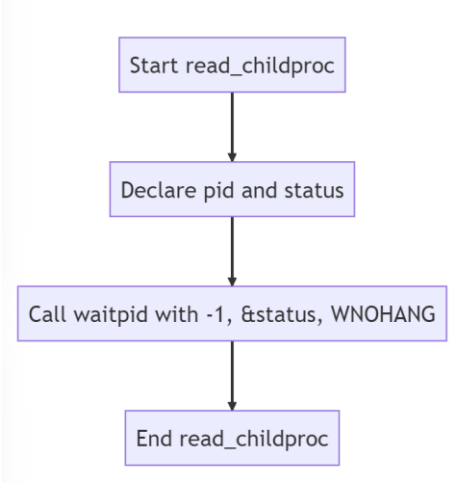

Start extractNonHidden
Allocate memory for new string
Create pointer p to new string
While origin is not end of string
Move origin pointer forward by one character
Copy current character to new string
Move new string pointer forward by one character
Add null terminator to new string
Return pointer to new string
End extractNonHidden

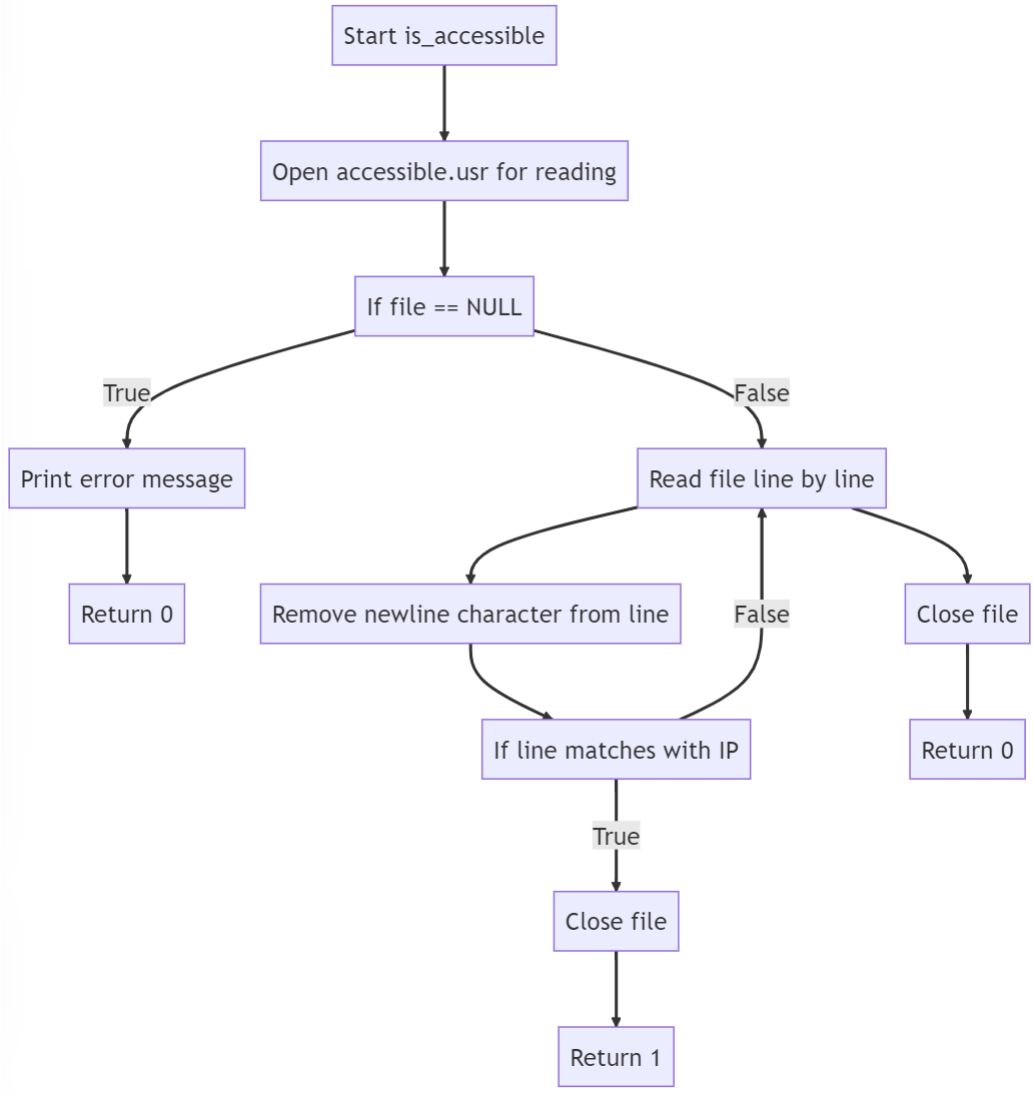

bubbleSort
allocate memory for temp_str
for i from 0 to height - 1
for j from 0 to height - i - 2
if str[j] is greater than str[j + 1] in alphabetical order
swap str[j] and str[j + 1]

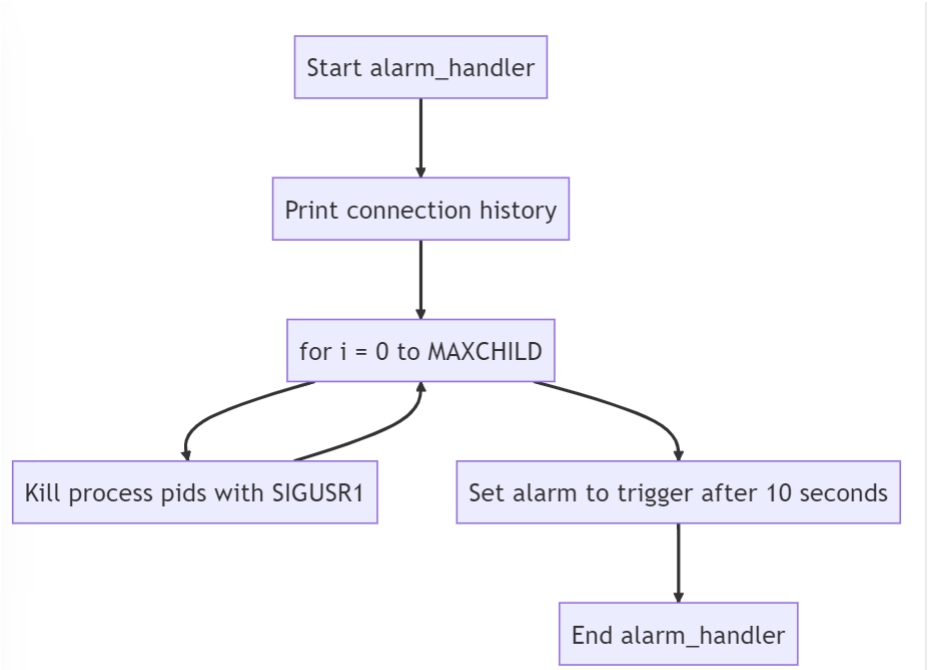

term_exit(int sig) called
Program exits

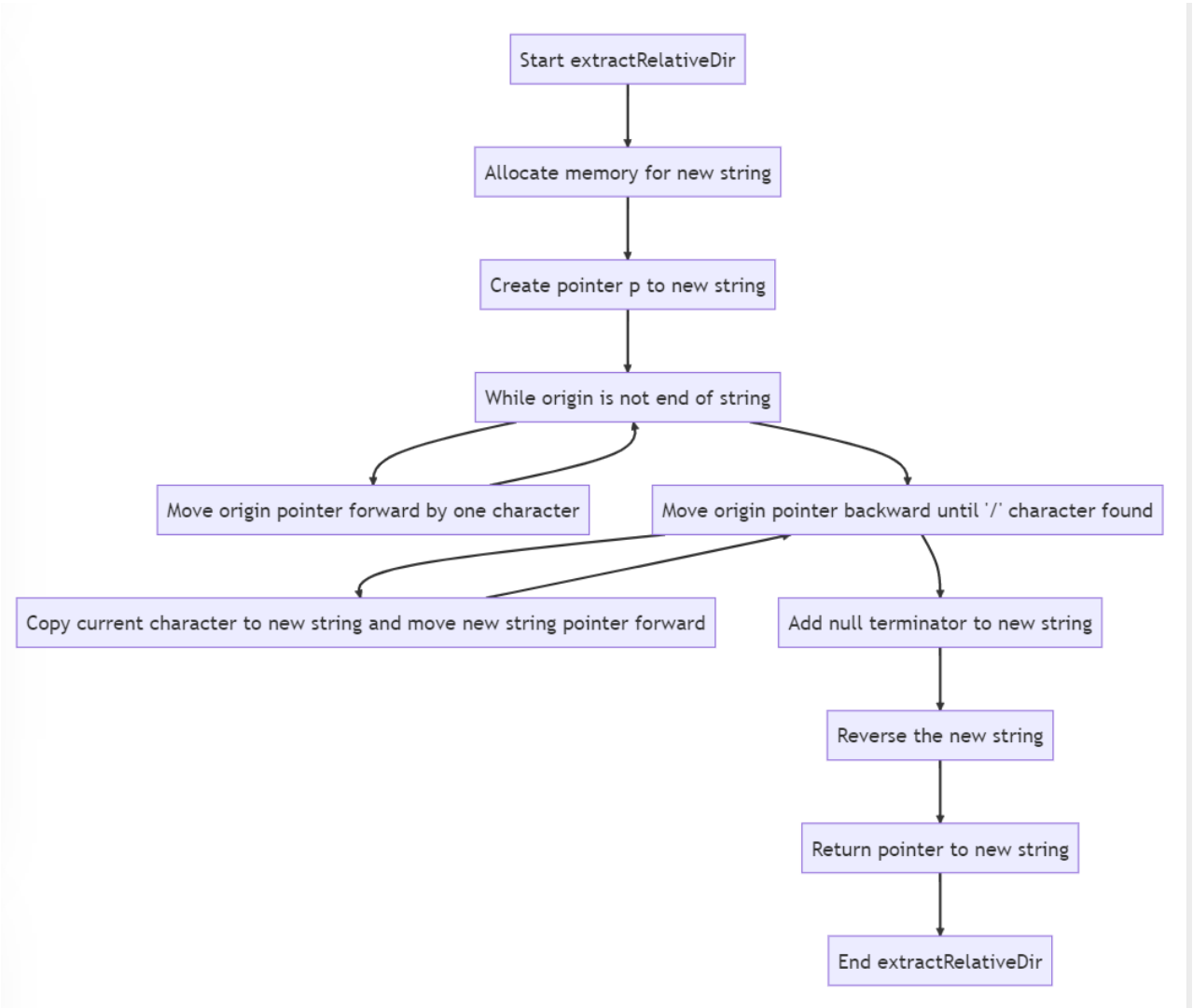

Start Sigint_handler
for i = 0 to MAXCHILD
Print process termination message
Kill child process i
Wait for child process i to terminate
End of Loop
Print server termination message
Exit the program


Start print_history
Get the process ID of the current process
for i = 0 to MAXCHILD
if child_pid == pids
for j = 0 to MAXCLIENT
Get client information from client list
if client number is 0
False
True
Print the client information
Break
End print_history


Start read_childproc
Declare pid and status
Call waitpid with -1, &status, WNOHANG
End read_childproc


Start is_accessible
Open accessible.usr for reading
If file == NULL
True
False
Print error message
Return 0
Read file line by line
Remove newline character from line
False
Close file
Return 0
If line matches with IP
True
Close file
Return 1


Start alarm_handler
Print connection history
for i = 0 to MAXCHILD
Kill process pids with SIGUSR1
Set alarm to trigger after 10 seconds
End alarm_handler


Start extractRelativeDir
Allocate memory for new string
Create pointer p to new string
While origin is not end of string
Move origin pointer forward by one character
Move origin pointer backward until '/' character found
Copy current character to new string and move new string pointer forward
Add null terminator to new string
Reverse the new string
Return pointer to new string
End extractRelativeDir

◆ Pseudo code

◆ Main

main_function

create a socket, and check for failure

assign the signal handler function (read_childproc) to the sa_handler field of the sigaction struct

clear all bits in the signal set represented by sa_mask

set the sa_flags field of the sigaction struct to 0

install the signal handler for the SIGCHLD signal using the sigaction function

set socket options

clear server address

set server address with the appropriate family, IP, and port

bind socket to server address, and check for failure

listen for connections

add signal for alarm to print history every 10 seconds

handle incoming connections

add a handler for the SIGINT signal that will execute the sigint_handler function

start a loop to pre-fork child processes up to the maximum number of child processes (MAXCHILD)

if in the parent process, print a message indicating a process has been forked and store the child process ID

if in the child process, get the current time, format it as a string, and store the process ID of the child process

enter an infinite loop to handle incoming connections

add handlers for the SIGUSR1 (execute print_history function), SIGTERM (execute term_exit function), and SIGINT (ignore the signal) signals

declare variables for client IP address, buffer, buffer size, stream, and other temporary buffers and pointers

accept a new connection, and check for failure

read client request

if IP is accessible and the request is valid, create and set up a new client

increment client count, and store client information in a new client struct

if client_list is full, remove the oldest client

add the new client to client_list

create a new process for the new client

get client IP address and print a message

read client request

if accept fails, print an error message and return 0

get client IP address and print a message with connected client information

read client request into buf

if reading is successful

declare variables: curdir, and isBackward

get current working directory and store in curdir

if getting working directory fails, print an error message

copy request from buf to tmp

print request message from the client

parse request method and URL, store in method and url variables

if the method is GET, extract the URL without the first character and update theisBackward flag

create a title for the HTML response with the client IP address, port, and requested URL

if the requested URL is empty, set it to the current directory

print the requested URL

check the status of the requested URL with lstat

if it fails, print a 404 Not Found error message and generate an HTML response with the error message

otherwise, handle the requested URL

if the URL is the current directory, set arguments to list files in the directory

if the URL is a directory, set arguments to list files in the directory

if the URL is a file

get the MIME type of the requested file

if the MIME type is supported, send the file contents as an HTTP response

if the file cannot be opened, print a 404 Not Found error message and generate an HTML response with the error message

if the MIME type is not supported, print an Unsupported Media Type error message and generate an HTML response with the error message

close the client file descriptor and continue to the next iteration

send an HTTP response header with a 200 OK status and Content-Type

generate the HTML structure for the response, including the title and a welcome message

if argc is 1

initialize HTML file

initialize current directory

```
write HTML headers and title
check for command line options
allocate memory for optarr

parse command line options using getopt
    if -a option
        set a_flag
    if -l option
        set l_flag
    if -h option
        set h_flag
    if -S option
        set S_flag
    if -r option
        set r_flag

store non-option arguments in optarr
if argument contains wildcard characters
    perform globbing to get list of matching files
    copy each matching file to optarr
else if invalid file path
    print error message

if argc is equal to optind
    call respective functions based on options
else if optcnt > 0
    for each option in optarr
        check if it is a file or folder
        increment filecnt or foldercnt accordingly

    allocate memory for filearray and folderarray
    store files and folders in respective arrays
    sort the arrays alphabetically using bubbleSort

    print output according to options
```

```
    return 0
```

◆ bubbleSort

```
function sizeBubbleSort(sizeary, str, height, weight, isRflag):

    Allocate memory for 'temp_str'

    For i in range 0 to height - 1
        Set 'temp_size' to 0

        For j in range 0 to height - i - 2
            If 'isRflag' is 0 and sizeary[j] < sizeary[j + 1]
                Swap str[j] and str[j + 1]
                Swap sizeary[j] and sizeary[j + 1]
            Else if 'isRflag' is 1 and sizeary[j] > sizeary[j + 1]
                Swap str[j] and str[j + 1]
                Swap sizeary[j] and sizeary[j + 1]
            Else if 'isRflag' is 1 and sizeary[j] == sizeary[j + 1]
                Handle special cases for hidden files and sorting

                If strcasecmp(str[j], str[j + 1]) < 0
                    Swap str[j] and str[j + 1]
                    Swap sizeary[j] and sizeary[j + 1]
            Else if 'isRflag' is 0 and sizeary[j] == sizeary[j + 1]
                Handle special cases for hidden files and sorting

                If strcasecmp(str[j], str[j + 1]) > 0
                    Swap str[j] and str[j + 1]
                    Swap sizeary[j] and sizeary[j + 1]

    Free memory for 'temp_str'
```

◆ initHeightWeight

```
function initHeightWeight(w, h, dir, isShowHidden)
    t_dirp <- null
```

```
    t_dir <- null

    if dir is not null
        open specified directory and store handle in t_dirp
    else
        open current directory and store handle in t_dirp

    if isShowHidden is true
        for each entry t_dir in the directory t_dirp
            increment h
            update w with the maximum length of the file name and the current w value
    else
        for each entry t_dir in the directory t_dirp
            if the file name does not start with '.'
                increment h
                update w with the maximum length of the file name and the current w
value

    increment w to add space for null character
    close the directory t_dirp
```

◆ optNotEqualPrint

```
function optNotEqualPrint(htmlfile, a_flag, l_flag, foldercnt, filecnt, folderarray, filearray, optcnt,
isHflag, isSflag, isRflag)
    initialize filecur and foldercur counters

    if both a_flag and l_flag are set
        loop through all options
            if files are left
                print detailed information about the file using printLFiles
                increment filecur and decrement filecnt
            else if no more files but there are folders left
                list contents of the directory using list_directory
                increment foldercur and decrement foldercnt
```

else if only a_flag is set
if files are left
print HTML headers for directory path and table
loop through all options
print name of the file using printAllOption
increment filecur and decrement filecnt
print closing HTML table tag
loop through all options
if no more files but there are folders left
print basic information about all files in the directory using printAllOption
increment foldercur and decrement foldercnt

else if only l_flag is set
loop through all options
if files are left
print detailed information about the file using printLFiles
increment filecur and decrement filecnt
else if no more files but there are folders left
list contents of the directory without showing hidden files using list_directory
increment foldercur and decrement foldercnt

else (neither a_flag nor l_flag are set)
if files are left
print HTML headers for directory path and table
loop through all options
print name of the file using printAllOption
increment filecur and decrement filecnt
print closing HTML table tag
loop through all options
if no more files but there are folders left
print basic information about all files in the directory without showing hidden files by calling printAllOption
increment foldercur and decrement foldercnt

◆ extractNonHidden

function extractNonHidden(origin)

allocate memory for newone string with the same length as origin + 1 for null terminator

set p equal to the beginning of the newone string

for each character in origin, starting from the second character

copy the current character from origin to newone

move both origin and newone pointers forward by one character

add a null terminator to the end of newone

return a pointer to the beginning of the newone string (p)

◆ list_directory

function list_directory(htmlfile, dir_path, isShowHidden, isHflag, isSflag, isRflag)

declare variables and structures (dir, entry, total_stat, file_stat, pwd, grp, date_str, total_size, prev_dir, dirto)

get current working directory (prev_dir)

if dir_path is ".", set dirto to prev_dir, otherwise set dirto to dir_path

initialize w, h, and filename array

initHeightWeight(w, h, dirto, isShowHidden)

allocate memory for filename array

open directory (dirto) and store in dir

if dir is NULL, print "Unable to open directory" and return

populate filename array based on isShowHidden flag

free memory allocated to directory entry (entry)

allocate memory for fileSizeArray

calculate total size of files in directory and store individual sizes in fileSizeArray

if isRflag or isSflag, sort the files by size using sizeBubbleSort

else, sort the files by name using bubbleSort

print directory path and total size to htmlfile
print table headers to htmlfile

for each file in the sorted filename array
skip file if its name is "html_ls.html"
retrieve file information and save into file_stat

print file details to htmlfile (name, permissions, owner, group, size, modification time) in a tabular format

close the table in htmlfile

free memory allocated to filename array and close the directory (dir)
print a newline character

◆ printLFiles

function printLFiles(htmlfile, filename, isHflag, isSflag, isRflag):

Check if 'filename' contains "html_ls.html", and if so, return

Get file information for 'filename' into 'file_stat'
If getting file information fails, print an error message and return

Get user information for 'file_stat.st_uid' into 'pwd'
Get group information for 'file_stat.st_gid' into 'grp'
Format the last modified date and time of 'file_stat' into 'date_str'

Write HTML tags and file information to 'htmlfile':
Directory path
File type and permissions (link, directory, or regular file)
Number of links
Owner and group name
Size with optional human-readable format (based on isHflag)
Last modified date and time

Close HTML tags for the table and row.

◆ printAllOption

function printAllOption(htmlfile, directory, isShowHidden):

Get the current working directory into 'prev_dir'

Set 'dirto' to 'directory' if it's not the current directory

Otherwise, set 'dirto' to 'prev_dir

If opening the directory fails and getting file information also fails

Print error message "cannot access 'directory' : No such directory"

Else

If the filename contains "html_ls.html", return

Write HTML tags and file information to 'htmlfile':

Color code based on file type (link, directory, or regular file)

Filename

Open the directory and store the pointer in 'dirp'

Write HTML tags for the table header to 'htmlfile'

Initialize 'w' and 'h' using 'initHeightWeight' function

Allocate memory for 'filename' array

If 'isShowHidden' is true

Copy all filenames from 'dirp' to 'filename' array

Else

Copy only non-hidden filenames from 'dirp' to 'filename' array

Close the directory using 'closedir' function

Sort the 'filename' array using the 'bubbleSort' function

For each sorted filename in the 'filename' array

If the filename contains "html_ls.html", skip it

Get file information into 'file_stat'

Write HTML tags and file information to 'htmlfile':

Color code based on file type (link, directory, or regular file)

Filename

Close HTML tags for the table

◆ extractRelativeDir

function extractRelativeDir(origin)
allocate memory for newone string with the same length as origin + 1 for null terminator
set p equal to the beginning of the newone string
move the origin pointer to the end of the string
starting from the end of origin, find the last '/' character
    copy the current character from origin to newone
    move the newone pointer forward by one character and origin pointer backward by one character
add a null terminator to the end of newone
reverse the newone string
    create pointers to the beginning and end of the newone string
    swap characters at the beginning and end pointers
    move the beginning pointer forward and end pointer backward
return a pointer to the beginning of the newone string (p)

◆ get_mimetype

function get_mimetype(filename)
use fnmatch to match MIME types (html, txt, image files)
if the filename matches "*.html" pattern (case insensitive)
    return "text/html"

else if the filename matches "*.txt" or "*.c" pattern (case insensitive)
    return "text/plain"
else if the filename matches "*.jpg", "*.jpeg", or "*.png" pattern (case insensitive)
    return "image/*"
else
    return NULL

◆ alarm_hander

On receiving an alarm signal
    print the connection history header
    display the total number of client requests
    print a table header for client details
    for the last 10 (or less) clients connected, in reverse order
        retrieve each client's details from the client list
        display the client's number, IP address, port, PID, and connection time

print the connection history footer

set a new alarm to trigger after 10 seconds

◆ read_childproc

wait for any child process to terminate without blocking (WNOHANG flag)

store the process ID of the terminated child process in variable `pid`

store the exit status of the terminated child process in variable `status`

◆ print_history

function print_history(signal)

get the process ID of the current process and store it in child_pid

for each index in the range of maximum child processes (0 to MAXCHILD-1)

if the current process ID (child_pid) matches the stored child process ID at the index in the pids array

for each index in the range of maximum clients (0 to MAXCLIENT-1)

get the client information at the index in the child's client list and store it in variable c

if the client number (c.no) is 0, indicating the end of the client list

break out of the inner loop

print the client number, IP address, port, PID, and connection time (converted to a string)

◆ sigint_handler

function sigint_handler(signal)

for each index in the range of maximum child processes (0 to MAXCHILD-1)

print a message indicating the termination of the process with its process ID (pids[i]) and current time (c_time)

send a termination signal (SIGTERM) to the child process using its process ID

wait for the child process to terminate

print a message indicating the termination of the server with the current time (c_time)

exit the program

◆ term_exit

exit the program

◆ is_accessible

function is_accessible(ip_address)

open the file named "accessible.usr" in read mode and store the file pointer in variable 'file'

    if the file cannot be opened, print an error message and return 0

declare a character array 'line' with a length of INET_ADDRSTRLEN

read the file line by line into the 'line' array

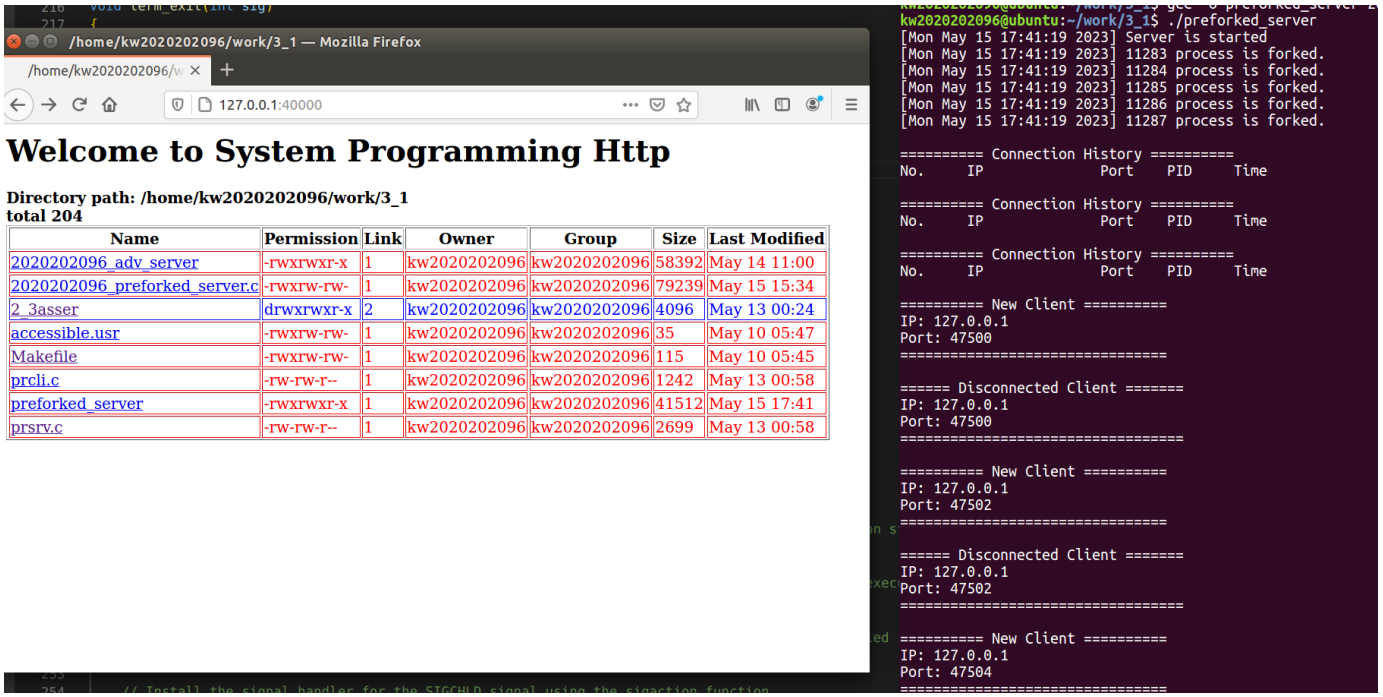
    remove the newline character from the end of the line

    use the 'fnmatch' function to compare the line with the given IP address

        if the line matches the IP address, close the file and return 1

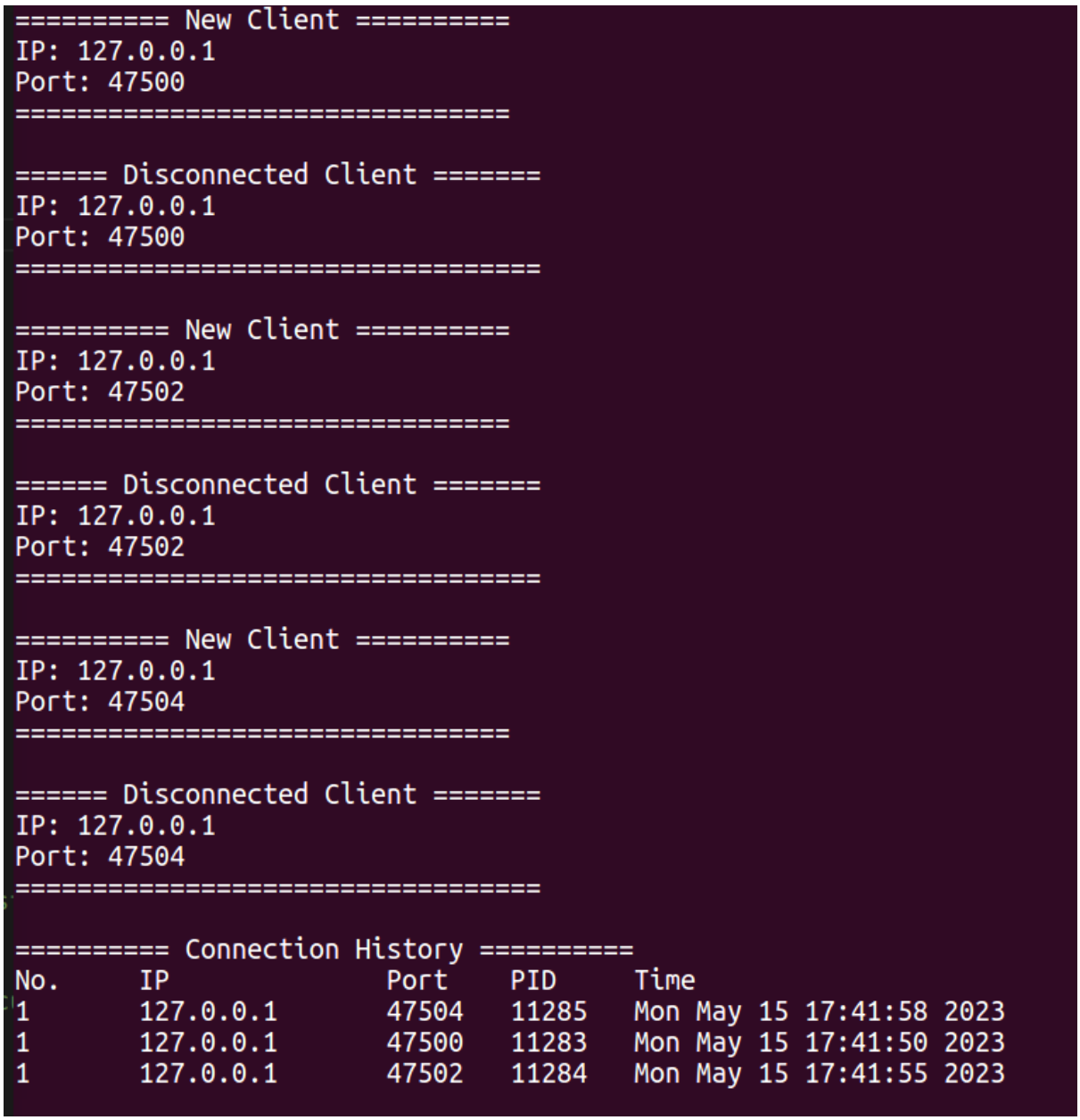
if the end of the file is reached and no match was found, close the file and return 0

◆ 결과화면



서버를 실행하니 서버의 실행 메시지와 프로세서들이 포크되었다는 메시지가 출력된다.

웹브라우저를 통해 클라이언트 요청을 실행하니 new client를 잘 인식하는 모습이다.이때 fork된 5개의 자식 프로세서를 통해 클라이언트와 소통한다.또한 10초에 한번씩 Connection History를 출력한다.



연결된 클라이언트들의 History를 각 자식 프로세서들이 출력한다. History부분의 PID를 보면 모두 각각 다른 자식프로세서와 연결된 것이 보이며 Port번호 또한 요청된 Client의 Port와 동일함을 확인할수있다. 연결 번호 또한 각각의 자식 프로세서들이 출력하므로 독립적이다.

```
========== Connection History ==========
No.     IP              Port    PID     Time
1       127.0.0.1       47504   11285   Mon May 15 17:41:58 2023
1       127.0.0.1       47500   11283   Mon May 15 17:41:50 2023
1       127.0.0.1       47502   11284   Mon May 15 17:41:55 2023

========== New Client ==========
IP: 127.0.0.1
Port: 47506
================================

====== Disconnected Client =======
IP: 127.0.0.1
Port: 47506
==================================

========== Connection History ==========
No.     IP              Port    PID     Time
1       127.0.0.1       47502   11284   Mon May 15 17:41:55 2023
1       127.0.0.1       47504   11285   Mon May 15 17:41:58 2023
1       127.0.0.1       47500   11283   Mon May 15 17:41:50 2023
2       127.0.0.1       47506   11283   Mon May 15 17:42:04 2023
```

추가적으로 연결되어도 잘 반영하는 모습이다. PID를 확인해보면 어떤 자식프로세서에 몇 개의 클라이언트가 있는지 확인할수있다.

```
^C[Mon May 15 17:41:19 2023] 11283 process is terminated.
[Mon May 15 17:41:19 2023] 11284 process is terminated.
[Mon May 15 17:41:19 2023] 11285 process is terminated.
[Mon May 15 17:41:19 2023] 11286 process is terminated.
[Mon May 15 17:41:19 2023] 11287 process is terminated.
[Mon May 15 17:41:19 2023] Server is terminated.
```

Cltr+C로 SIGINT를 발생시켜 서버를 종료시켰다.클라이언트가 먼저 종료되고 순차적으로 서버가 종료되는 모습이다.

```
kw2020202096@ubuntu:~/Work/3_1/final/Web3_1_D_2020202096 (1) (2)$ ./preforked_server
[Tue May 16 22:35:29 2023] Server is started
[Tue May 16 22:35:29 2023] 2999 process is forked.
[Tue May 16 22:35:29 2023] 3000 process is forked.
[Tue May 16 22:35:29 2023] 3001 process is forked.
[Tue May 16 22:35:29 2023] 3002 process is forked.
[Tue May 16 22:35:29 2023] 3003 process is forked.

========== New Client ==========
[Tue May 16 22:35:36 2023]
IP: 127.0.0.1
Port: 45978
================================

====== Disconnected Client =======
[Tue May 16 22:35:36 2023]
IP: 127.0.0.1
Port: 45978
================================

========== Connection History ==========
No.     IP              Port    PID     Time
1       127.0.0.1       45978   2999    Tue May 16 22:35:36 2023

========== New Client ==========
[Tue May 16 22:35:47 2023]
IP: 127.0.0.1
Port: 45980
================================

====== Disconnected Client =======
[Tue May 16 22:35:47 2023]
IP: 127.0.0.1
Port: 45980
================================

========== New Client ==========
[Tue May 16 22:35:49 2023]
IP: 127.0.0.1
Port: 45982
================================
```

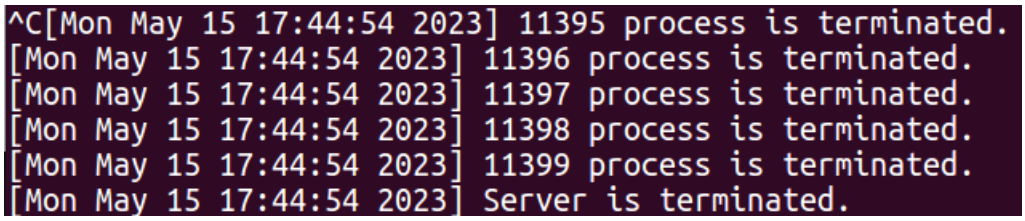
새로운 클라이언트가 연결될때의 시간과 History내부에 저장되어있는 새로운 클라이언트 구조체 내부 기록시간이 일치하다.또한 추가적인 클라이언트 요청시 시간이 잘 업데이트 되는 것을 볼수있다. 시간이 잘 기록되고 해당 내용이 구조체에 잘 기록된다고 볼수있다.

```
kw2020202096@ubuntu:~/work/3_1$ ./preforked_server
[Mon May 15 17:44:54 2023] Server is started
[Mon May 15 17:44:54 2023] 11395 process is forked.
[Mon May 15 17:44:54 2023] 11396 process is forked.
[Mon May 15 17:44:54 2023] 11397 process is forked.
[Mon May 15 17:44:54 2023] 11398 process is forked.
[Mon May 15 17:44:54 2023] 11399 process is forked.
```

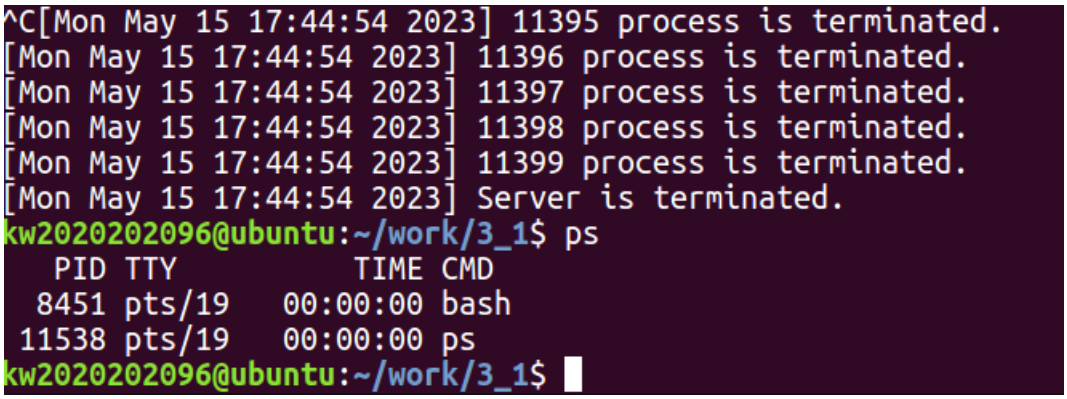
이번엔 클라이언트에서 많은 요청을 보낼때의 서버에서의 처리를 확인해보겠다.

```
kw2020202096@ubuntu: ~/work/3_1
========== Connection History ==========
No.     IP              Port    PID     Time
4       127.0.0.1       47554   11395   Mon May 15 17:46:27 2023
5       127.0.0.1       47564   11395   Mon May 15 17:46:29 2023
6       127.0.0.1       47574   11395   Mon May 15 17:46:30 2023
7       127.0.0.1       47584   11395   Mon May 15 17:46:32 2023
4       127.0.0.1       47560   11397   Mon May 15 17:46:28 2023
8       127.0.0.1       47594   11395   Mon May 15 17:46:33 2023
9       127.0.0.1       47604   11395   Mon May 15 17:46:34 2023
5       127.0.0.1       47570   11397   Mon May 15 17:46:30 2023
10      127.0.0.1       47614   11395   Mon May 15 17:46:36 2023
6       127.0.0.1       47580   11397   Mon May 15 17:46:31 2023
11      127.0.0.1       47624   11395   Mon May 15 17:46:37 2023
7       127.0.0.1       47590   11397   Mon May 15 17:46:32 2023
12      127.0.0.1       47634   11395   Mon May 15 17:47:08 2023
8       127.0.0.1       47600   11397   Mon May 15 17:46:34 2023
13      127.0.0.1       47644   11395   Mon May 15 17:47:09 2023
9       127.0.0.1       47608   11397   Mon May 15 17:46:35 2023
10      127.0.0.1       47618   11397   Mon May 15 17:46:36 2023
11      127.0.0.1       47628   11397   Mon May 15 17:46:38 2023
12      127.0.0.1       47636   11397   Mon May 15 17:47:08 2023
13      127.0.0.1       47646   11397   Mon May 15 17:47:10 2023
3       127.0.0.1       47550   11398   Mon May 15 17:46:18 2023
4       127.0.0.1       47562   11398   Mon May 15 17:46:29 2023
5       127.0.0.1       47572   11398   Mon May 15 17:46:30 2023
6       127.0.0.1       47582   11398   Mon May 15 17:46:31 2023
7       127.0.0.1       47592   11398   Mon May 15 17:46:33 2023
8       127.0.0.1       47602   11398   Mon May 15 17:46:34 2023
9       127.0.0.1       47612   11398   Mon May 15 17:46:35 2023
10      127.0.0.1       47622   11398   Mon May 15 17:46:37 2023
4       127.0.0.1       47556   11396   Mon May 15 17:46:28 2023
11      127.0.0.1       47632   11398   Mon May 15 17:47:08 2023
5       127.0.0.1       47566   11396   Mon May 15 17:46:29 2023
12      127.0.0.1       47642   11398   Mon May 15 17:47:09 2023
6       127.0.0.1       47576   11396   Mon May 15 17:46:30 2023
7       127.0.0.1       47586   11396   Mon May 15 17:46:32 2023
8       127.0.0.1       47596   11396   Mon May 15 17:46:33 2023
9       127.0.0.1       47606   11396   Mon May 15 17:46:34 2023
10      127.0.0.1       47616   11396   Mon May 15 17:46:36 2023
11      127.0.0.1       47626   11396   Mon May 15 17:46:37 2023
12      127.0.0.1       47638   11396   Mon May 15 17:47:09 2023
13      127.0.0.1       47648   11396   Mon May 15 17:47:11 2023
3       127.0.0.1       47548   11399   Mon May 15 17:46:18 2023
4       127.0.0.1       47558   11399   Mon May 15 17:46:28 2023
5       127.0.0.1       47568   11399   Mon May 15 17:46:29 2023
6       127.0.0.1       47578   11399   Mon May 15 17:46:31 2023
7       127.0.0.1       47588   11399   Mon May 15 17:46:32 2023
8       127.0.0.1       47598   11399   Mon May 15 17:46:33 2023
9       127.0.0.1       47610   11399   Mon May 15 17:46:35 2023
10      127.0.0.1       47620   11399   Mon May 15 17:46:37 2023
11      127.0.0.1       47630   11399   Mon May 15 17:46:38 2023
12      127.0.0.1       47640   11399   Mon May 15 17:47:09 2023
```

각 자식프로세서는 10개까지 접속한 클라이언트의 정보를 기록하고 출력한다.그러므로 10개 이상의 요청이 오면 가장 오래된 접속 클라이언트의 정보를 지우고 새로운 클라이언트의 접속정보를 저장한다.이때 각 자식 프로세스에서의 클라이언트 번호에 대한 내용은 과제에서 언급되지않아,고유번호 개념으로 각 자식 프로세서 내부에서는 클라이언트끼리 서로 같은 번호를 부여하지 않도록 하였다.위 사진을 확인해보면 각 프로세서 별로 클라이언트에 대한 고유번호가 부여되어있는 것을 확인할수있다.이때 처음에 접속한 1,2번의 클라이언트는 접속 클라이언트가 10개를 초과하였기에 보이지 않는 것을 확인할수있다.



SIGINT 발생시 자식 프로세스가 종료되고 서버 프로세스가 종료되는 모습이다.



좀비프로세스를 확인해보았다. 좀비프로세스 없이 잘 종료되는 모습을 보여준다.

◆ 고찰

이번과제에서는 프로세스 및 fork 그리고 시그널에 대한 깊은 이해를 경험하는 과제가 되었다.먼저 클라이언트들을 어떻게 관리해야 각 자식프로세서마다 클라이언트에 대한 정보를 출력할까라는 고민이 들었다.이를 2차원 배열을 통해 해결하였는데 다시 생각해보니 굳이 2차원일 필요가 없다는 생각이 들었다.왜냐하면 fork순간 각 프로세스 별로 독립된 메모리를 copy하기 때문에 코드에서는 같은 배열의 같은 인덱스로 접근하는 것을 보여도 실제 메모리상으론 다른 위치에 접근하는것이다.결론적으로 이 과제는 fork를 얼마나 잘 이해하고 쓸수있느냐에 대한 과제인것같다.또한 각 자식프로세스 들에 대한 PID를 PIDS라는 배열을 통해 모든 자식프로세서들이 자기 자신에 대한 인덱스를 위 2차원 배열과 공유하여 접근할수있도록 하여 해결하였는데 클라어인트 구조체 배열을 1차원으로 바꾸면 좀더 코드가 간결해지지 않았을까 라는 생각이 들었다.

◆ 참고

-강의자료