Assignment#4-1

시스템 프로그래밍 실습

제출일: 5월 31일 금요일

분 반: 화요일

담당 교수: 신영주

학 번: 2015722025

학 과: 컴퓨터정보공학부

이 름: 정용훈

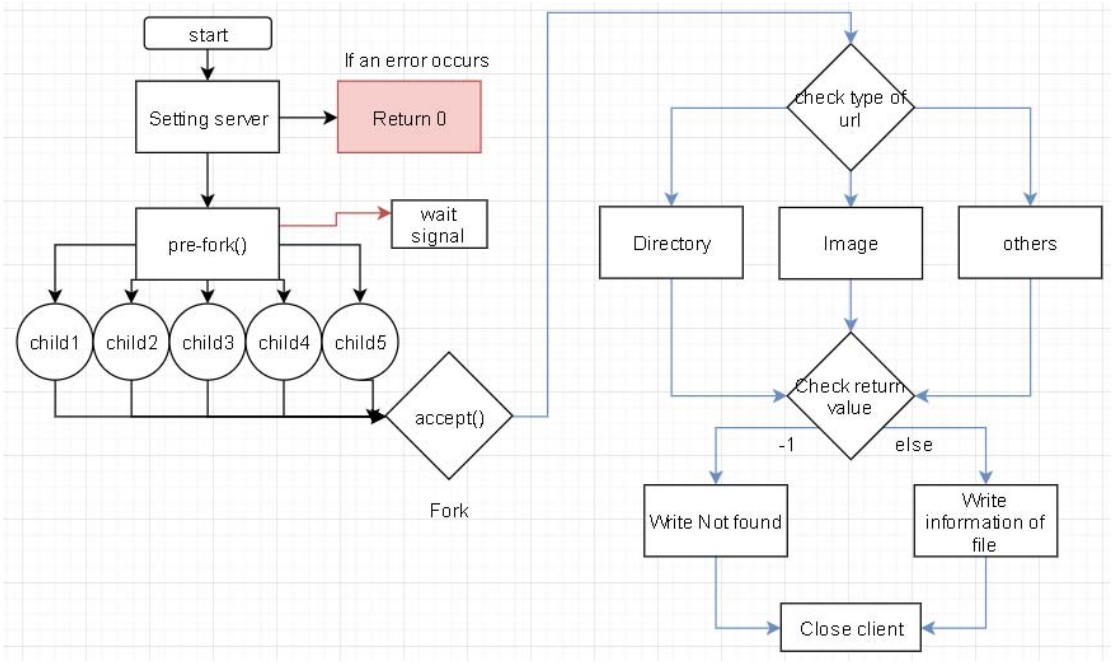
1. Introduction

4-1과제는 3-3을 기반으로 원래 client에서 요청이 들어오면, 그 순간 fork를 통하여 프로세스를 생성하였지만 이번 과제는 pre-fork기술을 사용함으로써 요청이 들어올 때 fork를 하는 시간을 단축하는데 목적을 가지고 있다. 그 외에는 보다 심화된 signal사용을 이해하고 배우는데 목적을 둔다.

2. Flow Chart

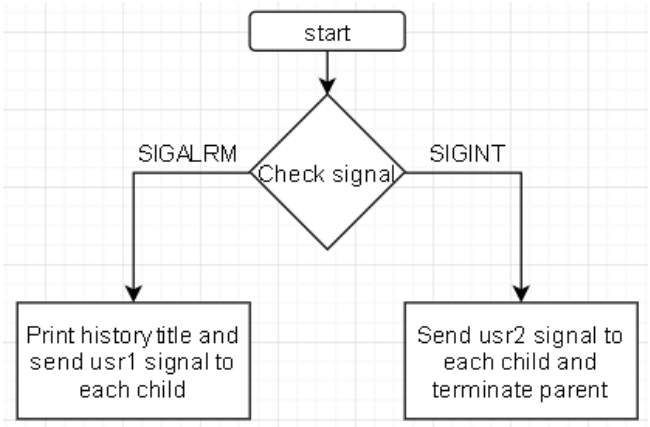
이번 4-1과제에서는 Main함수에 추가되는 pre-fork관련함수와 전 과제와 변형된 signal함수만 있을 뿐 다른 점은 없으며, 추가적으로 signal과 history에 관련된 함수가 추가된다.

Main



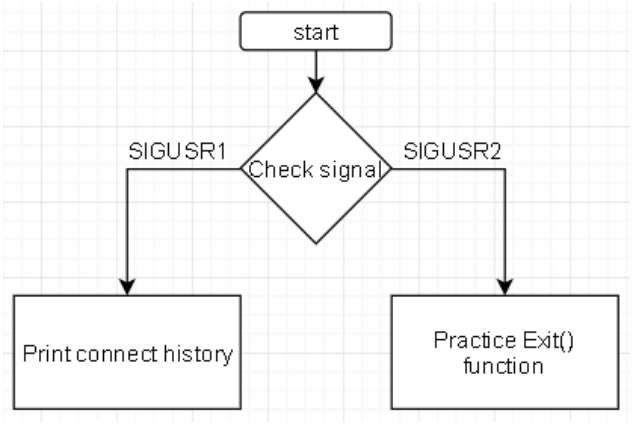
다음과 같이 accept()함수가 실행되기 전 먼저 fork를 통하여 child process를 생성 후 parent process는 다른 signal이 들어올 때 동안 pause상태를 유지한다. 페이지의 요청이 들어오면 각각의 child는 요청을 각각 수행하게 된다. 간단하게 왼쪽은 기존 main이고 오른쪽은 child main이다.

SignalHandler_parent



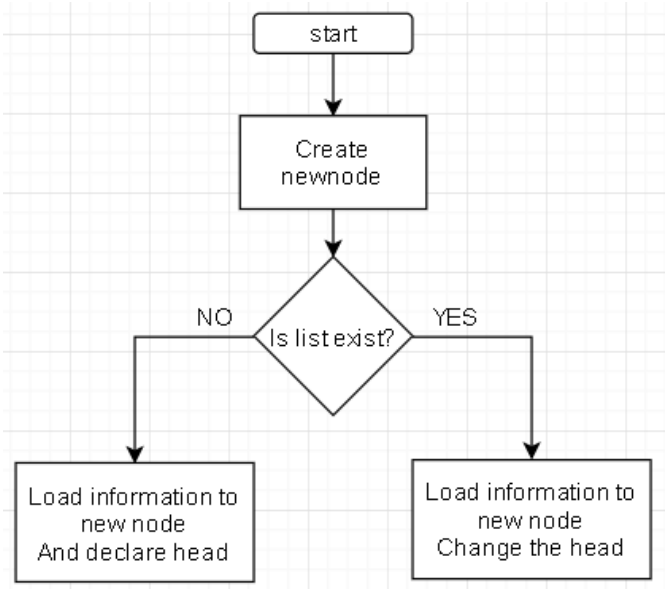
Parent의 signal을 정의해주는 함수로써 각 상황에 맞게 child에게 usr1 signal이나 usr2 signal을 보내준다.

SignalHandler_child



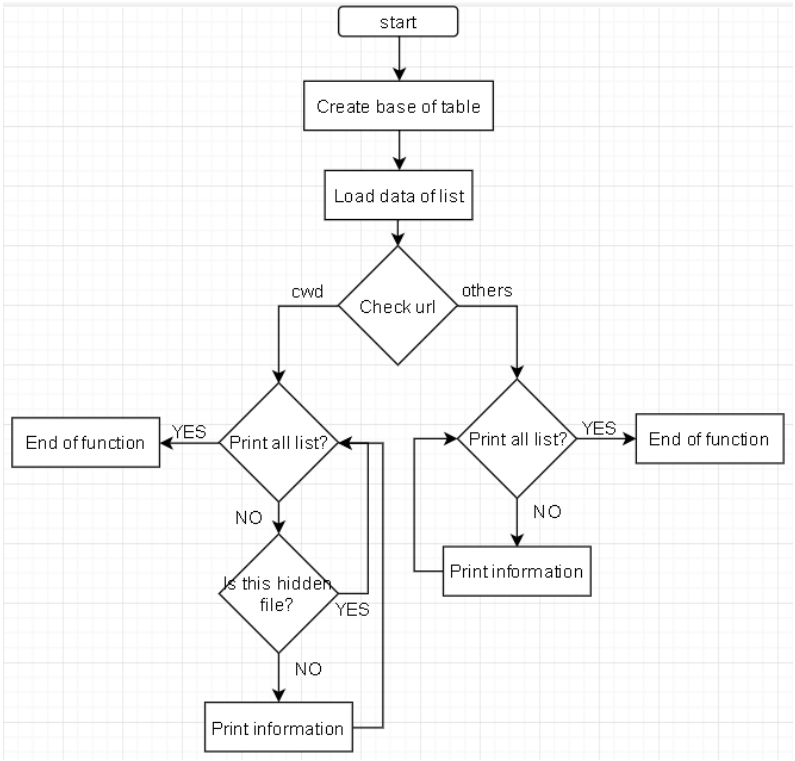
Child process의 signal을 정의해주는 함수다. History를 제외한 나머지 출력은 parent에서 담당하기 때문에 함수가 굉장히 간단하다.

UpdateInfo



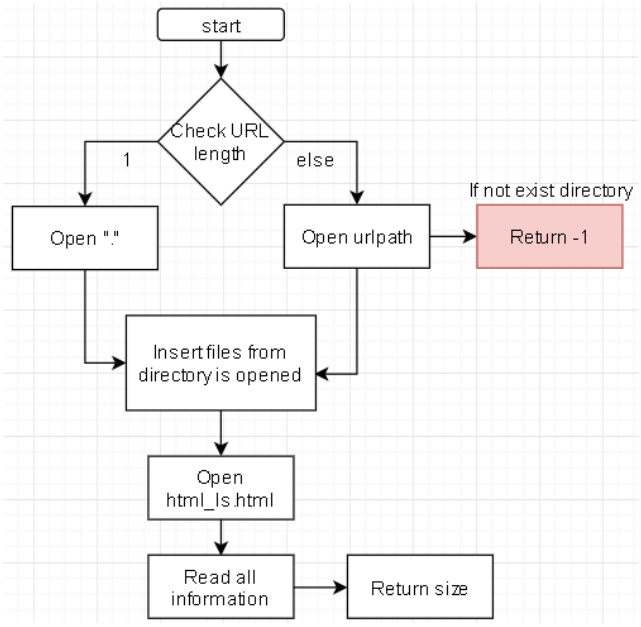
History출력을 위해 list를 생성하는 함수다. 특징으로는 가장 최신의 데이터가 먼저 보여지기 때문에 Head를 계속해서 update해주며 정보들을 link해준다. Link된 list들은 parent process에서 오는 USR1 signal을 통하여 출력된다.

PrintHTML(create file of html_ls.html)



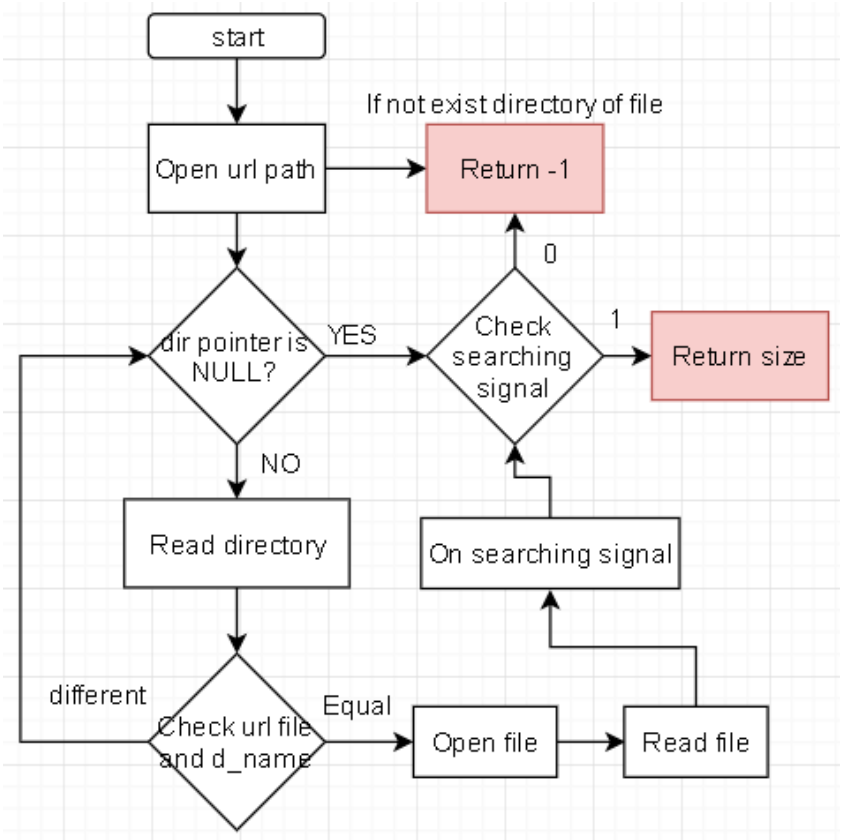
3-1 과제인 html_ls.hml을 만드는 과정으로 옵션 구현이 –a과 –al만 있기 때문에 구조만 동작 자체는 똑같지만 구조를 변경하게 되었다. 해당 함수는 Html함수에서 call하는 함수로써 아래 Html함수를 참고하면 이해하기 쉽다.

Html



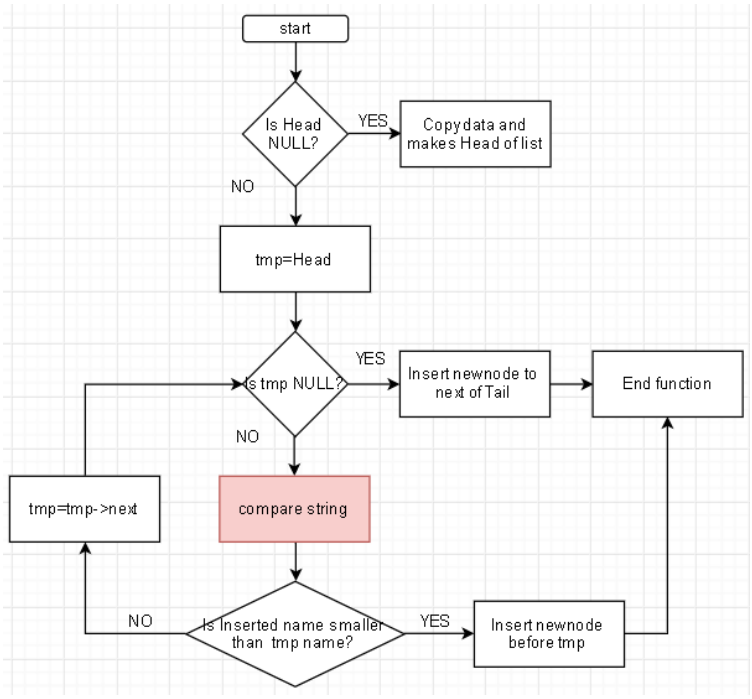
Open html을 하기 전 과정이 printHTML로써 html파일을 생성한 후 실행하게 된다.

Image & Normal (others)



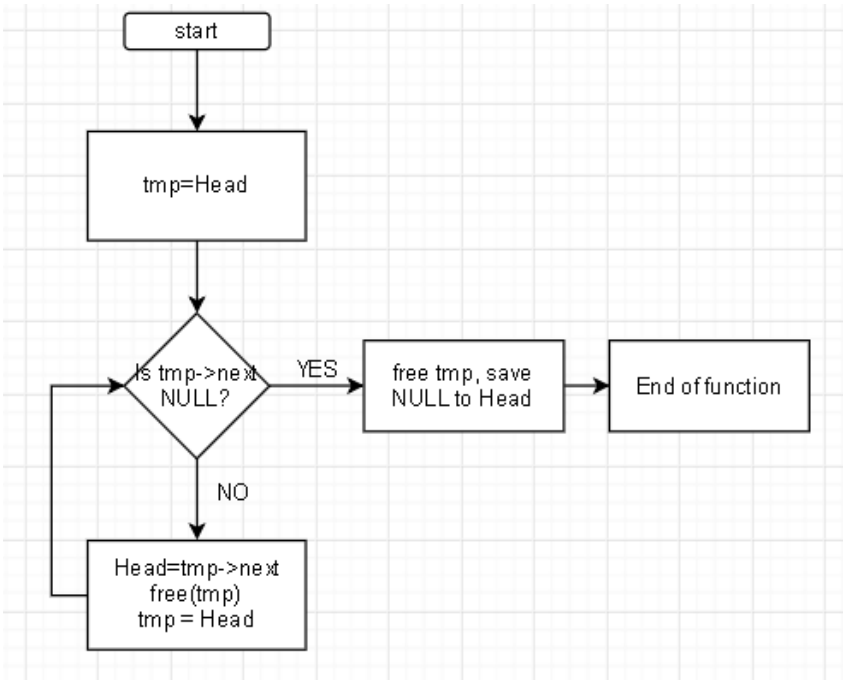
다음 함수는 Image와 나머지 다른 파일들을 처리해주는 함수로써 정보를 binary로 읽어와 write해주는 작업을 하게 된다 사실 image와 나머지 파일은 모두 binary를 통하여 읽을 수 있기 때문에 나눠줄 필요는 없다고 생각된다.

Insertnode



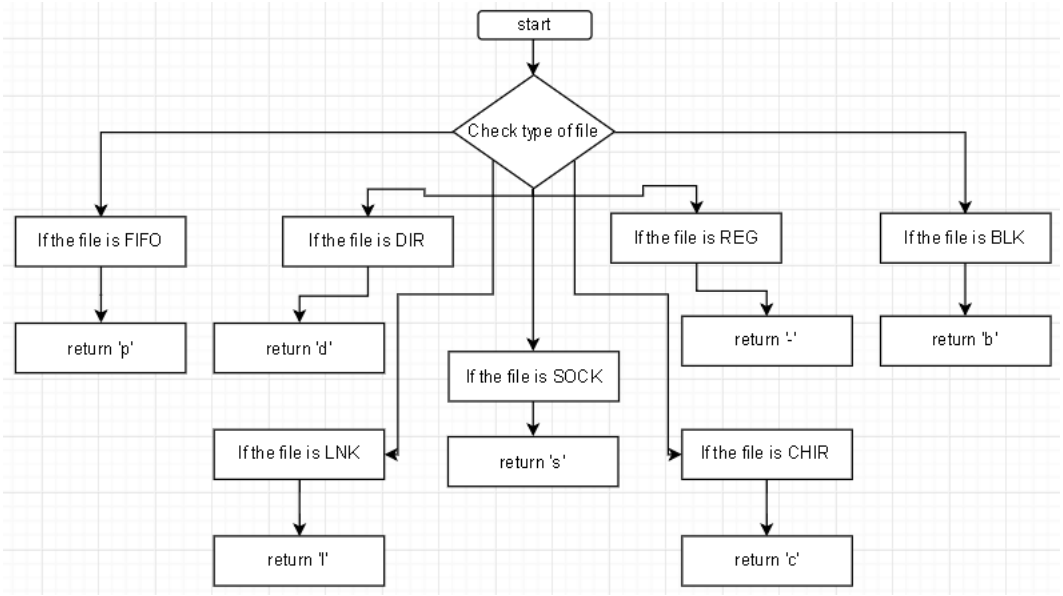
따로 sort함수를 사용하지 않고 insert를 실행 할 때부터 sort가 되며, list가 생성된다. Insert함수는 기존 함수와 동일하게 사용되므로 전체적인 변화는 없지만, S옵션을 사용면 size를 비교해야 하므로 compare를 하는 부분이 변경되게 된다. 해당 문제는 함수는 같고 조건만 바꿔주어 해결하였다.

Deletelist



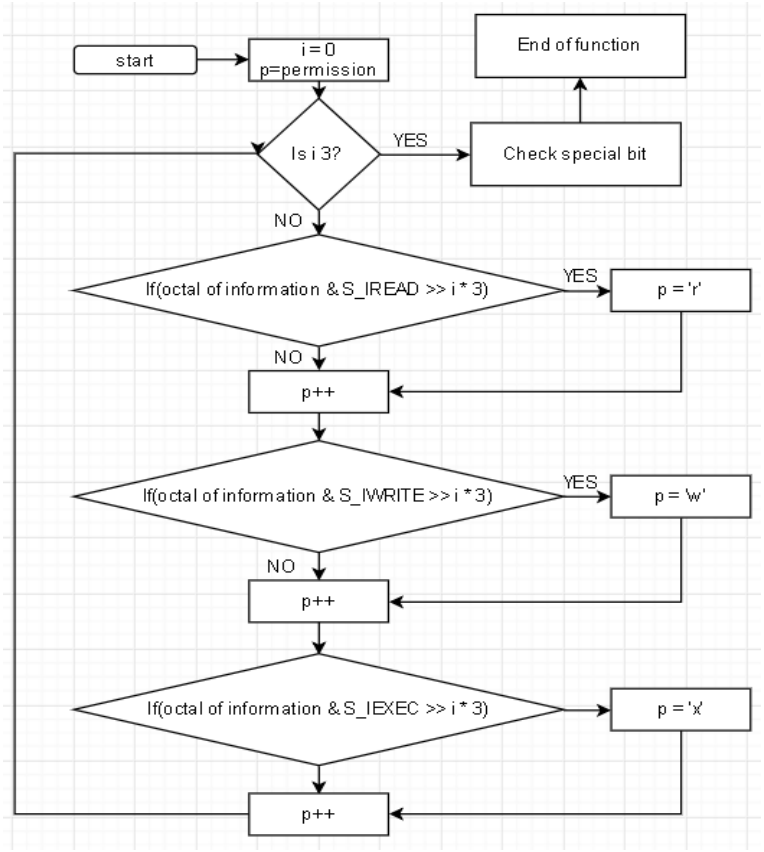
모든 정보 출력 후 linked list를 제거하는 함수다.

printType



파일의 정보를 받아와 st_mode를 통하여 파일의 type을 정의하는 함수다

printPerm



St_mode를 받아와 해당 파일의 permission을 확인하여, 최종적인 permission을 출력할 수 있도록 도와주는 함수다.

3. Pseudo code

Main

```
int main(int argc, char** argv)
{
        Setting signal;
        Declare value for using main function;

        ////////////////////////Server value////////////////////////
        struct sockaddr_in server_addr, client_addr;
        int socket_fd, client_fd;
        int len, len_out;
        int opt = 1;
        ////////////////////////////////////////////////////////////

        load current working directory;

        ////////////////////////For Connecting Server////////////////////////
        setting socket;
        setting socket opt;

        memset(&server_addr, 0, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        server_addr.sin_port = htons(PORTNO);

        bind;

        listen(socket_fd, 5);
```

```
//////////////////////////Make child process/////////////////////////////////////
        pids=create array;
        for(int i=0;i<5;i++)
        {
                fork();
                if(parent process)
                {
                        print information of time;
                        continue;
                }

                else
                {
                        Declare child signal;
                        break;
                }

        }

        if(parent process)
        {
                alarm(10);
                while(1)
                        pause();
        }
////////////////////////////////////////////////////////////////////////////////
```

————————————————아래로 child process————————————————

```
while (1) //start server
{
        Setting variable;

        len = sizeof(client_addr);
        client_fd = accept;

        Check accessible user;
        load url;

        continue parent process;
        child process practice below code;
```

```
////////////////////////////Declare head/////////////////////////////////////////////////
if(The signal is image file)
{
        long unsigned int filesize=0; //file size
        unsigned char image_message[3000000]={0, }; //message buffer
        filesize=Image(urlname,client_fd,image_message); //return size of file

        if(filesize==-1) //check not found
                send Not found response

        else
                send information of file

        write(client_fd, response_header, strlen(response_header)); //send header
        write(client_fd, image_message, filesize); //send entity
}
```

```
		else if(The signal is directory)
		{		long unsigned int filesize=0; //file size
				unsigned char html_message[3000000]={0, }; //message buffer
				filesize=Html(url,client_fd,html_message); //return size of file

				if(filesize==-1) //check not found
					send Not found response

				else
					send information of file

				write(client_fd, response_header, strlen(response_header)); //send header
				write(client_fd, html_message, filesize); //send entity
		}

		else //type of others
		{
				long unsigned int filesize=0; //file size
				unsigned char normal_message[3000000]={0, }; //message buffer
				filesize=Normal(urlname,client_fd,normal_message); //return size of file

					if(filesize==-1) //check not found
					send Not found response

				else
					send information of file

				write(client_fd, response_header, strlen(response_header)); //send header
				write(client_fd, normal_message, filesize); //send entity

		}

			close(client_fd); //close client
			continue;

	}
	close(socket_fd); //close socket
	return 0;
}
```

SignalHandler_parent

```
void signalHandler_parent(int sig)
{
        if(sig == SIGALRM)
        {
                Print title of history;
                Send usr1 signal to each child;

        }

        if(sig == SIGINT)
        {
                Send usr1 signal to each child;

                for(int i=0;i<5;i++)
                {
                        Wait each child;
                        Print terminated information;
                }

                Print information of Server;
                exit(0);
        }

        if(sig == SIGCHLD)
                wait(&status);
}
```

SignalHandler_child

```
void signalHandler_child(int sig)
{
        if(sig == SIGUSR1) //alram signal
                PrintInfo(); //print information of history


        if(sig==SIGUSR2)
                exit(0);

}
```

UpdateInfo

```
void UpdateInfo(struct sockaddr_in client_addr,int PID)
{
        create new node;

        setting time;

        if(List not exist)
        {
                load information of client to new node;
                Head=newnode;
        }

        else
        {
                load information of client to new node;
                newnode->next=Head;
                Head-newnode;
        }
}
```

Normal & Image

```
int Normal & Image(char *urlname,int client_fd,unsigned char *normal_message)
{
        char urlpath[256]={0};
        char urlfile[256]={0};
        char cwd[256]={0};
        char Openpath[256]={0};
        char Dirpath[256]={0};
        int searching=0;

        unsigned char response_message[3000000]={0, };
        struct stat buf;
        DIR *dirp;
        struct dirent *dir;

        load current working directory;
        urlpath = urlname

        Make Open path;

//////////////////////open dir for checking image file////////////////////////////
        if(open directory==NULL)
                return -1;

        change directory;

                do
                {
                        read directory;

                        If(read is NULL)
                                return -1;

                        else if(urlfile is equal d_name)
                        {
                                searching=1;
                                FILE *file=NULL;
                                int ch;

                                file=Open d_name file;

                                while(Repeat read file before EOF)
                                        normal_message[count++]=ch;

                                close file;
                                break;
                        }
                } while (1);
        change directory to home;

        if(searching==1)
                return count;

        return -1;
}
```

Html

```
int Html(char *url,int client_fd,unsigned char *html_message)
{
        DIR *dirp;
        struct dirent *dir;
        struct stat buf;
        struct group *gid;
        struct passwd *uid;
        struct tm *time;
        FILE *htmlfile;
        struct sockaddr_in server_addr, client_addr;
        struct in_addr inet_client_address;

        int total = 0,count=0;
        char response_message[3000000] = { 0, };
        char NULLpath[256]={0, };

        ////////////////////////////////////Open DIR and wirte//////////////////////.
        Open html file that created from printHTML function;

        if(strlen(url)==1)
        {
                Opswitch=1;
                open current directory;
                change directory;

                do
                {
                        read directory;

                        if (read is NULL)
                                break;
```

```
                        else
                        {
                                load information of file to buf;
                                total += buf.st_blocks / 2;
                                Insert node;
                        }

                } while (1);
        }

        else
        {
                Opswitch=0;
                char urlpath[256]={0,};
                make url;

                if(Not exist directory)
                        return -1;
                change working directory;

                do
                {
                        read directory;

                        if (read is NULL)
                                break;

                  else
                  {
                        load information of file to buf;
                        total += buf.st_blocks / 2;
                        Insert node;
                  }

          } while (1);
}


if(root path)
{
        strcpy(response_message, "<h1>Welcome to System Programming Http</h1><br>₩n"); //copy
        fprintf(htmlfile,"<h1>Welcome to System Programming Http</h1><br>₩n"); //write to file
}

else
{
        strcpy(response_message, "<h1>System Programming Http</h1><br>₩n"); //copy
        fprintf(htmlfile,"<h1>System Programming Http</h1><br>₩n"); //write to file
}
```

```
                practice printHTML; //create html file

                deletelist(); //delete list
                change working directory;
                close html file;

                FILE *file=NULL;
                Open html file;

                int ch;
                while(Repeat read file before EOF)
                        html_message[count++]=ch;

                close file;

                return count;
}
```

PrintlistHTML

```
void printHTML(int client_fd, FILE* file, int flagA, int flagL, int total, char *dirpath)
{
        struct group *gid;
        struct passwd *uid;
        struct tm *time;
        char buff[256];
        char cwd[256];
        struct stat buf;
        struct dirent *dir;

        int k = 0, m = 0;
        char subpath[256] = { 0 };
        char subdirpath[256] = { 0 };
        char checkpath[256] = { 0 };
        char color[256] = { 0 };

        int createflag = 0;
        char content[BUFSIZE] = { 0, };

        load current working directory;
        For making path;

        Declare information of Directory(path, total);
```

```
        if (createflag == 0)
        {
                Declare format of table;
                createflag = 1;
        }

        for (Node* tmp=Head; tmp; tmp = tmp->next) //Repeat function for printing all list
        {
                if(Opswitch==1&&tmp->filename[0]=='.')
                        continue;
                if(!strcmp(tmp->filename,"html_ls.html"))
                        continue

                lstat(tmp->filename, &buf);
                Declare color of file;
                Make hyperlink;

                if(url's mean is current("."))
                        urlpath[strlen(urlpath)-1]='₩0';
                else
                        strcat(urlpath,tmp->filename); //make full format of url path

                uid = getpwuid(buf.st_uid);
                gid = getgrgid(buf.st_gid);

                time = load information of local time;

                if (File type is Link)
                        Make path use "->"


                Print full format;

        }

        End table and html
        return;
}
```

Nodeinsert

```
void NodeInsert(char* name)
{
        char Ename[256];
        char Etmpname[256];
        Node* tmp = Head;
        Node* prevnode;

        if (Head of list is NULL)
        {
                Makes newnode;
                strcpy(newnode->filename, name);
                newnode->next = NULL;
                newnode->prev = NULL;

                Head = newnode;
                return;
        }

    else
    {
            while (tmp is not NULL)
            {
                    strcpy(Ename, name);
                    strcpy(Etmpname, tmp->filename);

                    if (checkstring(Ename))
                            Eliminate character of '.'
                    if (checkstring(Etmpname))
                            Eliminate character of '.'

                    if Etmpname is bigger than name)
                    {
                            if (tmp is Head)
                            {
                                    Makes newnode;
                                    strcpy(newnode->filename, name);
                                    newnode->prev = NULL;

                                    newnode->next = Head;
                                    Head->prev = newnode;
                                    Head = newnode;
                                    return;
                            }
```

```
                                        else
                                        {
                                                Makes newnode;
                                                strcpy(newnode->filename, name);
                                                newnode->next = tmp;
                                                newnode->prev = tmp->prev;
                                                tmp->prev->next = newnode;
                                                tmp->prev = newnode;
                                                return;
                                        }
                                }
                                else continue;
                        }
                        Move prev to Tail;
                        Makes newnode;

                        strcpy(newnode->filename, name);
                        newnode->next = NULL;
                        prevnode->next = newnode;
                        newnode->prev = prevnode;
                        Tail = newnode;
                        return;

                }
        }
```

Function of others

```
void Eliminate(char *str, char ch)
{
	while (;befor check all character of string;str++)
	{
		if (*str == ch)
		{
			strcpy(str, str + 1);
			str--;
			return;
		}
	}
}

void deletelist()
{
	Node* tmp = Head;
	for (; tmp->next != NULL;)
	{
		Head = tmp->next;
		free(tmp);
		tmp = Head;
	}
	free(tmp);
	Head = NULL;
}
```

```
char printType(mode_t mode)
{
        switch (mode & S_IFMT)
        {
        case S_IFREG:
                return('-');
        case S_IFDIR:
                return('d');
        case S_IFCHR:
                return('c');
        case S_IFBLK:
                return('b');
        case S_IFLNK:
                return('l');
        case S_IFIFO:
                return('p');
        case S_IFSOCK:
                return('s');
        }

        return('?');
}

char *printPerm(mode_t mode)
{
        int i;
        char *p;
        static char perms[10];
        p = perms;
        strcpy(perms, "---------");

        for (i = 0; i < 3; i++)
        {
                if (mode & (S_IREAD >> i * 3))
                        *p = 'r';
                p++;

                if (mode & (S_IWRITE >> i * 3))
                        *p = 'w';
                p++;

                if (mode & (S_IEXEC >> i * 3))
                        *p = 'x';
                p++;
        }
        if ((mode & S_ISUID) != 0)
                perms[2] = 's';

        if ((mode & S_ISGID) != 0)
                perms[5] = 's';

        if ((mode & S_ISVTX) != 0)
                perms[8] = 't';

        return(perms);
}
```

```
void printOph(long int size)
{
	int k=0,flag=0;
	double sub_size;

	sub_size=(double)size;

	for(Repeat until undivided)
	{
		if(sub_size>(double)1024)
			sub_size/=1024;

		else
			break;
	}

	Execute unit alignment process

	if(If the decimal point is not zero) //check first decimal point
		flag=1;

	int size_int=0;

	if(k==0)
	{

		size_int=(int)sub_size;
		printf(integer output of size_int);

	}

	else if(k==1)
	{
		if(flag==1)
		{
			size_int=(int)sub_size;
			printf(integer output of size_int,"K");
		}
		else
			printf(float output of sub_size,"K");
	}

	else if(k==2)
	{
		if(flag==1)
		{
			size_int=(int)sub_size;
			printf(integer output of size_int,"M");
		}
		else
			printf(float output of sub_size,"M");
	}
```

```
        else if(k==3)
        {
                if(flag==1)
                {
                        size_int=(int)sub_size;
                        printf(integer output of size_int,"G");
                }
                else
                        printf(float output of sub_size,"G");
        }

        else if(k==4)
        {
                if(flag==1)
                {
                        size_int=(int)sub_size;
                        printf(integer output of size_int,"T");
                }
                else
                        printf(float output of sub_size,"T");
        }
        return;
}
```

```
int matchfunction(char (*paname)[256], int argc)
{
	char Matchcmd[256][256]={0};
	int matchcount=0;

////////////////////////////////////////////////////////searching match command////////////////////////////////
	while(read all command)
	{
		while(read all character of one string)
		{
			if(string has '*' or '?') //if the command has '*' or '?'
			{
				copy string to command vector
				matchcount++; //increase
				break;
			}
			else if(string has '[x-y]' format) //if the command has 'index'
			{
				copy string to command vector
				matchcount++; //increase
				break;
			}
			else if(string has '[x]' format)

			{
				copy string to command vector
				matchcount++; //increase
				break;
			}
		}
	}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////delete match cmd//////////////////////////////////////////////////////

	Remove extracted commands from existing vectors

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
int newargc=argc; //integer
int flag=0; //integer
struct dirent *dir;
struct stat buf;
DIR *dirp;
char pathname[256][256]={0};
char cmd[256][256]={0};

while(checking all string) //for check match command
{
        To separate into path and command parts
}


while(Repeat by match count) //for check command
{
        int currentflag=0; //declare flag

        if(string length is 0) //check path name
                dirp=opendir(".");
        else
        {
                currentflag=1; //on flag
                dirp=opendir(pathname[j]); //open pathname directory
        }
        change directory;

    if(In case there is no matching command) //Check null
    {
            Save back to original vector;
            newargc++;
            continue;
    }
    do
    {

            dir = readdir(dirp); //read file
            if (dir == NULL) //check NULL
                    break; //stop repeat functsion

            else
            {
                    if(!fnmatch(pattern[i],file name,0)&&(dir->d_name[0]!='.')) //function of match
                    {
                            Perform a function that makes it the original path;

                            Save the filename that matches the original vector;
                            newargc++;
                            flag=1; //on flag
                    }
            }

    } while (1);


    if(flag==0) //if flag is 0
    {
            Save back to original vector;
            newargc++;
    }
    flag=0; //off flag
    change pointer dirp to start
}
return newargc; //return new vector count

}
```

4. Result

```
sp20157522025@ubuntu:~$ ./preforked_server_40229
[Tue May 28 17:26:26 2019] Server is started.
[Tue May 28 17:26:26 2019] 2664 process is forked.
[Tue May 28 17:26:26 2019] 2665 process is forked.
[Tue May 28 17:26:26 2019] 2666 process is forked.
[Tue May 28 17:26:26 2019] 2667 process is forked.
[Tue May 28 17:26:26 2019] 2668 process is forked.
```

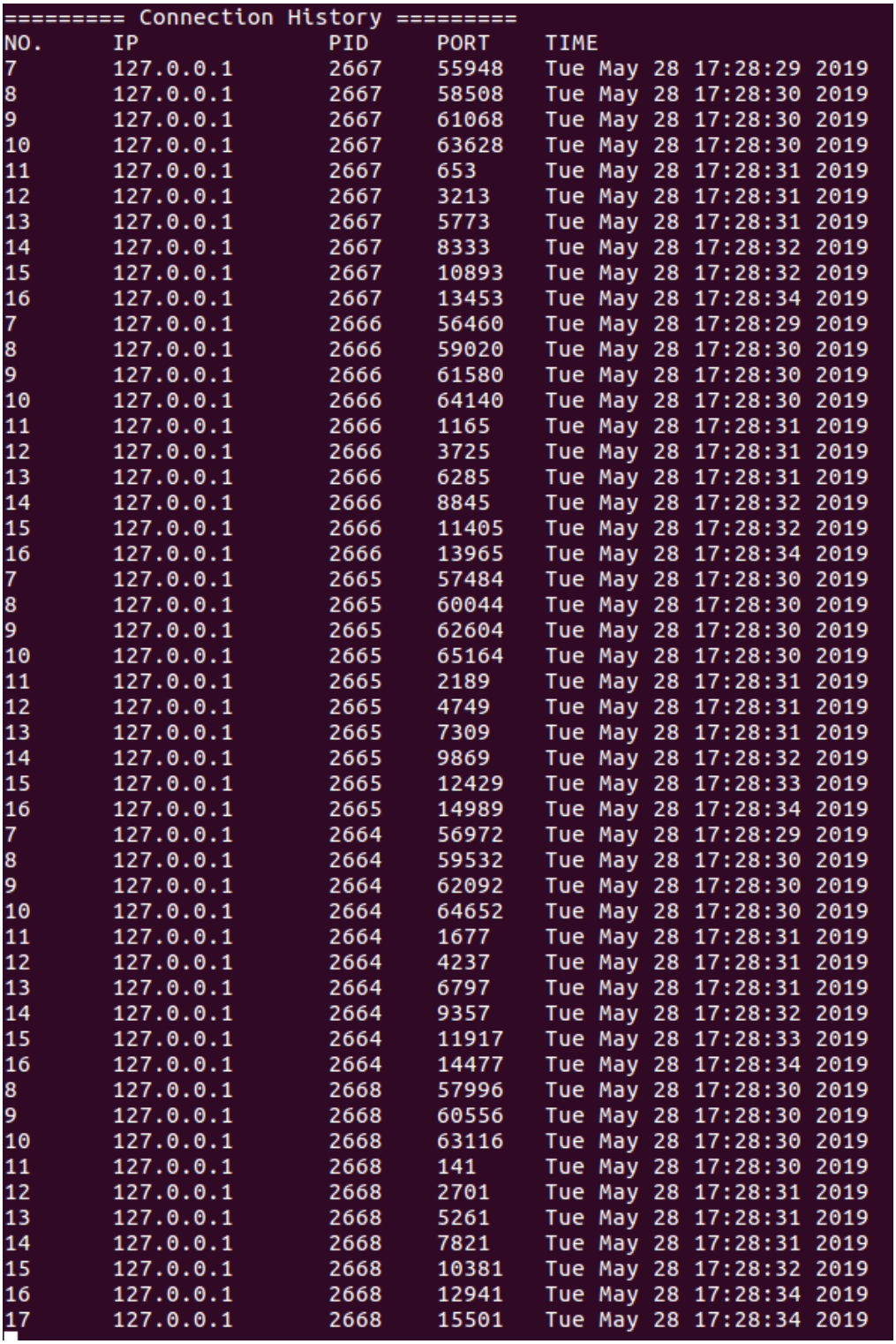
다음 결과 화면은 PDF의 결과 화면과 같이 Server가 시작 되고, child process가 생성되는 것을 콘솔에서 확인할 수 있는 결과 화면이다.

```
============= New client =============
[Tue May 28 17:27:03 2019]
IP : 127.0.0.1
Port : 37004
=======================================
========= Disconeected Client ========
[Tue May 28 17:27:03 2019]
IP : 127.0.0.1
Port : 37004
=======================================
========= Connection History =========
NO.     IP              PID     PORT    TIME
1       127.0.0.1       2668    37004   Tue May 28 17:27:03 2019
```

위 사진은 클라이언트가 서버에 접속하는 모습으로 접속을 감지하고 new client문구를 띄운 후 요청 작업이 완료되면 바로 연결을 끊어주는 동작을 확인할 수 있다. 또한 연결을 하나만 했고, 10초의 시간이 지나면서 History또한 함께 출력된 것을 확인할 수 있다.

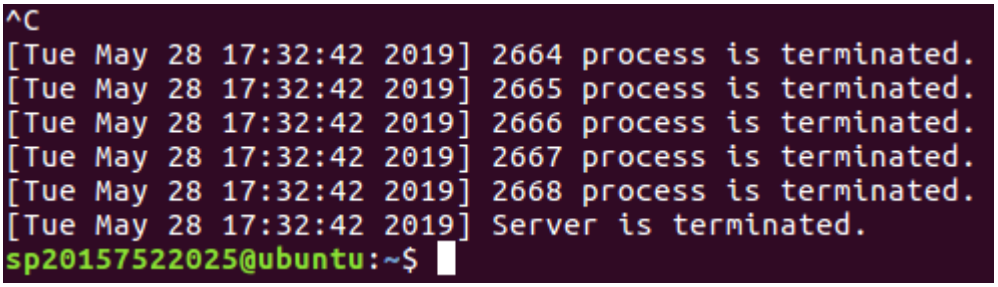
```
============= New client =============
[Tue May 28 17:27:40 2019]
IP : 127.0.0.1
Port : 42124
=======================================
========= Disconeected Client ========
[Tue May 28 17:27:40 2019]
IP : 127.0.0.1
Port : 42124
=======================================
========= Connection History =========
NO.     IP              PIDPORT     TIME
1       127.0.0.1       2664        40076   Tue May 28 17:27:38 2019
1       127.0.0.1       2665        40588   Tue May 28 17:27:39 2019
1       127.0.0.1       2668        37004   Tue May 28 17:27:03 2019
2       127.0.0.1       2668        41100   Tue May 28 17:27:39 2019
1       127.0.0.1       2667        39052   Tue May 28 17:27:36 2019
2       127.0.0.1       2667        41612   Tue May 28 17:27:40 2019
1       127.0.0.1       2666        39564   Tue May 28 17:27:37 2019
2       127.0.0.1       2666        42124   Tue May 28 17:27:40 2019
```

연결 request를 하나가 아닌 좀 더 여러 개를 시도한 후 History를 출력한 형태다. 결과를 보면, 한 개의 process만 계속 동작하는 것이 아니라 여러 개의 process가 순차적으로 동작하는 것을 확인할 수 있다. No를 보면 각 process가 기록하고 있는 연결 history를 확인할 수 있다.



```
========= Connection History =========
NO.     IP              PID     PORT    TIME
7       127.0.0.1       2667    55948   Tue May 28 17:28:29 2019
8       127.0.0.1       2667    58508   Tue May 28 17:28:30 2019
9       127.0.0.1       2667    61068   Tue May 28 17:28:30 2019
10      127.0.0.1       2667    63628   Tue May 28 17:28:30 2019
11      127.0.0.1       2667    653     Tue May 28 17:28:31 2019
12      127.0.0.1       2667    3213    Tue May 28 17:28:31 2019
13      127.0.0.1       2667    5773    Tue May 28 17:28:31 2019
14      127.0.0.1       2667    8333    Tue May 28 17:28:32 2019
15      127.0.0.1       2667    10893   Tue May 28 17:28:32 2019
16      127.0.0.1       2667    13453   Tue May 28 17:28:34 2019
7       127.0.0.1       2666    56460   Tue May 28 17:28:29 2019
8       127.0.0.1       2666    59020   Tue May 28 17:28:30 2019
9       127.0.0.1       2666    61580   Tue May 28 17:28:30 2019
10      127.0.0.1       2666    64140   Tue May 28 17:28:30 2019
11      127.0.0.1       2666    1165    Tue May 28 17:28:31 2019
12      127.0.0.1       2666    3725    Tue May 28 17:28:31 2019
13      127.0.0.1       2666    6285    Tue May 28 17:28:31 2019
14      127.0.0.1       2666    8845    Tue May 28 17:28:32 2019
15      127.0.0.1       2666    11405   Tue May 28 17:28:32 2019
16      127.0.0.1       2666    13965   Tue May 28 17:28:34 2019
7       127.0.0.1       2665    57484   Tue May 28 17:28:30 2019
8       127.0.0.1       2665    60044   Tue May 28 17:28:30 2019
9       127.0.0.1       2665    62604   Tue May 28 17:28:30 2019
10      127.0.0.1       2665    65164   Tue May 28 17:28:30 2019
11      127.0.0.1       2665    2189    Tue May 28 17:28:31 2019
12      127.0.0.1       2665    4749    Tue May 28 17:28:31 2019
13      127.0.0.1       2665    7309    Tue May 28 17:28:31 2019
14      127.0.0.1       2665    9869    Tue May 28 17:28:32 2019
15      127.0.0.1       2665    12429   Tue May 28 17:28:33 2019
16      127.0.0.1       2665    14989   Tue May 28 17:28:34 2019
7       127.0.0.1       2664    56972   Tue May 28 17:28:29 2019
8       127.0.0.1       2664    59532   Tue May 28 17:28:30 2019
9       127.0.0.1       2664    62092   Tue May 28 17:28:30 2019
10      127.0.0.1       2664    64652   Tue May 28 17:28:30 2019
11      127.0.0.1       2664    1677    Tue May 28 17:28:31 2019
12      127.0.0.1       2664    4237    Tue May 28 17:28:31 2019
13      127.0.0.1       2664    6797    Tue May 28 17:28:31 2019
14      127.0.0.1       2664    9357    Tue May 28 17:28:32 2019
15      127.0.0.1       2664    11917   Tue May 28 17:28:33 2019
16      127.0.0.1       2664    14477   Tue May 28 17:28:34 2019
8       127.0.0.1       2668    57996   Tue May 28 17:28:30 2019
9       127.0.0.1       2668    60556   Tue May 28 17:28:30 2019
10      127.0.0.1       2668    63116   Tue May 28 17:28:30 2019
11      127.0.0.1       2668    141     Tue May 28 17:28:30 2019
12      127.0.0.1       2668    2701    Tue May 28 17:28:31 2019
13      127.0.0.1       2668    5261    Tue May 28 17:28:31 2019
14      127.0.0.1       2668    7821    Tue May 28 17:28:31 2019
15      127.0.0.1       2668    10381   Tue May 28 17:28:32 2019
16      127.0.0.1       2668    12941   Tue May 28 17:28:34 2019
17      127.0.0.1       2668    15501   Tue May 28 17:28:34 2019
```

위 사진은 각 child process에 history기록이 제한 개수만큼 기록되어있는 모습이다. Pre-fork를 통해 child를 5개 만들었고 각각의 process는 10개의 기록을 가질 수 있으므로 50개의 history기록이 있는 것을 확인할 수 있다. 다음 상태에서 연결을 계속 시도하면, 최근 기록으로 업데이트 되지만 출력 개수는 변함이 없다.



Ctrl+C를 통하여, SIGINT를 발생시켰을 때 동작이다. 각각의 child process를 먼저 종료시킨 후 최종적으로 서버를 닫는 것을 확인할 수 있다.

5. Conclusion

4-1과제는 3-3과제를 조금 더 응용하는 과제였습니다. 실습에서 사용했던 코드를 참고하여 원래 main function과 child function으로 나누어 fork가 완료된 후 child process들은 각각의 child function을 call하면서 동작하게 됩니다. Fork의 동작을 이해한다면 process를 생성하는 것은 큰 어려움이 없던 과제입니다. 과제를 진행하면서 난해한 부분은 signal을 정의해주는데 있었습니다. Signal을 parent와 child로 나누어 정의해야 하므로 정의해주지 않은 signal이 들어가는 경우가 생기고 code위치에따라 signal에 반응하기도 하고 안 하는 문제가 있어 잘못된 부분을 찾는데 시간이 지체된 부분이 있었습니다. Signal을 부분부분으로 정의하는 것이 아니라 정의하는 signal에 대하여 두 process모두 정의해준다면 쉽게 해결되는 문제입니다.