Assignment#4-2

시스템 프로그래밍 실습

제출일: 6월 07일 금요일

분 반: 화요일

담당 교수: 신영주

학 번: 2015722025

학 과: 컴퓨터정보공학부

이 름: 정용훈

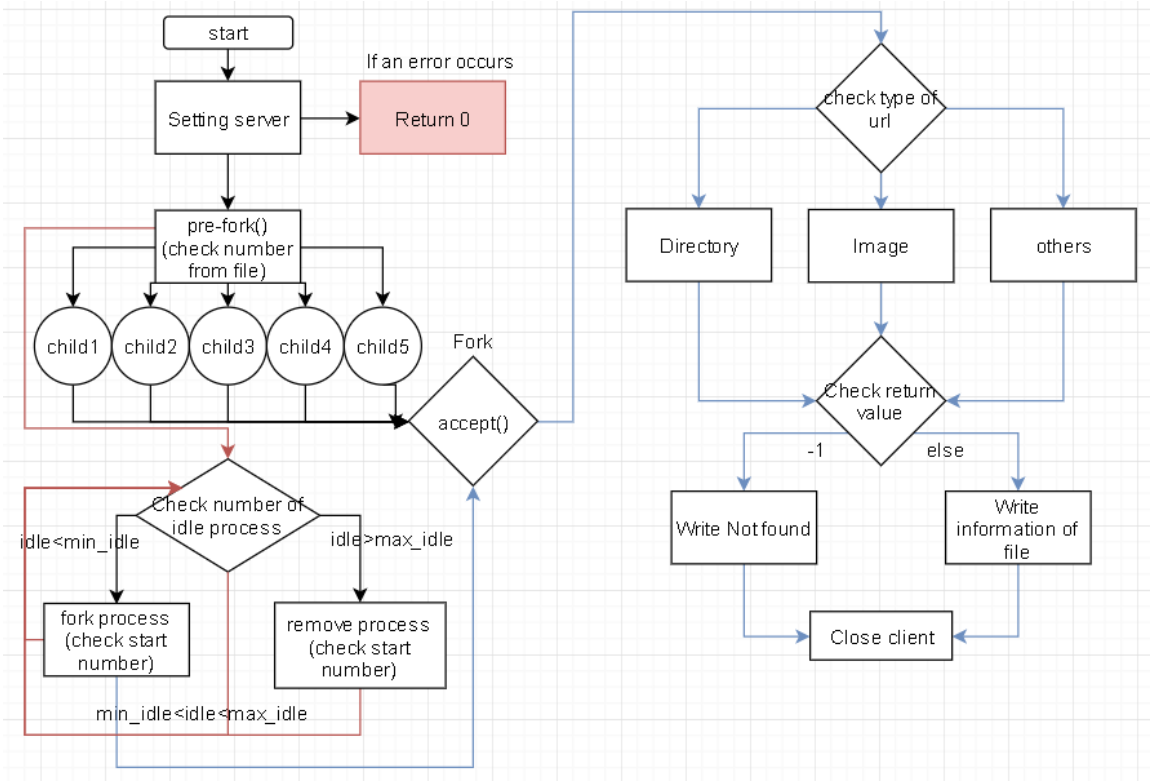
1. Introduction

4-2과제는 전 과제를 기반으로 공유메모리를 사용하여 history를 공유하고, idle프로세스와 busy프로세스의 개수를 구분하여, 프로세스의 개수를 계속해서 조절해주는 프로그램을 구현하는 것이다. 동기화 문제로는 mutex_lock와 unlock을 이용하며, 공유메모에 접근 할 때는 thread를 사용해야 한다.

2. Flow Chart

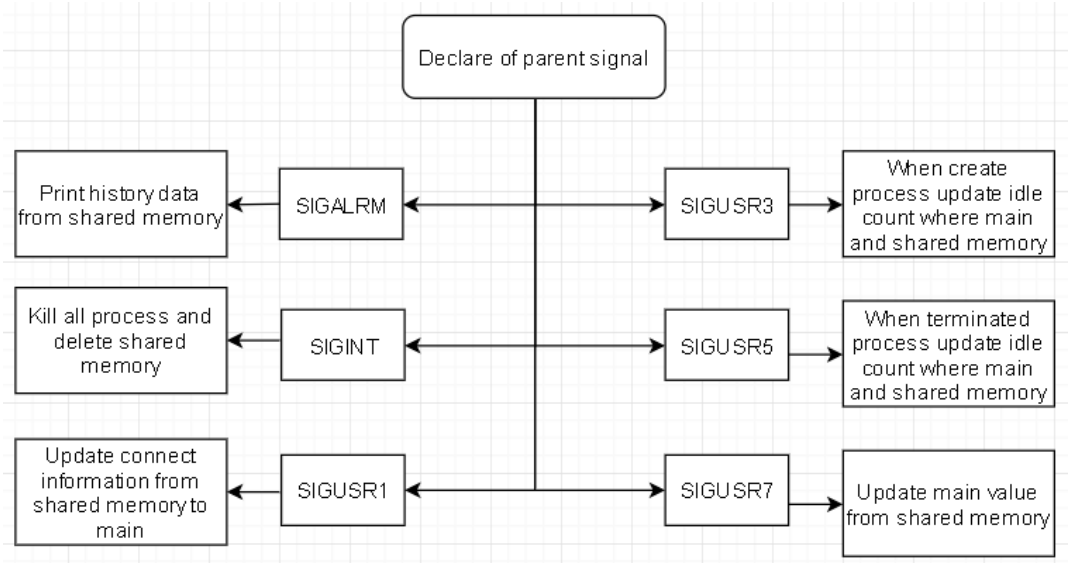
전 과제와 마찬가지로 prefork기법은 동일하게 사용한다. 단 해당 개수는 파일에서 읽어와 실행하게 되며, 생성된 프로세서의 개수는 idle수를 기반으로 프로세서의 개수가 가변적으로 계속 변할 수 있다. 관련된 flow chart는 아래와 같이 나타낼 수 있다.

Main



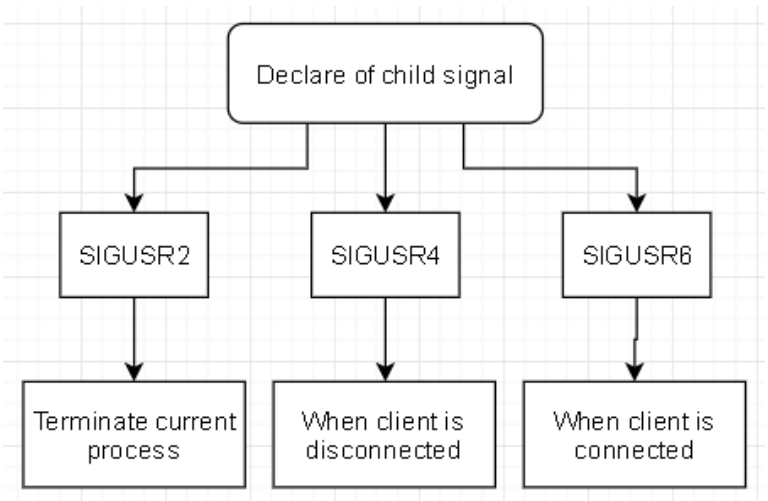
전 과제에서는 prefork를 수행 후 main함수는 pause상태를 유지하며, SIGINT신호를 기다렸지만 이번 main함수에서는 공유메모리에서 제공되는 idle count를 계속해서 check하여 프로세스의 개수를 조절해주는 역할을 해야 하기 때문에 왼쪽 아래와 같은 함수가 추가적으로 실행된다. Child main에서는 정보를 업데이트해주어야 하기 때문에 연결 하는 순간과 연결이 끝날 때 parent에 signal을 주면서 정보를 최신화 시켜야 한다.

SignalHandler_parent



Parent의 signal을 정의해주는 함수로써 다음과 같은 Signal들을 정의해준다. SIGUSR1, 2가 아닌 다른 것들은 Posix에서 정의한 1~30번이 아닌 사용하지 않는 signal 50번부터 정의했으며, Signal이 많은 이유는 처음부터 체계적으로 설계하지 못한 이유가 있다.

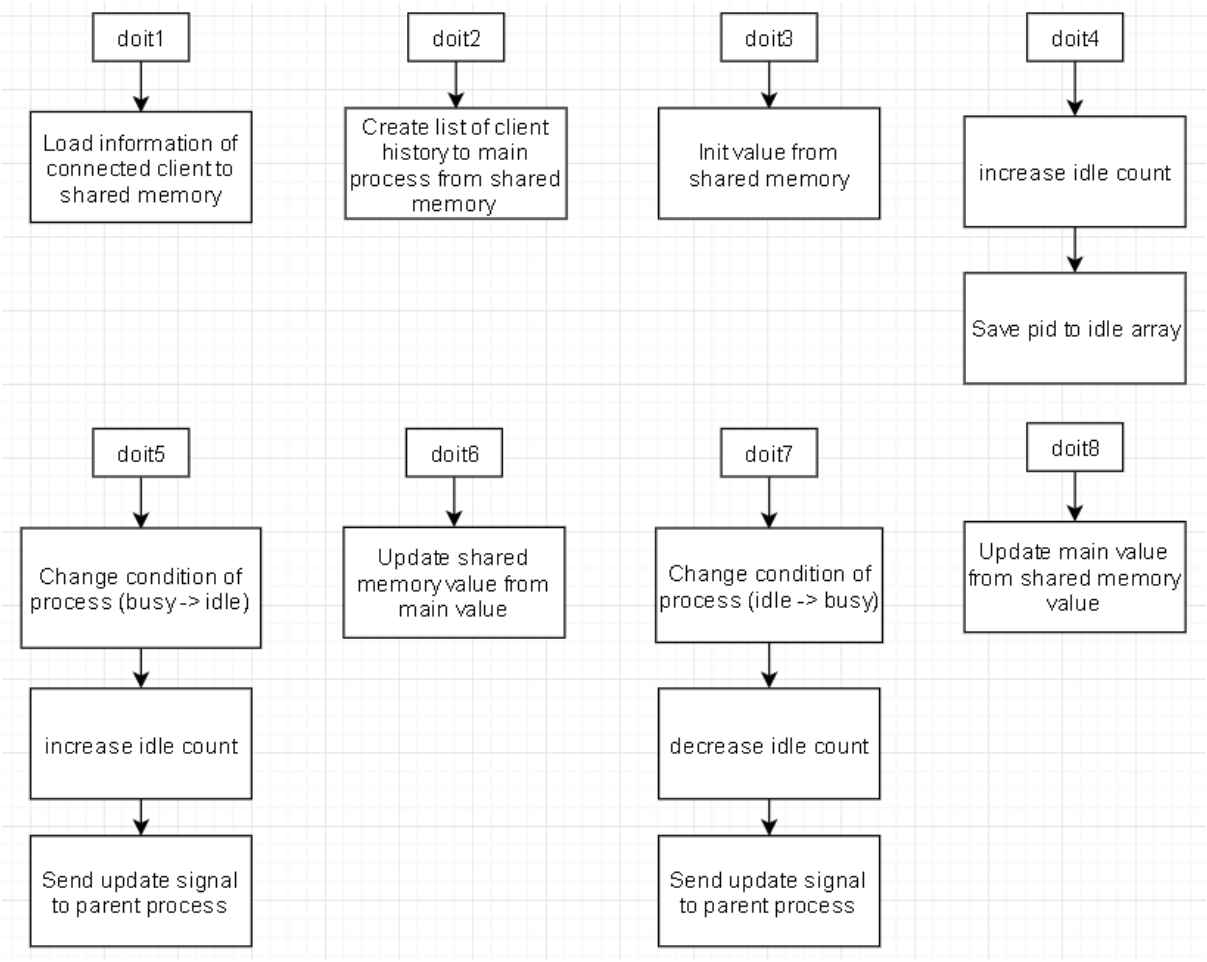
SignalHandler_child



Idle process의 수를 client가 연결되었을 때 연결이 끊어질 때 경우도 조절해주어야 하기 때문에 클라이언트가 연결되는 경우와 끊어지는 경우를 Signal 주면서 해당 Thread에서 공유메모리에 접근하여 idle수를 조절해주는 동작을 한다.

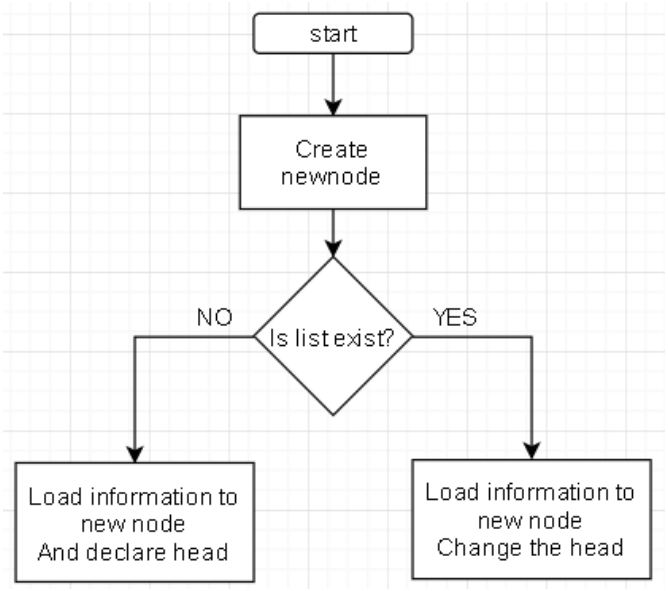
각 Thread에서 사용하는 doitN함수들

체계적으로 설계하지 못하면서 함수들이 난잡하게 여러 개가 존재합니다. 아래 사진은 각각의 thread가 사용하는 doit동작입니다.



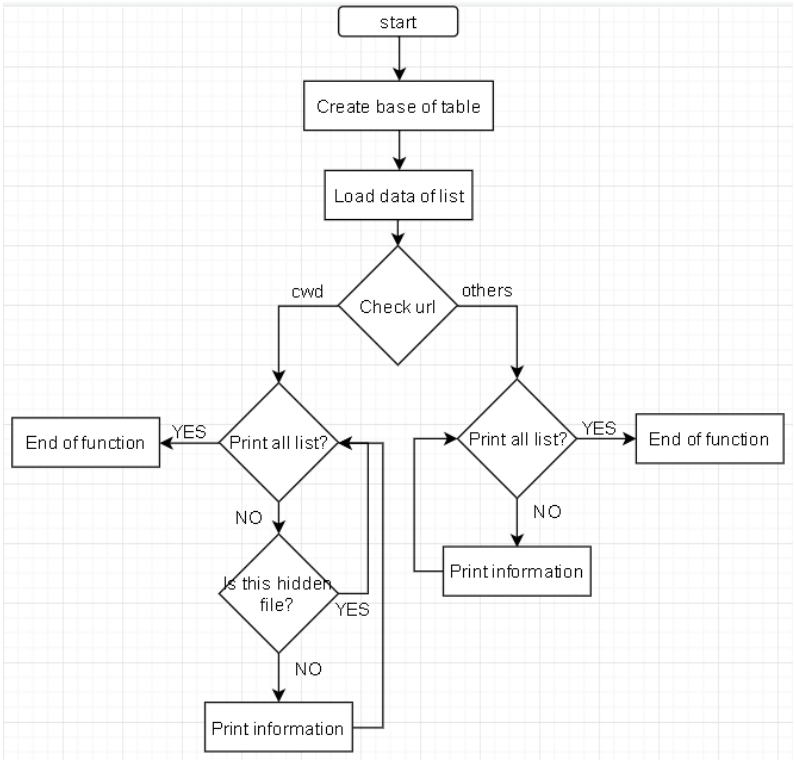
각각 함수의 종류는 다양하지만 동작 자체는 굉장히 간단하다. Doit1함수는 client가 연결되는 순간 history에 정보를 저장하기 위한 동작이고, doit2는 저장된 정보를 기반으로 main함수에서 list를 생성한다. (생성될 때는 최근 연결된 것부터 앞에 생성되므로, 시간기준 내림차순으로 생성된다.) doit3는 공유메모리의 값을 따로 초기화 시켜주기 위한 thread로 변수들의 값을 모두 0으로 만들어준다. Doit4는 fork되어 process가 생성 되는 경우 제어해주는 역할을 한다. Doit5와 doit7은 각 client들이 연결되고 끊어질 때 array와 idle count를 관리해주는 thread이며, 나머지 thread함수들은 보통 shared memory와 main의 변수를 update해주는 working을 담당하고 있다.

UpdateInfo



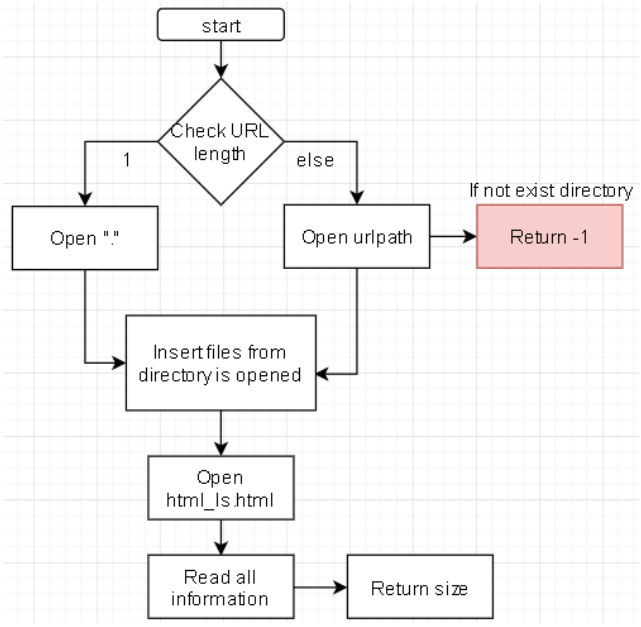
공유메모리의 history뿐 아니라 main process에서도 list를 관리해주기 때문에 update하는 함수가 필요하다.

PrintHTML(create file of html_ls.html)



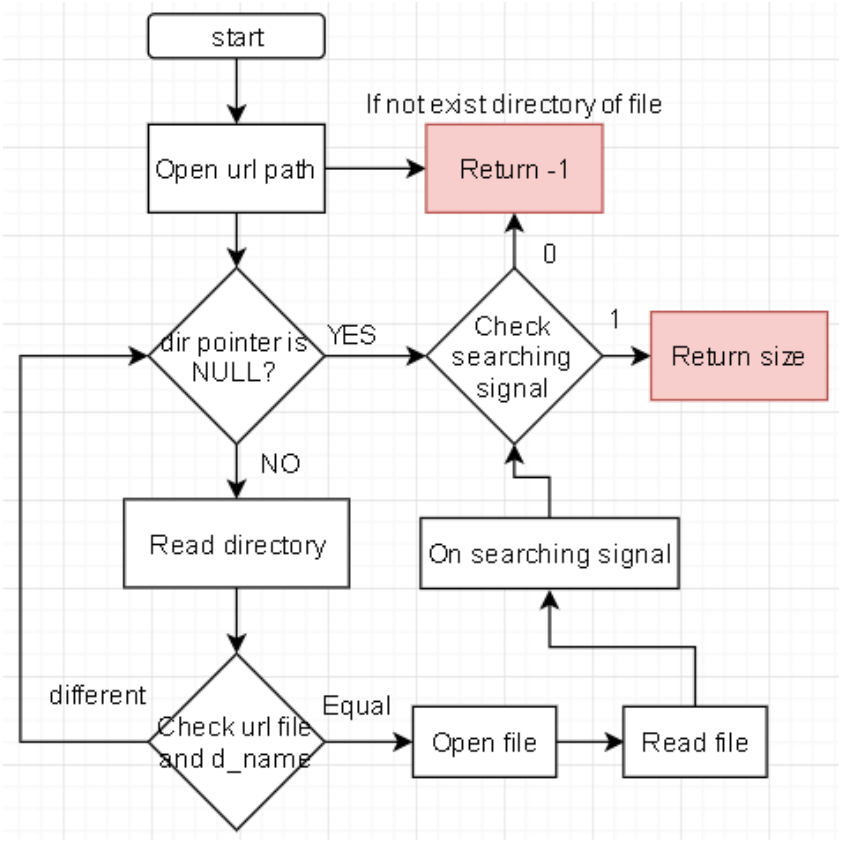
3-1 과제인 html_ls.hml을 만드는 과정으로 옵션 구현이 –a과 –al만 있기 때문에 구조만 동작 자체는 똑같지만 구조를 변경하게 되었다. 해당 함수는 Html함수에서 call하는 함수로써 아래 Html함수를 참고하면 이해하기 쉽다.

Html



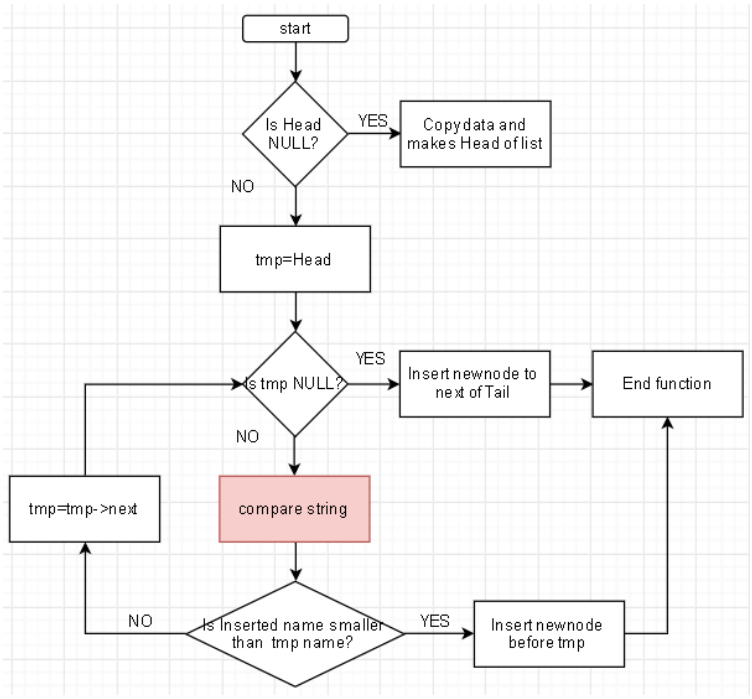
Open html을 하기 전 과정이 printHTML로써 html파일을 생성한 후 실행하게 된다.

Image & Normal (others)



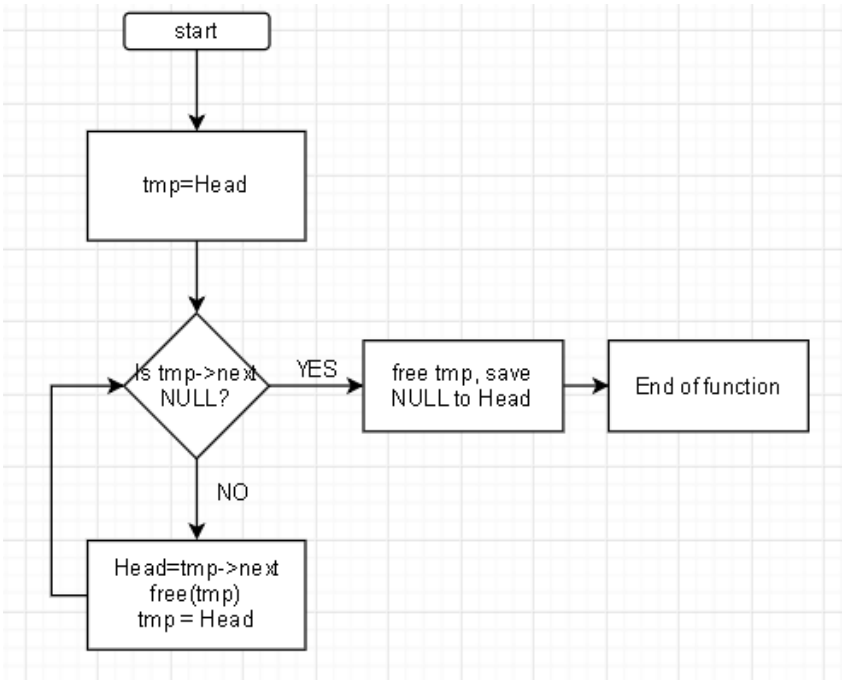
다음 함수는 Image와 나머지 다른 파일들을 처리해주는 함수로써 정보를 binary로 읽어와 write해주는 작업을 하게 된다 사실 image와 나머지 파일은 모두 binary를 통하여 읽을 수 있기 때문에 나눠줄 필요는 없다고 생각된다.

Insertnode



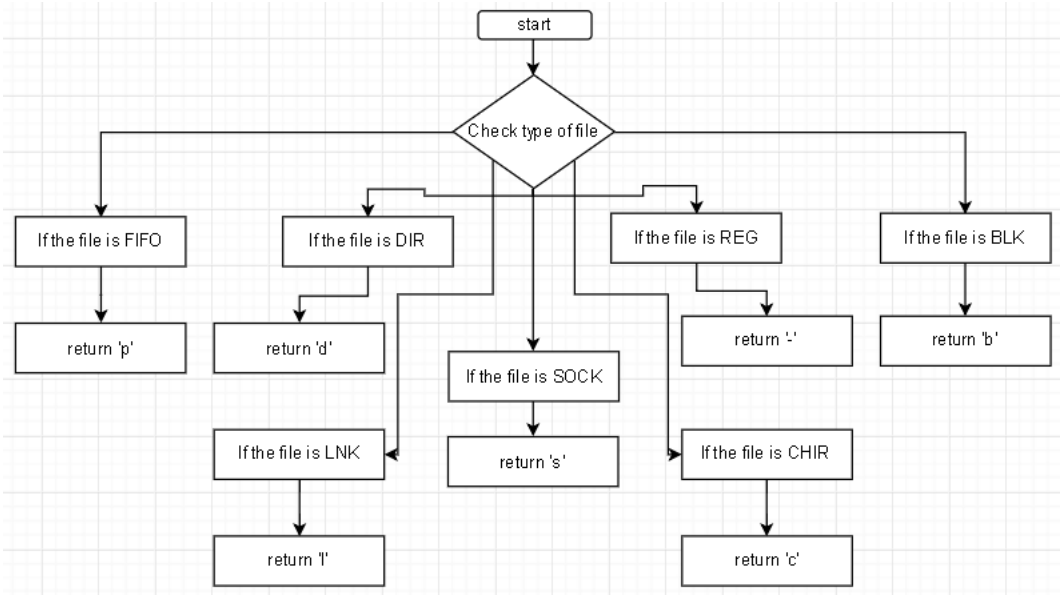
따로 sort함수를 사용하지 않고 insert를 실행 할 때부터 sort가 되며, list가 생성된다. Insert함수는 기존 함수와 동일하게 사용되므로 전체적인 변화는 없지만, S옵션을 사용면 size를 비교해야 하므로 compare를 하는 부분이 변경되게 된다. 해당 문제는 함수는 같고 조건만 바꿔주어 해결하였다.

Deletelist



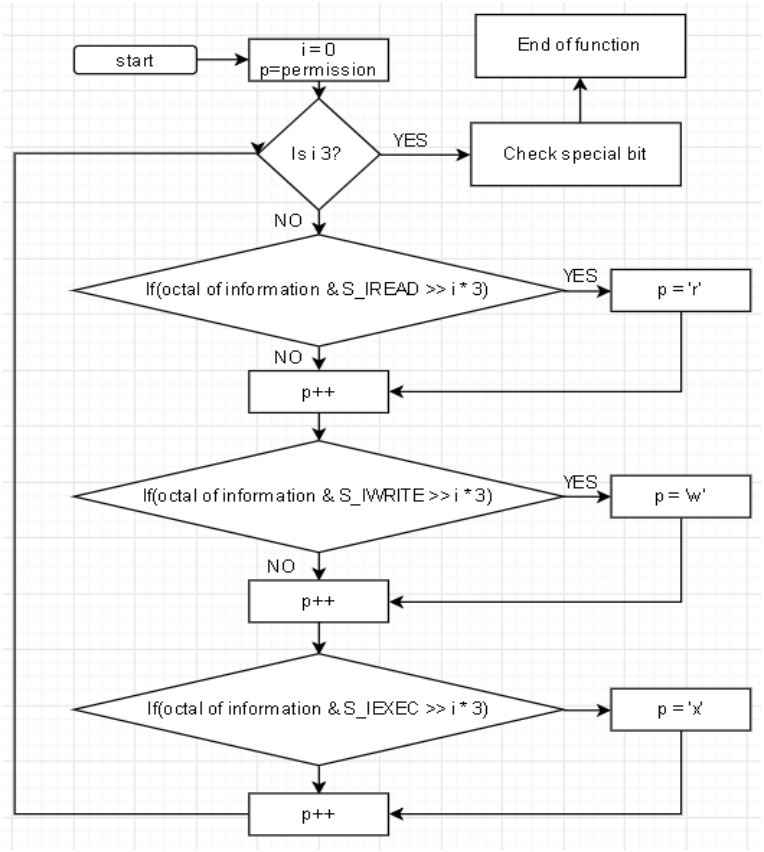
모든 정보 출력 후 linked list를 제거하는 함수다.

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/////////////////////////////////////
        for(int i=0;i<StartProcess;i++)
        {
                if((PID=fork())>0) //parent process
                {
                        print information start condition;
                        continue;
                }

                else if(PID==0) //child process
                {
                        Declare child signal;
                        break;
                }

                else //error
                {
                        Error;
                        return;
                }
        }

        if(PID>0) //parent
        {
                alarm(10); //alarm
                while(1)
                {
                        pause(); //wait parent process
                        if(Idle process smaller thaan min_idle)
                        {
                                while(idle_porcess != start_idle_number)
                                {
                                        if((PID=fork())>0)
                                        {
                                                print information;
                                        }

                                        else if(PID==0)
                                        {
                                                Declare child signal;
                                                go to child main;
                                        }

                                        else //error
                                        {
                                                Error;
                                                return;
                                        }
                                }

                        }
```

```
                        else if(idle_number > max_idle_number)
                        {
                                while(idle_number>max_process_number)
                                {
                                        delete idle process;
                                        print_current_condition;
                                }
                        }
                        else
                                continue;
                }

        }
        go to child main;
        close(socket_fd); //close socket
        return 0;
}
```

————————————————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;
}
```

각종 therad에 사용되는 doit 함수

```
void *doit1(void *vptr) //for sned history information to shared memory
{
        Get shared memory;
        Apply shared memory to process;

        lcok;

        update client information;

        unlock

        return NULL;
}
```

```
void *doit2(void *vptr)
{
        Get shared memory;
        Apply shared memory to process;

        lcok;

        shm_info=(Sh*)shm_addr;
        update main value form shared memory;

        unlock;
        return NULL;
}
```

```
void *doit3(void *vptr) //Init thread
{
        Get shared memory;
        Apply shared memory to process;

        Declare 0 to all shared memory value;

        return NULL;
}
```

```
void *doit4(void *vptr) //Just create process
{
        Get shared memory;
        Apply shared memory to process;

        lock

        load address of shared memory;
        printf("[%s] IdleProcessCount : %d₩n",time_buf,++shm_info->idle);
        ++shm_info->process;

        Save PID to main array;
        Save PID to shared memory array;

        Cur_idle=shm_info->idle;
        Cur_process=shm_info->process;

        unlock
        return NULL; //return NULL
}
```

```
void *doit5(void *vptr)
{
        Get shared memory;
        Apply shared memory to process;

        lcok;

        load address of shared memory;
        Update idle array and busy array;
        printf("[%s] IdleProcessCount : %d₩n",time_buf,++shm_info->idle); //print condition of idle
        sned SIGUSR7 signal to parent

        unlock
        return NULL; //return NULL
}
```

```
void *doit7(void *vptr)
{
        Get shared memory;
        Apply shared memory to process;

        lcok;

        load address of shared memory;
        Update idle array and busy array;
        printf("[%s] IdleProcessCount : %d₩n",time_buf,--shm_info->idle); //print condition of idle
        sned SIGUSR7 signal to parent

        unlock
        return NULL; //return NULL
}
```

```
void *doit6(void *vptr)
{
        Get shared memory;
        Apply shared memory to process;

        lcok;

        load address of shared memory;
        printf("[%s] IdleProcessCount : %d₩n",time_buf,--shm_info->idle);
        --shm_info->process;
        Update array information from shared memory;
        Cur_idle=shm_info->idle; //update value of idle
        Cur_process=shm_info->process;

        unlock;

        return NULL;
}
```

```
void *doit8(void *vptr)
{
        Get shared memory;
        Apply shared memory to process;

        lcok;

        load address of shared memory;
        Update shared memory array information from main;
        Cur_idle=shm_info->idle; //update value of idle
        Cur_process=shm_info->process;

        unlock;
        return NULL; //return NULL
}
```

SignalHandler_parent

```
void signalHandler_parent(int sig)
{
        int status;
        if(sig == SIGALRM) //alram signal
        {
                printHistroy();
                alarm(10);
        }

        if(sig == SIGINT) //Check signal of ^C
        {
                exit all process, and main process;
        }

        if(sig==SIGUSR1)
                Update history();

        if(sig==SIGUSR3)
                create_process_working_function();

        if(sig==SIGUSR5)
                terminated_process_working_function();

        if(sig==SIGUSR7)
                updateidle();
}
```

SignalHandler_child

```
void signalHandler_child(int sig)
{
        if(sig==SIGUSR2)
                exit(0);
        if(sig==SIGUSR6)
                connect_client();
        if(sig==SIGUSR4)
                disconnect_client();
}
```

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:~$ ./ipc_server_40229
[Thu Jun  6 08:21:10 2019] Server is started.
[Thu Jun  6 08:21:10 2019] 12448 process is forked.
[Thu Jun  6 08:21:10 2019] IdleProcessCount : 1
[Thu Jun  6 08:21:10 2019] 12450 process is forked.
[Thu Jun  6 08:21:10 2019] IdleProcessCount : 2
[Thu Jun  6 08:21:10 2019] 12452 process is forked.
[Thu Jun  6 08:21:10 2019] IdleProcessCount : 3
[Thu Jun  6 08:21:10 2019] 12454 process is forked.
[Thu Jun  6 08:21:10 2019] IdleProcessCount : 4
[Thu Jun  6 08:21:10 2019] 12456 process is forked.
[Thu Jun  6 08:21:10 2019] IdleProcessCount : 5
```

서버를 시작하는 경우 start문구와 process들이 생성되는 것을 확인할 수 있으며, idle_count에 의해 process count가 증가한다.

```
============= New client =============
[Thu Jun  6 08:22:47 2019]
IP : 127.0.0.1
Port : 21733
======================================
[Thu Jun  6 08:22:47 2019] IdleProcessCount : 4
```

Process에 연결요청이 들어오면 다음과 같이 client 연결 메시지와 함께 idle count를 공유메모리에서 관리한다.

```
========= Disconeected Client ========
[Thu Jun  6 08:22:47 2019]
IP : 127.0.0.1
Port : 21733
======================================
[Thu Jun  6 08:22:52 2019] IdleProcessCount : 5
```

Process의 sleep time이 지나면 연결이 끊어지면 Disconnect 메시지 출력과 함께 idle수를 관리한다.

```
============= New client =============
[Thu Jun  6 08:28:02 2019]
IP : 127.0.0.1
Port : 36581
======================================
[Thu Jun  6 08:28:02 2019] IdleProcessCount : 4
============= New client =============
[Thu Jun  6 08:28:03 2019]
IP : 127.0.0.1
Port : 37605
======================================
[Thu Jun  6 08:28:03 2019] IdleProcessCount : 3
[Thu Jun  6 08:28:03 2019] 12963 process is forked.
[Thu Jun  6 08:28:03 2019] IdleProcessCount : 4
[Thu Jun  6 08:28:03 2019] 12965 process is forked.
[Thu Jun  6 08:28:03 2019] IdleProcessCount : 5
```

다음과 같이 클라이언트를 두 개 연결요청이 들어오면 최소 Idle process의 개수보다 적기 때문에 start process number가 될 때까지 process를 생성해주는 모습이다.

```
========= Disconeected Client ========
[Thu Jun  6 08:30:27 2019]
IP : 127.0.0.1
Port : 40165
======================================
[Thu Jun  6 08:30:32 2019] IdleProcessCount : 6
========= Disconeected Client ========
[Thu Jun  6 08:30:27 2019]
IP : 127.0.0.1
Port : 40677
======================================
[Thu Jun  6 08:30:32 2019] IdleProcessCount : 7
[Thu Jun  6 08:30:32 2019] 12994 process is terminated.
[Thu Jun  6 08:30:32 2019] IdleProcessCount : 6
[Thu Jun  6 08:30:32 2019] 12992 process is terminated.
[Thu Jun  6 08:30:32 2019] IdleProcessCount : 5
```

또한 연결되었던 process가 돌아오면서 7개까지 Idle process가 늘어나는데, 이는 초대 Idle process의 값보다 크기 때문에 다시 start process number까지 process를 제거해주는 모습이다.

```
========= Connection History =========
NO.     IP              PID     PORT    TIME
1       127.0.0.1       13061   46309   Thu Jun  6 08:31:51 2019
2       127.0.0.1       13051   45797   Thu Jun  6 08:31:50 2019
3       127.0.0.1       13049   45285   Thu Jun  6 08:31:49 2019
4       127.0.0.1       13039   44773   Thu Jun  6 08:31:49 2019
5       127.0.0.1       13037   44261   Thu Jun  6 08:31:49 2019
6       127.0.0.1       12996   43749   Thu Jun  6 08:31:48 2019
7       127.0.0.1       13014   43237   Thu Jun  6 08:31:48 2019
8       127.0.0.1       13012   42725   Thu Jun  6 08:31:48 2019
9       127.0.0.1       13000   42213   Thu Jun  6 08:31:48 2019
10      127.0.0.1       12998   41701   Thu Jun  6 08:31:47 2019
```

다음은 history출력으로 가장 최근에 접속된 client의 정보가 1번에 최신화 되는 모습으로, 공유메모리의 정보를 기반으로 모든 child process의 동작을 함께 관리한다.

```
[Thu Jun  6 08:33:35 2019] 12998 process is terminated.
[Thu Jun  6 08:33:35 2019] IdleProcessCount : 5
[Thu Jun  6 08:33:35 2019] 13000 process is terminated.
[Thu Jun  6 08:33:35 2019] IdleProcessCount : 4
[Thu Jun  6 08:33:35 2019] 13012 process is terminated.
[Thu Jun  6 08:33:35 2019] IdleProcessCount : 3
[Thu Jun  6 08:33:35 2019] 13014 process is terminated.
[Thu Jun  6 08:33:35 2019] IdleProcessCount : 2
[Thu Jun  6 08:33:35 2019] 12996 process is terminated.
[Thu Jun  6 08:33:35 2019] IdleProcessCount : 1
[Thu Jun  6 08:33:35 2019] 13061 process is terminated.
[Thu Jun  6 08:33:35 2019] IdleProcessCount : 0
[Thu Jun  6 08:33:35 2019] Server is terminated.
```

다음은 종료하는 모습으로 프로세스를 10개까지 만드는 과정에서 존재하는 process가 6개가 되며, 6개 모두 순차적으로 종료되는 것을 확인할 수 있다.

5. Conclusion

4-2과제는 전 과제와는 다르게 공유메모리를 사용하여, 자식들이 공유하지 못하던 정보, 즉 history를 공유 가능하게 설계하며, idle process의 count또한 관리하면서 프로그램이 보다 원활하게 돌아갈 수 있도록 process의 수를 조절하는 기능 또한 추가해야 한다. 이번 과제에서 까다로웠던 점은 thread의 사용법과 공유메모리를 처음 사용하였기에 메모리적 문제가 발생하기도 하였으며, 이를 해결하는데 실습 자료뿐 아니라 검색을 통한 참고자료가 많이 필요했다. 또한 모든 process가 연결 중 request를 client에서요청시 동작하고 있던 process가 강제 종료 되는 문제가 발생하여 zombie process가 생기기도 하였는데 이는 새로 들어온 request요청이 write를 해야 하는데 대기중인 process가 없어 write함수 자체에서 SIGPIPE신호를 process자체에 신호를 주면서 강제 종료되는 것을 확인하였다. 해결방안으로는 SIGPIPE를 SIG_ING로 정의하여, signal을 무시할 수 있도록 설정하여 process가 자동으로 죽는 것을 방지하였다. Pre_fork에 비해 client의 요청에 대한 동작 상황을 하나씩 생각해주어야 했기 때문에 나름 까다로운 과제였다고 생각된다.