시스템프로그래밍실습 보고서

Assignment 3-2

과 목 시스템프로그래밍실습

담당교수 이기훈교수님

학 과 컴퓨터정보공학부

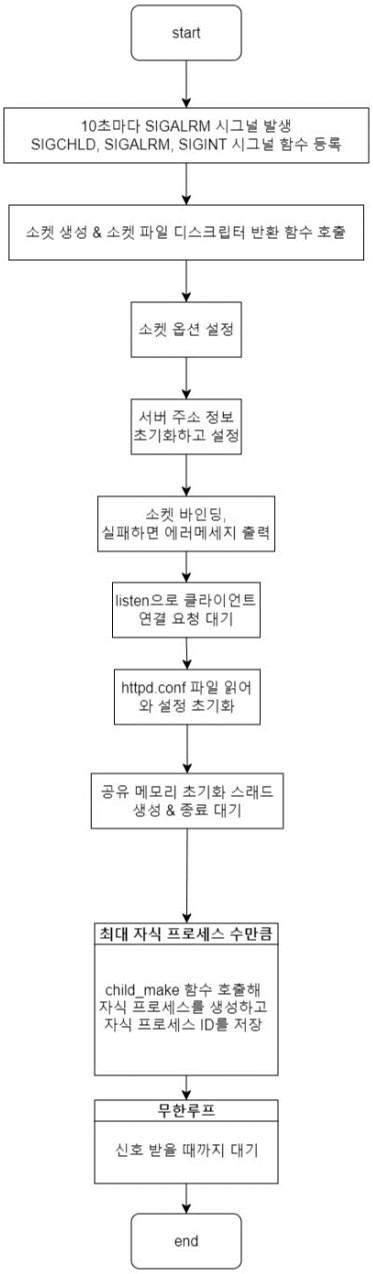
학 번 2021202058

이 름 송채영

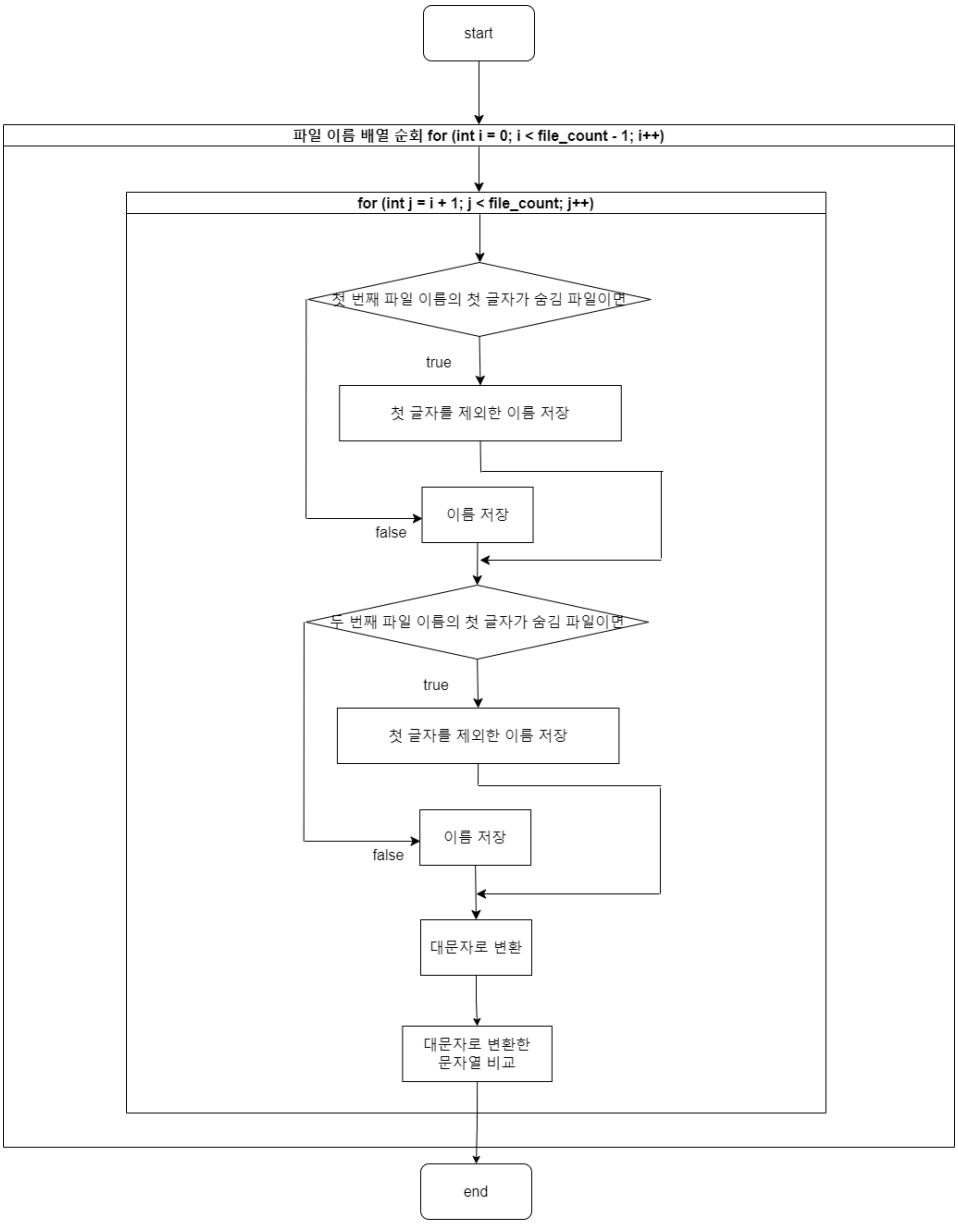
1. Introduction

이번 과제는 Assignment 3-1 과제를 기반으로 해 Process 에서 공유메모리에 접근할 때 thread 를 생성한 뒤 이를 활용해서 서버를 구현하는 것이다. 이전 과제와 마찬가지로 fork 함수를 활용하여 StartProcess 개의 자식 프로세스를 생성하며, 공유 메모리는 프로세스 간의 통신 및 동기화에 사용이 된다. client 의 연결을 처리하지 않는 자식 프로세스인 Idle process 는 특정 수의 Idle process 를 유지하도록 관리한다.

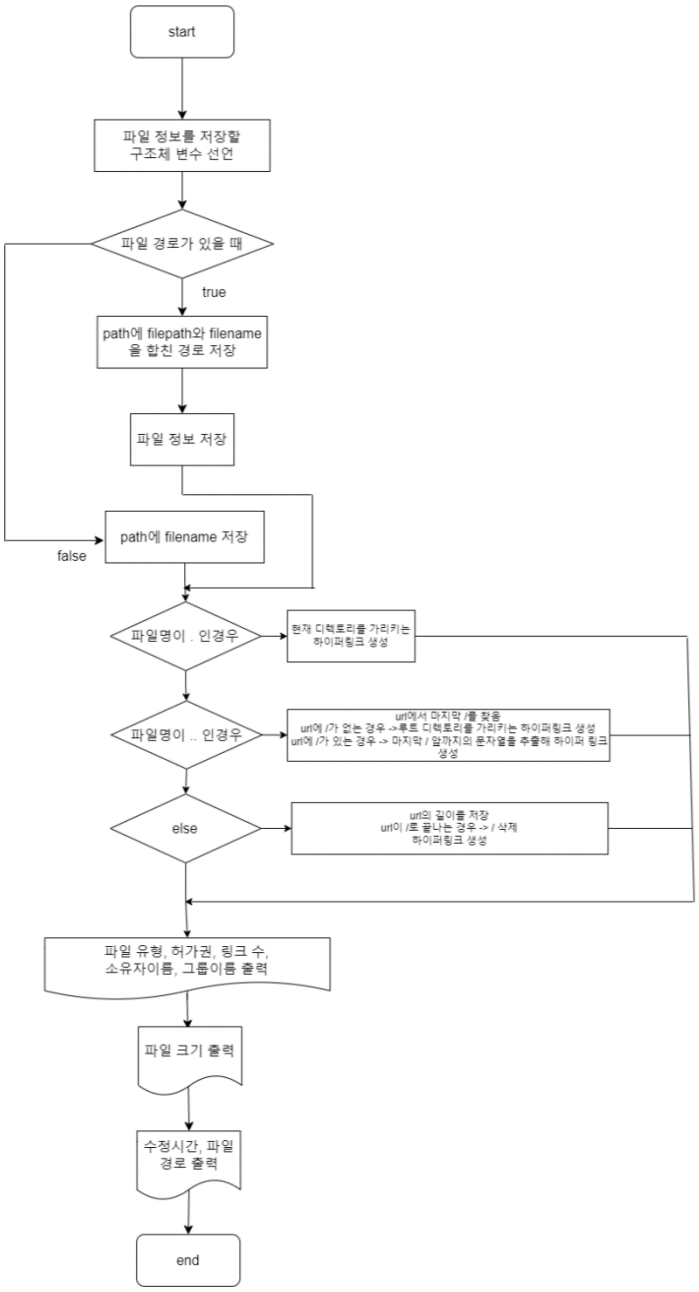
2. Flow chart



코드의 전체적인 흐름을 flow chart 로 나타내었다. 우선 main 함수에 대해 자세히 설명해보면 다음과 같다. 프로그램이 시작하면 10 초마다 SIGALRM 시그널이 발생하도록 alarm 을 설정하고 SIGCHLD, SIGALRM, SIGINT 시그널 핸들러를 handle_signal 로 등록한다. 소켓을 생성하고 소켓 파일 디스크립터 반환 함수를 호출한 후 소켓 옵션을 설정한다. memset 을 사용하여 서버 주소를 초기화 하고 설정해준다. 소켓 바인딩을 실행하는데, 이때 실패하면 에러 메세지를 출력해주고, listen() 함수를 사용해 클라이언트의 연결 요청을 대기한다. httpd.conf 파일을 읽어와 설정을 초기화 해준다. 이후 공유 메모리 초기화 thread 를 생성하고 종료를 대기한다. 최대 자식 프로세스만큼 child_make 함수를 호출해 자식 프로세스를 생성하고 자식 프로세스 ID 를 저장한다. 무한루프를 돌아 신호를 받을 때까지 대기한다. 프로그램을 종료한다.

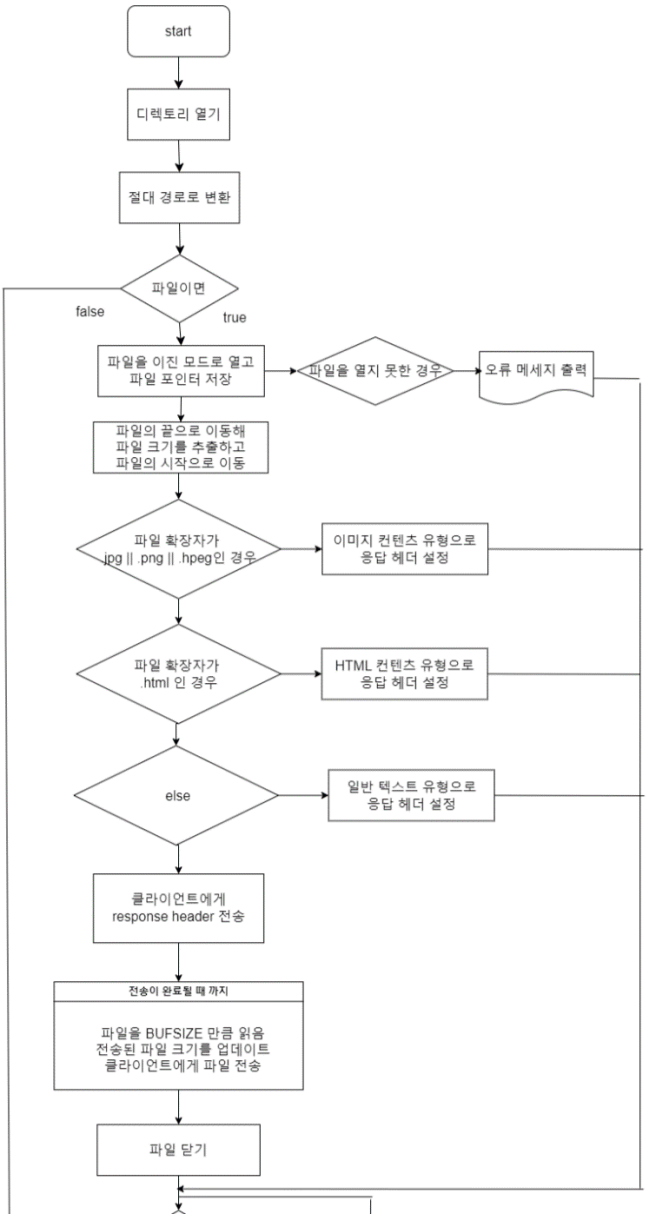


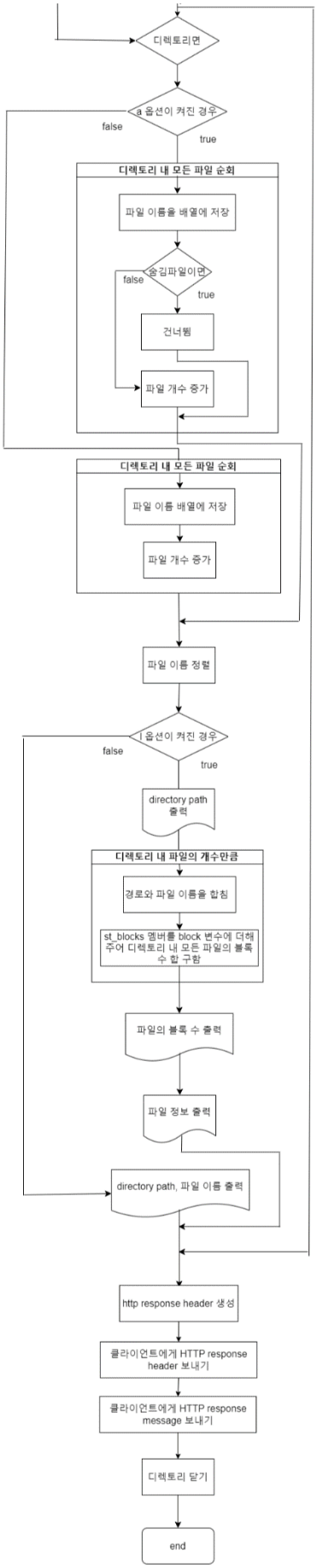
다음은 sort 함수의 flow chart 이다. 우선 파일의 개수만큼 반복한다. 첫 번째 파일이 숨김 파일일 경우, 파일의 이름을 길이만큼 반복하고 첫 글자를 제외한 이름을 저장한다. 숨김파일이 아닐 경우 이름을 저장해준다. 두 번째 파일도 동일하게 진행한다. 먼저 대문자로 변환해준 후 대문자로 변환한 문자열을 비교한다.



다음은 print_file_info, 파일 정보를 출력하는 함수의 flow chart 이다. 파일 정보를 읽어 구조체 변수인 filestat 에 저장한다. 파일 경로가 있고, filepath 와 filename 이 같지 않으면 파일 경로가 존재하는 경우이므로 경로와 파일 이름을 합치며, 파일 경로가 없는 경우 파일이름을 정보로 읽는다. 파일명이 ., .., else 인 경우로 나누어 하이퍼링크를 준다.

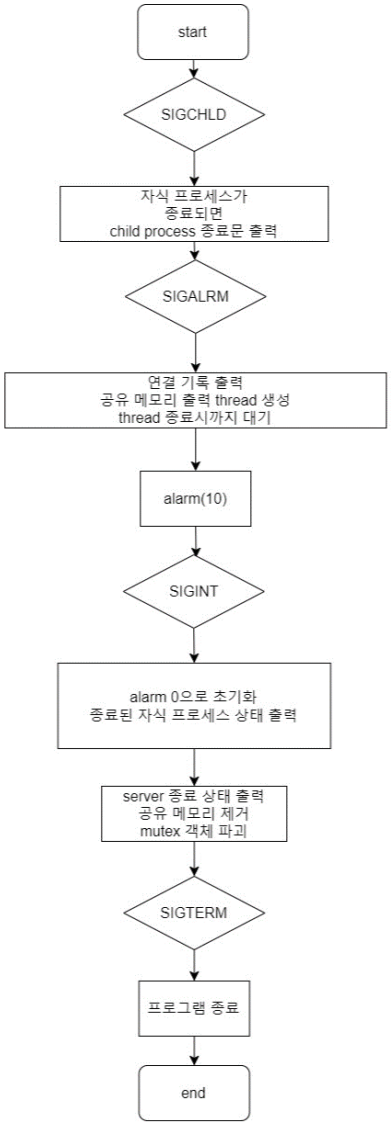
먼저 .인 경우 현재 디렉토리를 가리키는 하이퍼링크를 생성하며, ..일 경우 url 에서 마지막 /을 찾고 /이 있으면 루트 디렉토리를 가리키는 하이퍼링크를 생성하고, 없을 경우 마지막 / 앞까지의 문자열을 추출한 후 하이퍼링크를 생성해준다. else 인 경우 url 의 길이를 저장한 후 url 이 /로 끝나면 /을 삭제한 후 하이퍼링크를 생성하도록 하였다. 파일 유형, 허가권, 링크 수, 소유자,호출해주며, 파일 크기, 수정 시간, 파일 경로를 출력한 후 프로그램을 종료한다.



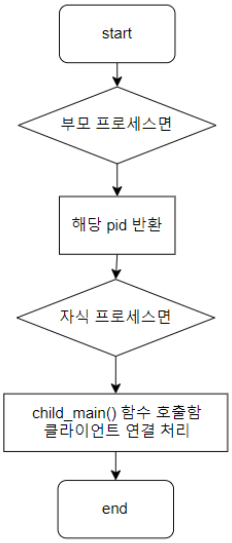

디렉토리면
a 옵션이 켜진 경우
false
true
디렉토리 내 모든 파일 순회
파일 이름을 배열에 저장
숨김파일이면
false
true
건너뜀
파일 개수 증가
디렉토리 내 모든 파일 순회
파일 이름 배열에 저장
파일 개수 증가
파일 이름 정렬
l 옵션이 켜진 경우
false
true
directory path 출력
디렉토리 내 파일의 개수만큼
경로와 파일 이름을 합침
st_blocks 멤버를 block 변수에 더해 주어 디렉토리 내 모든 파일의 블록 수 합 구함
파일의 블록 수 출력
파일 정보 출력
directory path, 파일 이름 출력
http response header 생성
클라이언트에게 HTTP response header 보내기
클라이언트에게 HTTP response message 보내기
디렉토리 닫기
end

다음은 list files 함수의 flow chart 이다. 파일을 열어준 후 절대경로로 변환한다. 연 파일이 파일일때와 디렉토리 일 때로 나눈다. 파일일 때, 파일을 이진 모드로 열고 파일 포인터를 file 변수에 저장한다. 파일을 열지 못했을 경우 오류 메세지를 출력한다. 파일의 끝으로 이동해 파일 크기를 추출한 후 파일의 시작으로 이동한다. 파일 확장자에 따라 content-type 을 설정하였는데, .jpg, .png, .jpeg 일 때 이미지 컨텐츠 유형으로 응답 헤더를 설정한다. 파일 확장명이 .html 인 경우 HTML 컨텐츠 유형으로 응답 헤더를 설정한다. else 일 경우 일반 텍스트 유형으로 응답 헤더를 설정하고 클라이언트에게 response header 를 전송하였다. 전송이 완료될 때 까지 반복하며 파일을 BUFSIZE 만큼 읽어서 전송된 파일 크기를 업데이트 하고 클라이언트에게 파일을 전송한다. 파일을 닫는다. 디렉토리일 경우 a 옵션이 꺼져 있을 때와 켜져 있을 때, l 옵션이 켜져 있을 때로 나눈다. a 옵션이 꺼져 있으면 디렉토리 내 모든 파일을 순회하며 파일이름을 배열에 저장한다. .과 ..은 건너뛰고 그 외의 파일은 파일 개수를 증가시킨다. a 옵션이 켜져 있을 경우, 디렉토리 내 모든 파일을 파일 이름을 배열에 저장하고 파일 개수를 증가시킨다. 파일 이름 정렬 함수를 호출해 정렬한다. l 옵션이 켜져 있으면, 디렉토리 경로와 total 을 출력해주고 파일 정보 출력 함수를 호출해 출력한다. HTTP response header 를 생성해 작성해 준 후, 클라이언트에게 HTTP response header 와 message 를 보낸 후 디렉토리를 닫고 프로그램을 종료한다.

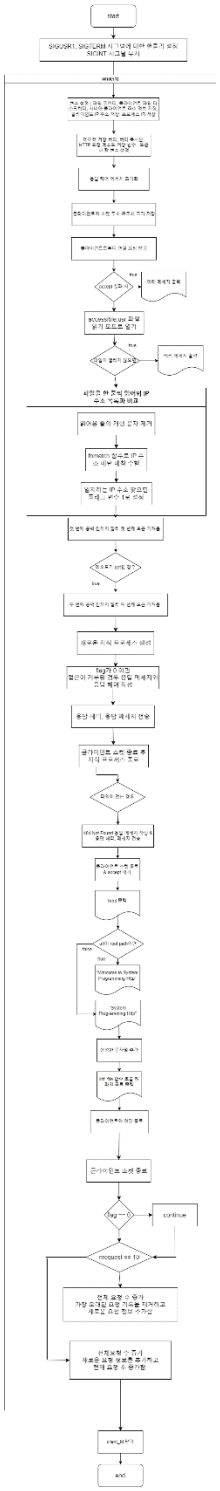
위 함수에서 html 에 directory path 와 total 그리고 filename 을 table 형태로 출력해주기 위해 html 코드로 출력하는 부분을 추가했다. 또한 html_ls.html 파일일 경우 건너뛰는 코드를 추가해 html_ls.html 파일이 출력되지 않도록 하였다. flow chart 의 흐름은 기존과 변함이 없다.



다음으로 handle_signal 함수에 대한 flow chart 이다. 우선 SIGCHLD 시그널 핸들러일 경우 자식 프로세서가 종료되면, 모든 종료된 자식 프로세스를 처리하며 Child process 가 종료되었다는 구문을 출력해준다. signal 이 SIGALRM 일 경우 연결 기록을 출력한다. 공유 메모리 출력 thread 를 생성하고 thread 가 종료될 때까지 대기한다. alarm 을 10 으로 설정한다. 다음으로 SIGINT 시그널인 경우 alram 을 0 으로 초기화한 후 종료된 자식 프로세스의 상태를 출력한다. 이어서 부모 프로세스 종료를 출력하고 공유 메모리를 제거하고 mutex 객체를 파괴한다. 마지막으로 SIGTERM 의 경우 프로그램을 종료한다.

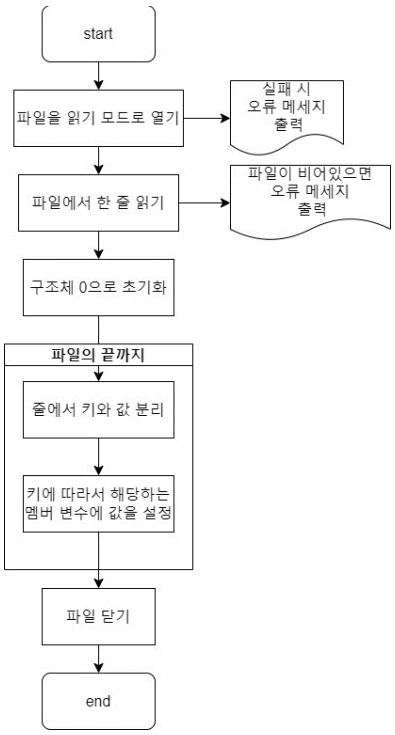


다음으로 child_make 함수이다. 부모 프로세스의 경우 해당 pid 를 반환하며 자식 프로세스의 경우 child_main 함수를 호출해 클라이언트 연결을 처리한다.

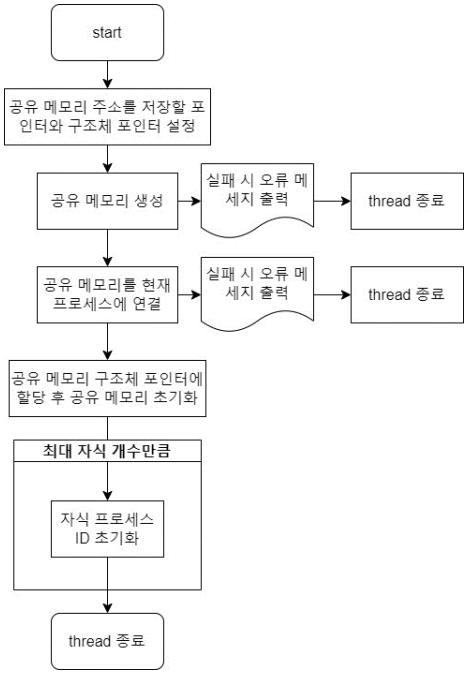


다음으로 child_main 함수이다. SIGUSR1, SIGTERM 시그널에 대한 핸들러를 설정하고, SIGINT 시그널은 무시한다. 필요한 변수를 선언한 후 while 이 1 일 때 클라이언트 IP 주소 저장 변수와 클라이언트로부터 받은 데이터를 저장할 버퍼, 버퍼 복사본, HTTP 요청 메소드 저장 변수, HTTP 요청 메시지 토큰화 할 포인터를 선언한 후 응답 헤더와 메시지를 초기화한다. 클라이언트의 소켓 주소 구조체 크기를 저장한 후

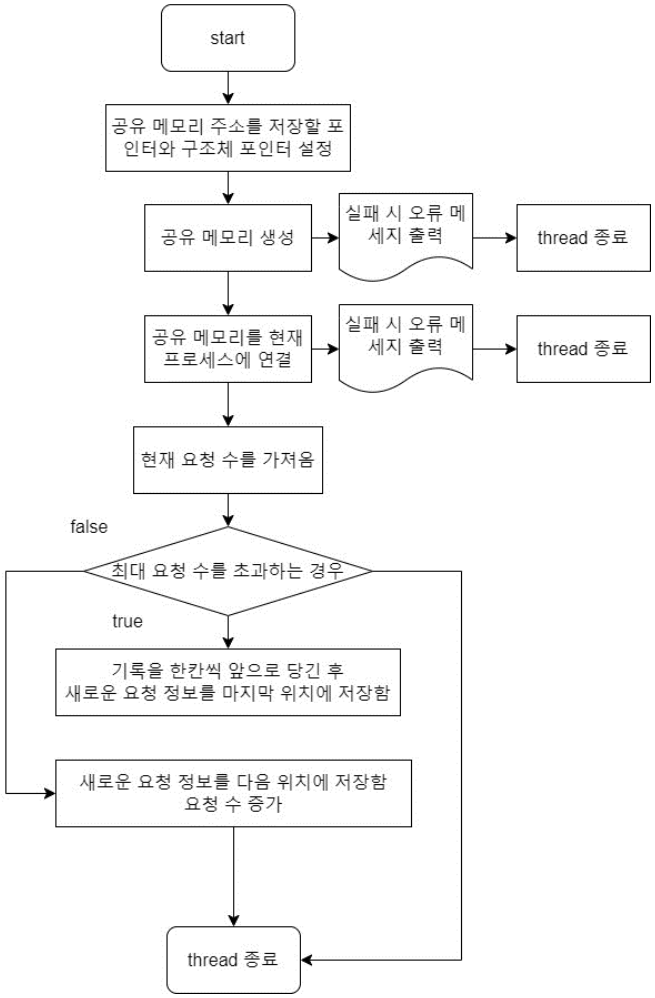
클라이언트로부터 연결 요청을 받으며, accept 를 실패하면 에러 메세지를 출력해준다. accessible.usr 파일을 읽기 모드로 열어 파일을 열지 못했을 경우 오류 메시지를 출력한다. 파일을 한 줄씩 읽어와 IP 주소 목록과 비교한다. 읽어온 줄의 개행 문자를 제거하고 fnmatch 함수를 사용하여 IP 주소 패턴 매칭을 수행한다. 이때 일치하는 IP 주소를 찾으면 플래그 변수 1 로 설정한다. 첫 번째 공백 전까지 문자열을 잘라서 첫 번째 토큰을 가져오는데 이때 토큰이 없는 경우 무시한다. url 배열을 초기화한 후 메서드가 GET 인 경우 두 번째 공백 전까지 문자열을 잘라서 두 번째 토큰을 가져온다. flag 가 0 이면 접근이 거부된 경우 응답 메시지와 응답 헤더를 작성하고 응답 헤더와 메시지를 전송한다. 클라이언트 소켓을 종료하고 다음 클라이언트를 요청하고 자식 프로세스를 종료한다. 현재 작업 디렉토리를 cwd 에 저장하고 요청한 파일의 경로를 만들어, 경로가 없는 경우 404 Not Found 응답 메시지를 작성한다. 응답 헤더와 메시지를 전송한 후 클라이언트 소켓을 종료한다. 다시 accept 대기로 돌아간 후 html 출력을 위해 필요한 부분을 구현한다. url 이 root path 인 경우 lflag 를 1 로 설정해주고 "Welcome to System Programming Http"을 출력한다. 하위디렉토리일 경우 aflag 와 lflag 를 1 로 설정해주고 "System Programming Http"를 출력한다. cwd 의 파일 목록을 출력한 후 클라이언트와 연결과 소켓과의 연결을 종료한다. flag 가 0 이면 다음 클라이언트 요청 처리를 위해 반복문의 처음으로 이동한다. nrequest 가 10 일 경우 전체 요청 수를 증가하고 가장 오래된 요청 기록을 제거한 후 새로운 요청 정보를 추가한다. else 일 경우 전체 요청 수를 증가하고 새로운 요청 정보를 추가하고 현재 요청 수를 증가한다. client_fd 를 닫고 프로그램을 종료한다.



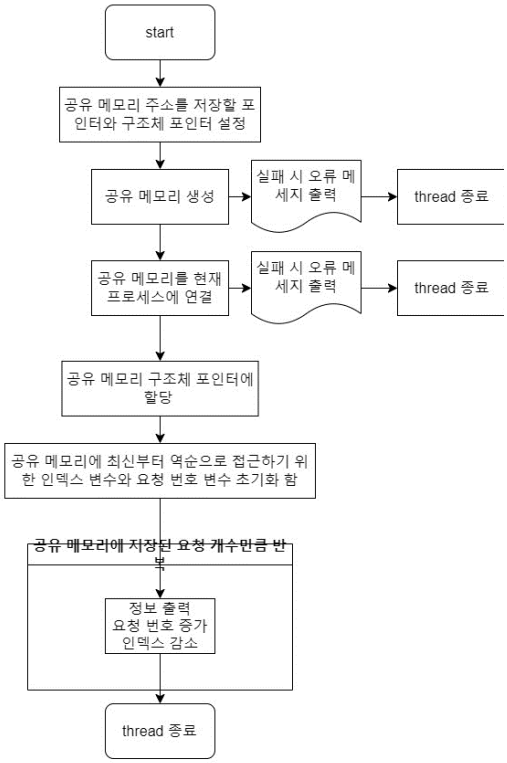
다음으로 initailizeHttpdconf 함수이다. Httpdconf 파일을 초기화 하는 함수로 먼저 파일을 읽기 모드로 열며, 파일 열기에 실패하면 오류 메시지를 출력한다. 파일에서 한 줄씩 읽는데 파일이 비어 있을 경우 오류 메세지를 출력한다. httpConf 구조체를 0 으로 초기화 한 후 파일의 끝까지 while 문을 돌며, 줄에서 키와 값을 분리하고 공백과 개행 문자를 제거한다. 이어서 키에 따라서 해당하는 멤버 변수에 값을 설정한 후 파일의 끝을 도달하면 파일을 닫고 프로그램을 종료한다.



shared_memory_initialize 함수로, 공유 메모리를 초기화 하는 thread 함수이다. 우선 공유 메모리 주소를 저장할 포인터와 구조체 포인터를 설정한다. 공유 메모리를 생성하며 생성 실패 시 오류 메시지를 출력한 후 thread 를 종료한다. 공유 메모리를 현재 프로세스에 연결하며 실패 시 오류 메시지를 출력한 후 thread 를 종료한다. 공유 메모리를 구조체 포인터에 할당한 후 공유 메모리를 0 으로 초기화 한다. 최대 자식 개수만큼 for 문을 활용하여 자식 프로세스 ID 를 초기화 한 후 thread 를 종료한다.



shared_memory 함수로 공유 메모리에 데이터를 저장하는 thread 함수이다. 우선 공유 메모리 주소를 저장할 포인터와 구조체 포인터를 설정한다. 공유 메모리를 생성하며 생성 실패 시 오류 메시지를 출력한 후 thread 를 종료한다. 공유 메모리를 현재 프로세스에 연결하며 실패 시 오류 메시지를 출력한 후 thread 를 종료한다. 공유 메모리를 구조체 포인터에 할당한 후 현재 요청 수를 가져온다. 만약 최대 요청 기록 수를 초과하는 경우 기록을 한 칸 씩 앞으로 당기며 새로운 요청 정보를 마지막 위치에 저장한다. 그렇지 안은 경우 새로운 요청 정보를 다음 위치에 저장하고 요청 수를 증가한다. thread 를 종료한다.



shared_memory_pirnt 함수로 공유 메모리에 저장된 데이터를 출력하는 thread 함수이다. 우선 공유 메모리 주소를 저장할 포인터와 구조체 포인터를 설정한다. 공유 메모리를 생성하며 생성 실패 시 오류 메시지를 출력한 후 thread 를 종료한다. 공유 메모리를 현재 프로세스에 연결하며 실패 시 오류 메시지를 출력한 후 thread 를 종료한다. 공유 메모리를 구조체 포인터에 할당한 후 공유 메모리에서 최신부터 역순으로 접근하기 위한 인덱스 변수 n 과 요청 번호 변수 i 를 초기화 한다. while 문을 활용해 공유 메모리에 저장된 요청 개수만큼 반복하며 정보를 출력하고 요청번호는 증가, 인덱스는 감소시킨다. thread 를 종료한다.

3. Pseudo Code

```
initializeHttpdConf
{
	파일 읽기 모드로 열기
	if(파일 열기 실패)
	{
		오류 메세지 출력 & 프로그램 종료
	}
	if(파일이 비어있는 경우)
	{
		오류 메세지 출력 & 프로그램 종료
	}
	httpdConf 구조체 0으로 초기화
	while(파일의 끝까지)
	{
		줄에서 키와 값을 분리함
		if(key와 value가 NULL이 아니면)
		{
			공백 및 개행 문자 제거
			키에 따라서 해당 멤버 변수에 값 설정
		}
	}
	파일 닫기
}
```

initializeHttpdConf 함수의 pseudo code 이다.

```
선택정렬() //sort
{
	for(파일 수만큼 파일 이름 배열 순회)
	{
		if(첫번째 파일 첫 글자가 숨김 파일이면)
		{
			filename[i]의 길이만큼 반복하고 첫 글자 제외하고 저장
		}
		else(숨김파일이 아니면)
		{
			이름 저장
		}
		if(두번째 파일 첫 글자가 숨길 파일이면)
		{
			filename[i]의 길이만큼 반복하고 첫 글자 제외하고 저장
		}
		else(숨김파일이 아니면)
		{
			이름 저장
		}
		hidden_file 1 대문자로 변환
		hidden_file 2 대문자로 변환
		대문자로 변환한 문자열 비교
}
```

sort 함수의 pseudo code 이다.

```
파일 정보 출력() //print_file_info
{
	파일 정보 읽어 구조체 변수 filestat에 저장
	파일 권한 정보 저장할 변수 선언
	if(파일 경로가 있으면)
	{
		파일 경로와 파일 이름을 합치고 stat함수로 파일 정보 읽음
	}
	else
	{
		파일 이름으로 정보 읽음
	}
	if(파일명이 .인 경우) 현재 디렉토리를 가리키는 하이퍼링크 생성
	else if(파일명이 ..인 경우)
	{
		url에서 마지막 /를 찾음
		url에 /가 없는 경우 -> 루트 디렉토리를 가리키는 하이퍼링크 생성
		url에 /가 있는 경우 -> 마지막 / 앞까지의 문자열을 추출 후 하이퍼링크 생성
	}
	else
	{
		url의 길이 저장
		url이 /로 끝나는 경우 -> /를 삭제
		하이퍼링크 생성
	}
	// 파일 유형, 허가권, 링크 수 , 소유자, 그룹, 출력
	if(파일 유형이 디렉토리면) permission 배열의 0번째는 d
	else if(파일 유형이 링크면) permission 배열의 0번째는 l
	else permission 배열의 0번째는 -
	파일 허가권 정보 설정

	if(디렉토리면) 파란색으로 출력
	else if(link 파일이면) 초록색으로 출력
	else 빨간색으로 출력

	파일 크기 출력
	수정시간, 파일 경로 출력
}
```

print_file_info 함수의 pseudo code 이다.

```
파일 목록 출력() //list_files
{
	디렉토리 열기
	절대경로로 변환
	if(파일이면)
	{
		파일을 이진 모드로 열고 파일 포인터 저장
		if(파일 열지 못하면) 에러메세지 출력
		파일 끝으로 이동해 파일 크기 추출한 후 파일 시작으로 이동

		파일 확장자에 따라 content-type 설정
		if (jpg, png, jpeg)
		{
			이미지 콘텐츠 유형으로 응답 헤더 설정
		}
		else if(.html)
		{
			HTML 컨텐츠 유형으로 응답 헤더 설정
		}
		else
		{
			일반 텍스트 유형으로 응답 헤더 설정
		}
		클라이언트에게 response header를 전송
		while(전송이 완료될 때까지)
		{
			파일을 BUFSIZE 만큼 읽음
			전송된 파일 크기를 업데이트
			클라이언트에게 파일을 전송
		}
		파일 닫기
	}
	else //디렉토리면
	{
		if(a 옵션이 꺼져있는 경우)
		{
			while(디렉토리 내 모든 파일 순회)
			{
				html_ls.html 파일은 건너뜀
				숨김파일이면 넘어감
				아니면 파일 개수 증가
			}
		}
		else
		{
			while(디렉토리 내 모든 파일 순회)
			{
				html_ls.html 파일은 건너뜀
				파일 이름 배열에 저장
			}
		}
		파일 이름 정렬
		if(l 옵션이 켜져있는 경우)
		{
			directory path, total 출력
			HTML 파일에 테이블 태그 출력
			HTML 파일에 테이블 헤더 삽입
			파일 정보 출력
		}
		HTTP response header 생성
		클라이언트에게 response header, message 보내기
	}
	디렉토리 닫기
}
```

list_files 함수의 pseudo code 이다.

```
handle_signal
{
	if(signal == SIGCHLD)
	{
		자식 프로세서가 종료되면, 모든 종료된 자식 프로세서 처리
	}
	if(signal == SIGALRM)
	{
		연결 기록 출력
		공유 메모리 출력 thread 생성
		thread 종료시까지 대기
		alarm(10)
	}
	if(signal == SIGUNT)
	{
		alarm 0으로 초기화

		startProcess 개수만큼 종료된 자식 프로세스 상태 출력
		부모 프로세스 종료 출력
		공유 메모리 제거
		mutex 객체 파괴 % 동적 할당 해제
		프로그램 종료
	}
	if(signal == SIGTERM)
	{
		프로그램 종료
	}
}
```

handle_signal 함수의 pseudo code 이다.

```
child_make
{
	자식 프로세스 생성
	if(부모 프로세스면)
	{
		해당 pid 반환
	}
	if(자식 프로세스면)
	{
		child_main() 함수 호출해 클라이언트 연결 처리
	}
}
```

child_make 함수의 pseudo code 이다.

```
child_main
{
	SIGUSR1, SIGTERM 시그널 핸들러 설정
	SIGINT 시그널 무시

	while(1)
	{
		응답 헤더, 메세지 초기화
		클라이언트로부터 연결 요청 받음
		if(accept 실패 시) 에러 메세지 출력

		accessible.usr 파일을 읽기 모드로 열기
		if(파일을 열지 못하면) 오류메세지 출력

		while(파일에서 한 풀씩 읽어와 IP 주소 목록과 비교)
		{
			읽어온 풀의 개행 문자 제거
			fnmatch 함수 사용해서 IP 주소 패턴 매칭 수행
				일치하는 IP 주소 찾으면 플래그 변수 1로 설정
				break;
		}
		파일 닫기

		첫번째 공백 전까지 문자열 잘라서 첫 번째 토큰 가져옴
		if(토큰이 없는 경우) continue
		url 배열 초기화
		if(method가 GET인 경우)
		{
			두번째 공백 전까지 문자열 잘라서 두 번째 토큰 가져옴
		}
		현재 작업 디렉토리 cwd변수에 저장

		새로운 자식 프로세스 생성

		접근이 거부된 경우 응답 메세지와 헤더 작성
		응답 헤더와 메세지 전송
		continue

		if(파일이 없는 경우)
		{
			404 Not Found 응답 메세지 작성
			응답 헤더, 메세지 전송
			클라이언트 소켓 종료
			accpet대기로 돌아감
			continue
		}
		new client 출력
		if(url이 root path)
		{
			lflag = 1
			"Welcome to System Programming Http" 태그
		}
		else //하위 디렉토리
		{
			aflag =1, lfalg = 1
			"System Programming Http"
		}

		클라이언트의 IP주소, 포트 번호, PID, 현재 시간을 연결 기록에 저장
		list_files 함수 호출해 cwd의 파일 목록 출력
		Disconnected client 출력
		shared_memory 함수를 실행하는 thread를 생성
		thread가 종료될 때까지 대기
		클라이언트 연결 종료
	}
}
```

child_main 함수의 pseudo code 이다.

```
shared_memory_initialize
{
	공유 메모리 주소를 저장할 포인터와 구조체 포인터 설정
	공유 메모리 생성
	if 실패 시 오류 메세지 출력 & tread 종료
	공유 메모리를 현재 프로세스에 연결
	if 실패 시 오류 메세지 출력 & tread 종료

	공유 메모리를 구조체 포인터에 할당 후 초기화
	for(최대 자식 프로세스 개수만큼)
	{
		자식 프로세스 ID 초기화
	}
	thread 종료
}
```

shared_memory_initialize 함수의 pseudo code 이다.

```
shared_memory
{
	공유 메모리 주소를 저장할 포인터와 구조체 포인터 설정
	공유 메모리 생성
	if 실패 시 오류 메세지 출력 & tread 종료
	공유 메모리를 현재 프로세스에 연결
	if 실패 시 오류 메세지 출력 & tread 종료

	공유 메모리를 구조체 포인터에 할당
	현재 요청 수 가져옴
	if(최대 요청 기록 수를 초과하는 경우)
	{
		기록을 한 칸씩 앞으로 당김 -> 새로운 요청 정보를 마지막 위치에 저장
	}
	else
	{
		새로운 요청 정보를 다음 위치에 저장
		요청 수 증가
	}
	thread 종료
}
```

shared_memory 함수의 pseudo code 이다.

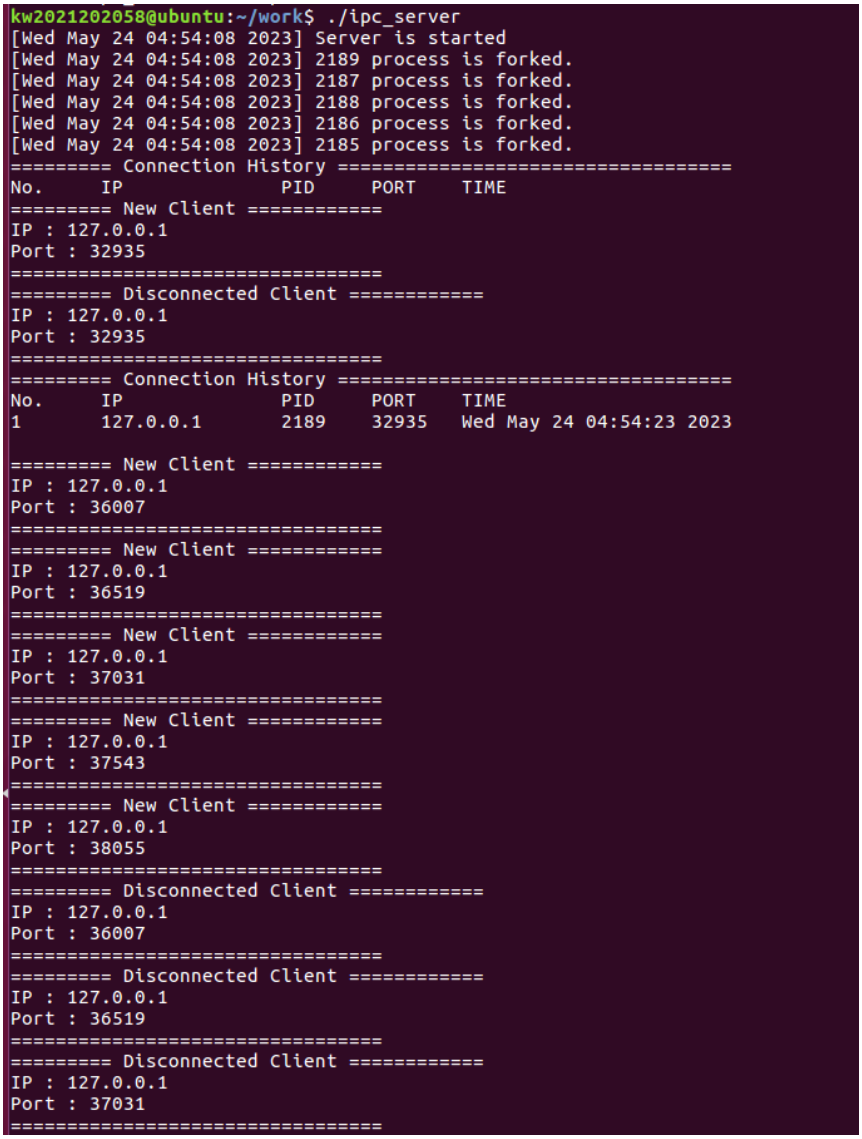
```
shared_memory_print
{
	공유 메모리 주소를 저장할 포인터와 구조체 포인터 설정
	공유 메모리 생성
	if 실패 시 오류 메세지 출력 & tread 종료
	공유 메모리를 현재 프로세스에 연결
	if 실패 시 오류 메세지 출력 & tread 종료

	공유 메모리를 구조체 포인터에 할당
	공유 메모리에서 최신부터 역순으로 접근하기 위한 인덱스 n 선언
	요청 번호 변수 i 선언
	while(공유 메모리에 저장된 요청 개수만큼 반복)
	{
		정보 출력
		요청 정보 증가
		인덱스 감소
	}
	thread 종료
}
```

shared_memory_print 함수의 pseudo code 이다.

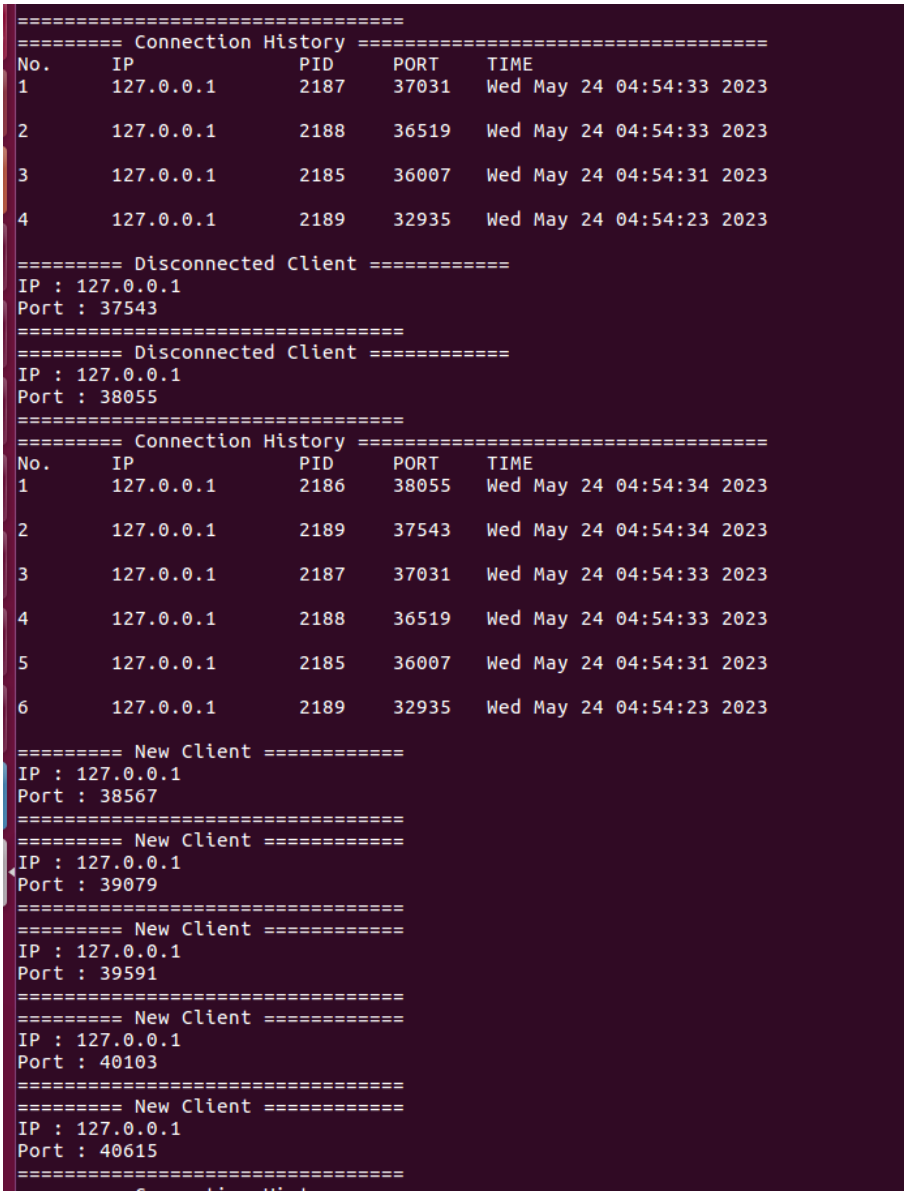
```
main
{
	10초마다 SIGALRM 시그널 발생
	SIGCHLD, SIGALRM, SIGINT 시그널 핸들러 함수로 등록

	소켓 생성 & 소켓 파일 디스크립터 반환 함수 호출
	실패 -> 에러 메세지 출력

	소켓 옵션 설정

	서버 주소 초기화 하고 설정함

	소켓 bind
	실패 -> 에러 메세지 출력

	클라이언트 연결 요청 대기

	httpd.conf 파일 읽어와 설정 초기화

	공유 메모리 초기화 thread 생성 후 종료 대기

	자식 프로세스 수 = 5

	for(최대 자식 프로세스 수만큼 반복)
	{
		child_make 함수 호출 -> 자식 프로세스 생성 -> 자식 프로세스 ID를 저장
	}
	for(무한루프)
	{
		신호 받을 때 까지 대기
	}
```

main 함수의 pseudo code 이다.

4. 결과화면



server program 이 시작되고 Server is started 구문이 잘 출력되며, Child process 가 startporcess 값 만큼 생성될 때 해당 pid process is forked 의 출력화면으로 문제에서 요구하는 대로 잘 출력이 되는 것을 확인할 수 있다. 이어서 Connection History 가 10 초마다 잘 출력되는 것 역시 확인할 수 있다.



client 를 접속하고 종료했을 때 New client 와 Disconnected client 가 잘 출력이 된다. 또한 10 초마다 공유메모리에 저장된 history 기록을 최신 순서대로(NO.) 출력하는 것을 확인할 수 있다.

```
========= Connection History ==================================
No.     IP              PID     PORT    TIME
1       127.0.0.1       2186    38055   Wed May 24 04:54:34 2023

2       127.0.0.1       2189    37543   Wed May 24 04:54:34 2023

3       127.0.0.1       2187    37031   Wed May 24 04:54:33 2023

4       127.0.0.1       2188    36519   Wed May 24 04:54:33 2023

5       127.0.0.1       2185    36007   Wed May 24 04:54:31 2023

6       127.0.0.1       2189    32935   Wed May 24 04:54:23 2023

========= Disconnected Client ============
IP : 127.0.0.1
Port : 38567
================================
========= Disconnected Client ============
IP : 127.0.0.1
Port : 39079
================================
========= Disconnected Client ============
IP : 127.0.0.1
Port : 39591
================================
========= Disconnected Client ============
IP : 127.0.0.1
Port : 40103
================================
========= Disconnected Client ============
IP : 127.0.0.1
Port : 40615
================================
========= Connection History ==================================
No.     IP              PID     PORT    TIME
1       127.0.0.1       2186    40615   Wed May 24 04:54:55 2023

2       127.0.0.1       2189    40103   Wed May 24 04:54:54 2023

3       127.0.0.1       2187    39591   Wed May 24 04:54:54 2023

4       127.0.0.1       2188    39079   Wed May 24 04:54:54 2023

5       127.0.0.1       2185    38567   Wed May 24 04:54:54 2023

6       127.0.0.1       2186    38055   Wed May 24 04:54:34 2023

7       127.0.0.1       2189    37543   Wed May 24 04:54:34 2023

8       127.0.0.1       2187    37031   Wed May 24 04:54:33 2023

9       127.0.0.1       2188    36519   Wed May 24 04:54:33 2023

10      127.0.0.1       2185    36007   Wed May 24 04:54:31 2023
```

```
========= New Client ============
IP : 127.0.0.1
Port : 41127
================================
========= Connection History ====================================
No.     IP              PID     PORT    TIME
1       127.0.0.1       2186    40615   Wed May 24 04:54:55 2023

2       127.0.0.1       2189    40103   Wed May 24 04:54:54 2023

3       127.0.0.1       2187    39591   Wed May 24 04:54:54 2023

4       127.0.0.1       2188    39079   Wed May 24 04:54:54 2023

5       127.0.0.1       2185    38567   Wed May 24 04:54:54 2023

6       127.0.0.1       2186    38055   Wed May 24 04:54:34 2023

7       127.0.0.1       2189    37543   Wed May 24 04:54:34 2023

8       127.0.0.1       2187    37031   Wed May 24 04:54:33 2023

9       127.0.0.1       2188    36519   Wed May 24 04:54:33 2023

10      127.0.0.1       2185    36007   Wed May 24 04:54:31 2023

========= Disconnected Client ============
IP : 127.0.0.1
Port : 41127
================================
========= Connection History ====================================
No.     IP              PID     PORT    TIME
1       127.0.0.1       2185    41127   Wed May 24 04:55:16 2023

2       127.0.0.1       2186    40615   Wed May 24 04:54:55 2023

3       127.0.0.1       2189    40103   Wed May 24 04:54:54 2023

4       127.0.0.1       2187    39591   Wed May 24 04:54:54 2023

5       127.0.0.1       2188    39079   Wed May 24 04:54:54 2023

6       127.0.0.1       2185    38567   Wed May 24 04:54:54 2023

7       127.0.0.1       2186    38055   Wed May 24 04:54:34 2023

8       127.0.0.1       2189    37543   Wed May 24 04:54:34 2023

9       127.0.0.1       2187    37031   Wed May 24 04:54:33 2023

10      127.0.0.1       2188    36519   Wed May 24 04:54:33 2023
```

MaxHistory 의 수가 10 이므로 공유메모리에 저장된 history 의 수가 10 과 같으면 가장 오래된 history 를 제거하는 것을 확인하는 결과 화면이다. 첫 번째 사진에서 가장 마지막 port 번호가 36007, 그 다음이 36519 인 것을 확인할 수 있고 이어서 두 번째 사진에서 41127 의 port 번호가 Disconnected 된 후 출력된 connection History 에서 36519 가 가장 마지막 port 번호가 되었으므로 조건에 만족하는 것을 확인할 수 있다.

```
^C
[Wed May 24 04:55:32 2023] 2189 process is terminated.
[Wed May 24 04:55:32 2023] 2188 process is terminated.
[Wed May 24 04:55:32 2023] 2187 process is terminated.
[Wed May 24 04:55:32 2023] 2186 process is terminated.
[Wed May 24 04:55:32 2023] 2185 process is terminated.
[Wed May 24 04:55:32 2023] Server is terminated.
```

ctrl + c 로 종료 시, 즉 SIGINT signal 이 발생했을 때 child process 를 먼저 종료한 후 server 를 종료하며 각각에 알맞은 출력문이 출력 되는 것을 확인할 수 있다.

5. 고찰

우선 이번 과제에 대한 이해가 부족했던 것 같다. 공유 메모리 부분을 구현하면서 처음에는 공유 메모리를 초기화 하고 데이터를 저장하고 저장된 데이터를 출력하는 thread 부분을 한 함수에 구현하고 pthread_create 의 인자로 그 함수를 넘겨주었다. 하지만 이는 segmentation fault 로 이어졌다. 우선 공유 메모리를 초기화하는 부분은 한 번만 이루어져야 했기에 함수로 따로 뺀 후 main 에서 초기화 함수를 호출하도록 하였다. 이어서 데이터를 저장하는 함수에서는 변수를 계속해서 수정해서 새로운 데이터를 주어야 하기 때문에 lock 을 걸어주어야 했기에 이 역시 함수로 따로 빼었다. 앞서 말한 이유는 추측으로 든 생각이지만 이 부분은 동기화가 필요하기 때문에 따로 구현하는 것이 맞다고 생각이 들었다. 이번 과제에서는 Idle process 를 관리하는 부분까지 구현하지 못하였다. thread 에 대한 개념이 부족했던 것도 있지만, 전역 변수로 선언하였던 변수를 지역 변수로 추가로 선언해서 값이 제대로 들어가지 않거나 초기화를 잘못해주거나, 선언된 위치, thread 를 create 하고 join 하는 위치가 적절하지 않아 segmentation fault 가 뜨거나 올바른 출력 값이 나오지 않거나 중간에 Disconnected 가 뜨기도 전에 terminated 되거나 하는 문제가 발생하였고 이러한 문제를 해결하는데 시간을 많이 썼던 것 같다. 이번 과제에서는 함수의 흐름을 이해하려고 노력했으며 추가적으로 구현하기 보다는 기존에 구현하였던 부분을 적절하게 옮겨가며 사용하여 구현하였던 것 같다.

6. Reference

시스템프로그래밍실습 강의자료

시스템프로그래밍 강의자료