Term Project #2

-archive of homeworks-

학 과	컴퓨터공학과
학 번	201211704
이 름	김기홍
제 출 일	2015.11.30

목 차

1. 개요 ····· 3

- 머리말 ····· 3

- 배경 ····· 3

2. 본문 ····· 3

- 구현 과정 ····· 3

· HW5. ls -laR과 유사한 동작을 하는 프로그램 ll ····· 3

· HW6. 간단한 쉘프로그램 mysh ····· 5

· HW7. IPC 계산기 프로그램 ····· 6

① HW7-1. mmap, pipe 사용 ····· 6

② HW7-2. Message Queue, Shared Memory 사용 ····· 9

③ HW7-3. 네트워크 통신 활용 ····· 10

- 결과 분석 ····· 13

· ll 프로그램 ····· 13

· mysh 쉘 프로그램 ····· 14

· IPC 계산 프로그램 ····· 16

3. 결론 ····· 17

- 느낀 점 및 배운 점 ····· 17

그림 차례

[그림 1] ll 프로그램의 실행 절차 ····4

[그림 2] ll 프로그램의 main 함수 ····4

[그림 3] look 함수 내부에서 파일의 정보를 리스트에 삽입하는 모습 ····4

[그림 4] 리스트를 탐색하며 조건 만족 시 재귀호출 하는 모습 ····5

[그림 5] mysh 프로그램의 실행 절차 ····5

[그림 6] 입력 받은 문자열(command)를 토큰화 하는 모습 ····5

[그림 7] 자식 프로세스에서 명령어를 실행하는 모습 ····6

[그림 8] 메모리 매핑 방식 ····7

[그림 9] 메모리 매핑 방식 IPC 계산 프로그램 실행 절차 ····7

[그림 10] 이름 없는 파이프 방식 ····8

[그림 11] 이름 없는 파이프 방식 IPC 계산 프로그램 실행 절차 ····8

[그림 12] 이름 있는 파이프 방식 ····8

[그림 13] Message Queue 방식 ····9

[그림 14] Message Queue 방식 IPC 계산 프로그램 실행 절차 ····9

[그림 15] Shared Memory 방식 ····10

[그림 16] 네트워크 통신 방식 ····10

[그림 17] 네트워크 방식 IPC 계산 프로그램 ····11

[그림 18] ls -laR [dir] 와 ll [dir] 실행화면 비교 ····12

[그림 19] mysh 프로그램 기본 명령어 입력 화면 ····13

[그림 20] mysh 프로그램 cd 명령어 입력 화면 ····14

[그림 21] mysh 프로그램 exit 명령어 입력 화면 ····14

[그림 22] mysh 프로그램 log.txt 파일 ····15

[그림 23] IPC 계산 프로그램 실행 화면 ····15

1. 개요

- 머리말

본 보고서는 '시스템 프로그래밍' 교과목의 기말 Term Project로서 부제목에서도 알 수 있다시피 archive of homeworks 즉, 숙제들의 모음집이다. 각 과제들에 대한 구현 과정, 결과 비교, 필자가 이해한 것을 바탕으로 여러 방법간의 상호 비교 등을 기술할 것이며 Term Project로서의 과제 제출의 목적도 있지만, 필자가 공부한 내용을 잘 정리하여 보관하려는 목적 또한 가지고 있다.

본 보고서의 큰 구조는 각 과제들의 구현 과정을 설명하고 결과를 분석하고 그 후에 필자의 생각을 정리하고자 한다.

- 배경

본 보고서에 기록되어지는 모든 과제들은 VMware Player(가상 머신)의 Linux Mint mate_17.2-64bit 환경에서 구현 및 실행 되었다.

또한 각 과제들은 Smart-LMS(lms.pknu.ac.kr)에서 요구되어진 내용들에 충실하게 만족시켰음을 알린다.

2. 본문

- 설명 및 구현 과정

· HW5. ls -laR과 유사한 동작을 하는 프로그램 ll

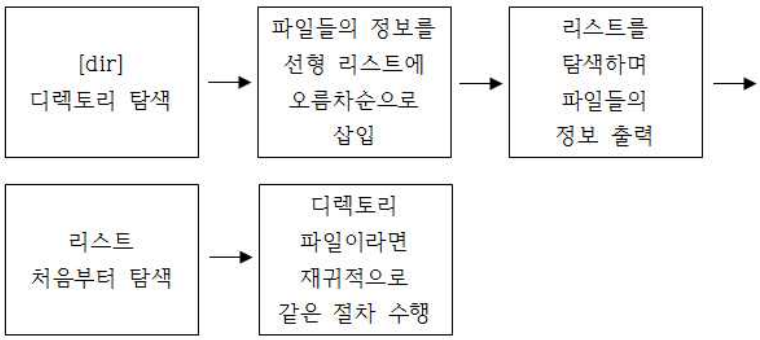
먼저 ls -laR이 어떤 동작을 하는지 살펴보자. ls 명령어에는 여러가지 옵션이 있는데 그 중에서 -l 옵션, -a 옵션, -R 옵션의 기능은 아래 표와 같다.

옵션	기능
-l	접근권한, 소유자, 그룹, 크기, 날짜 등을 출력
-a	디렉토리 내의 모든 파일 출력
-R	하위 디렉토리의 내용까지 재귀적으로 모두 출력

[표 1] ls 명령어의 옵션 설명

결과적으로 ls -laR을 입력하면 다음과 같은 결과가 터미널 상에 나타난다. '디렉토리 내의 모든 파일의 접근권한, 소유자, 그룹, 크기, 날짜 등이 출력되며 모든 하위 디렉토리까지 재귀적으로 출력'

필자의 목표는(요구되어진 내용은) 이와 유사한 동작을 하는 프로그램 'll'의 구현이다. 사용법은 ls [dir] 이며, 입력받은 디렉토리에 대해 위와 같은 출력을 해내는 프로그램이다. 프로그램의 실행 절차를 아래의 그림을 통해 알아보도록 하자.



[그림 1] ll 프로그램의 실행 절차

ll [dir] 입력에서 [dir]을 얻기 위해 argc, argv를 이용했다.

```c
#include "main.h"
#include "list.h"

int main(int argc, char **argv)
{
    look(argv[1]);

    return 0;
}
```

[그림 2] ll 프로그램의 main 함수

main 함수 본문에 보이는 look(char *s) 함수는 look.c에서 정의되어있으며 그 기능은 매개변수로 받은 문자열의 이름을 가진 디렉토리 내부의 파일들의 정보를 struct dirent와 struct stat을 이용하여 구한 뒤 오름차순으로 삽입을 행하는 선형 리스트에 삽입하고, 리스트의 내용을 출력하며, 다시 탐색하며 디렉토리 파일일 경우 다시 look 함수를 호출한다. ll 프로그램에서 가장 핵심적인 함수라고 할 수 있다.

```c
while(dent = readdir(dp))
{
    sprintf(path, "%s/%s", s, dent->d_name);
    stat(path, &buf);
    Insert(&l, dent, buf);
}
```

[그림 3] look 함수 내부에서 파일의 정보를 리스트에 삽입하는 모습

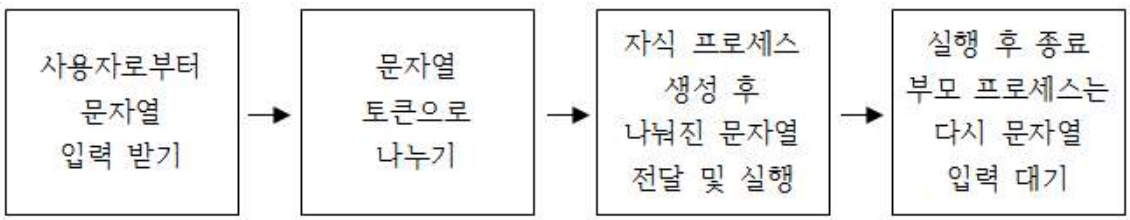
```
while(l.cur != NULL)
{
    if(strcmp(l.cur->d->d_name, ".") && strcmp(l.cur->d->d_name, ".."))
    {
        sprintf(path, "%s/%s", s, l.cur->d->d_name);
        stat(path, &sbuf);
        if(S_ISDIR(sbuf.st_mode))
            look(path);
    }

    l.cur = l.cur->link;
}
```

[그림 4] 리스트를 탐색하며 조건 만족 시 재귀호출 하는 모습

바로 위의 코드에서 파일 이름이 "." 일 경우와 ".." 일 경우를 제외하는 것에 주목하자! 그 이유는 .(현재 디렉토리), ..(상위 디렉토리)를 제외하지 않으면 재귀적으로 호출하는 과정이 무한적으로 반복되기 때문인데 이를 방지하기 위해서는 반드시 조건문을 사용하여 제외시켜주어야 한다. 추가로 S_ISDIR(struct stat)을 통해 파일이 디렉토리 파일인지를 구별할 수 있었다.

· HW6. 간단한 쉘프로그램 mysh

mysh 프로그램은 말 그대로 우리가 주로 사용하는 bash shell과 유사한 동작을 하는 프로그램이다. 예를 들면 명령어를 입력하면 그에 상응하는 실행 결과를 보인다. 그리고 입력된 명령어는 log.txt 파일에 시간과 함께 기록된다. 그러면 바로 아래 그림을 통해 mysh 프로그램의 전체적인 실행 절차를 알아보자.



[그림 5] mysh 프로그램의 실행 절차

문자열은 strok(char*, char*)을 통해 토큰화 하였으며, 각각의 토큰화 된 문자열은 배열에 저장하였다. 그 이유는 입력 받은 문자열(명령어)을 자식 프로세스에서 실행시키기 위해 execvp(char *, char **) 함수를 사용하는데 매개변수로 사용하기 위함이다.

```
char token[] = " \n";
argv[m] = strtok(command, token);
while(argv[m] != NULL)
{
    argc++;
    argv[++m] = strtok(NULL, token);
}
```

[그림 6] 입력 받은 문자열(command)를 토큰화 하는 모습

```
else if(execvp(argv[0], argv) == -1)
{
    printf("Can't Find The Command\n");
    exit(1);
}
```

[그림 7] 자식 프로세스에서 명령어를 실행하는 모습

로그는 자식 프로세스에서 fopen(file_name, mode) 함수를 통해서 로그 파일을 연 후에 fprintf(fd, form, param) 함수로 틀에 맞춰 기록 후 fclose(fd) 함수로 닫는다. 물론 파일을 열 때의 모드는 "a"(append)로 한다.

· HW7. IPC 계산기 프로그램

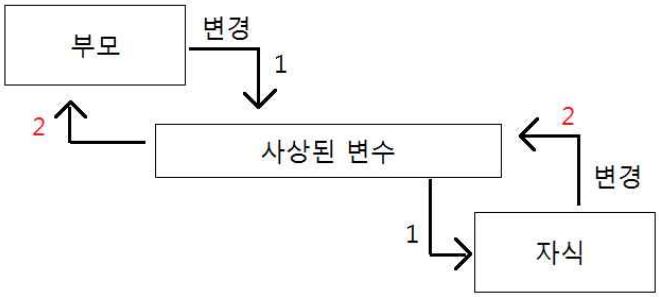
mmap를 사용한 경우를 제외하고는 모두 계산 프로그램과 식을 입력 받고 결과 값을 출력하는 프로그램을 따로 두어 구현하였다. fork() 함수를 최대한 사용하지 않고 구현하려고 한 의도는 상호 프로세스간의 데이터 전송을 부모와 자식 프로세스 사이로 특정 짓고 싶지 않았기 때문이다. 필자는 fork() 함수를 이용하는 편이 훨씬 쉬웠을 거라 생각하지만 조금 더 어렵게 구현함으로써 더 깊게 배울 수 있었다고 생각한다.

① HW7-1. mmap, pipe 사용

mmap(메모리 매핑)은 파일을 프로세스의 메모리에 사상시키는 것을 의미한다. 즉, 프로세스에 전달할 데이터를 직접 프로세스의 가상 주소 공간으로 사상하는 방식이다.

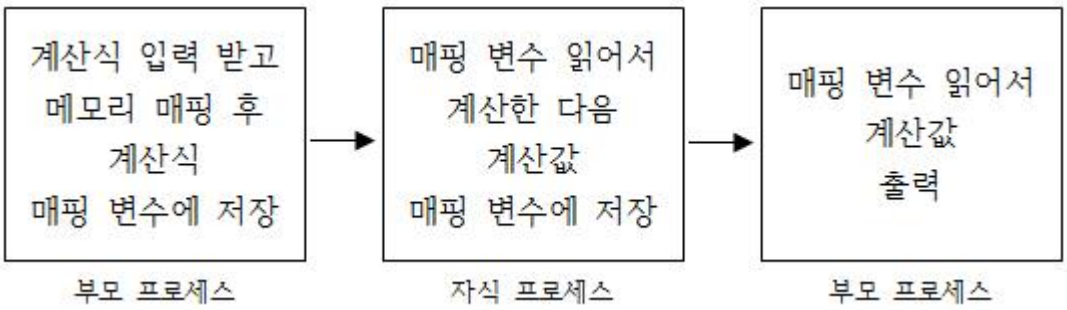
먼저 mmap()를 사용하기 위해서는 파일 기술자와 해당 파일의 정보(크기)가 필요하다. 그렇게 caddr_t형 변수에 메모리를 사상시키고 나서, fork()를 통해 자식 프로세스를 만들면 앞서 메모리를 사상시킨 caddr_t형 변수를 사용해서 부모자식 프로세스 간에 데이터를 주고받을 수 있다.

메모리 매핑을 할 파일을 미리 만들어 둘 것이라면 stat()를 이용해 파일의 크기를 알아낸 다음 open을 하면 아무 문제가 없지만, 만약 파일이 없는 상태에서 open에 O_CREAT 플래그로 파일을 생성한 뒤에는 stat()를 이용할 수 없기 때문에 문제가 될 수도 있다. 이 때는 현재 열려 있는 파일의 정보를 가져오는 fstat() 함수를 사용하면 쉽게 해결할 수 있다.



[그림 8] 메모리 매핑 방식

IPC 계산기 프로그램은 먼저 부모 프로세스에서 계산식을 입력받은 후, 메모리 매핑된 변수에 복사를 한다. 그 다음 자식 프로세스에서 그 식을 이용해 계산 값을 구한 다음 그 값을 다시 매핑된 변수에 복사를 하고, 최종적으로 부모 프로세스에서는 그 변수를 읽어서 출력을 하면 구현이 완료된다.



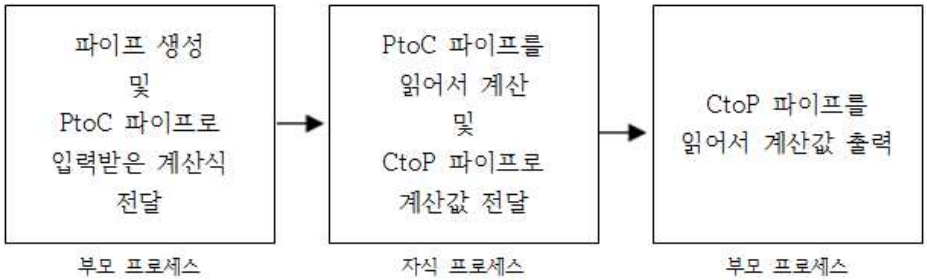
[그림 9] 메모리 매핑 방식 IPC 계산 프로그램 실행 절차

다음으로 pipe(파이프)방식은 두 가지로 구현했는데 하나는 이름 없는 파이프, 다른 하나는 이름 있는 파이프를 사용해서 구현하였다.

먼저, 이름 없는 파이프는 fork()를 통해서 부모자식 프로세스 간 통신을 구현하였고 이름 있는 파이프는 서로 다른 프로세스(Server 와 Client) 간 통신을 구현하였다.

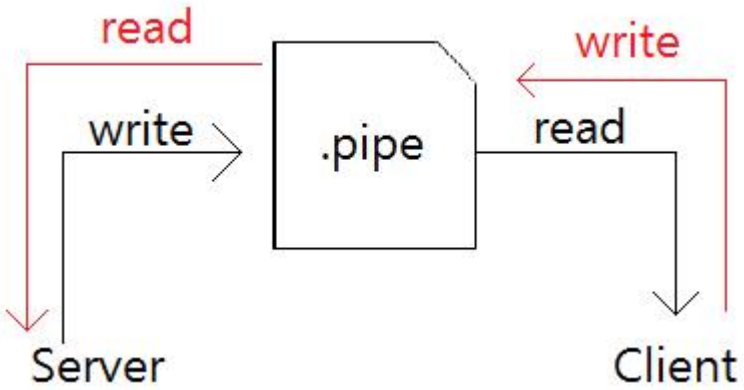
이름 없는 파이프를 통한 IPC 계산 프로그램은 정말 쉽게 구현 할 수 있었다. 준비물은 그저 파일기술자 2개가 끝이다. pipe()를 통해 파일기술자 2개는 각각 단 방향 파이프의 읽기와 쓰기를 담당하게 된다. fork() 호출 시 파이프 자체도 복사가 되기 때문에 부모자식 프로세스 간 통신이 가능하다. 하지만 여기서 한 가지 주의할 점은 파이프는 단방향이기 때문에 파이프가 2개가 있어야 부모 프로세스에서 자식 프로세스로, 반대로 자식 프로세스에서 부모 프로세스로 데이터 전송이 가능하기 때문에 파일기술자 2개를 한 쌍으로 하여 총 두 쌍이 필요하다.

[그림 10] 이름 없는 파이프 방식



[그림 11] 이름 없는 파이프 방식 IPC 계산 프로그램 실행 절차

다음으로 이름 있는 파이프 방식은 파일 입출력을 통해 여기서 쓰고 저기서 읽거나, 저기서 쓰고 여기서 읽는 형태의 방식인데, 해당 파일이 파이프 파일(특수 파일)이라는 점에서 다르다. 파이프 파일은 mkfifo()를 이용해서 생성하며, read(), write()를 사용해서 읽거나 쓰는 동작을 수행한다.



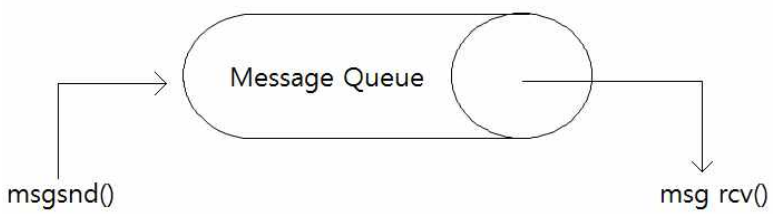
[그림 12] 이름 있는 파이프 방식

실행 절차는 앞서 보인 이름 없는 파이프 방식과 매우 유사하기 때문에 생략하도록 하겠다.

② HW7-2. Message Queue, Shared Memory 사용

Message Queue, Shared Memory 방식은 시스템 V 계열 유닉스에서 개발해서 제공하는 프로세스 간 통신 방법이다. 그런 만큼 굉장히 편리함을 느낄 수 있다.

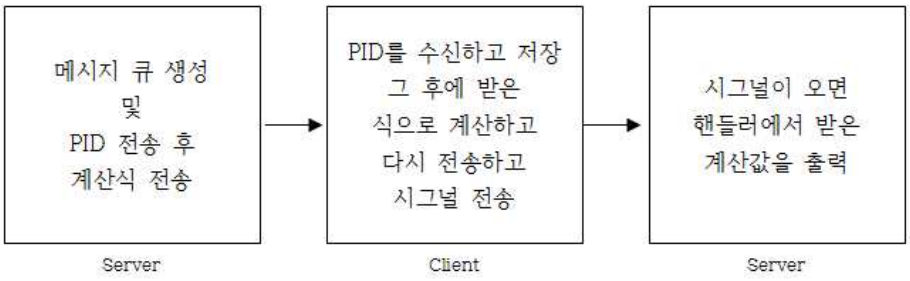
먼저 Message Queue 방식은 파이프와 유사하지만 파이프는 스트림 기반으로 동작하는 것에 반해 메시지 큐는 메시지 단위로 동작한다.



[그림 13] Message Queue 방식

위의 그림과 같이 운영체제에서 제공하는 큐를 이용해서 상호 프로세스 간 데이터를 주고받을 수 있다. 필자는 Message Queue 방식의 IPC 계산 프로그램을 구현할 때 하나의 메시지 큐로 구현하였다. 꽤 우여곡절이 많았지만 본 보고서에 기재할 만큼의 문제점은 아니었으므로 바로 구현방법에 대해 설명하도록 하겠다.

우선, 동기화를 맞추기 위해 시그널 전송 방식을 택했다. 따라서 다른 프로세스로의 시그널 전송을 하기 위해서는 PID가 필요하다. 다른 프로세스의 PID를 알기 위해서는 어떻게 해야 할까? 필자는 Message Queue 방식이니 PID도 같은 방법으로 전송하였다. Message Queue 방식 IPC 계산 프로그램의 실행 절차를 한번 살펴보자.

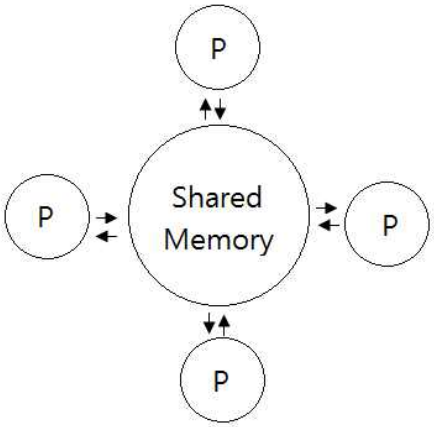


[그림 14] Message Queue 방식 IPC 계산 프로그램 실행 절차

다음으로 Shared Memory 방식을 살펴 볼 텐데 이 방식 또한 정말 편리함을 알 수 있다. 그리고 앞서 설명하였던 메모리 매핑 방식과도 아주 유사하기 때문에 이해도가 높을 것이라고 필자는 생각한다.

이 방법은 데이터를 주고받는다는 표현 보다는 같은 공간을 쓰기 때문에 한쪽에서 변경된 값이 다른 한쪽에서도 변경되어 사용할 수 있다는 표현이 더 맞는 것 같다. shmget()을 통해 생성된 공유 메모리는 같은 키 값인 조건에 한해서 다른 프로세스들이 공유할 수 있다. 그리고 필자는 여기서도 시그널 전송을 통해 동기화를 하였는데, PID를 다른 프로세스에게 알리기 위해 shmid_ds 구조체 필드

중에 shm_cpid(공유 메모리를 생성한 프로세스ID)가 있는데 이를 이용해서 PID를 알 수 있었다.

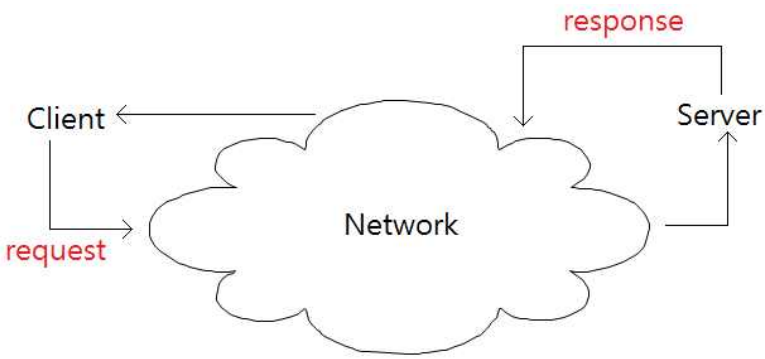


[그림 15] Shared Memory 방식

Shared Memory 방식 IPC 계산 프로그램 실행 절차 또한 Memory Queue 방식과 너무 유사하기 때문에 다른 설명은 생략하도록 하겠다. 다만 다른 점이라 하면 PID를 얻는 방법(Shared Memory 방식에서는 shmid_ds 구조체 활용)과 전송, 수신이 아닌 공유 메모리의 데이터를 변경시켜서 다른 프로세서도 변경된 데이터를 얻을 수 있다는 점이다.

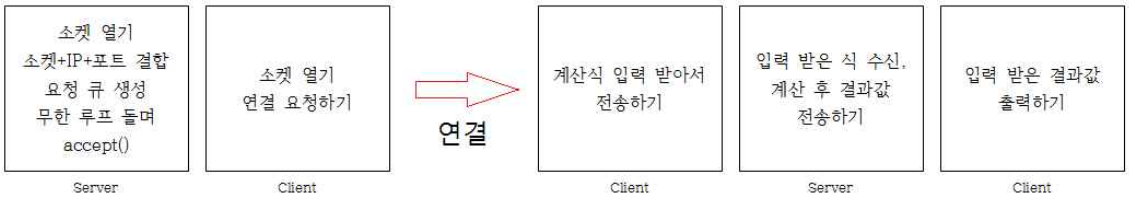
③ HW7-3. 네트워크 통신 활용

필자는 네트워크 통신을 활용하여 IPC 계산 프로그램을 구현하기 위하여 'Beej의 네트워크 프로그래밍 안내' 웹 페이지를 적극 참고 하였으며, 예제로 나온 소스의 내용을 숙지 후 약간씩만 변형시켜 목표로 했던 기능을 구현할 수 있었다. 먼저 Server 프로그램과 Client 프로그램 간 Connect만 시켜주면, 그 이후는 send()와 recv()를 이용해 쉽게 데이터를 주고받을 수 있다.



[그림 16] 네트워크 통신 방식

앞선 HW7-1, HW7-2 에서는 Server 에서 식을 입력 받아 Client 에서 계산을 하였지만, 이번 HW7-3에서는 Server 와 Client 의 기능을 바꾸었다. 그 이유는 요청하는 입장과 응답하는 입장을 고려해 봤을 때 이 명칭이 더 적합하다고 생각했기 때문이다. 이제 네트워크 방식 IPC 계산 프로그램 실행 절차를 한 번 살펴보자.

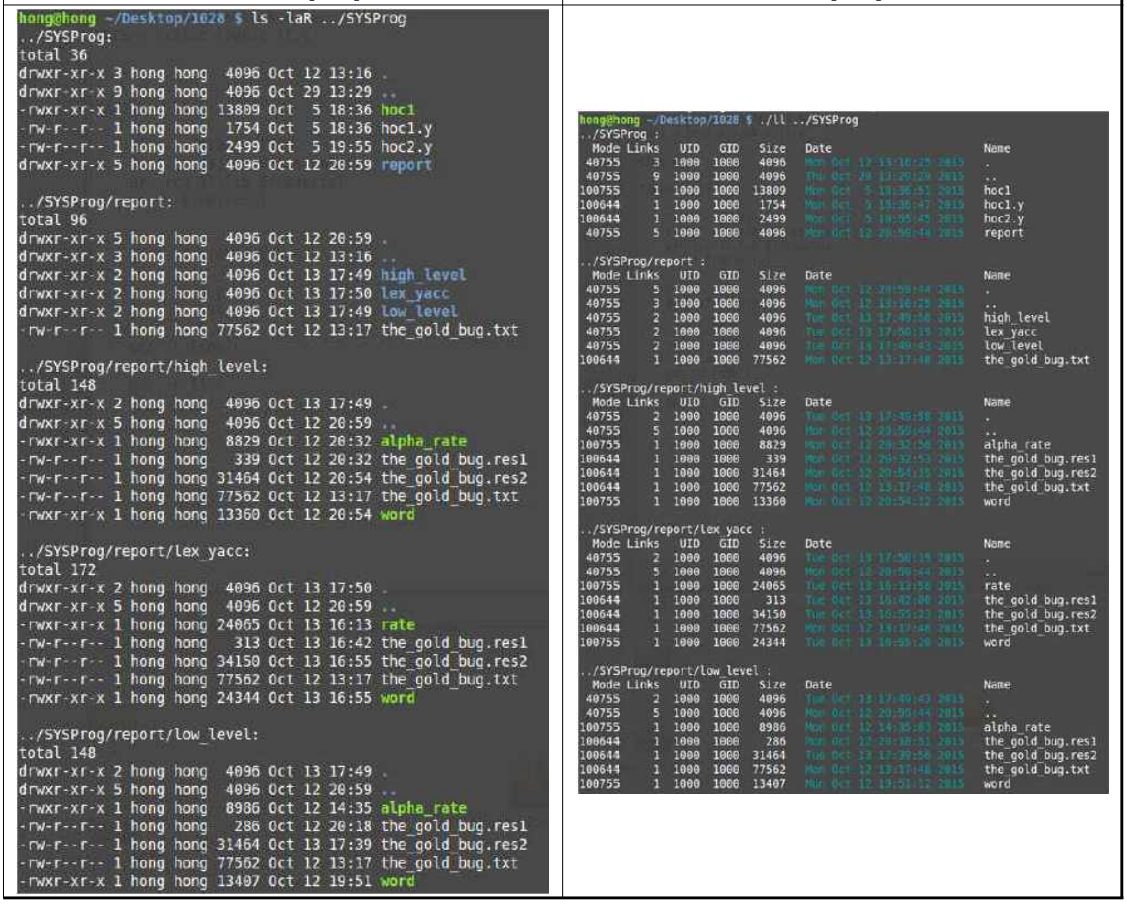


[그림 17] 네트워크 방식 IPC 계산 프로그램

- 결과 분석

· ll 프로그램

ls -laR [dir]	ll [dir]



```
hong@hong ~/Desktop/1028 $ ls -laR ../SYSProg
../SYSProg:
total 36
drwxr-xr-x 3 hong hong  4096 Oct 12 13:16 .
drwxr-xr-x 9 hong hong  4096 Oct 29 13:29 ..
-rwxr-xr-x 1 hong hong 13809 Oct  5 18:36 hoc1
-rw-r--r-- 1 hong hong  1754 Oct  5 18:36 hoc1.y
-rw-r--r-- 1 hong hong  2499 Oct  5 19:55 hoc2.y
drwxr-xr-x 5 hong hong  4096 Oct 12 20:59 report

../SYSProg/report:
total 96
drwxr-xr-x 5 hong hong  4096 Oct 12 20:59 .
drwxr-xr-x 3 hong hong  4096 Oct 12 13:16 ..
drwxr-xr-x 2 hong hong  4096 Oct 13 17:49 high_level
drwxr-xr-x 2 hong hong  4096 Oct 13 17:50 lex_yacc
drwxr-xr-x 2 hong hong  4096 Oct 13 17:49 low_level
-rw-r--r-- 1 hong hong 77562 Oct 12 13:17 the_gold_bug.txt

../SYSProg/report/high_level:
total 148
drwxr-xr-x 2 hong hong  4096 Oct 13 17:49 .
drwxr-xr-x 5 hong hong  4096 Oct 12 20:59 ..
-rwxr-xr-x 1 hong hong  8829 Oct 12 20:32 alpha_rate
-rw-r--r-- 1 hong hong   339 Oct 12 20:32 the_gold_bug.res1
-rw-r--r-- 1 hong hong 31464 Oct 12 20:54 the_gold_bug.res2
-rw-r--r-- 1 hong hong 77562 Oct 12 13:17 the_gold_bug.txt
-rwxr-xr-x 1 hong hong 13360 Oct 12 20:54 word

../SYSProg/report/lex_yacc:
total 172
drwxr-xr-x 2 hong hong  4096 Oct 13 17:50 .
drwxr-xr-x 5 hong hong  4096 Oct 12 20:59 ..
-rwxr-xr-x 1 hong hong 24065 Oct 13 16:13 rate
-rw-r--r-- 1 hong hong   313 Oct 13 16:42 the_gold_bug.res1
-rw-r--r-- 1 hong hong 34150 Oct 13 16:55 the_gold_bug.res2
-rw-r--r-- 1 hong hong 77562 Oct 12 13:17 the_gold_bug.txt
-rwxr-xr-x 1 hong hong 24344 Oct 13 16:55 word

../SYSProg/report/low_level:
total 148
drwxr-xr-x 2 hong hong  4096 Oct 13 17:49 .
drwxr-xr-x 5 hong hong  4096 Oct 12 20:59 ..
-rwxr-xr-x 1 hong hong  8986 Oct 12 14:35 alpha_rate
-rw-r--r-- 1 hong hong   286 Oct 12 20:18 the_gold_bug.res1
-rw-r--r-- 1 hong hong 31464 Oct 13 17:39 the_gold_bug.res2
-rw-r--r-- 1 hong hong 77562 Oct 12 13:17 the_gold_bug.txt
-rwxr-xr-x 1 hong hong 13407 Oct 12 19:51 word
```

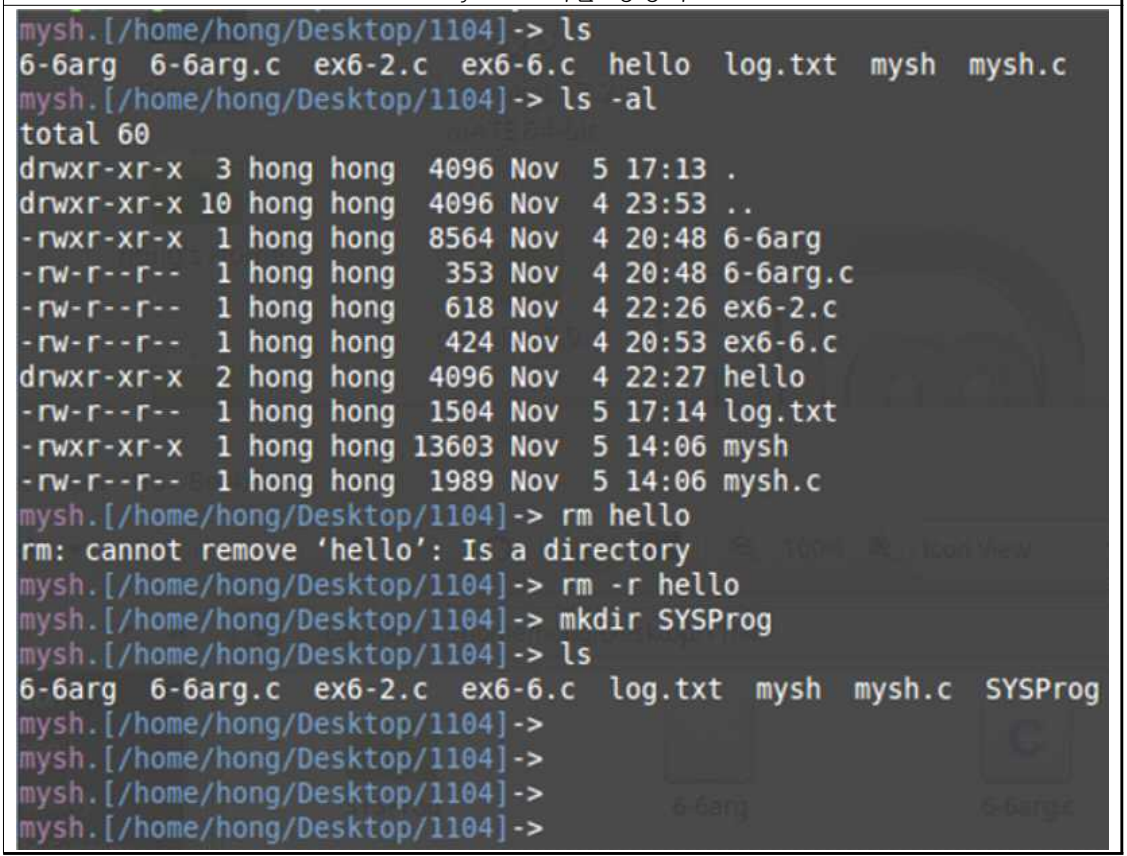
```
hong@hong ~/Desktop/1028 $ ./ll ../SYSProg
../SYSProg :
 Mode Links   UID   GID   Size   Date                    Name
 40755     3  1000  1000   4096   Mon Oct 12 13:16:25 2015  .
 40755     9  1000  1000   4096   Thu Oct 29 13:29:29 2015  ..
100755     1  1000  1000  13809   Mon Oct  5 18:36:51 2015  hoc1
100644     1  1000  1000   1754   Mon Oct  5 18:36:47 2015  hoc1.y
100644     1  1000  1000   2499   Mon Oct  5 19:55:45 2015  hoc2.y
 40755     5  1000  1000   4096   Mon Oct 12 20:59:44 2015  report

../SYSProg/report :
 Mode Links   UID   GID   Size   Date                    Name
 40755     5  1000  1000   4096   Mon Oct 12 20:59:44 2015  .
 40755     3  1000  1000   4096   Mon Oct 12 13:16:25 2015  ..
 40755     2  1000  1000   4096   Tue Oct 13 17:49:56 2015  high_level
 40755     2  1000  1000   4096   Tue Oct 13 17:50:19 2015  lex_yacc
 40755     2  1000  1000   4096   Tue Oct 13 17:49:42 2015  low_level
100644     1  1000  1000  77562   Mon Oct 12 13:17:48 2015  the_gold_bug.txt

../SYSProg/report/high_level :
 Mode Links   UID   GID   Size   Date                    Name
 40755     2  1000  1000   4096   Tue Oct 13 17:49:56 2015  .
 40755     5  1000  1000   4096   Mon Oct 12 20:59:44 2015  ..
100755     1  1000  1000   8829   Mon Oct 12 20:32:56 2015  alpha_rate
100644     1  1000  1000    339   Mon Oct 12 20:32:53 2015  the_gold_bug.res1
100644     1  1000  1000  31464   Mon Oct 12 20:54:15 2015  the_gold_bug.res2
100644     1  1000  1000  77562   Mon Oct 12 13:17:48 2015  the_gold_bug.txt
100755     1  1000  1000  13360   Mon Oct 12 20:54:12 2015  word

../SYSProg/report/lex_yacc :
 Mode Links   UID   GID   Size   Date                    Name
 40755     2  1000  1000   4096   Tue Oct 13 17:50:19 2015  .
 40755     5  1000  1000   4096   Mon Oct 12 20:59:44 2015  ..
100755     1  1000  1000  24065   Tue Oct 13 16:13:56 2015  rate
100644     1  1000  1000    313   Tue Oct 13 16:42:00 2015  the_gold_bug.res1
100644     1  1000  1000  34150   Tue Oct 13 16:55:23 2015  the_gold_bug.res2
100644     1  1000  1000  77562   Mon Oct 12 13:17:48 2015  the_gold_bug.txt
100755     1  1000  1000  24344   Tue Oct 13 16:55:20 2015  word

../SYSProg/report/low_level :
 Mode Links   UID   GID   Size   Date                    Name
 40755     2  1000  1000   4096   Tue Oct 13 17:49:42 2015  .
 40755     5  1000  1000   4096   Mon Oct 12 20:59:44 2015  ..
100755     1  1000  1000   8986   Mon Oct 12 14:35:03 2015  alpha_rate
100644     1  1000  1000    286   Mon Oct 12 20:18:51 2015  the_gold_bug.res1
100644     1  1000  1000  31464   Tue Oct 13 17:39:56 2015  the_gold_bug.res2
100644     1  1000  1000  77562   Mon Oct 12 13:17:48 2015  the_gold_bug.txt
100755     1  1000  1000  13407   Mon Oct 12 19:51:11 2015  word
```

[그림 18] ls -laR [dir] 와 ll [dir] 실행화면 비교

실행화면을 살펴보면 거의 똑같지만 차이가 나는 것은 접근권한, UID, GID가 문자열로 출력되었는지 아니면 정수로 출력 되었는 지이다. 구현하는데 있어서 특별한 어려움은 없었으며 먼저 ls -la를 구현 한 뒤, 리스트를 다시 탐색하면서 디렉토리 파일일 경우 재귀적으로 호출하는 것이 포인트이다.

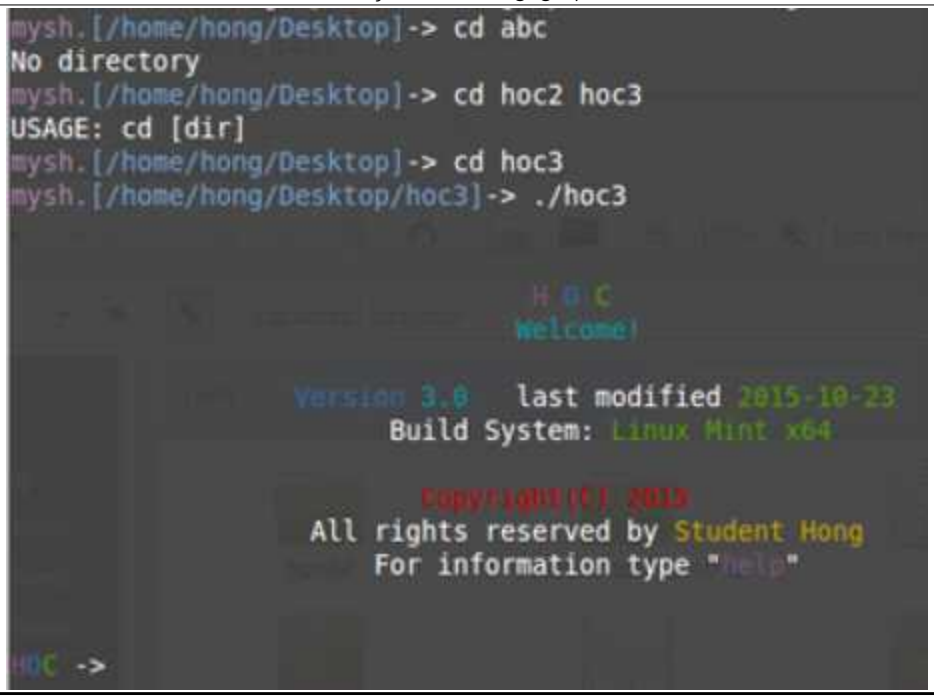
· mysh 쉘 프로그램

mysh - 기본 명령어



[그림 19] mysh 프로그램 기본 명령어 입력 화면

mysh - cd 명령어



[그림 20] mysh 프로그램 cd 명령어 입력 화면

mysh - exit 명령어

[그림 21] mysh 프로그램 exit 명령어 입력 화면

mysh - log 기록

```
ls                      Thu Nov  5 17:14:47 2015

ls -al                  Thu Nov  5 17:14:49 2015

rm hello                Thu Nov  5 17:14:59 2015

rm -r hello             Thu Nov  5 17:15:02 2015

mkdir SYSProg           Thu Nov  5 17:15:10 2015

ls                      Thu Nov  5 17:15:11 2015
```

[그림 22] mysh 프로그램 log.txt 파일

위 결과 화면들에서 알 수 있듯이 mysh 프로그램은 간단한 쉘 프로그램으로서는 전혀 손색이 없다. 기본명령어 실행은 당연하며, 프롬프트에 현재경로도 표시되고, Change Directory(cd) 명령어, Exit 명령어도 실행이 잘 되는 것을 알 수 있다. 또한 log.txt에 저장되는 기록들도 칸에 맞게 정렬되어 깔끔하게 저장되는 모습이다.

· IPC 계산기 프로그램

IPC 계산기 프로그램의 결과는 모두 비슷하기 때문에 각각의 방법에 따른 스크린샷 보다는 부모자식 프로세스 간 통신했을 경우의 결과와 상호 다른 프로세스 간 통신했을 경우의 결과 이 두 가지로 구분하여 살펴보려한다. (사실 통신 방법의 차이지 IPC 계산기 프로그램이 갖는 특징이 변하는 것은 아니다.)

부모자식 프로세스	다른 프로세스
Input the formula : 3*7 Child Process : Calculating... Parent Process : Wait... The Answer : 21	hong@hong ~/Desktop/1125 $ ./server& [1] 19831 hong@hong ~/Desktop/1125 $ ./client1 Client============= Input the formula : 3*7 Server============= server: got connection from 127.0.0.1 Server============= Claculating... Client============= The answer : 21

[그림 23] IPC 계산 프로그램 실행 화면

결과만 봐서는 잘 알 수 없겠지만 현재 프로세스의 정보를 최대한 출력하려고 노력한 모습이다. 부모자식 프로세스간의 통신 실행 화면에 쓰인 방식은 메모리 매핑 방식이고 다른 프로세스간의 통신 실행 화면에 쓰인 방식은 네트워크 방식이다. 개인적으로 이번 HW7 중에서 가장 흥미로웠던 방식은 네트워크 방식이다.

3. 결론

- 느낀점 및 배운점

각각의 HW를 직접 구현하는 과정에서의 배운 점도 많지만, 또 이렇게 Term Project를 통해서 하나의 보고서로 만드는 과정에서 배운 점도 많습니다. 아직 많이 부족하지만 처음에 비하면 비약적인 발전을 했다고 스스로 생각되어집니다. 이번 수업을 통해서 수업 내용에만 국한된 것이 아니라 리눅스 환경에도 익숙해지고, 보고서를 쓰는 연습, 또 오일러 프로젝트를 통한 사고력 향상까지 할 수 있어서 여러모로 배울 수 있었습니다. 수업 내용 또한 lex와 yacc의 활용, 여러 가지 방법을 이용한 상호 프로세스 간 통신 방법 등 정말 흥미로운 내용이었습니다.