

# 운영체제및실습 01분반 실습 6회차

세마포어

2021.04.28(수)

# Contents

## 1. 리눅스 명령어(5)

- 1) id, chmod, umask, (file mode), su, passwd

## 2. 실습

- 1) PIPE & Redirection & FIFO
- 2) Semaphore & Named Semaphore

## 3. 과제

- 1) Ping-Pong Game (using FIFO & Named Semaphore)

# 리눅스 명령어(5)

1. id, chmod, umask, (file mode), su, passwd

# 리눅스 명령어(5)

## ■ id

: id 출력

```
03:58 $ id
uid=1000(joonhee) gid=1000(joonhee) groups=1000(joonhee),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
```

\*uid : 사용자 ID

\*gid : 주요 그룹 ID

\*groups : 소속된 그룹들

```
04:12 $ file /etc/shadow
/etc/shadow: regular file, no read permission
✓ ~/os_prac/06 [main|+1...37]
04:12 $ less /etc/shadow
/etc/shadow: Permission denied
✗1 ~/os_prac/06 [main|+1...37]
```

## ■ file mode

```
04:12 $ > foo.txt
✓ ~/os_prac/06 [main|+1...38]
04:15 $ ls -l foo.txt
-rwxrwxr-- 1 joonhee joonhee 0 4월 28 04:15 foo.txt
```

Owner	Group	World
rwx	rwx	rwx

파일 타입

- : regular 파일

d : 디렉토리 / l : 링크

c : character special (e.g. 키보드, 마우스)

b : block special (e.g. 주변 장치)

p : 파이프 / s : 소켓

r : 읽기 (4) Read

w : 쓰기 (2) Write

x : 실행 (1) eXecute

# 리눅스 명령어(5)

## ■ chmod

: change file mode

Owner	Group	World
rWX	rWX	rWX

r : 읽기 (4) Read

w : 쓰기 (2) Write

x : 실행 (1) eXecute

Table 9-4: File Modes in Binary and Octal

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwX

```
joonhee@joonhee-laptop:~$ > foo.txt
joonhee@joonhee-laptop:~$ ls -l foo.txt
-rw----- 1 joonhee joonhee 0 4월 28 04:29 foo.txt
joonhee@joonhee-laptop:~$ chmod 600 foo.txt
joonhee@joonhee-laptop:~$ ls -l foo.txt
-rw----- 1 joonhee joonhee 0 4월 28 04:29 foo.txt
joonhee@joonhee-laptop:~$ chmod 776 foo.txt
joonhee@joonhee-laptop:~$ ls -l foo.txt
-rwxrwxrw- 1 joonhee joonhee 0 4월 28 04:29 foo.txt
```

# 리눅스 명령어(5)

## ■ umask : set default permissions

(참고) chown, chgrp : 소유자, 그룹 변경  
/ useradd, adduser : 유저 추가  
궁금하면 검색 ㄱㄱ

```
joonhee@joonhee-laptop:~$ umask  
0002
```

```
joonhee@joonhee-laptop:~$ ls -l foo.txt  
-rw-rw-r-- 1 joonhee joonhee 0 4월 28 04:47 foo.txt  
joonhee@joonhee-laptop:~$ rm foo.txt  
joonhee@joonhee-laptop:~$ umask 0000  
joonhee@joonhee-laptop:~$ > foo.txt  
joonhee@joonhee-laptop:~$ ls -l foo.txt  
-rw-rw-rw- 1 joonhee joonhee 0 4월 28 04:48 foo.txt
```

Original file mode	--- rw- rw- rw-
Mask	000 000 000 010
Result	--- rw- rw- r--

## ■ su : 다른 사용자 또는 그룹의 ID로 셸을 실행

```
joonhee@joonhee-laptop:~$ su -  
Password:  
root@joonhee-laptop:~# exit  
logout
```

```
joonhee@joonhee-laptop:~$ su -c 'file /etc/shadow'  
Password:  
/etc/shadow: ASCII text  
joonhee@joonhee-laptop:~$ file /etc/shadow  
/etc/shadow: regular file, no read permission
```

-c옵션: super user 권한으로 커맨드 실행 (sudo랑 유사)

## ■ passwd : 비밀번호 변경

```
joonhee@joonhee-laptop:~$ passwd  
Changing password for joonhee.  
Current password:  
New password:  
Retype new password:  
passwd: password updated successfully
```

# 실습

1. PIPE & Redirection & FIFO
2. Semaphore & Named Semaphore



# pipe & redirection

## ■ pipe & redirection

- ◆ pipe : 명령어(프로세스)의 출력 -> 다른 명령어(프로세스)의 입력
- ◆ redirection : 명령어(프로세스)의 표준/입/출/에러 목적지 **재정의**

소스코드

```
1 #include <stdio.h>
2
3 int main()    input.c
4 {
5     printf("hello\n");
6     return 0;
7 }
```

```
1 #include <stdio.h>
2
3 int main()    output.c
4 {
5     char buf[64];
6     fgets(buf, 64, stdin);
7     printf("input : %s", buf);
8     return 0;
9 }
```

각 파일 실행결과

```
✓ ~/os_prac/06 [main| 5+ 1...39]
03:33 $ ./input
hello
✓ ~/os_prac/06 [main| 5+ 1...39]
03:33 $ ./output
joonhee
input : joonhee
```

파이프

```
03:33 $ ./input | ./output
input : hello
```

리다이렉션

```
03:34 $ ./input > msg
✓ ~/os_prac/06 [main| 5+ 1...39]
03:34 $ cat msg
hello
✓ ~/os_prac/06 [main| 5+ 1...39]
03:34 $ ./output < msg
input : hello
```

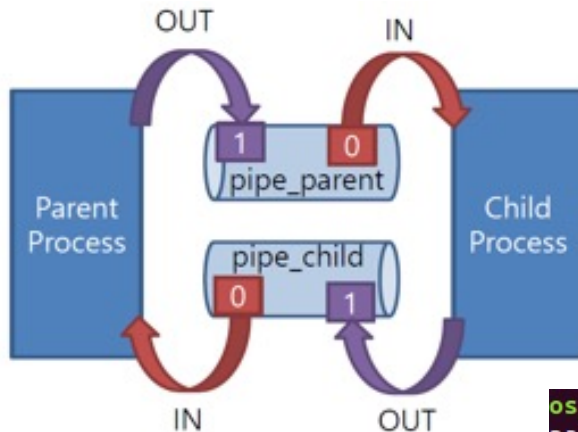


# pipe()

## ■ pipe()를 통한 부모/자식 프로세스 간 데이터 통신

◆ **int pipe(int pipefd[2]);** // unistd.h

- pipefd[0] : 읽기 (출구)
- pipefd[1] : 쓰기 (입구)



### 실행결과

```
os_ta@OS:~/Desktop/Week6/prac2$ ./pipe
parent : (child) test_pipe
child : (parent) test_pipe
```

◆ **ssize\_t write(int fd, const void \*buf, size\_t n);**

- buf에서 n 바이트만큼 읽어서 fd에 쓴다

◆ **ssize\_t read(int fd, void\*buf, size\_t n);**

- fd에서 n 바이트만큼 읽어서 buf에 쓴다
- 0 리턴하면 eof(end-of-file)

◆ **void \*memset(void \*s, int c, size\_t n);** // string.h

- s에 n 바이트를 각각 c로 초기화 (c가 3이면 각 바이트 3)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUF_SIZE 100

int main(void)
{
    int    pipe_parent[2];
    int    pipe_child[2];
    char   buf[BUF_SIZE];
    pid_t  pid;

    pipe(pipe_parent);
    pipe(pipe_child);

    pid = fork();

    if (pid == 0)
    {
        sprintf(buf, "(child) test_pipe");
        write(pipe_child[1], buf, strlen(buf));
        memset(buf, 0x00, BUF_SIZE);

        read(pipe_parent[0], buf, BUF_SIZE);
        printf("child : %s\n", buf);
    }
    else
    {
        sprintf(buf, "(parent) test_pipe");
        write(pipe_parent[1], buf, strlen(buf));
        memset(buf, 0x00, BUF_SIZE);

        read(pipe_child[0], buf, BUF_SIZE);
        printf("parent : %s\n", buf);
    }
    sleep(1);
    return 0;
}
```

# PIPE vs FIFO

## ■ PIPE vs FIFO

- ◆ Name PIPE (명명된 파이프)
- ◆ PIPE나 FIFO나 선입선출(First-In First-Out)의 구조임
- ◆ PIPE는 단방향이며, 커널의 버퍼를 이용하고, 프로세스가 종료되면 정리됨.
- ◆ FIFO는 양방향이며, 생성 후 일반 파일처럼 취급됨. 프로세스 종료되어도 파일이 남음.
- ◆ 그러나 FIFO는 생성 후 양쪽 끝이 동시에 연결상태가 될 때까지는 Block (읽기가 연결되면 반대쪽에 쓰기가 연결될 때까지 Block, 반대 경우도 마찬가지)
- ◆ mkfifo()로 생성

# FIFO

[https://github.com/JoonheeJeong/cnu\\_os\\_prac/tree/main/06](https://github.com/JoonheeJeong/cnu_os_prac/tree/main/06)

## ■ FIFO를 이용하여 두 프로세스 간 데이터 통신

◆ send.c : 데이터 송신

◆ recv.c : 데이터 수신

■ `int mkfifo(const char *pathname, mode_t mode);`

◆ `pathname`  
: 파일 경로

◆ `mode`  
: 접근 권한  
(umask 이용)

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <fcntl.h>
5
6 #define BUF_SIZE 100
7 #define FIFO_PATH "./fifo_temp"
8
9 int main()
10 {
11     int cnt = 0;
12     int fd;
13     char buf[BUF_SIZE];
14
15     fd = open(FIFO_PATH, O_WRONLY);
16     while(1) {
17         memset(buf, 0, BUF_SIZE);
18         fgets(buf, BUF_SIZE, stdin);
19         write(fd, buf, strlen(buf));
20     }
21     close(fd);
22     return 0;
23 }
```

write\_fifo.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4 #include <fcntl.h>
5 #include <sys/stat.h>
6
7 #define FIFO_PATH "./fifo_temp"
8 #define BUF_SIZE 100
9
10 int main()
11 {
12     int cnt = 0;
13     int fd;
14     char buf[BUF_SIZE];
15
16     mkfifo(FIFO_PATH, 0600);
17     fd = open(FIFO_PATH, O_RDONLY);
18     while(1) {
19         memset(buf, 0, BUF_SIZE);
20         read(fd, buf, BUF_SIZE);
21         printf("%d: %s", ++cnt, buf);
22     }
23     close(fd);
24     return 0;
25 }
```

read\_fifo.c

```
03:40 $ ls -l fifo_temp
prw----- 1 joonhee joonhee 0  4월 28 02:33 fifo_temp
```

# FIFO

## ■ FIFO 실행결과

- ◆ write\_fifo 실행 → FIFO\_FILE 생성됨 → 대기
- ◆ 다른 터미널 열기 → read\_fifo 실행 후 메시지 입력
- ◆ read\_fifo 실행시킨 터미널 확인
- ◆ 반드시 read\_fifo 먼저 실행(fifo 생성)/종료(버퍼 외부 접근 문제)



```
02:33 $ ./write_fifo
hi
hello
joonhee
202112345
^C
```

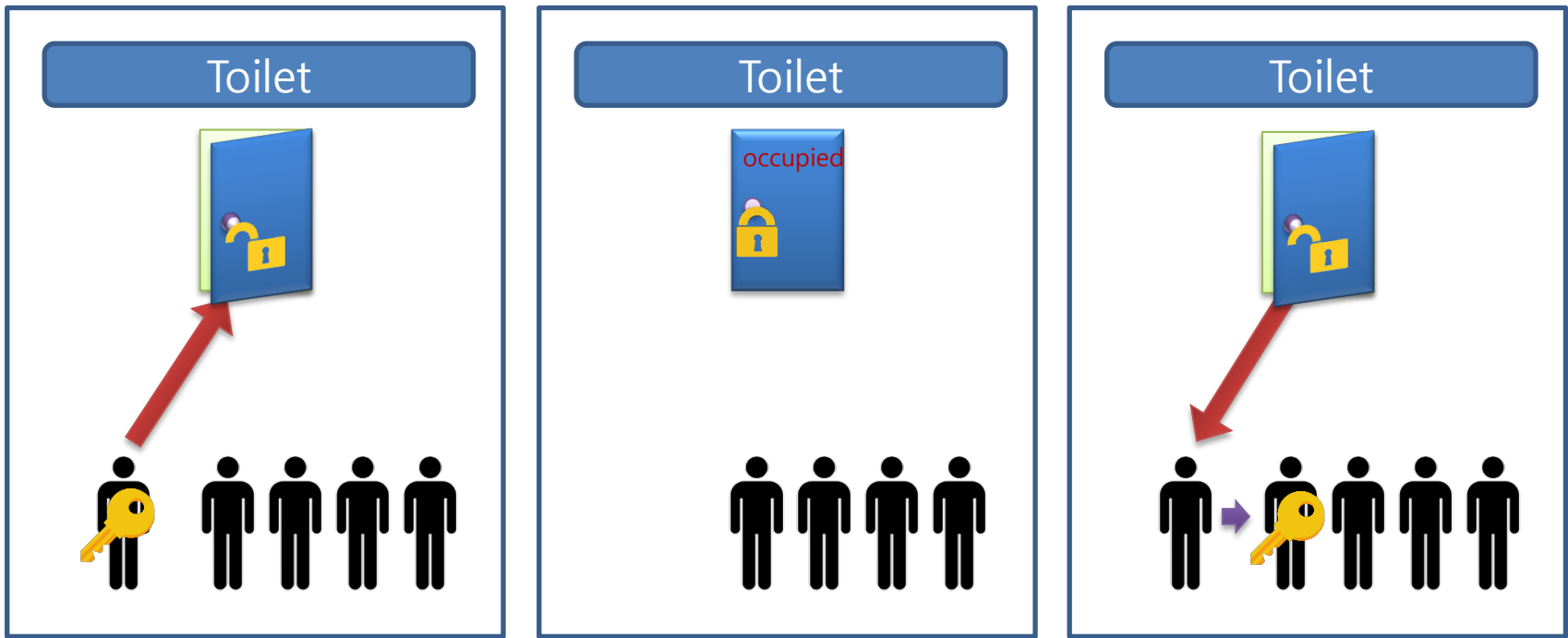
```
03:40 $ ./read_fifo
1: hi
2: hello
3: joonhee
4: 202112345
^C
```

send

# Concurrency

## ■ Mutual Exclusion



- ◆ Critical Section에 대해 한 번에 하나의 실행단위만 접근 허용
  - 소유 → 소유한 실행단위가 해제 후 다른 실행단위가 접근 가능
- ◆ 예제 (  : 실행단위,  : critical section 소유)

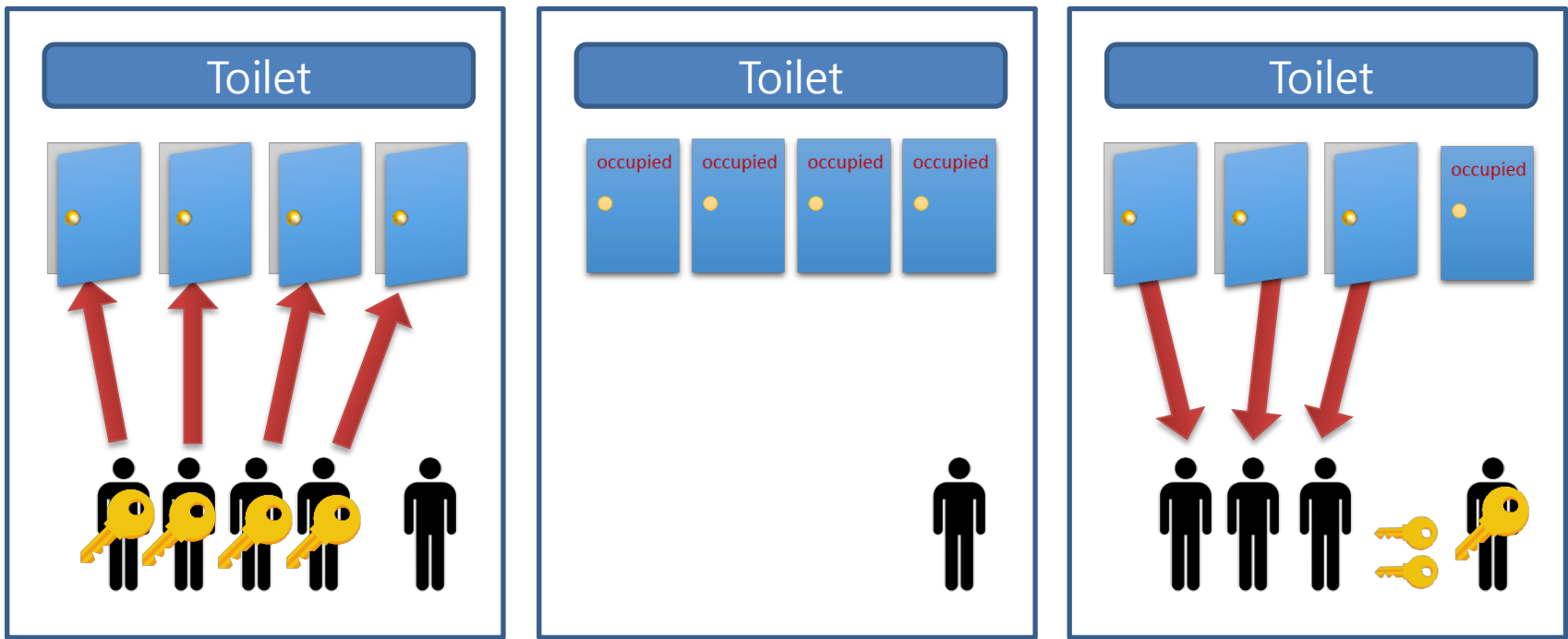


[참고] <https://www.youtube.com/watch?v=DvF3AsTgIUU>

# Concurrency

## ■ Semaphore

- ◆ 최대 허용치만큼 동시 접근 가능
- ◆ 예제 (  : 실행단위, 최대 허용치 :  x4)



[참고] <https://www.youtube.com/watch?v=DvF3AsTgIUU>

# Concurrency

## ■ Semaphore

Semaphore 관련 함수	설명
<code>sem_t *<b>sem_open</b>(const char* name, int oflag, mode_t mode, int value)</code>	Named 세마포어 생성
<code><b>sem_unlink</b>(const char* name)</code>	Named 세마포어 제거
<code><b>sem_close</b>(sem_t* sem)</code>	Named 세마포어 사용 종료
<code>int <b>sem_init</b>(sem_t* sem, int pshared, unsigned value)</code>	세마포어 할당
<code><b>sem_destroy</b>(sem_t* sem)</code>	세마포어 반환
<code>int <b>sem_wait</b>(sem_t* sem)</code>	세마포어 lock
<code>int <b>sem_post</b>(sem_t* sem)</code>	세마포어 unlock
<code>int <b>sem_getvalue</b>(sem_t *<i>sem</i>, int *<i>sval</i>)</code>	Sval 위치에 sem 값 세팅



# Concurrency

## ■ Semaphore

```
03:02 $ ./semaphore
342599424 In (key:3)
317421312 In (key:2)
201324288 In (key:1)
325814016 In (key:0)
325814016 Out (key:1)
342599424 Out (key:2)
317421312 Out (key:3)
201324288 Out (key:4)
334206720 In (key:3)
334206720 Out (key:4)

03:02 $ ./semaphore
2606249728 In (key:3)
2581071616 In (key:0)
2597857024 In (key:1)
2589464320 In (key:2)
2606249728 Out (key:1)
2581071616 Out (key:2)
2572678912 In (key:1)
2589464320 Out (key:2)
2597857024 Out (key:3)
2572678912 Out (key:4)
```

### sem\_wait

- 카운트 1 감소
- 0이면 대기

### sem\_post

- 카운트 1 증가

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5
6 #define MAN 5
7 int key = 4;
8 sem_t sem;
9
10 void use();
11
12 int main()
13 {
14     pthread_t man[MAN];
15
16     sem_init(&sem, 0, key);
17     for (int i = 0; i < 5; ++i)
18         pthread_create(&man[i], NULL, (void *) use, NULL);
19     for (int i = 0; i < 5; ++i)
20         pthread_join(man[i], NULL);
21     sem_destroy(&sem);
22     return 0;
23 }
24
25 void use()
26 {
27     sem_wait(&sem);
28     key--;
29     printf("%u In (key:%d)\n", (unsigned) pthread_self(), key);
30     usleep(100);
31     key++;
32     printf("%u Out (key:%d)\n", (unsigned) pthread_self(), key);
33     sem_post(&sem);
34 }
semaphore.c" 34 lines --2%--
```

# Concurrency

## ■ Named Semaphore

```
1 #include <stdio.h>      // perror
2 #include <stdlib.h>     // close
3 #include <unistd.h>     // open, write, read
4 #include <sys/stat.h>   // sem_open
5 #include <fcntl.h>      // mode, flag
6 #include <sys/mman.h>   // mmap: memory manager -- mmap()
7 #include <semaphore.h>
8
9 int main()
10 {
11     const int nloop = 5;
12     const int zero = 0;
13     int fd;
14     int *p_fd;
15     int i;
16     sem_t *p_sem;
17
18     fd = open("log.txt", O_RDWR | O_CREAT, S_IRWXU);
19     write(fd, &zero, sizeof(int));
20     p_fd = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
21     close(fd);
22
23     sem_unlink("my_sem");
24     if ((p_sem = sem_open("my_sem", O_CREAT, 0600, 1)) == SEM_FAILED) {
25         perror("sem_open");
26         exit(1);
27     }
28 }
```

읽기/쓰기, 공유 가능한 가상 메모리로 매핑

읽기/쓰기 없으면 생성 소유자 읽기/쓰기/실행 권한

소유자 읽/쓰 권한 (octal) 세마포어 변수 1 초기화

# Concurrency

## ■ Named Semaphore

### 세마포어 사용 / 미사용 결과

```
03:12 $ ./named_sem
parent: 0
parent end
child: 1
child end
parent: 2
parent end
child: 3
child end
parent: 4
parent end
child: 5
child end
parent: 6
parent end
child: 7
child end
child: 8
child end
parent: 9
parent end
```

```
03:12 $ ./named_sem
parent: 0
parent end
child: 1
child end
parent: 2
parent end
child: 3
parent: 4
parent end
child end
parent: 5
parent end
child: 6
parent: 7
parent end
child end
parent end
child: 8
child end
child: 9
child end
```

```
29     if (fork() == 0) { // child
30         for (i = 0; i < nloop; ++i) {
31             sem_wait(p_sem);
32             printf("child: %d\n", *p_fd);
33             (*p_fd)++;
34             usleep(100);
35             printf("child end\n");
36             sem_post(p_sem);
37             usleep(100);
38         }
39     } else { // parent
40         for (i = 0; i < nloop; ++i) {
41             sem_wait(p_sem);
42             printf("parent: %d\n", *p_fd);
43             (*p_fd)++;
44             usleep(100);
45             printf("parent end\n");
46             sem_post(p_sem);
47             usleep(100);
48         }
49     }
50     sem_close(p_sem);
51     return 0;
52 }
```

named\_sem.c" 52 lines --100%--

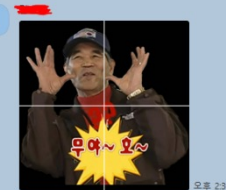
부모든 자식이든 일단 시작하면 끝을 봐야 함  
// 부모연속 자식연속의 경우는 발생 가능

# 과제

1. Ping-Pong Game (using FIFO & Named Semaphore)



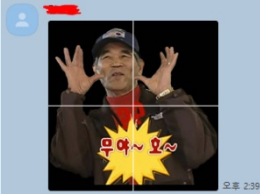
# 과제



[https://github.com/JoonheeJeong/cnu\\_os\\_prac/blob/main/06/pp\\_server\\_hw.c](https://github.com/JoonheeJeong/cnu_os_prac/blob/main/06/pp_server_hw.c)  
git fetch-full해서 쓰시거나 그냥 복붙하세요~

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <semaphore.h>
4  #include <sys/stat.h>
5  #include <fcntl.h>
6  #include <string.h>
7
8  #define FIFO_PATH "fifo_temp"
9  #define SEM_NAME "sem_pp"
10 #define BUF_SIZE 8
11 #define TURN 5
12
13
14 int main()
15 {
16     const char *msg = "ping\n";
17     int fd;
18     int cnt;
19     int score = 100;
20     sem_t *p_sem;
21     char buf[BUF_SIZE];
22
23     /*
24      Homework
25      Fill your codes here and make the correct result.
26      Refer to the included header files and given variables.
27      You can complete this file from this skeleton without creating another variable,
28      but also modify anything as you want.
29      As long as your result is correct, it doesn't matter how you write codes.
30      However, you must use fifo, named semaphore.
31
32      And based on this file, you have to create another source for the client.
33      It would be similar to this.
34     */
35
36     printf("Done! Your score: %d\n", score);
37     return 0;
38 }
```

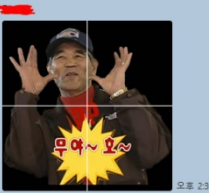
# 과제



- ping, pong을 순서대로 입력하는 게임을 구현하라.  
(pp\_server\_hw.c 샘플 코드 제공)
  - ◆ server와 client 두개의 프로그램으로 구성
  - ◆ server와 client는 각자의 차례까지 대기 – Named Semaphore 사용
    - 각 프로그램은 자기 차례에 상대가 입력한 결과를 볼 수 있어야 함 (FIFO 사용)
  - ◆ 각 프로그램은 점수 100점을 가지고 시작한다.
  - ◆ 실수한 경우 해당 프로그램은 점수 20점 감점.
    - server는 항상 ping, client는 항상 pong을 입력해야 함
  - ◆ 각 프로그램 당 5회 반복한 후, 남은 점수를 출력하면서 프로그램 종료.
  - ◆ 주의 – 무조건 server의 ping이 먼저 시작되어야 함
  - ◆ FIFO, Named Semaphore 필수 사용
  - ◆ Sample 코드는 사용하셔도 좋고, 처음부터 새로 만드셔도 좋습니다.



# 과제



## ■ 예시 답안

```
✓ ~/os_prac/06 [main|+1...23]
02:32 $ ./pp_server
Your turn!
ping
[opponent] pong
Your turn!
pong
wrong! -20
[opponent] pong
Your turn!
ping
[opponent] good
Your turn!
baboya
wrong! -20
[opponent] pong
Your turn!
ping
[opponent] ping
Done! Your score: 60
```

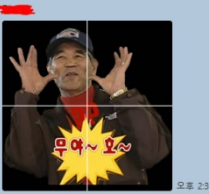
↔

```
✓ ~/os_prac/06 [main|+1...23]
02:32 $ ./pp_client
[opponent] ping
Your turn!
pong
[opponent] pong
Your turn!
pong
[opponent] ping
Your turn!
good
wrong! -20
[opponent] baboya
Your turn!
pong
[opponent] ping
Your turn!
ping
wrong! -20
Done! Your score: 60
```





# 과제



## ■ 포함 내용

### ◆ 소스코드

- pp\_server\_학번.c
- pp\_client\_학번.c

### ◆ 보고서

- 코드 이미지 및 설명 (그렇게 작성한 이유)
- 예상 결과 및 실제 결과 화면
- 느낀 점 (쪽지시험, 수업, 과제 재미, 난이도 기타 등등)
- OS01\_06\_학번\_이름.pdf

✓ 맥북 유저분들, pdf인지 확인하세요!

## ■ 제출기한 및 양식

### ◆ ~5월 4일(화) 23:59 (1주)

### ◆ OS01\_06\_학번\_이름.zip

- 소스코드 (src 폴더에 담아주시면 감사합니다.)
- 보고서



# Q&A

Thank you!!!