

IPC

운영체제 실습

조교 김형준

ledzep0830@korea.ac.kr

Table of Contents

- 1. IPC**
- 2. Shared Memory**
- 3. Semaphore**
- 4. Message Passing – PIPE**
- 5. Assignment**

Section 1

“ IPC ”

IPC (Inter-Process Communication)

- 프로세스 간에 데이터 및 정보를 주고받기 위한 메커니즘
 - 각 프로세스는 독립적인 메모리 공간을 가지므로 통신을 위한 별도의 메커니즘이 필요함

여러 프로세스가 서로 필요한 데이터를 교환하며 작업을 수행할 때 IPC를 사용

프로세스 간의 협력이 필요한 이유

- **정보 공유 (Information sharing)**: 여러 사용자가 동일한 정보에 흥미를 가질 수 있음에 따라, 병행적으로(concurrently) 접근할 수 있는 환경 제공
- **계산 가속화 (Computation speedup)**: task를 sub-task로 나누고, 병렬 처리를 통한 계산 가속화 (ex. MapReduce)
- **모듈성 (Modularity)**: 시스템 기능을 별도의 프로세스 또는 스레드로 나누어 모듈식 형태로 시스템 구성
- **편의성 (Convenience)**: 한 순간에 작업할 많은 task를 가질 수 있음

Shared Memory vs Message Passing

- 공유 메모리 (shared memory)

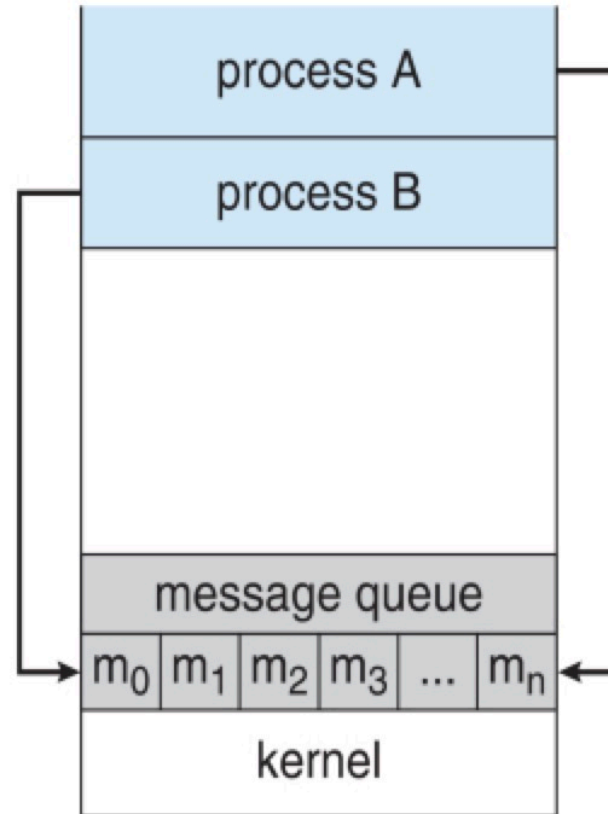
- 프로세스 간에 공유되는 메모리 영역
- 속도가 빠름 (공유 메모리 영역을 생성할 때만 시스템 호출을 사용)
- 많은 양의 데이터를 전달하는 경우에 적합함
- 별도의 동기화 기술이 필요함

- 메시지 전달 (message passing)

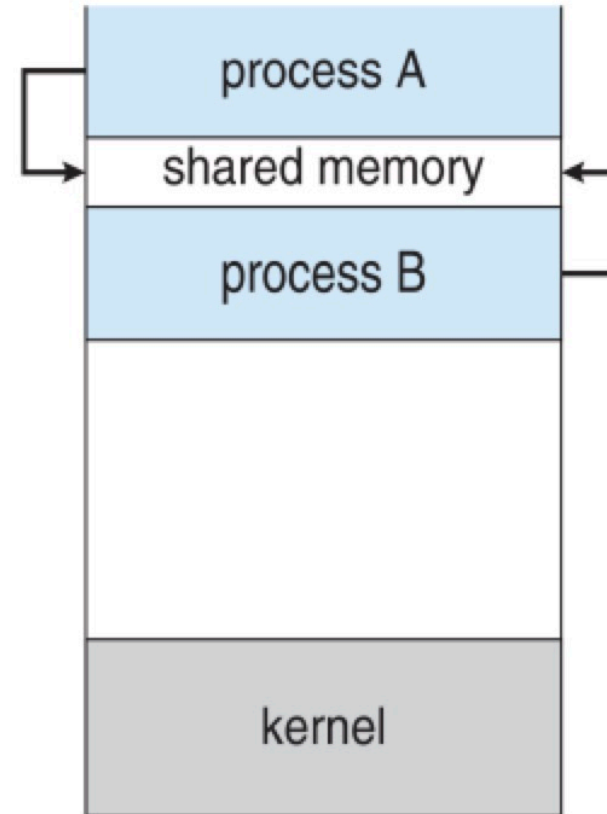
- 프로세스 간에 메시지 형태로 정보를 전달
- 구현이 간단함
- 적은 양의 데이터를 전달하는 경우에 적합함
- 비교적 속도가 느림 (커널을 통해서 최소 두 번의 복사가 이루어짐)

1. IPC

Shared Memory vs Message Passing



(a) Message Passing



(b) Shared Memory

Section 2

“ Shared Memory ”

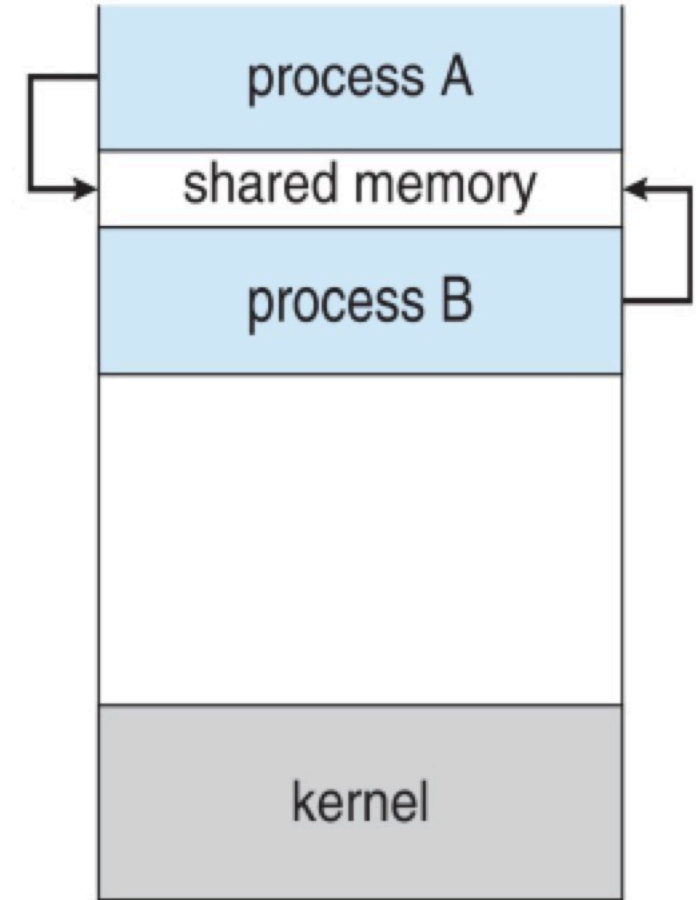
2. Shared Memory

Shared Memory

→ 여러 프로세스가 동시에 접근할 수 있는 메모리 영역

특징

- 과도한 데이터 복사를 막을 수 있음
- 공유 메모리 영역을 구축할 때만 시스템 호출이 필요
- 공유 메모리에 대한 접근은 커널의 도움이 필요 없음
- 많은 프로세스가 동시에 사용하는 경우에는 캐시 일관성 유지로 인해 성능 저하가 발생



2. Shared Memory

Shared Memory

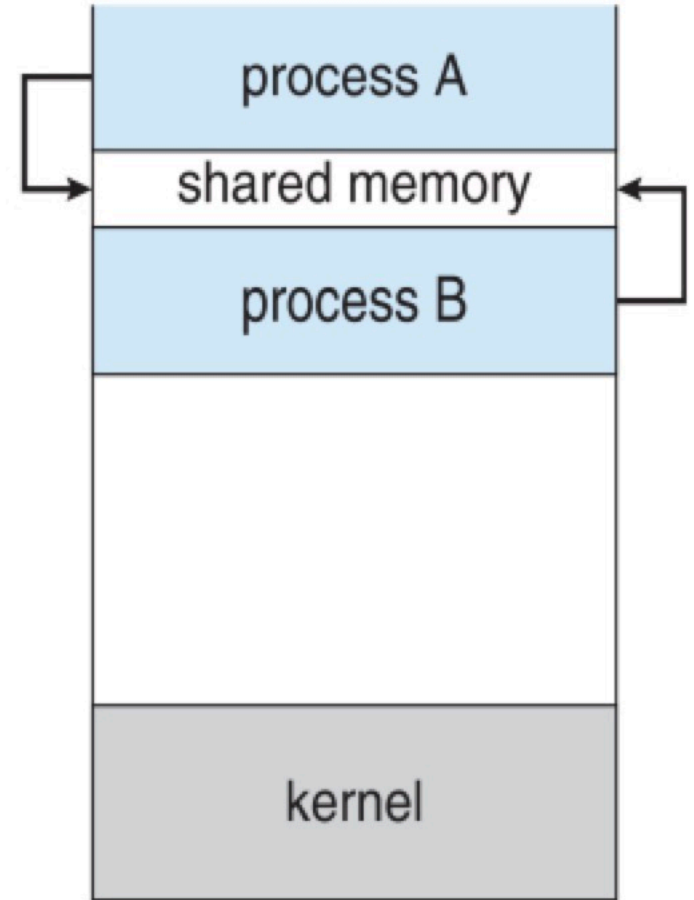
→ 여러 프로세스가 동시에 접근할 수 있는 메모리 영역

사용 조건

- 메모리 보호 제약 해제
- 동시에 동일한 위치에 쓰지 않도록 프로세스들이 책임을 져야 함

생산자-소비자 문제

- 클라이언트-서버 패러다임
- 공유 메모리를 사용해서 해결할 수 있음



2. Shared Memory

Shared Memory 함수

- 공유 메모리 생성 or shmid 얻기

```
int shmget(key_t key, size_t size, int shmflg);
```

- 공유 메모리 attach

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- 공유 메모리 detach

```
int shmdt(const void *shmaddr);
```

- 공유 메모리 컨트롤

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

2. Shared Memory

shared_memory_1.c

```
1 #include <sys/ipc.h>
2 #include <sys/shm.h>
3 #include <stdio.h>
4
5 #define KEY 1234
6
7 int main(){
8     int shmid;
9     int *num;
10    void *memory_segment=NULL;
11
12    if ((shmid = shmget(KEY, sizeof(int), IPC_CREAT|0666)) == -1) return -1;
13
14    printf("shmid : %d\n", shmid);
15
16    if ((memory_segment = shmat(shmid, NULL, 0)) == (void*)-1) return -1;
17
18    num = (int*)memory_segment;
19    (*num)++;
20    printf("num : %d\n", (*num));
21
22    return 0;
23 }
```

2. Shared Memory

shared_memory_1.c 실행 결과

```
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory > master ● gcc -o shared_memory_1 shared_memory_1.c
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory > master ● ./shared_memory_1
shmctl : 65537
num : 1
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory > master ● ./shared_memory_1
shmctl : 65537
num : 2
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory > master ● ./shared_memory_1
shmctl : 65537
num : 3
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory > master ● ipcs -m
IPC status from <running system> as of Mon Apr 18 16:31:43 KST 2022
T      ID      KEY      MODE      OWNER      GROUP
Shared Memory:
m 65536 0x000072a3 --rw-rw-rw-    root    wheel
m 65537 0x000004d2 --rw-rw-rw- kimhyungjun  staff
```

2. Shared Memory

shared_memory_2.c

```
1 #include <sys/ipc.h>
2 #include <sys/shm.h>
3 #include <stdio.h>
4
5 #define KEY 1234
6
7 int main(){
8     int shmid;
9     int *num;
10    void *memory_segment=NULL;
11
12    if ((shmid = shmget(KEY, sizeof(int), IPC_CREAT|0666)) == -1) return -1;
13
14    printf("shmid : %d\n", shmid);
15
16    if ((memory_segment = shmat(shmid, NULL, 0)) == (void*)-1) return -1;
17
18    num = (int*)memory_segment;
19    (*num)++;
20    printf("num : %d\n", (*num));
21
22    if (shmctl(shmid, IPC_RMID, NULL) == -1) return -1;
23
24    return 0;
25 }
```

IPC_RMID (0x00000000)

Remove the shared memory segment identifier *shmid* from the system and destroy the shared memory segment.

2. Shared Memory

shared_memory_2.c 실행 결과

```
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory } master ● gcc -o shared_memory_2 shared_memory_2.c
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory } master ● ./shared_memory_2
shmids : 65537
num : 4
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory } master ● ./shared_memory_2
shmids : 131073
num : 1
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory } master ● ./shared_memory_2
shmids : 196609
num : 1
kimhyungjun > ~/workspace/22-1-os/ipc_project/shared_memory } master ● ipcs -m
IPC status from <running system> as of Mon Apr 18 16:42:09 KST 2022
T      ID      KEY      MODE      OWNER      GROUP
Shared Memory:
m 65536 0x000072a3 --rw-rw-rw-    root    wheel
```

2. Shared Memory

Bounded Buffer

```
#define BUFFER_SIZE 10
typedef struct {
    ...
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- ✓ **in** – points to the next free position in the buffer
- ✓ **out** – points to the first full position in the buffer

Producer

```
item next_produced;
while (true) {
    /* produce an item in next_produced */
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Consumer

```
item next_consumed;
while (true) {
    while (in == out)
        ; /* do nothing */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;

    /* consume the item in next_consumed */
}
```


2. Shared Memory

Assignment 1-1 (10점)

목표: 공유 버퍼 구현하기 (단, Assignment 1-1에서는 동기화를 고려하지 않는다)

message_buffer.h

```
1 #define KEY 54321
2 #define BUFFER_SIZE 10
3
4 typedef struct {
5     int sender_id;
6     int data;
7 } Message;
8
9 typedef struct {
10     Message messages[BUFFER_SIZE];
11     int is_empty;
12     int account_id;
13 } MessageBuffer;
14
15 int init_buffer(MessageBuffer **buffer);
16 int attach_buffer(MessageBuffer **buffer);
17 int detach_buffer();
18 int destroy_buffer();
19 int produce(MessageBuffer **buffer, int sender_id, int data, int account_id);
20 int consume(MessageBuffer **buffer, Message **message);
```

2. Shared Memory

Assignment 1-1 (10점)

실행 방법

\$ chmod 777 run.sh

\$./run.sh

\$./consumer

// 다른 터미널에서

\$./producer 0 1000

// 공유 메모리 제거

\$./destroy

```
kimhyungjun ~/workspace/ipc_project_complete/assignment/assignment_1_1 $ chmod 777 run.sh
kimhyungjun ~/workspace/ipc_project_complete/assignment/assignment_1_1 $ ./run.sh
kimhyungjun ~/workspace/ipc_project_complete/assignment/assignment_1_1 $ ./consumer
init buffer
sender_id : 80551
account_id : 0
balance : 1000

sender_id : 80564
account_id : 0
balance : 900

sender_id : 80588
account_id : 0
balance : 2000
```

```
kimhyungjun ~/workspace/ipc_project_complete/assignment/assignment_1_1 $ ./producer 0 1000
attach buffer

produce message
kimhyungjun ~/workspace/ipc_project_complete/assignment/assignment_1_1 $ ./producer 0 -100
attach buffer

produce message
kimhyungjun ~/workspace/ipc_project_complete/assignment/assignment_1_1 $ ./producer 0 1100
attach buffer

produce message
```

2. Shared Memory

Shared Memory 문제점

동시에 여러 프로세스가 공유 메모리에 있는 값을 증가/감소시키려고 한다면?

→ **Race Condition**

Section 3

“ Semaphore ”

3. Semaphore

Semaphore

- IPC Semaphore는 공유하는 자료 구조에 대한 통제된 접근 방법을 제공하기 위해 사용

특징

- semaphore는 양의 정수 값을 가짐
- 자원에 접근할 때 semaphore 값을 감소시킴
- Semaphore 값이 0이면 커널은 semaphore 값이 양수가 될 때까지 프로세스를 블록시킴
- 프로세스가 보호된 자원에서의 작업을 마치면 semaphore 값을 증가

2. Shared Memory

Semaphore 함수

- semaphore 식별자 얻기

int semget(key_t key, int nsems, int semflg);

- semaphore 값 변경

int semop(int semid, struct sembuf *spos, size_t nsops);

- semaphore 컨트롤

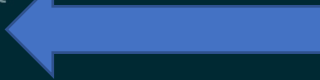
int semctl(int semid, int semnum, int cmd, union semun_arg);

```
Struct sembuf buf {  
    unsigned short sem_num;  
    short sem_op;  
    short sem_flg;  
}
```

3. Semaphore

no_semaphore.c

```
1 #include <sys/ipc.h>
2 #include <sys/shm.h>
3 #include <stdio.h>
4
5 #define SHM_KEY 1234
6
7 int main(){
8     int shmid;
9     int *num;
10    void *memory_segment=NULL;
11
12    // shared memory
13    if ((shmid = shmget(SHM_KEY, sizeof(int), IPC_CREAT|0666)) == -1) return -1;
14
15    printf("shmid : %d\n", shmid);
16
17    if ((memory_segment = shmat(shmid, NULL, 0)) == (void*)-1) return -1;
18
19    num = (int*)memory_segment;
20
21    for (int i=0; i<1000000; i++) {
22        (*num)++;
23    }
24    printf("num : %d\n", (*num));
25
26    return 0;
27 }
```



3. Semaphore

semaphore.c

```
1 #include <sys/ipc.h>
2 #include <sys/shm.h>
3 #include <sys/sem.h>
4 #include <sys/types.h>
5 #include <stdio.h>
6
7 #define SHM_KEY 4321
8 #define SEM_KEY 4321
9
10 union semun {
11     int val;
12     struct semid_ds *buf;
13     unsigned short *array;
14 };
15
16 void s_wait(int semid) {
17     struct sembuf buf;
18     buf.sem_num = 0;
19     buf.sem_op = -1;
20     buf.sem_flg = SEM_UNDO;
21
22     if (semop(semid, &buf, 1) == -1) {
23         printf("<s_wait> semop error!\n");
24         return ;
25     }
26 }
27
```


```
28 void s_quit(int semid) {
29     struct sembuf buf;
30     buf.sem_num = 0;
31     buf.sem_op = 1;
32     buf.sem_flg = SEM_UNDO;
33
34     if (semop(semid, &buf, 1) == -1) {
35         printf("<s_quit> semop error!\n");
36         return ;
37     }
38 }
39
```


3. Semaphore

semaphore.c

```
40 int main(){
41     int shmid;
42     int *num;
43     void *memory_segment=NULL;
44
45     int semid;
46     union semun sem_union;
47
48     // shared memory
49     if ((shmid = shmget(SHM_KEY, sizeof(int), IPC_CREAT|0666)) == -1) return -1;
50
51     printf("shmid : %d\n", shmid);
52
53     if ((memory_segment = shmat(shmid, NULL, 0)) == (void*)-1) return -1;
54
55     // semaphore
56     if ((semid = semget(SEM_KEY, 1, IPC_CREAT|IPC_EXCL|0666)) == -1) {
57         // try as a client
58         if ((semid = semget(SEM_KEY, 0, 0)) == -1) return -1;
59     } else {
60         sem_union.val = 1;
61         semctl(semid, 0, SETVAL, sem_union);
62     }
63
64     printf("semid : %d\n", semid);
```

```
66     num = (int*)memory_segment;
67
68     for (int i=0; i<1000000; i++) {
69         s_wait(semid);
70         (*num)++;
71         s_quit(semid);
72     }
73     printf("num : %d\n", (*num));
74
75     return 0;
76 }
```



3. Semaphore

실행 방법

```
$ chmod 777 run.sh
```

```
$ ./run.sh
```

실행 결과

```
root@hj-desktop:/home/hj/workspace/OS_22-1/ipc_project/semaphore# ./run.sh
shmids : 458756
shmids : 458756
num : 857970
num : 1013799
shmids : 458759
semids : 1
shmids : 458759
semids : 1
num : 1999793
num : 2000000
destroy shmids : 458756
destroy shmids : 458759
destroy semids : 1
```

3. Semaphore

Assignment 1-2 (5점)

목표: assignment 1-1에 semaphore 기능 추가하기

(단, s_quit(), s_wait()가 사용되어야 하는 위치에 대한 설명을 pdf 파일로 첨부할 것)

message_buffer_semaphore.h

```
1 #define SEM_KEY 54321
2 #define SHM_KEY 54321
3 #define BUFFER_SIZE 10
4
5 union semun {
6     int val;
7     struct semid_ds *buf;
8     unsigned short *array;
9 };
10
11 typedef struct {
12     int sender_id;
13     int data;
14 } Message;
15
16 typedef struct {
17     Message messages[BUFFER_SIZE];
18     int is_empty;
19     int account_id;
20 } MessageBuffer;
```

```
22 void init_sem();
23 void destroy_sem();
24 void s_wait();
25 void s_quit();
26
27 int init_buffer(MessageBuffer **buffer);
28 int attach_buffer(MessageBuffer **buffer);
29 int detach_buffer();
30 int destroy_buffer();
31 int produce(MessageBuffer **buffer, int sender_id, int data, int account_id);
32 int consume(MessageBuffer **buffer, Message **message);
```

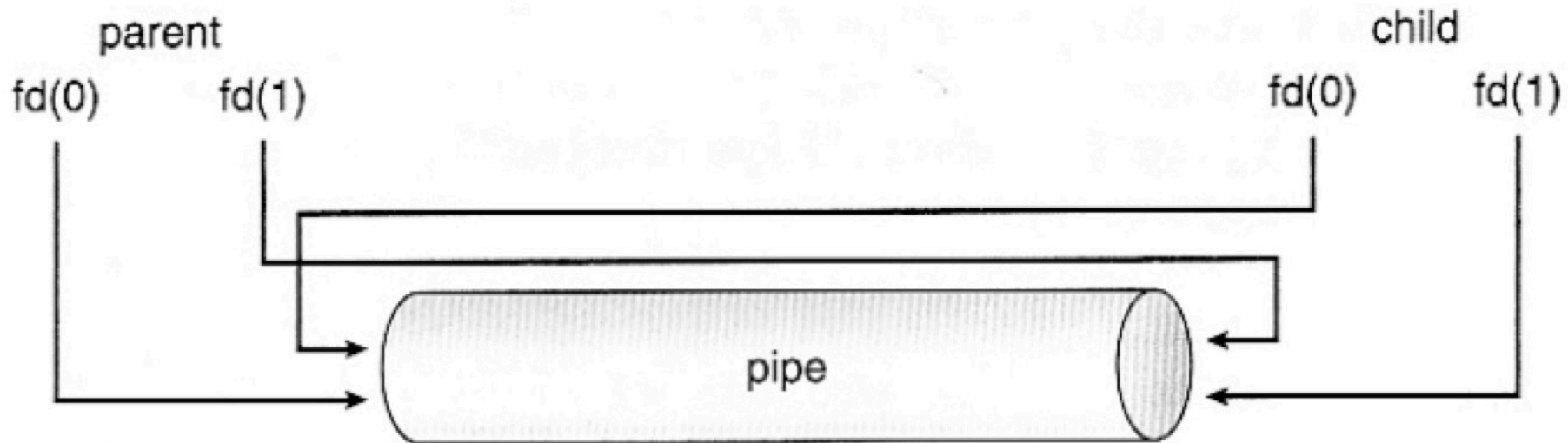
Section 4

“ Message Passing - PIPE ”

4. Message Passing - PIPE

PIPE

- 프로세스 사이에서 통신할 수 있는 전달자로서 동작함
- 한 프로세스가 파이프에 기록한 데이터를 커널이 다른 프로세스로 전달하며, 다른 프로세스는 해당 데이터를 읽을 수 있음
- 특수 파일 시스템 pipefs로 구현됨



4. Message Passing - PIPE

PIPE

- Uni-directional byte stream : 한 쪽에서는 데이터를 write만 하고, 다른 한 쪽에서는 데이터를 read만 함
- Name, ID가 없음 : 파이프를 생성한 프로세스 이외에는 접근할 수 없음
- 관련된 프로세스 간에 사용 가능 (ex. fork)

Named PIPE (FIFO)

- Uni-directional byte stream
- 파일 경로가 ID 역할을 수행 : 서로 관계가 없는 프로세스 간에 통신이 가능
- Named PIPE 생성과 open이 분리되어 있음
- open()을 수행할 때, 양쪽 모두 open 시도가 있어야 동기화되어 통신이 가능

4. Message Passing - PIPE

PIPE

```
int pipe(int pipefd[]);
```

```
pipefd[0] // read
```

```
pipefd[1] // write
```

```
$ gcc -o anonymous_pipe
```

```
anonymous_pipe.c
```

```
kimhyungjun ~/workspace/ipc_project/pipe gcc -o anonymous_pipe anonymous_pipe.c
kimhyungjun ~/workspace/ipc_project/pipe ./anonymous_pipe
child : hello, korea
```

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4
5 #define BUFFER_SIZE 25
6
7 int main(void) {
8     int n, pipefd[2], pid;
9
10    if (pipe(pipefd) < 0) return -1; // generate pipe
11
12    if ((pid = fork()) < 0) return -1;
13    else if (pid > 0) { // parent process
14        close(pipefd[0]);
15        char write_msg[BUFFER_SIZE] = "hello, korea\n";
16
17        write(pipefd[1], write_msg, strlen(write_msg)+1);
18        close(pipefd[1]);
19    } else { // child process
20        close(pipefd[1]);
21        char read_msg[BUFFER_SIZE];
22
23        n = read(pipefd[0], read_msg, BUFFER_SIZE);
24        printf("child : %s\n", read_msg);
25        close(pipefd[0]);
26    }
27    return 0;
28 }
```

4. Message Passing - PIPE

Named PIPE (FIFO)

named_pipe_reader.c

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5 #include <sys/stat.h>
6
7 #define BUFFER_SIZE 100
8 #define PIPENAME "./named_pipe_file"
9
10 int main(void) {
11     char msg[BUFFER_SIZE];
12     int pipefd;
13
14     // 기존 named_pipe 제거
15     if (access(PIPENAME, F_OK) == 0) {
16         unlink(PIPENAME);
17     }
18
19     // 접근 권한이 0666인 named_pipe 생성
20     if (mkfifo(PIPENAME, 0666) == -1) return -1;
21
22     if ((pipefd=open(PIPENAME, O_RDWR)) == -1) return -1;
```

```
        while (1) {
            if (read(pipefd, msg, sizeof(msg)) == -1) return -1;
            if (!strcmp(msg, "quit")) return 0;
            printf("receive msg : %s\n", msg);
        }
        return 0;
    }
```


4. Message Passing - PIPE

Named PIPE (FIFO)

named_pipe_writer.c

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5
6 #define BUFFER_SIZE 100
7 #define PIPENAME "./named_pipe_file"
8
9 int main(void) {
10     char msg[BUFFER_SIZE];
11     int pipefd;
12
13     if ((pipefd = open(PIPENAME, O_WRONLY)) == -1) return -1;
14
15     for (int i=0; i<5; i++) {
16         sprintf(msg, "hello %d\n", i);
17         if (write(pipefd, msg, sizeof(msg)) == -1) return -1;
18
19         sleep(1);
20     }
21     sprintf(msg, "quit");
22     if (write(pipefd, msg, BUFFER_SIZE) == -1) return -1;
23     return 0;
24 }
```

4. Message Passing - PIPE

Named PIPE (FIFO)

```
$ gcc -o named_pipe_reader named_pipe_reader.c
```

```
$ gcc -o named_pipe_writer named_pipe_writer.c
```

```
kimhyungjun > ~/workspace/ipc_project/pipe > ./named_pipe_reader &  
[1] 8877  
kimhyungjun > ~/workspace/ipc_project/pipe > ./named_pipe_writer  
receive msg : hello 0  
  
receive msg : hello 1  
  
receive msg : hello 2  
  
receive msg : hello 3  
  
receive msg : hello 4  
  
[1] + 8877 done      ./named_pipe_reader
```

4. Message Passing - PIPE

Assignment 2 (5점)

Client가 Server에 정수 값을 보내고, Server가 받은 값을 제공해서 Client에 보낼 수 있도록 코드를 작성하시오.
(단, 추가한 코드의 역할과 필요성에 대한 설명을
pdf 파일로 첨부할 것)

실행 방법

```
$ chmod 777 run.sh
```

```
$ ./run.sh
```

출력 결과

```
client : send 12  
  
server : receive 12  
server : send 144  
client : receive 144  
  
server : receive 13  
server : send 169  
client : receive 169  
  
  
client : send 14  
server : receive 14  
server : send 196  
client : receive 196  
  
  
client : send 15  
server : receive 15  
server : send 225  
client : receive 225
```

Section 5

“ Assignment ”

5. Assignment

Assignment 1-1 (10점)

Assignment 1-2 (5점)

Assignment 2 (5점)

Git 주소 : <https://github.com/ledzep0830/ipc-project>

제출폴더명: os_학번 (Ex. os_2020123123)

- 소스 코드 내의 빈칸을 채워서 제출
- 1-2, 2의 경우 보고서를 pdf 파일로 첨부하여 폴더에 포함해서 압축하여 제출
- 기간: 5월 4일 오전 10시 30분까지 (온라인 수업에서 답을 공유할 예정이라 이후에는 받지 않습니다)
- 온라인 수업 : 5월 4일 오전 10시 30분 ~

The End