**강의명: 유닉스 시스템**

**실습 번호: 10**

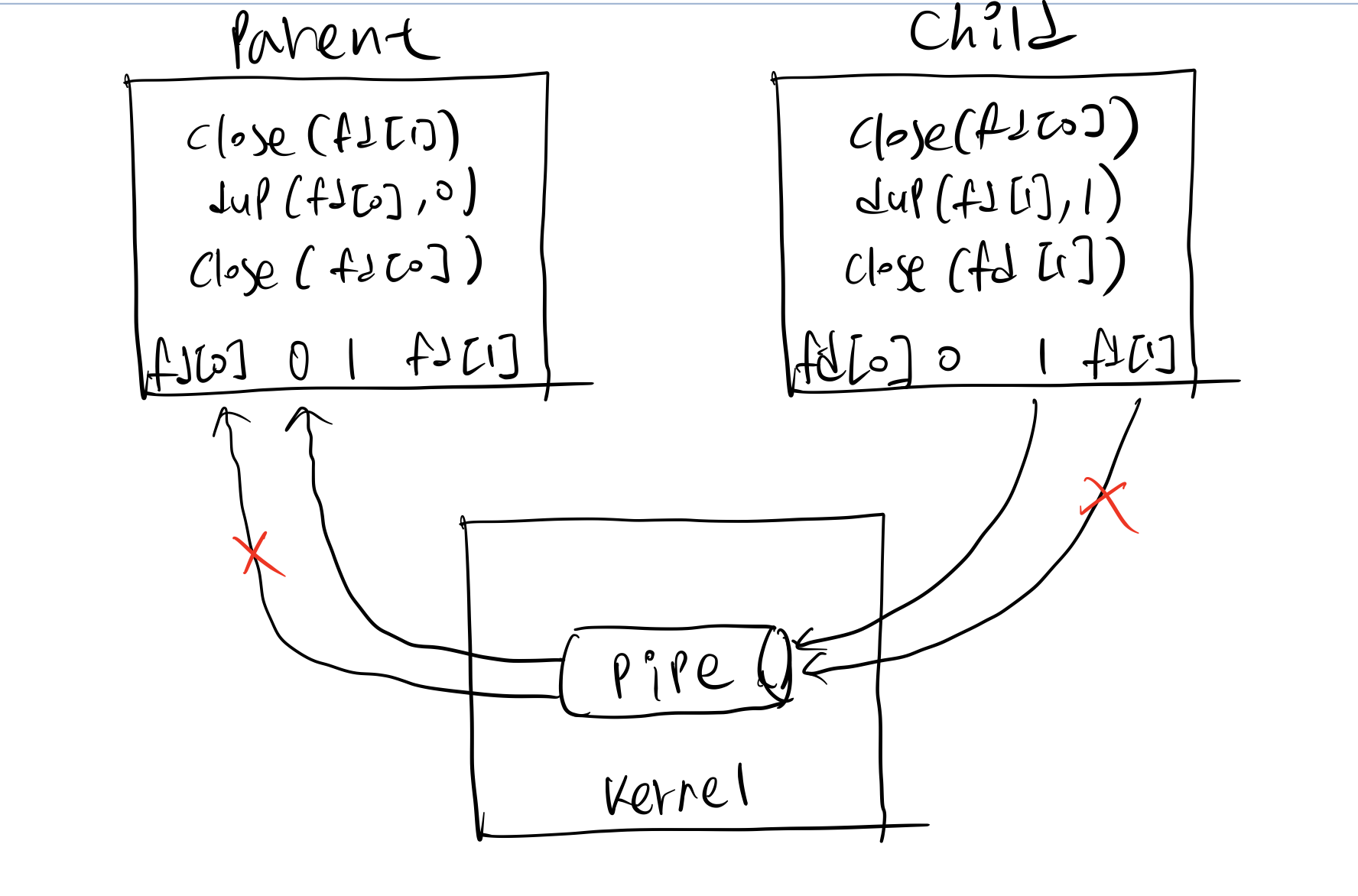
**실습 제목: 프로세스간 통신(Inter-Process Communication)**

**학생 이름: 황귀훈**

**학번: 201710885**

**1. 파이프**

**1.1**



번호 1 파이프는 파이프 IPC를 사용하여 argv[1]에 주어진 명령어 com1과 argv[2]에 주어진 명령어 com2에 대하여 파이프라인 "com1 | com2"를 실행하는 프로그램이다. Parent process는 pipe()함수를 사용하여 fd[1]에 write하고 fd[0]에서 read하는 pipe하나를 만든다. Pipe()호출 후 fork()를 호출하여 child process를 생성하는데 이 때 parent process의 address space를 복사하여 똑같이 fd[1]에 write하고 fd[0]에서 read하는 pipe(parent 와 chile는 같은 pipe를 사용)를 갖게 된다. 그 후 parent process가 fd[1]을 close한 후에, fd[0]을 0에 복제하고, fd[0]을 close하고 child process가 fd[0]을 close한 후에 fd[1]을 1에 복사 후 fd[1]을 close하면 위의 그림과 같이 child process의 standard output은 pipe를 통해 parent에서 standard input이 되어 주어진 명령에 대하여 파이프라인이 실행된다.

**1.2**

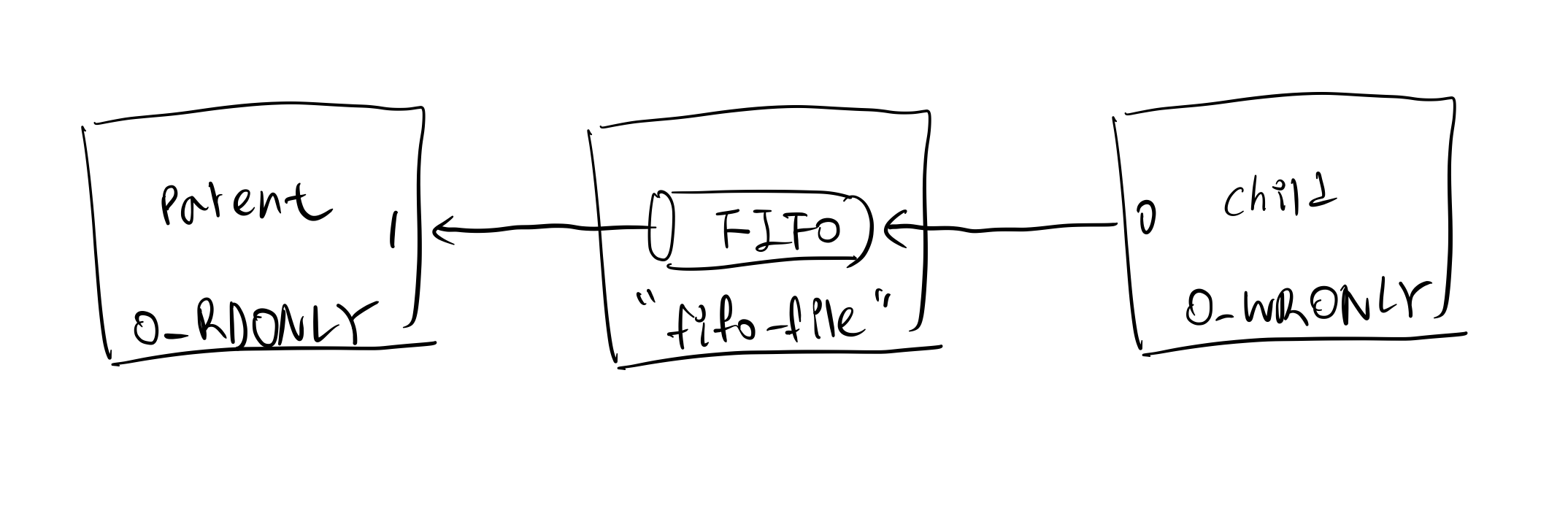
**텍스트이(가) 표시된 사진

자동 생성된 설명**

명령 줄 "./pipe "cat /etc/passwd" wc”와 "cat /etc/passwd | wc"를 수행한 결과는 위와 같다. 위 프로그램은 파이프 IPC를 사용하여 argv[1]에 주어진 명령어 com1과 argv[2]에 주어진 명령어 com2에 대하여 파이프라인 "com1 | com2"를 실행하는 프로그램으로 명령 줄 "./pipe "cat /etc/passwd" wc”와 "cat /etc/passwd | wc"를 수행한 결과가 같음을 알 수 있고, argv[1]인 “cat /etc/passwd” 명령어의 결과가 parent process의 입력이 되어, argv[1]인 “wc” 명령에게 전달된 결과이다. 아래 유닉스 명령어 "cat /etc/group | wc"는 | 앞의 명령의 표준 출력이 뒤의 명령 wc의 표준 입력으로 들어간다.

**2. FIFO**

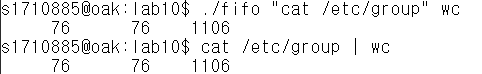
**2.1**



Fifo는 pipe와는 달리 파일 시스템에 pathname을 가지는 file을 실제 생성하기 때문에 “named pipe” 라고 불린다. Pipe는 parent/child process 사이에서만 통신이 가능하지만 fifo는 어떤 두 process 사이에서도 통신이 가능하다. 보통 어떤 프로세스가 그 fifo 파일을 open하여 서로 통신한다. 위 프로그램은 FIFO IPC를 사용하여 argv[1]에 주어진 명령어 com1과 argv[2]에 주어진 명령어 com2에 대 하여 파이프라인 "com1 | com2"를 실행하는 프로그램이다. Mkfifo()함수를 사용하여 “fifo-file”이라는 이름의 mode 0644 fifo file을 생성한다. 그 후 fork()함수를 사용하여 child process를 생성하는데 이 때 address space를 복사하여 같은 fifo file로 두 process가 통신한다.

Child process는 write only로 fifo-file을 열고 dup()함수를 통해 fd1을 표준출력에 복사하고 fd1을 close함수를 통해 닫는다. Parent process는 read only로 fifo-file을 열고 dup()함수를 통해 fd0를 표준 입력으로 복사한 뒤 close()함수를 통해 fd0를 닫는다. 결과적으로 child process의 표준 입력이 Parent process의 표준 출력이 되어 두 프로세스가 통신하게 된다.

**2.2**

****

명령 줄 "./fifo "cat /etc/group" wc”와 "cat /etc/group | wc"를 수행한 결과는 위와 같다. 위 프로그램은 FIFO IPC를 사용하여 argv[1]에 주어진 명령어 com1과 argv[2]에 주어진 명령어 com2에 대 하여 파이프라인 "com1 | com2"를 실행하는 프로그램이다. argv[1]인 “cat /etc/passwd” 명령어의 결과가 parent process의 입력이 되어, argv[1]인 “wc” 명령에게 전달된 결과이다. 아래 유닉스 명령어 "cat /etc/group | wc"는 | 앞의 명령의 표준 출력이 뒤의 명령 wc의 표준 입력으로 들어간다.

**3. 메시지 큐**

**3.1**

메시지 큐는 커널이 관리하는 메시지의 linked list이며 각 메시지 큐는 고유의 ID를 가진다. 메시지 큐를 서비스하는 시스템 함수는 다음과 같다.

msgget(): 새로운 큐를 생성하거나 기존의 큐를 열고 ID를 넘겨준다.

msgctl(): 주어진 ID의 큐를 제어한다.

msgsnd(): 주어진 ID의 큐에 어떤 메시지를 쓴다.

msgrcv(): 주어진 ID의 큐로부터 하나의 메시지를 읽는다.

**struct {**

**long mtype;**

**char mtext[MSGSIZE];**

**} msg\_snd, msg\_rcv;**

이 프로그램에서 사용하는 구조체이다.

**msg\_id = msgget(IPC\_PRIVATE, 0644 | IPC\_CREAT);**

이 프로그램에서 먼저 msgget()을 호출하는데, 첫 번째 인수 key로 IPC\_PRIVATE를, 두 번째 인수 msgflg에 0644 | IPC\_CREAT을 주어 새로운 queue를 생성하고, 새로 생성된 queue의 message id를 int형의 msg\_id에 저장한다.

**msg\_snd.mtype = 1;**

**strcpy(msg\_snd.mtext, "This is a message 1.");**

**msgsnd(msg\_id, (void \*) &msg\_snd, MSGSIZE, 0);**

**printf("msgsnd: %s\n", msg\_snd.mtext);**

이후 msg\_snd 구조체의 mtype을 1로 초기화하고, strcpy() 함수를 사용해 msg\_snd의 mtext에 “This is a message 1.” 이라는 문자열을 복사하고, msgsnd()함수를 사용해 msg\_id의 queue에 msg\_snd에 저장된 MSGSIZE 크기의 message를 쓴다. 따라서 msg\_snd의 mtext에 저장된 “This is a message 1.”가 msg\_id의 queue에 write된다. 그 후 msg\_snd 구조체의 mtext를 printf한다.

**msgrcv(msg\_id, (void \*) &msg\_rcv, MSGSIZE, 0, 0);**

**printf("msgrcv: %s\n", msg\_rcv.mtext);**

msgrcv()함수를 사용해 msg\_id의 queue로부터 MSGSIZE크기의 message를 읽어서 msg\_rcv에 저장한다. 그 후 msg\_rcv 구조체의 mtext를 출력한다.

위 프로그램은 메시지 1, 2, 3을 msgsnd()함수를 통해 보낸 뒤 msgrcv()함수를 통해 받아 오면서 그 결과를 확인하는 프로그램이다.

**3.2**

**테이블이(가) 표시된 사진

자동 생성된 설명**

명령 줄 "./msg"를 수행한 결과는 위와 같다. 처음 세 줄은 msgsnd()를 사용해 write한 문자열을 출력하고 다음 세 줄은 msgrcv()를 통해 read한 문자열을 출력한 결과이다.

**4. 프로세스 메모리 지도**

**4.1**

**테이블이(가) 표시된 사진

자동 생성된 설명**

프로세스 메모리 지도는 위와 같다.

번호 4 프로세스 메모리 지도는 위의 프로세스 메모리 지도의 각 영역의 메모리 주소를 출력하는 프로그램이다.

**char data[DATA\_SIZE] = { 0, 0, };**

**char bss[BSS\_SIZE];**

int main(int argc, char \*argv[])

{

char stack[STACK\_SIZE];

int shm\_id;

char \*heap, \*shm;

**// text area**

**printf("text area: %p\n", main);**

text area는 프로그램의 코드(instruction)를 저장한다. 따라서, main함수의 시작 주소를 출력하면, 이 process의 text area의 시작주소를 출력할 수 있다.

**// data area**

**printf("data area: %p to %p\n", &data[0], &data[DATA\_SIZE - 1]);**

data area는 초기화 된 전역 변수를 저장하는 공간이다. 따라서 **char data[DATA\_SIZE] = { 0, 0, };**

의 시작주소와 끝 주소를 출력하면 data area의 시작되고 끝나는 주소를 알 수 있다.

**// bss area**

**printf("bss area: %p to %p\n", &bss[0], &bss[BSS\_SIZE - 1]);**

bss는 초기화 안된 전역 변수를 저장하는 공간이다. 따라서 **char bss[BSS\_SIZE];** 의 시작과 끝 index를 출력하면, bss area의 시작되고 끝나는 주소를 알 수 있다.

**// heap area**

**if ((heap = malloc(HEAP\_SIZE)) == NULL) {**

**perror("malloc");**

**exit(1);**

**}**

**printf("heap area: %p to %p\n", heap, heap + HEAP\_SIZE - 1);**

heap은 malloc한 메모리를 저장하는 공간이다. 따라서 malloc()을 사용하여 heap 변수에 저장공간을 할당하고 heap과 heap + HEAP\_SIZE – 1을 출력하여 heap area의 주소를 출력할 수 있다.

**// shared memory**

**if ((shm\_id = shmget(IPC\_PRIVATE, SHM\_SIZE, 0600)) == -1) {**

**perror("shmget");**

**exit(1);**

**}**

**if ((shm = shmat(shm\_id, 0, 0)) ==(void \*) -1) {**

**perror("shmat");**

**exit(1);**

**}**

**printf("shm area: %p to %p\n", shm, shm + SHM\_SIZE - 1);**

**if (shmctl(shm\_id, IPC\_RMID, 0) == -1) {perror("shmctl");**

**exit(1);**

**};**

먼저 shmget()을 호출하여 SHM\_SIZE크기의 새로운 shared memory를 생성했다. 그 후, shmat()을 호출하여 주어진 shm\_id주소의 shared memory를 주어진 주소에 attach한다. 두 번째 argument가 0이기 때문에, 커널이 찾은 첫 번째 가능한 주소를 attach하고, attach된 shared memory에 대한 주소를 return한다. 따라서 이 return값과 이 return값에 return값에 SHM\_SIZE-1만큼을 더한 값을 출력하면 공유 메모리의 주소를 알 수 있다. 이후 shmctl()함수에 두 번째 인수로 IPC\_RMID를 주어 shared memory를 제거한다.

**// stack area**

**printf("stack area: %p to %p\n", &stack[0], &stack[STACK\_SIZE - 1]);**

stack area는 지역 변수를 저장한다. 따라서 지역 변수 char stack[STACK\_SIZE];의 첫 번째 인수의 주소와 마지막 인수의 주소를 출력하면, stack area의 시작 주소와 끝 주소를 출력한다.

**// args area**

**printf("args area: %p\n", argv[0]);**

**exit(0);**

**}**

arguments area는 명령 줄 인수, 환경 변수가 저장되기 때문에 argv[0]를 출력하면, arguments area의 시작 주소를 알 수 있다.

**4.2**

텍스트이(가) 표시된 사진

자동 생성된 설명

명령 줄 "./map"을 수행한 결과 process memory map의 각 영역의 메모리 주소를 보여준다.