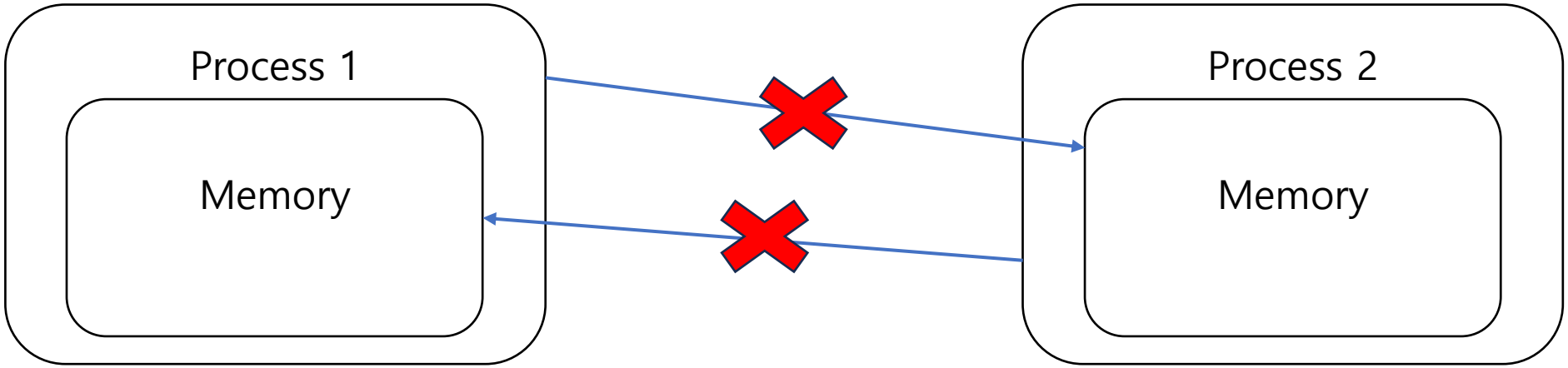
Process 간 통신

Data 전송 → Memory 공유

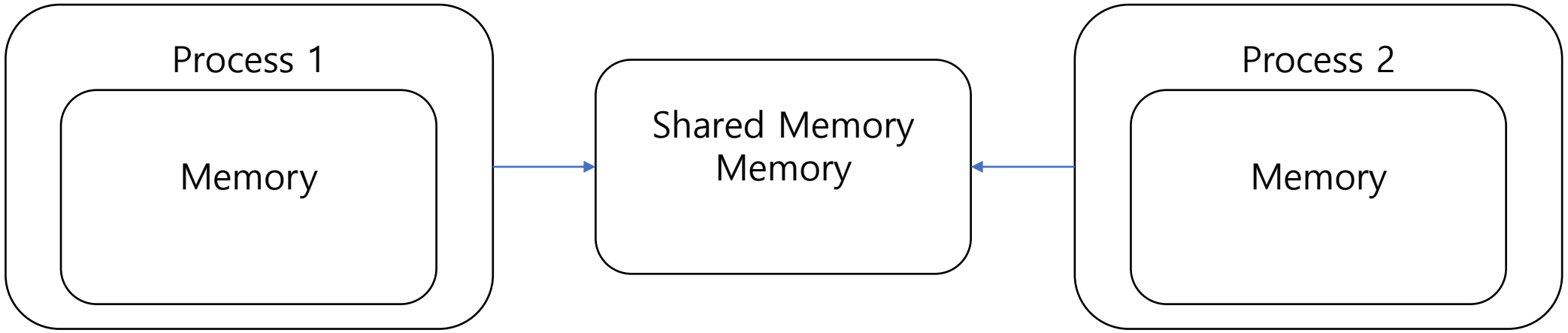
문제

상대 Process의 Memory에 접근 불가능

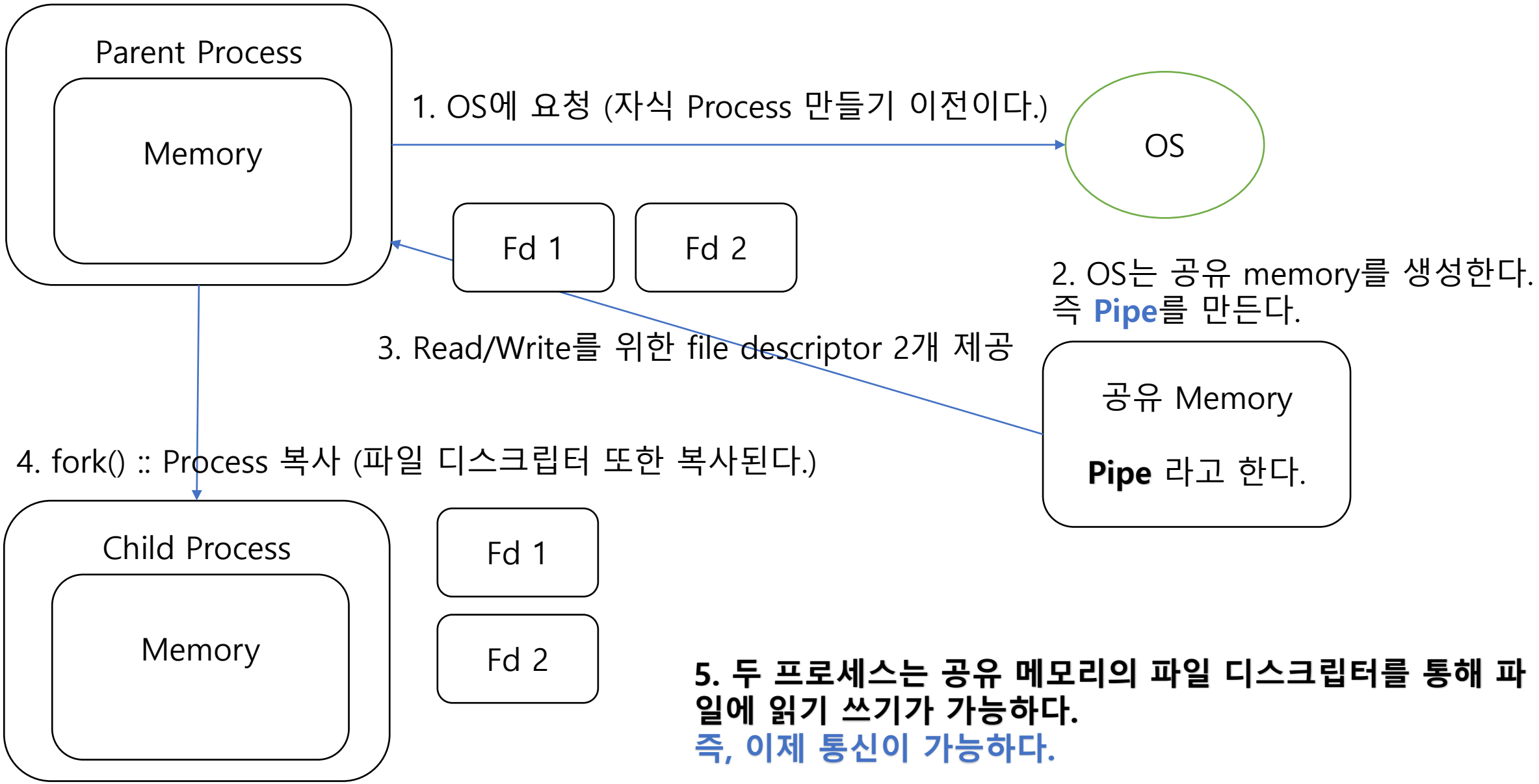


해결

공유 메모리 생성



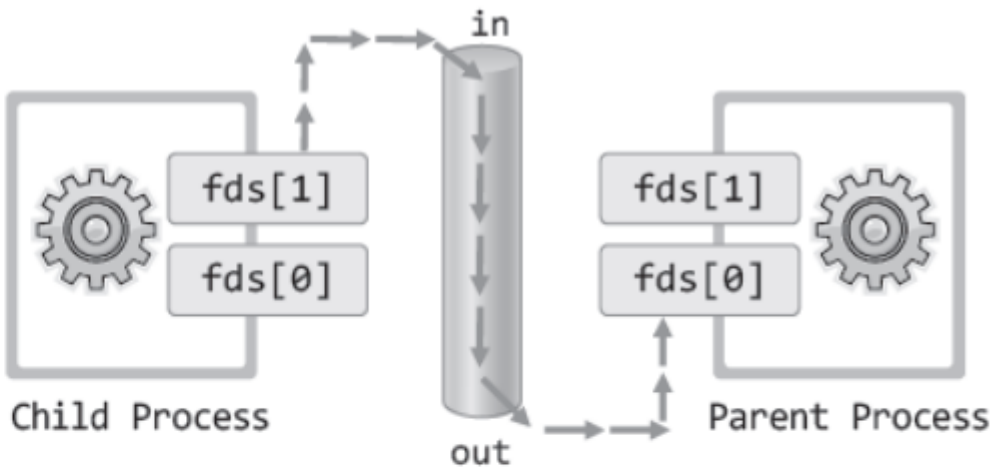
Process 간 통신



5. 두 프로세스는 공유 메모리의 파일 디스크립터를 통해 파일에 읽기 쓰기가 가능하다.
즉, 이제 통신이 가능하다.

Process 간 통신

```
int main(int argc, char *argv[])
{
    int fds[2];
    char str[]="Who are you?";
    char buf[BUF_SIZE];
    pid_t pid;
    pipe(fds);
    pid=fork();
    if(pid==0)
    {
        write(fds[1], str, sizeof(str));
    }
    else
    {
        read(fds[0], buf, BUF_SIZE);
        puts(buf);
    }
    return 0;
}
```



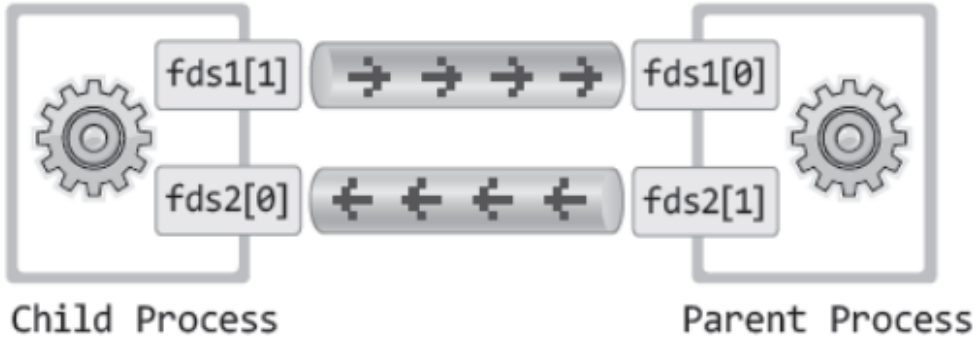
핵심은 파이프의 생성과 디스크립터의 복사에 있다!

실행결과

```
root@my_linux:/tcpip# gcc pipe1.c -o pipe1
root@my_linux:/tcpip# ./pipe1
Who are you?
```

Process 간 통신 - 양방향 통신 주의사항

```
int main(int argc, char *argv[])
{
    int fds1[2], fds2[2];
    char str1[]="Who are you?";
    char str2[]="Thank you for your message";
    char buf[BUF_SIZE];
    pid_t pid;
    pipe(fds1), pipe(fds2);
    pid=fork();
    if(pid==0)
    {
        write(fds1[1], str1, sizeof(str1));
        read(fds2[0], buf, BUF_SIZE);
        printf("Child proc output: %s \n", buf);
    }
    else
    {
         read(fds1[0], buf, BUF_SIZE);
         printf("Parent proc output: %s \n", buf);
         write(fds2[1], str2, sizeof(str2));
         sleep(3);
    }
    return 0;
}
```



양방향 통신을 위해서는 이렇듯 두 개의 파이프를 생성해야 한다. 그래야 입출력의 타이밍에 따라서 데이터의 흐름이 영향을 받지 않는다.

그렇지 않은 경우, R/W 타이밍을 신경 써야 한다.

실행결과

```
root@my_linux:/tcpip# gcc pipe3.c -o pipe3
root@my_linux:/tcpip# ./pipe3
Parent proc output: Who are you?
Child proc output: Thank you for your message
```