2022-Spring
운영체제

학번	2015150220
이름	이윤수

[OS Practice 2. Inter Process Communication]

Assignment 2. pipe

Question :

추가한 코드의 역할과 필요성, 작동 방식

Explanation

1. server.c

```
12  int main(void) {
13      char receive_msg[BUFFER_SIZE], send_msg[BUFFER_SIZE];
14      int receive_fd, send_fd;
15      int value;
16
17      /*---------------------------------------*/
18      /* TODO 3 : make pipes and init fds      */
19      /* (1) make NP_RECEIVE pipe              */
20      /* (2) make NP_SEND pipe                 */
21      /* (3) init receive_fd and send_fd       */
22
23      //(1) make NP_RECEIVE pipe
24      if(access(NP_RECEIVE, F_OK)==0){
25          unlink(NP_RECEIVE);
26      }
27      if(mkfifo(NP_RECEIVE, 0666) == -1) return -1;
28
29      //(2) make NP_SEND pipe
30      if(access(NP_SEND, F_OK)==0){
31          unlink(NP_SEND);
32      }
33      if(mkfifo(NP_SEND, 0666) == -1) return -1;
34
35      //(3) init receive_fd and send_fd
36      if((receive_fd = open(NP_RECEIVE, O_RDWR)) == -1) return -1;
37      if((send_fd = open(NP_SEND, O_RDWR)) == -1) return -1;
38      /* TODO 3 : END                          */
39      /*---------------------------------------*/
40
```

[line 23 ~ line 37]

(1) access

Access() 함수를 통해서 NP_RECEIVE 파이프와 NP_SEND 파이프가 이미 존재하는 지를

확인했습니다. 해당 함수는 이미 존재하는 파이프에 대해서 0을 리턴합니다. 만약 파이프가 이미 존재한다면 기존의 파이프에 대한 연결을 해제합니다.

(2) mkfifo

다음으로 mkfifo 함수를 통해서 named_pipe를 생성했습니다. 두 개의 파이프를 생성했고 각각의 Pipe name이 앞서 언급한 NP_RECEIVE, NP_SEND 입니다. 두 파이프 이름은 모두 파일경로에 대한 매크로에 해당합니다. 생성에 실패하는 경우 -1이 반환됩니다.

(3) open

만들어진 파이프를 open 하는 함수입니다. 이때 파일 권한을 설정할 수 있는데 이후 다룰 client.c의 파이프들과는 달리 server의 파이프들은 O_RDWR로 설정해야 합니다. 읽고 쓰기 권한을 모두 부여하는 것입니다. RECEIVE 파이프는 RDONLY로, SEND 파이프는 WRONLY로 설정해도 된다고 생각하기 쉬우나, 이 경우 처음 server.c를 먼저 실행할 때, 상대편 파이프에 읽거나 쓸 프로세스가 존재하지 않아 곧바로 0이 리턴되어 프로세스가 종료됩니다. 따라서 두 파이프 모두 O_RDWR로 설정해 파이프 반대편에 프로세스가 존재하지 않더라도 계속 대기하도록 만들어 줍니다. 이때 리턴되는 receive_fd와 send_fd는 integer 값으로 이후 read, write 함수에서 사용됩니다. 단 open이 실패할 경우 -1을 반환합니다.

```
41      while (1) {
42          /*--------------------------------------*/
43          /* TODO 4 : read msg                    */
44
45          if(read(receive_fd, receive_msg, sizeof(receive_msg)) == -1) return -1;
46
47          /* TODO 4 : END                         */
48          /*--------------------------------------*/
49
50          printf("server : receive %s\n", receive_msg);
51
52          value = atoi(receive_msg);
53
54          sprintf(send_msg, "%d", value*value);
55          printf("server : send %s\n", send_msg);
56
57          /*--------------------------------------*/
58          /* TODO 5 : write msg                   */
59
60          if(write(send_fd, send_msg, sizeof(send_msg)) == -1) return -1;
61
62          /* TODO 5 : END                         */
63          /*--------------------------------------*/
64      }
65      return 0;
66  }
```

[line 45, line 60]

(4) read : 파이프로 연결된 상대 프로세스가 write한 메시지를 읽어옵니다. 읽는 데 실패할 경우 -1이 반환됩니다.
(5) write : 파이프로 연결된 상대 프로세스에게 메시지를 전송합니다. 마찬가지로 전송에 실패하면 -1이 반환됩니다.

2. client.c

```c
int main(void) {
    char receive_msg[BUFFER_SIZE], send_msg[BUFFER_SIZE];
    int receive_fd, send_fd;
    /*----------------------------------------*/
    /* TODO 1 : init receive_fd and send_fd  */

    if((receive_fd = open(NP_RECEIVE, O_RDONLY)) == -1) return -1;
    if((send_fd = open(NP_SEND, O_WRONLY)) == -1) return -1;

    /* TODO 1 : END                          */
    /*----------------------------------------*/

    for (int i=12; i<16; i++) {
        printf("client : send %d\n", i);
        sprintf(send_msg, "%d", i);
        /*----------------------------------------*/
        /* TODO 2 : send msg and receive msg     */

        if(write(send_fd, send_msg, sizeof(send_msg)) == -1) return -1;
        if(read(receive_fd, receive_msg, sizeof(receive_msg)) == -1) return -1;
        /* TODO 2 : END                          */
        /*----------------------------------------*/

        printf("client : receive %s\n\n", receive_msg);

        usleep(500*1000);
    }
    return 0;
}
```

(1) open

Client 쪽에서도 open 시도가 있어야만 동기화가 되어 프로세스 간 통신이 가능해집니다. 따라서 서버에 존재하는 두 파이프(NP_RECEIVE, NP_SEND)에 대해 open 을 시도합니다. 이때 주의할 점은 서버에서의 receive_fd, send_fd 가 클라이언트의 그것과 반대로 연결되어야 합니다. 이를 위해 client.c 에서 두 파이프 이름은 server.c 와 반대의 파일경로로 define 되어 있습니다.

또한 서버와는 다르게 클라이언트는 목적에 따라 한 가지 기능만 수행하게 파이프를 오픈할 수 있습니다. 예를 들어 receive_fd 는 read-only, send_fd 는 write-only 로 설정할 수 있습니다. 이때

만약 서버가 종료된다면, 파이프를 통해 쓰거나 읽어줄 상대방 프로세스를 잃게 되어, client 의 프로세스도 자동으로 종료됩니다.

(2) write, read

클라이언트 측에서 먼저 파이프를 통해 서버에 12 부터 15 까지의 숫자를 차례로 전송합니다. 이때 각 숫자에 대해서 서버는 해당 숫자를 receive 했다는 메시지를 출력한 뒤, 그것의 제곱을 다시 클라이언트에게 전송합니다. 이후 클라이언트는 서버로부터 자신이 보낸 숫자의 제곱을 읽어와 그것을 출력합니다. 아래의 Result 에 실제 실행 결과를 첨부했습니다..

Result