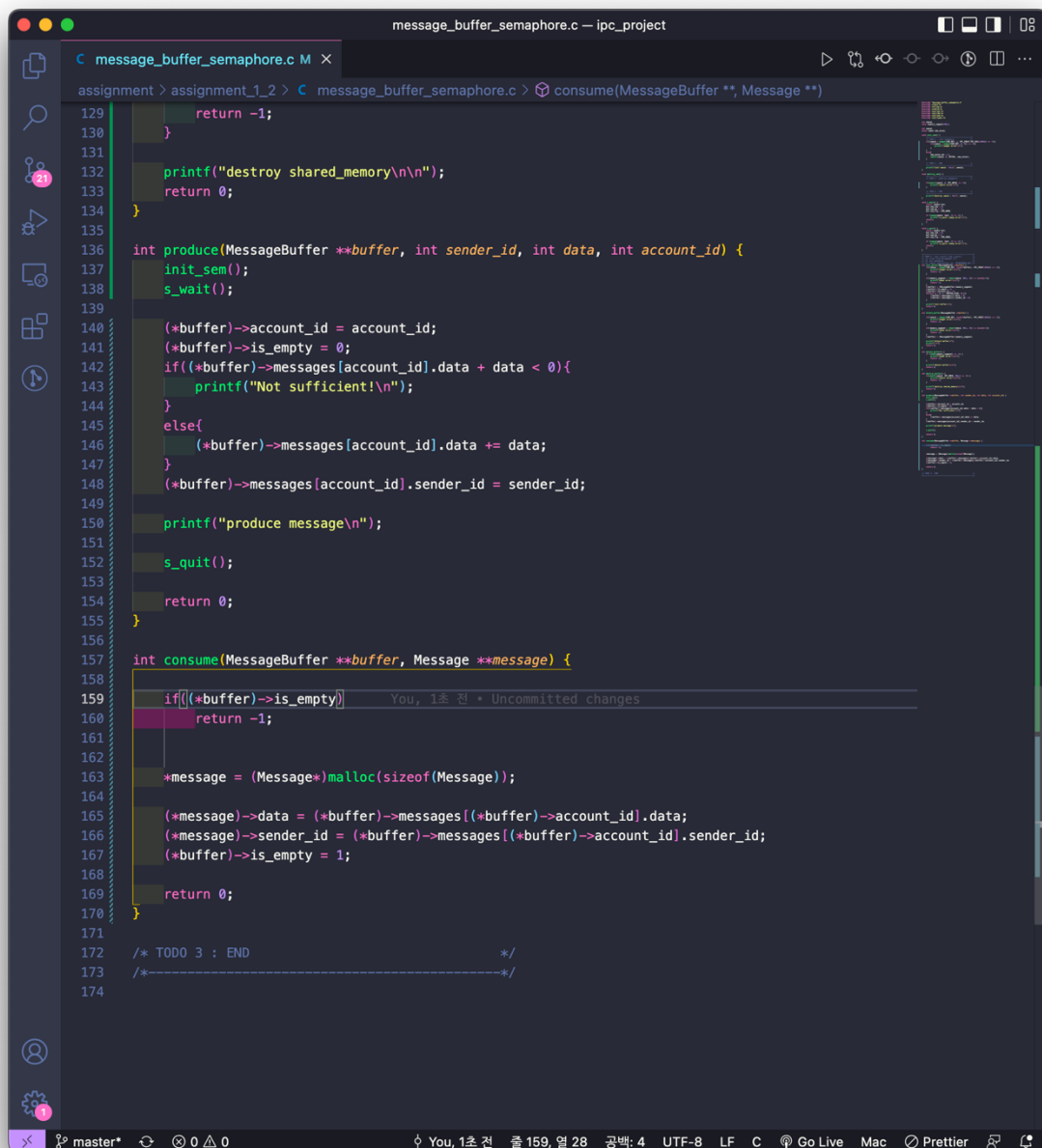


Assignment 1-2



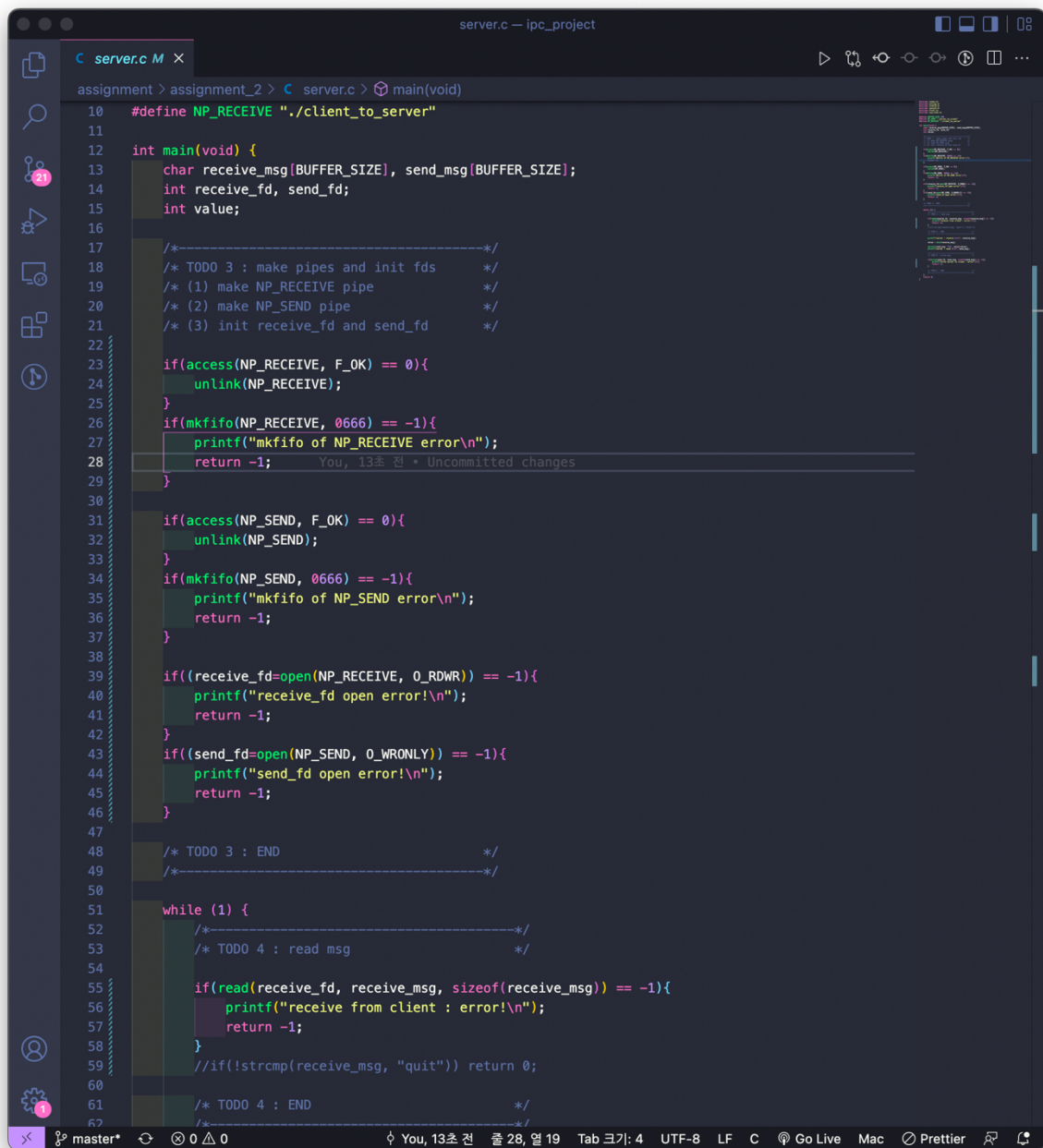
```
message_buffer_semaphore.c — ipc_project

C message_buffer_semaphore.c M X
assignment > assignment_1_2 > C message_buffer_semaphore.c > consume(MessageBuffer **, Message **)

129     return -1;
130 }
131
132 printf("destroy shared_memory\n\n");
133 return 0;
134 }
135
136 int produce(MessageBuffer **buffer, int sender_id, int data, int account_id) {
137     init_sem();
138     s_wait();
139
140     (*buffer)->account_id = account_id;
141     (*buffer)->is_empty = 0;
142     if((*buffer)->messages[account_id].data + data < 0){
143         printf("Not sufficient!\n");
144     }
145     else{
146         (*buffer)->messages[account_id].data += data;
147     }
148     (*buffer)->messages[account_id].sender_id = sender_id;
149
150     printf("produce message\n");
151
152     s_quit();
153
154     return 0;
155 }
156
157 int consume(MessageBuffer **buffer, Message **message) {
158
159     if(!(*buffer)->is_empty) You, 1초 전 + Uncommitted changes
160         return -1;
161
162     *message = (Message*)malloc(sizeof(Message));
163
164     (*message)->data = (*buffer)->messages[*buffer->account_id].data;
165     (*message)->sender_id = (*buffer)->messages[*buffer->account_id].sender_id;
166     (*buffer)->is_empty = 1;
167
168     return 0;
169 }
170
171
172 /* TODO 3 : END */
173 /*-----*/
174
```

위의 작업이 원자적으로 수행되기 위해서는 produce 작업을 할 때 여러 프로세스가 동시에 접근하지 못하도록 해야한다. 이를 위해 produce() 함수의 시작에 s_wait()을 걸어주어 다른 프로세스의 접근을 차단하고 produce() 함수의 마지막에 s_quit()으로 풀어주면 그 때 다른 프로세스가 produce() 함수를 사용할 수 있게 된다. 이렇게 하면 produce 작업을 원자적으로 실행할 수 있다.

Assignment 2



```
server.c — ipc_project

C server.c M x
assignment > assignment_2 > C server.c > main(void)
10 #define NP_RECEIVE "./client_to_server"
11
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     if(access(NP_RECEIVE, F_OK) == 0){
24         unlink(NP_RECEIVE);
25     }
26     if(mkfifo(NP_RECEIVE, 0666) == -1){
27         printf("mkfifo of NP_RECEIVE error\n");
28         return -1;
29     }
30
31     if(access(NP_SEND, F_OK) == 0){
32         unlink(NP_SEND);
33     }
34     if(mkfifo(NP_SEND, 0666) == -1){
35         printf("mkfifo of NP_SEND error\n");
36         return -1;
37     }
38
39     if((receive_fd=open(NP_RECEIVE, O_RDWR)) == -1){
40         printf("receive_fd open error!\n");
41         return -1;
42     }
43     if((send_fd=open(NP_SEND, O_WRONLY)) == -1){
44         printf("send_fd open error!\n");
45         return -1;
46     }
47
48     /* TODO 3 : END */
49     /*-----*/
50
51     while (1) {
52         /*-----*/
53         /* TODO 4 : read msg */
54
55         if(read(receive_fd, receive_msg, sizeof(receive_msg)) == -1){
56             printf("receive from client : error!\n");
57             return -1;
58         }
59         //if(!strcmp(receive_msg, "quit")) return 0;
60
61         /* TODO 4 : END */
62     }
63 }
```

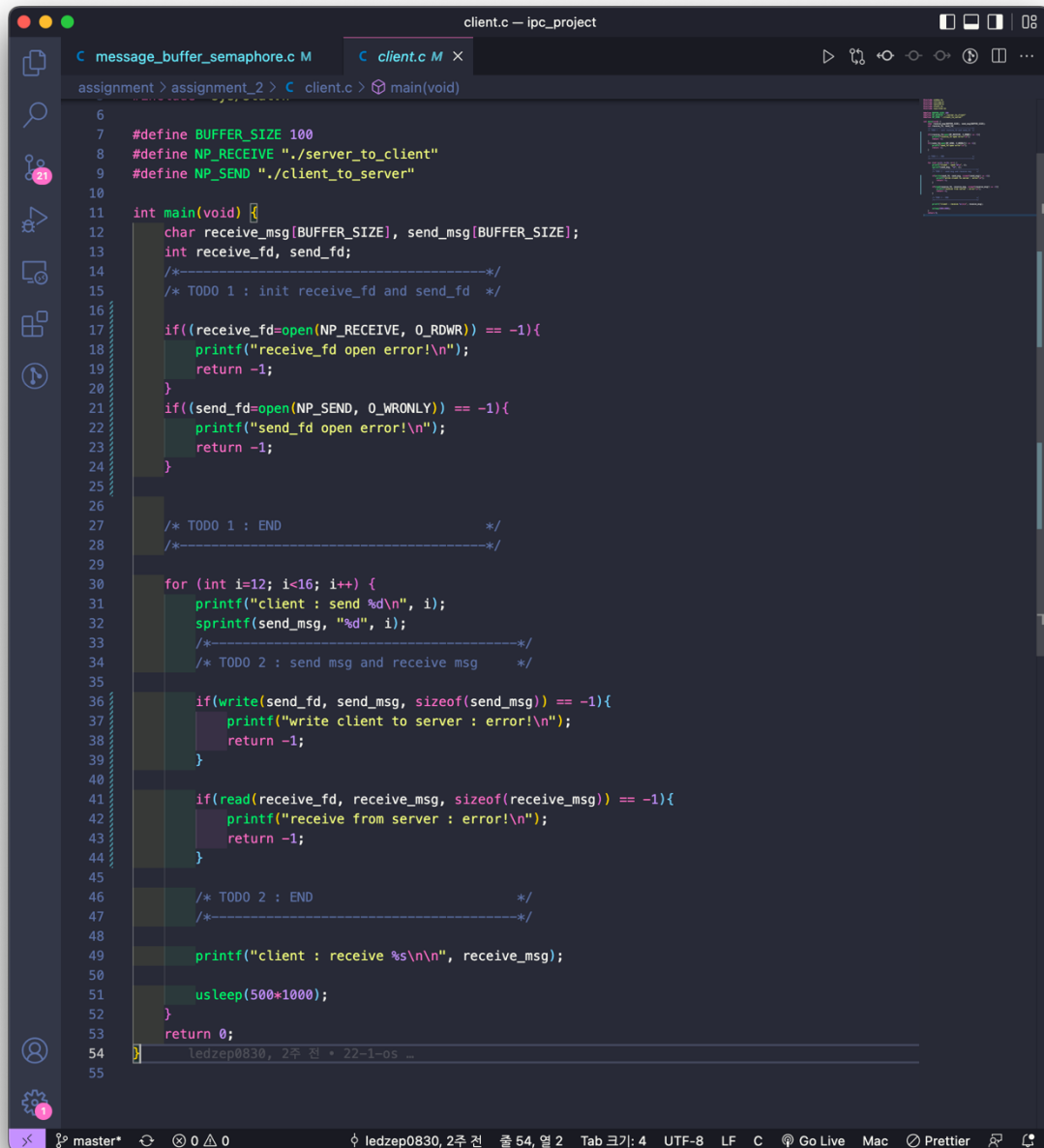
server.c — ipc_project

server.c M X

assignment > assignment_2 > C server.c > main(void)

```
33     }
34     if(mkfifo(NP_SEND, 0666) == -1){
35         printf("mkfifo of NP_SEND error\n");
36         return -1;
37     }
38
39     if((receive_fd=open(NP_RECEIVE, 0_RDWR) == -1){
40         printf("receive_fd open error!\n");
41         return -1;
42     }
43     if((send_fd=open(NP_SEND, 0_WRONLY) == -1){
44         printf("send_fd open error!\n");
45         return -1;
46     }
47
48     /* TODO 3 : END */
49     /*-----*/
50
51     while (1) {
52         /*-----*/
53         /* TODO 4 : read msg */
54
55         if(read(receive_fd, receive_msg, sizeof(receive_msg)) == -1){
56             printf("receive from client : error!\n");
57             return -1;
58         }
59         //if(!strcmp(receive_msg, "quit")) return 0;
60
61         /* TODO 4 : END */
62         /*-----*/
63
64         printf("server : receive %s\n", receive_msg);
65
66         value = atoi(receive_msg);
67
68         sprintf(send_msg, "%d", value*value);
69         printf("server : send %s\n", send_msg);
70
71         /*-----*/
72         /* TODO 5 : write msg */
73
74         if(write(send_fd, send_msg, sizeof(send_msg)) == -1){
75             printf("write server to client : error!\n");
76             return -1;
77         }
78
79         /* TODO 5 : END */
80         /*-----*/
81     }
82     return 0;
83 }
84
```

master* 0 0 You, 13초 전 클 28, 열 19 Tab 크기: 4 UTF-8 LF C Go Live Mac Prettier



```
client.c — ipc_project
C message_buffer_semaphore.c M C client.c M
assignment > assignment_2 > C client.c > main(void)

6
7 #define BUFFER_SIZE 100
8 #define NP_RECEIVE "./server_to_client"
9 #define NP_SEND "./client_to_server"
10
11 int main(void) {
12     char receive_msg[BUFFER_SIZE], send_msg[BUFFER_SIZE];
13     int receive_fd, send_fd;
14     /*-----*/
15     /* TODO 1 : init receive_fd and send_fd */
16
17     if((receive_fd=open(NP_RECEIVE, O_RDWR)) == -1){
18         printf("receive_fd open error!\n");
19         return -1;
20     }
21     if((send_fd=open(NP_SEND, O_WRONLY)) == -1){
22         printf("send_fd open error!\n");
23         return -1;
24     }
25
26
27     /* TODO 1 : END */
28     /*-----*/
29
30     for (int i=12; i<16; i++) {
31         printf("client : send %d\n", i);
32         sprintf(send_msg, "%d", i);
33         /*-----*/
34         /* TODO 2 : send msg and receive msg */
35
36         if(write(send_fd, send_msg, sizeof(send_msg)) == -1){
37             printf("write client to server : error!\n");
38             return -1;
39         }
40
41         if(read(receive_fd, receive_msg, sizeof(receive_msg)) == -1){
42             printf("receive from server : error!\n");
43             return -1;
44         }
45
46         /* TODO 2 : END */
47         /*-----*/
48
49         printf("client : receive %s\n\n", receive_msg);
50
51         usleep(500*1000);
52     }
53     return 0;
54 }
55
56 ledzep0830, 2주 전 · 22-1-os ...
```

우선 server에서 mkpipo()를 이용해 Named PIPE를 생성하고 open()를 이용해 open을 한다. Client 측에서도 PIPE를 open 하면 동기화되어 통신이 가능하게 된다.

Named PIPE는 파일 경로를 ID 삼아 통신하기 때문에 client의 receive와 server의 send가 ./server_to_client로 경로가 같고 client의 send와 server의 receive가 ./client_to_server로 경로가 같다. Client에서 ./client_to_server 경로에 write()을 이용해 메시지를 작성하게 되면 server에서도 같은 경로에서 read()를 통해 메시지를 읽어 값을 저장한다. 그 후 server에서 값을 계산하여 ./server_to_client의 경로에 write()를 이용해 메시지를 작성하고 이를 client가 같은 경로에서 read()를 통해 읽어들이어 print 하게 된다.

이에 따른 실행 결과는 다음과 같다.

```
~/.../운영 체제 (Prof.유현창)/과제/실습2/ipc_project/assignment_2 master !6 ?7 ./run.sh
~/.../운영 체제 (Prof.유현창)/과제/실습2/ipc_project/assignment_2 master !8 ?11 client : send 12
server : receive 12
server : send 144
client : receive 144

client : send 13
server : receive 13
server : send 169
client : receive 169

client : send 14
server : receive 14
server : send 196
client : receive 196
~/.../운영 체제 (Prof.유현창)/과제/실습2/ipc_project/assignment_2 master !8 ?11
```