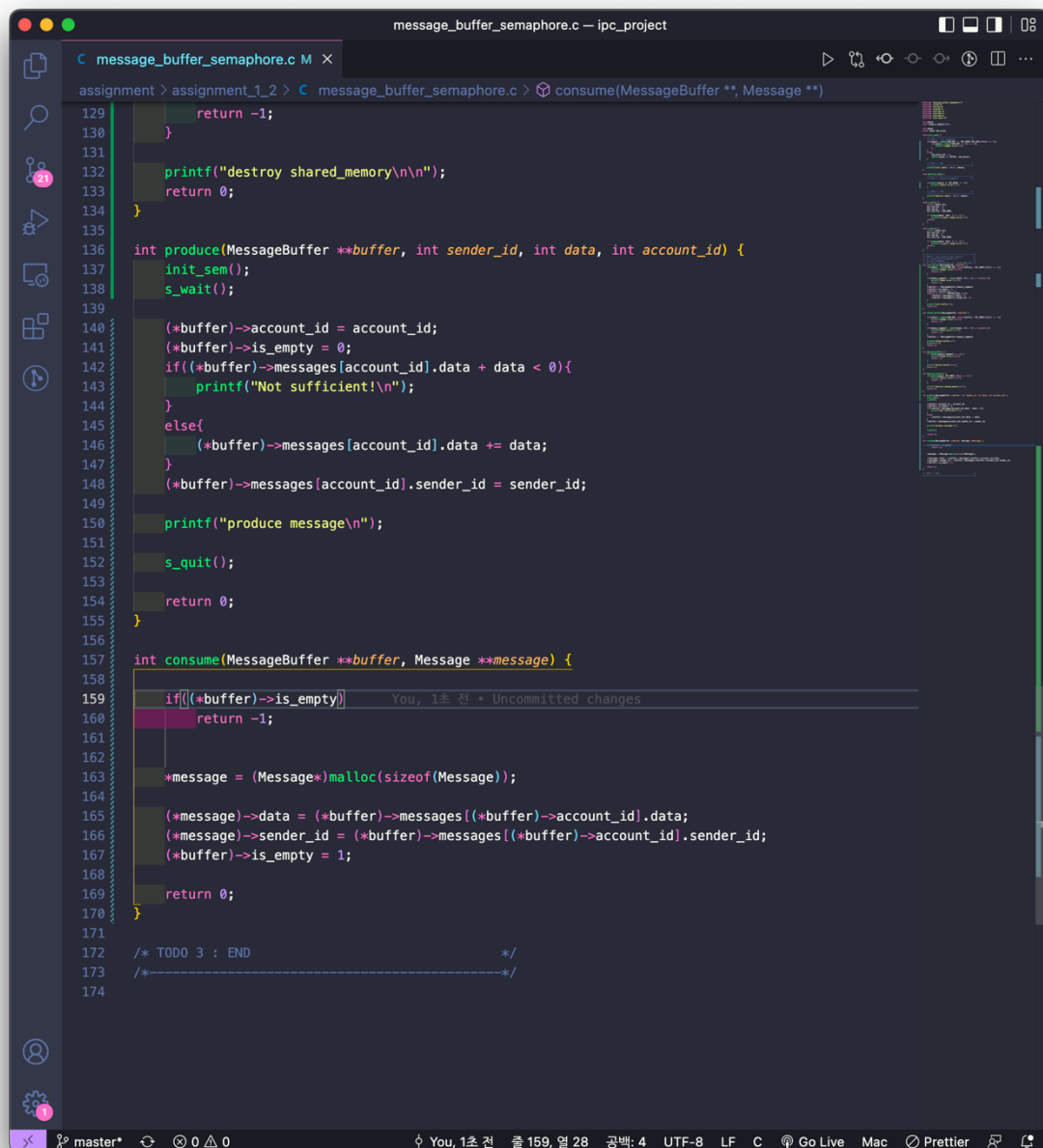


Assignment 1-2



```
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 /* TODO 3 : END */
172 /*-----*/
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.