OS 실습과제 5

2021320308 박한준

assignment_1_2 (semaphore)

assignment > assignment_1_2 > C message_buffer_semaphore.c

```
120 int produce(MessageBuffer **buffer, int sender_id, int data, int account_id) {
121
122     /*---------------------------------------*/
123     /* TODO 3 : produce message              */
124     s_wait();
125     (*buffer)->messages[account_id].sender_id = sender_id;
126     (*buffer)->messages[account_id].data += data;
127     (*buffer)->account_id = account_id;
128     (*buffer)->is_empty = 0;
129     s_quit();
130     /* TODO 3 : END                          */
131     /*---------------------------------------*/
132
133     printf("produce message\n");
134
135     return 0;
136 }
137
138 int consume(MessageBuffer **buffer, Message **message) {
139
140     /*---------------------------------------*/
141     /* TODO 4 : consume message              */
142     s_wait();
143     if((*buffer)->is_empty == 1){
144         s_quit();
145         return -1;
146     }
147     *message = &((*buffer)->messages[(*buffer)->account_id]);
148     (*buffer)->is_empty = 1;
149     s_quit();
150     /* TODO 4 : END                          */
151     /*---------------------------------------*/
152     return 0;
```

- **line 124**: produce 함수에서, 공유 메모리에 해당하는 *buffer에 접근하기 전에 `s_wait()`을 써서 세마포어로 락을 걸어주어 원자성을 보장한다.
- **line 129**: produce 함수에서, 공유 메모리 작업이 끝난 뒤 `s_quit()`을 써서 교착이 일어나지 않도록락을 release한다.
- **line 142**: consume 함수에서, 공유 메모리에 접근하기 전에 `s_wait()`을 써서 원자성을 보장한다.
- **line 144**: 만약 메모리가 버퍼가 비어있어 consume할 게 없을 경우, 작업을 끝내고 리턴하게 되므로 `s_quit()`을 써서 교착이 일어나지 않도록 release한다.

- **line 149**: 마찬가지로 작업이 끝난 뒤 `s_quit()`을 써서 락을 release 한다.

→ 위 코드들로 공유 메모리에 동시 접근 시 발생하는 race condition 문제를 해결할 수 있다.

assignment_1_2에 대한 test.sh

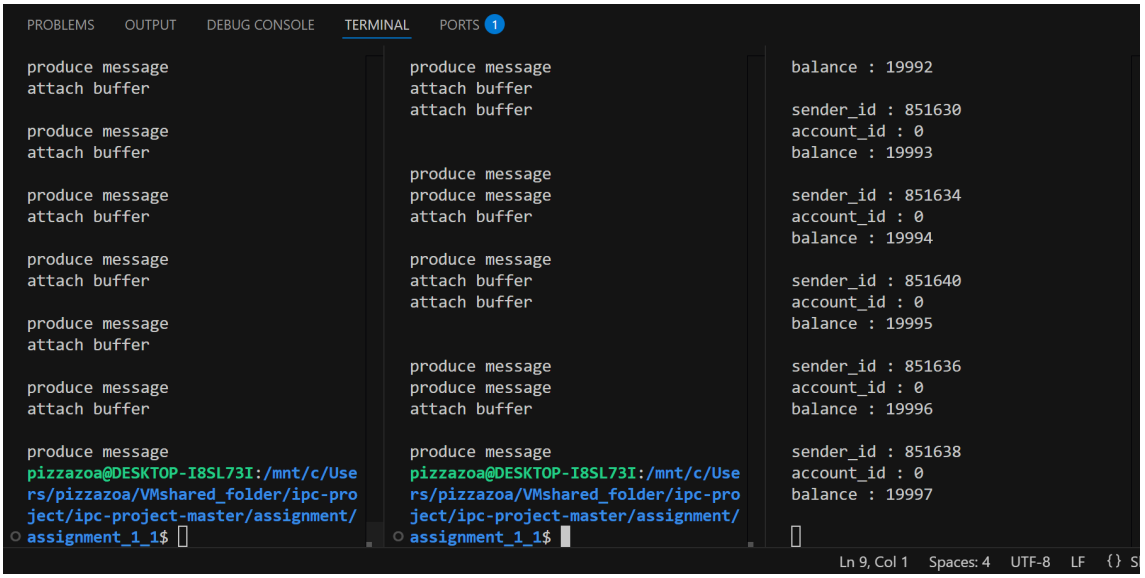
```
#!/bin/bash

for j in {1..100}; do
    for i in {1..100}; do
        ./producer 0 1 &
    done
done

wait
```

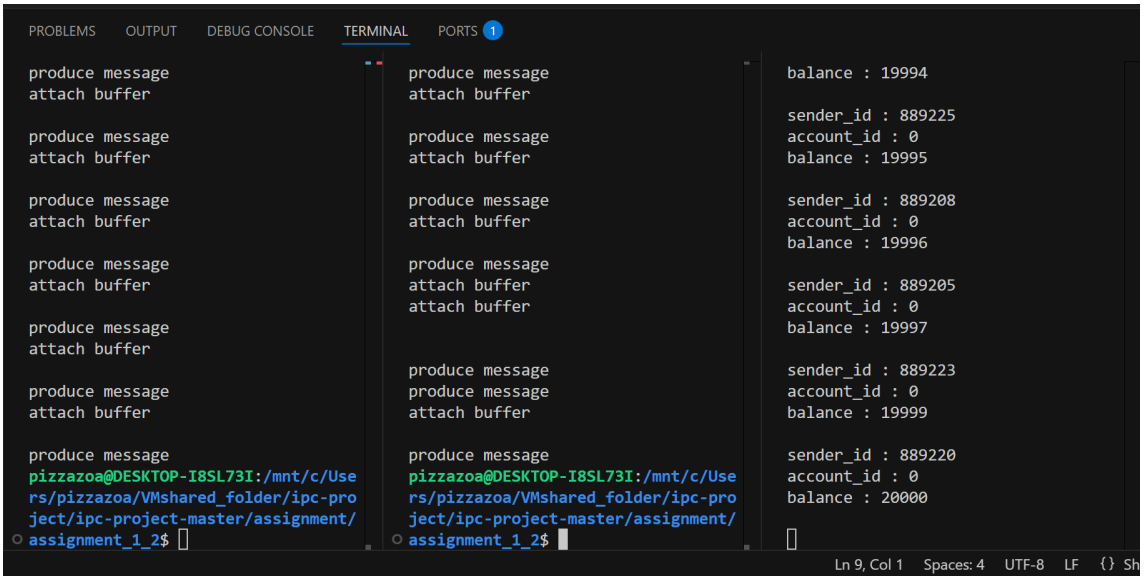
- 터미널 3개를 띄워서, 하나는 ./consumer를 실행하여 consumer를 켠다.
- 나머지 2개의 터미널에서 ./test.sh를 동시에 실행한다.
  동시에 producer가 실행될 때 발생할 수 있는 캐시 비일관성 문제에 대해 1_1과 1_2에서의 결과를 비교한다.

1_1에서의 결과 (세마포어 없을 때)



- 예상 기대값 20000과 불일치하는 문제가 발생했다.

1_2에서의 결과 (세마포어 있을 때)



- 세마포어로 임계 구역을 보호한 덕에 공유 메모리에 대한 연산이 원자적으로 수행되어 예상 기대값 20000과 일치하는 모습을 확인할 수 있다. (중간에 값(19998)이 비는 문제는 producer의 연산이 consumer보다 빠르게 두 번 수행되었기 때문으로 해석할 수 있다)

assignment_2 (named_pipe)

client.c

assignment > assignment_2 > C client.c

```c
#define NP_SEND "./client_to_server"

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  */
    sleep(1);
    if((send_fd = open(NP_SEND, O_WRONLY)) == -1) return -1;
    if((receive_fd = open(NP_RECEIVE, O_RDWR)) == -1) return -1;
    /* TODO 1 : END                          */
    /*---------------------------------------*/

    for (int i=1; i<10; 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;
        sleep(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;
}
```

- **line 16**: `sleep(1)`

  server.c가 파이프를 생성하는 역할을 맡고 있으므로, 안전하게 생성 후 연결할 수 있도록 잠시 대기한다.

- **line 17, 18**:

  `if((send_fd = open(NP_SEND, O_WRONLY)) == -1) return -1;`

  `if((receive_fd = open(NP_RECEIVE, O_RDWR)) == -1) return -1;`

  각각 "`client_to_server`" 파이프와 "`server_to_client`" 파이프를 open하여 연결한다. 이때, 양쪽에서 연결될 때까지 대기하게 되므로 server.c에서의 open 순서와 같은 순서로 파이프를 open한다. 성공적으로 양쪽이 연결될 경우 해당 파이프로 권한(O_WRONLY 등)에 따라 작업이 가능해진다.

- **line 27**: `if (write(send_fd, send_msg, sizeof(send_msg)) == -1) return -1;`

  send_fd(NP_SEND == `client_to_server`) 파이프를 이용해 send_msg를 전달한다(write).

- **line 29**: `if (read(receive_fd, receive_msg, sizeof(receive_msg)) == -1) return -1;`

  receive_fd(NP_RECEIVE == `server_to_clinet`) 파이프로부터 받은 메시지를 receive_msg에 저장한다(read).

server.c

assignment > assignment_2 > C server.c

```c
int main(void) {
    char receive_msg[BUFFER_SIZE], send_msg[BUFFER_SIZE];
    int receive_fd, send_fd;
    int value;

    /*---------------------------------------*/
    /* TODO 3 : make pipes and init fds      */
    /* (1) make NP_RECEIVE pipe              */
    /* (2) make NP_SEND pipe                 */
    /* (3) init receive_fd and send_fd       */
    //make client_to_server pipe
    if (mkfifo(NP_RECEIVE, 0666) == -1) return -1;

    //make server_to_client pipe
    if (mkfifo(NP_SEND, 0666) == -1) return -1;

    if((receive_fd = open(NP_RECEIVE, O_RDWR)) == -1) return -1;
    if((send_fd = open(NP_SEND, O_WRONLY)) == -1) return -1;
    /* TODO 3 : END                          */
    /*---------------------------------------*/
```

assignment > assignment_2 > C server.c

```c
int main(void) {
    /*---------------------------------------*/

    while (1) {
        /*---------------------------------------*/
        /* TODO 4 : read msg                     */
        if (read(receive_fd, receive_msg, sizeof(receive_msg)) == -1) return -1;
        /* TODO 4 : END                          */
        /*---------------------------------------*/

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

        value = atoi(receive_msg);

        sprintf(send_msg, "%d", value*value);
        printf("server : send %s\n", send_msg);

        /*---------------------------------------*/
        /* TODO 5 : write msg                    */
        if (write(send_fd, send_msg, sizeof(send_msg)) == -1) return -1;
        /* TODO 5 : END                          */
        /*---------------------------------------*/
    }
    return 0;
}
```

- **line 23:** `if (mkfifo(NP_RECEIVE, 0666) == -1) return -1;`

  `mkfifo` 함수로 NP_RECEIVE(= `client_to_server`) 파이프를 생성한다. 이렇게 생성하지 않으면 파이프에 연결할 수 없다.

- **line 26**: `if (mkfifo(NP_SEND, 0666) == -1) return -1;`

  NP_SEND(= `server_to_client`) 파이프를 생성한다.

- **line 28, 29**:

  `if((receive_fd = open(NP_RECEIVE, O_RDWR)) == -1) return -1;`

  `if((send_fd = open(NP_SEND, O_WRONLY)) == -1) return -1;`

  각각 receive_fd와 send_fd에 open한 파이프를 연결한다. client.c에서 말한대로, 서로 open하는 순서를 맞춰준다(`client_to_server` 먼저, `server_to_client` 나중에). 그렇지 않으면 서로 연결을 대기하는 교착 상태에 빠질 수 있다.

- **line 36**: `if (read(receive_fd, receive_msg, sizeof(receive_msg)) == -1) return -1;`

  receive_fd 파이프로부터 받은 메시지를 receive_msg에 저장한다(read).

- **line 49**: `if (write(send_fd, send_msg, sizeof(send_msg)) == -1) return -1;`

  send_fd 파이프를 통해 send_msg(=제곱한 결과)를 클라이언트에 보낸다(write).

실행결과

```
pizzazoa@DESKTOP-I8SL73I:~/VMshared_folder/ipc-project-master/assignment/assignment_2$ chmod 777
pizzazoa@DESKTOP-I8SL73I:~/VMshared_folder/ipc-project-master/assignment/assignment_2$ ./run.sh
client : send 1
server : receive 1
server : send 1
client : receive 1

client : send 2
server : receive 2
server : send 4
client : receive 4

client : send 3
server : receive 3
server : send 9
client : receive 9

client : send 4
server : receive 4
server : send 16
client : receive 16

client : send 5
server : receive 5
server : send 25
client : receive 25

client : send 6
server : receive 6
server : send 36
client : receive 36

client : send 7
server : receive 7
server : send 49
client : receive 49

client : send 8
server : receive 8
server : send 64
client : receive 64

client : send 9
server : receive 9
server : send 81
client : receive 81

pizzazoa@DESKTOP-I8SL73I:~/VMshared_folder/ipc-project-master/assignment/assignment_2$
```

(비고: vs code의 마운트 환경에서는 파이프를 생성하는 권한이 없는 오류가 있어 WSL 우분투 환경에서 실행)