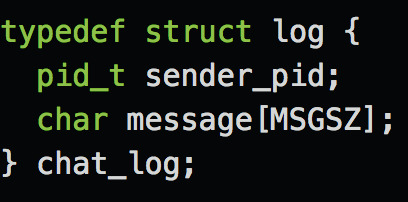
**시스템프로그래밍 과제 – IPC**

**2011020534 이원종**

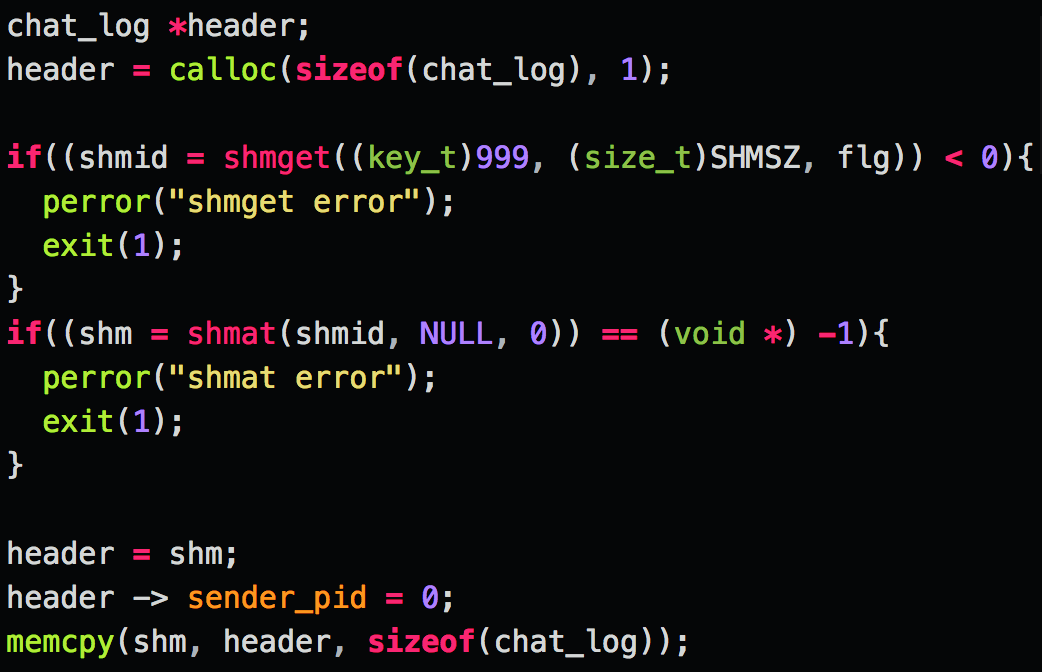
**2016025041 하태성**

**1.**  **Server.c**

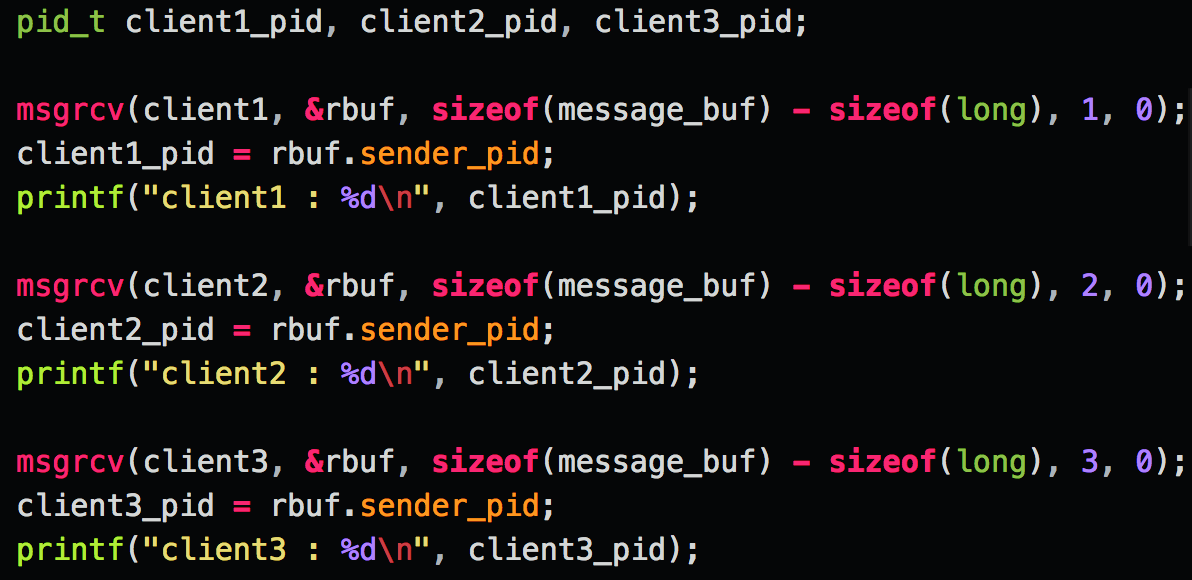
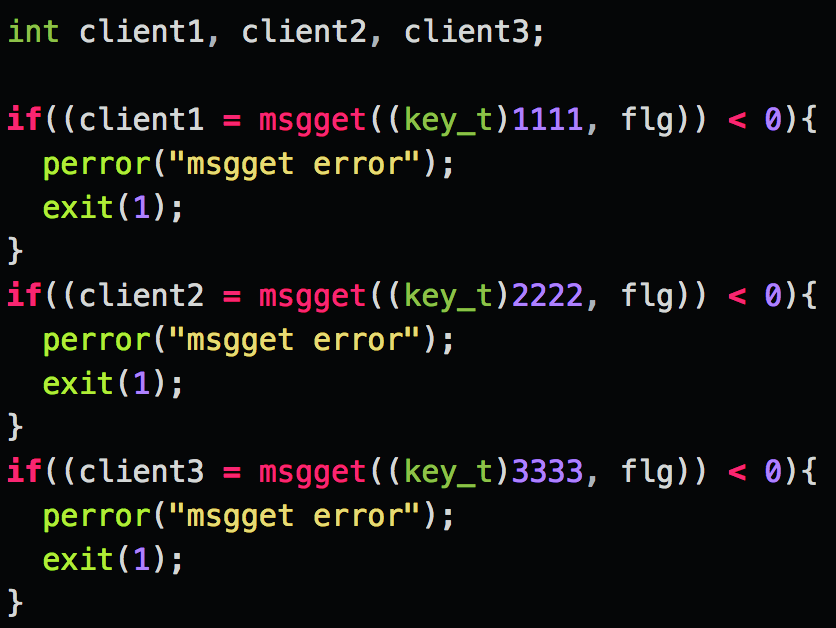
- message queue로 데이터를 주고 받을 message\_buf 구조체와 shared memory에 저장할 형태인 chat\_log 구조체를 정의. 필수적으로 들어가는 mtype과 message를 담을 mtext 필드 이외에 과제에 필요한 pid를 주고받을 필드를 정의. shared memory를 타입 캐스팅해서 사용하기 위해 저장할 채팅 로그를 구조체로 표현.



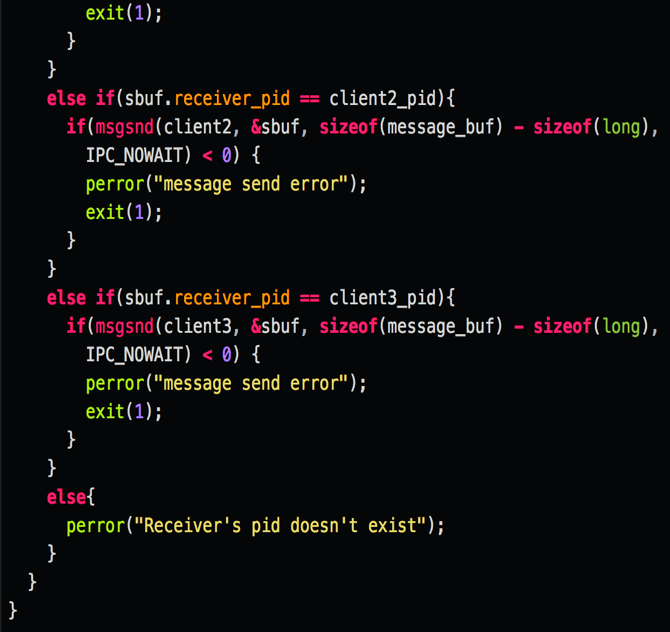
- shared memory의 크기를 chat\_log 100개가 아닌 101개로 잡고 맨 처음 chat\_log만큼을 헤더로 사용. 채팅 로그를 저장, 출력 혹은 관리를 하기 위해 header의 sender\_pid를 현재 shared memory(chat log)에 저장되어있는 chat\_log 갯수를 나타내는 필드로 사용.



- 처음에 각 프로세스들을 실행했을 때는 서버가 client 프로세스들의 pid를 알지 못함. 따라서 IPC\_NOWAIT 옵션을 주지않고 message queue를 생성한 뒤 각 client에게 메시지를 하나씩 받아 pid를 저장.

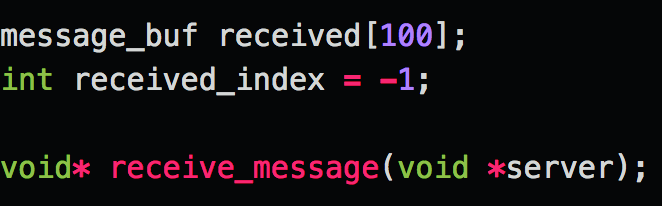


- 각 프로세스 별로 while문을 돌며 message queue에서 프로세스가 보낸(type : 1,2,3) 메시지를 확인. IPC\_NOWAIT 옵션을 넣어 만약 message queue가 비어있다면 while문을 나가 다음 프로세스의 message queue를 확인. 각 프로세스 별 while문을 전체 while문으로 묶어 계속 반복해가며 message queue를 확인. message의 receiver\_pid가 0인 경우는 header에 갯수를 증가시키고 chat\_log에 저장하고 0이 아닌 경우엔 해당 pid에 해당하는 client와 연결된 message queue로 message 전송.



**2. Client.c**

- server.c와 마찬가지로 message\_buf와 chat\_log 구조체를 선언하고 그 외에 서버와 달리 자신에게 온 message를 따로 읽는 receive\_message 함수와 읽어서 저장할 received 배열, 몇개의 메시지를 받았는지를 나타낼 received\_index를 추가적으로 선언.

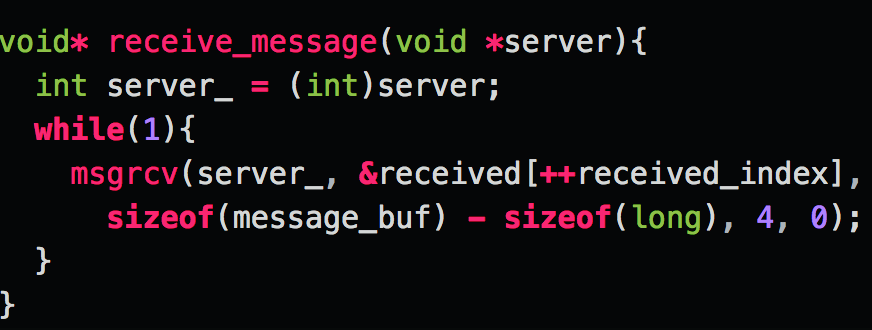


- server에서 처음에 pid를 저장할 수 있도록 본격적으로 user interface에 들어가기전에 server로 메시지를 전송. mtype은 프로세스 번호로 지정. msgsnd를 할 때 3번째 인자로 들어가는 size에는 type은 항상 들어가기때문에 man page를 참고해 type에 해당하는 long만큼을 빼고 전송해야함.

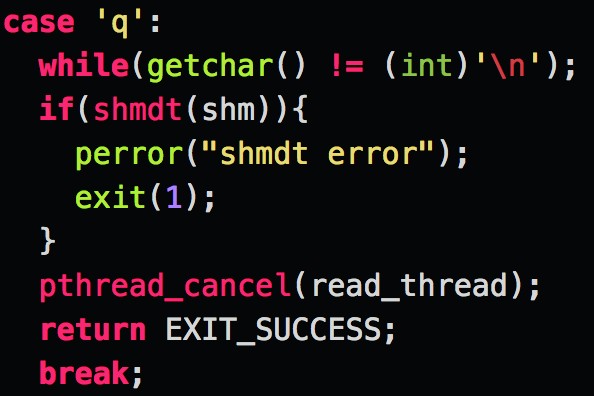
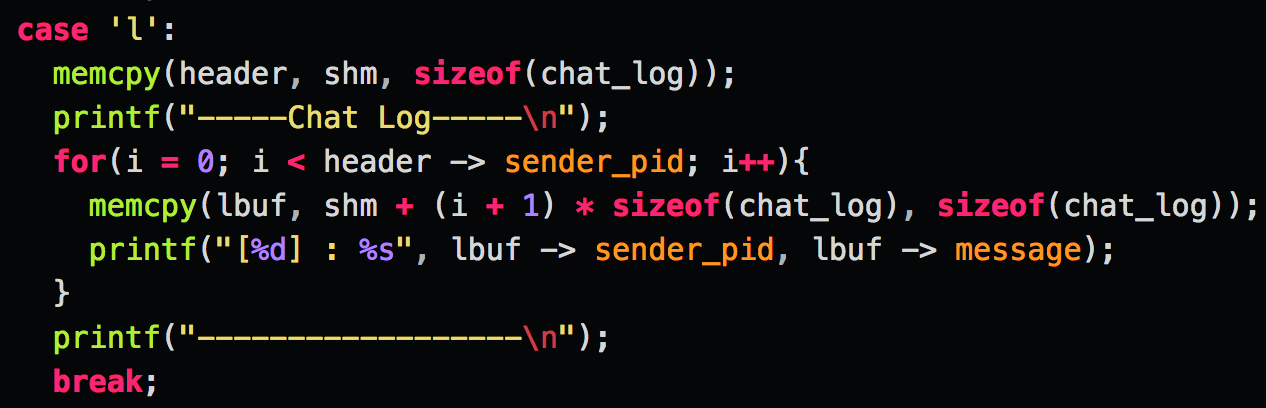
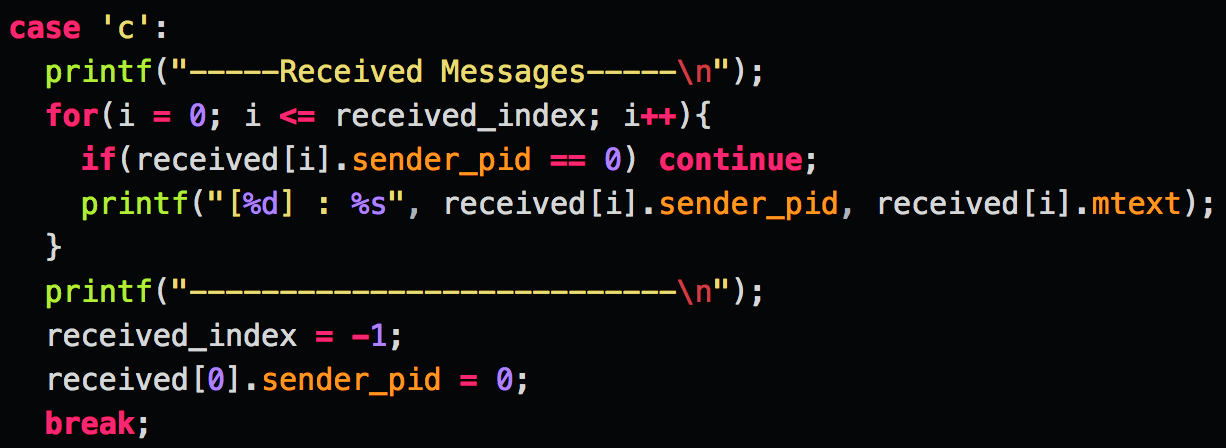
- 자신에게 도착한 message를 따로 읽어 저장하는 역할을 하는 thread를 생성. message queue id를 인자로 넘겨줌. 쓰레드를 따로 생성하지않고 읽을 수도 있지만 그럴 경우 user가 받은 메시지를 읽지 않고 계속 서로 보내는 경우 message queue가 금방 차 통신이 불가능한 경우가 생김.(message queue가 터짐)



- 위에서 만든 쓰레드가 수행할 함수. while문을 통해 message queue를 계속 확인하며 서버가 전달한 메시지(type : 4)만 받아 received 배열에 저장.



- do ~ while문과 switch ~ case 문을 통해 user interface를 구현. ‘c’의 경우 위에서 만든 쓰레드가 저장해놓은 메시지들을 출력. ‘s’의 경우 message를 만들어서 서버로 전송. mtext의 경우 stdin 버퍼에서 개행문자가 들어가는 오류가 발생해 while(getchar() != (int)’\n’); 으로 버퍼를 받은 후 띄워쓰기가 포함된 메시지가 들어오므로 fgets함수를 통해 메시지 내용을 받음. ‘l’의 경우 먼저 shared memory에 헤더를 받아온 뒤 헤더에 저장된 갯수만큼 shared memory를 읽어 chat log를 출력. 마지막으로 ‘q’의 경우 shared memory를 detach 하고 쓰레드를 종료시키고 프로세스 종료.



- shared memory는 서버에서 shared memory에 기록하기 때문에 explicit lock이나 semaphore 같은 sync를 따로 관리 해줄 필요없으며 message queue도 서버->클라이언트와 클라이언트->서버를 타입을 따로 해서 통신을 하기 때문에 sync를 따로 관리해주지 않아도됨. 만약 프로세스가 많아지게 될 경우 여러 개의 프로세스가 동시에 서로 채팅을 보내고 동시발생적으로 chat log를 쓸 경우 서버에 부하가 가서 while문에서 처리하는 속도때문에 각 프로세스간에 동기화가 안될 수도 있기때문에 그럴 경우 lock이나 semaphore 등을 통해 sync 문제를 해결해 주어야 됨.

**3. Delete\_ipc.c**

- 과제 명세에는 없지만 과제를 구현하고 실행하면서 필요에 의해 만든 프로그램. ipc는 키로 커널에서 할당을 받는데 IPC\_CREAT 플래그의 경우 해당 키에 해당하는 ipc가 존재할 경우 truncate 시킨 후 새로 만들어서 return하지않고 기존의 것을 그대로 return하기 때문에 기존에 오류가 나거나 사용중인 프로세스가 있을 경우 그 shared memory나 message queue를 그대로 받아와 오류가 발생 할 수 있음. 따라서 msgctl 함수와 shmctl 함수에 IPC\_RMID 플래그를 통해 기존에 존재하는 ipc를 삭제해주는 프로그램. Makefile에 make clean 명령어에 추가.