Dear all,

There is a change in the assignment because of a problem in Assgn 5 we missed carelessly, our fault. Pls note the problem and the solution. Also, given this change is coming on Tuesday, you may not submit a fully working code upto what was told in the intermediate submission this week, as long as most of the code is there (upto the point mentioned), it is ok.

The problem is that though two processes, creating two sockets, will not be able to bind to the same IP-port, the socket id returned by their socket calls can be the same as socket id is on a per-process basis. So the UDP socket id created when a user process makes m_socket() call or m_bind() call, is not the socket id that init process understands, so S and R cannot use it to send/receive. So to solve it, init process needs to create the socket and bind it when m_socket() or m_bind() call is made. This needs some additional synchronization. The solution proposed is below. It makes one assumption that only one process will call socket()/bind() at one time (that can be removed, but makes the solution bit more complex, so we allow it).

SC has graciously coded it and it works. If you find any boundary problems, let us know. But essentially, you will have to make all calls on the actual UDP socket from init process only.

Note that whenever you access a shared memory, you have to ensure mutual exclusion, that is understood, so not written separately.

Regards,

-AG

Init process, in addition to what we have given, does this:
1. Create a shared memory structure SOCK_INFO with the following fields: sock_id, IP, port,, errno. All are initialized to 0.
2. Create 2 semaphores Sem1 and Sem2. Both are initialized to 0.
3. The main thread, after creating S and R and doing everything else, does the following:
(a) wait on Sem1
(b) On being signaled, look at SOCK_INFO.
(c) If all fields are 0, it is a m_socket call. Create a UDP socket. Put the socket id returned in the sock_id field of SOCK_INFO. If error, put -1 in sock_id field and errno in errno field. Signal on Sem2.
(d) if sock_id, IP, and port are non-zero, it is a m_bind call. Make a bind() call on the sock_id value, with the IP and port given. If error, reset sock_id to -1 in the structure and put errno in errno field. Signal on Sem2.
(e) Go back to wait on Sem1

m_socket call:
Looks at SM to find a free entry as usual. Then signals on Sem1 and then waits on Sem2. On being woken, checks sock_id field of SOCK_INFO. If -1, return error and set errno correctly. If not, put UDP socket id returned in that field in SM table (same as if m_socket called socket()) and return the index in SM table as usual. In both cases, reset all fields of SOCK_INFO to 0.

m_bind call:
Find the corresponding actual UDP socket id from the SM table. Put the UDP socket ID, IP, and port in SOCK_INFO table. Signal Sem1. Then wait on Sem2. On being woken, checks sock_id field of SOCK_INFO. If -1, return error and set errno correctly. If not return success.
In both cases, reset all fields of SOCK_INFO to 0.

Rest remains the same.

Assumption. No two processes will make m_socket or m_bind calls together (or the signals can go to wrong process. Handling it will require putting process id, checking, or making SOCK_INFO an array etc. Lets keep things simple).