This lab concerns the use of two of the components of the System V IPC suite: message queues and shared memory. It also uses the fork() system call.

1. Get copies of the sample program message.c and the makefile (if necessary, copy makefile.unx to Makefile) in one of your subdirectories. Compile and link the program by typing:

   make message

2. Run the program first to see what it does. Make a printout and study the code so that you understand the order in which things are done before and after the fork(). Even though both parent and offspring can read and write the shared memory, synchronization is not a problem in this simple program. In a more complicated program this might not be the case and we might have to use semaphores for synchronization if more than one process needed to write to shared memory.

3. Run the program and stop at the "pausing" message. Open a second xterm (or console) and check all message queues, shared memory (and semaphores) with the command

   ipcs -mqs < cr >

   Can you find the message queue and shared memory segment belonging to you? Now hit < Enter > and end the program. Check again with

   ipcs -mqs < cr >

   Your message queue and shared memory segment should be gone. Note that message queues and shared memory segments are persistent structures. It your program didn’t remove them you should manually use ipcrm to do so.

Assignment. Modify the program in the following way (note that there is a critical section of code you will have to protect):

   1. Instead of having the offspring process end the program after it has read MESG_INIT from shared memory have it (the offspring process, not the parent) prompt the user for a string (e.g. “hello, this is from the user.”) from the console,

   2. when it has the received the string, the offspring should copy the string to shared memory (e.g. in shm_ptr->buf),

   3. it should then send (via msgsnd()) a message to the parent indicating that there is a message in shared memory for the parent, and the offspring should then go into sigsuspend().

   4. The parent, instead of just waiting for SIGCLD like in message.c should call msgrcv() and wait for one more message from the offspring,

   5. when the parent returns from msgrcv() it (the parent, not the offspring) should read shared memory to get the string, display it on the screen, send SIGUSR2 to the offspring, and wait for SIGCLD before exiting,

   6. the offspring, on breaking out of of sigsuspend() should write a message to the screen indicating it is exiting and then exit.

When you are satisfied your program is working correctly, e-mail me and tell me the path to the source file (that’s all I need).