Programming Assignment 1

Due by 11:55pm Friday, March 13th 2015

This assignment has the following goals:

1. To solidify your understanding of **IPC principles**.
2. To develop greater appreciation for the different **IPC mechanisms**.
3. To gain hands-on experience using **shared memory**.
4. To gain hands-on experience using **message queues**.
5. To gain hands-on experience using **signals**.
6. To learn how to combine shared memory and message queues in order to implement a practical application where **the sender process sends information to the receiver process**.

Overview

In this assignment you will use your knowledge of shared memory and message queues in order to implement an application which synchronously transfers files between two processes.

You shall implement two related programs: a *sender program* and the *receiver program* as described below:

• **sender:** this program shall implement the process that sends files to the receiver process. It shall perform the following sequence of steps:

1. The sender shall be invoked as `./sender file.txt` where sender is the name of the executable and `file.txt` is the name of the file to transfer.
2. The program shall then attach to the shared memory segment, and connect to the message queue both previously set up by the receiver.
3. Read a predefined number of bytes from the specified file, and store these bytes in the chunk of shared memory.
4. Send a message to the receiver (using a message queue). The message shall contain a field called size indicating how many bytes were read from the file.
5. Wait on the message queue to receive a message from the receiver confirming successful reception and saving of data to the file by the receiver.
6. Go back to step 3. Repeat until the whole file has been read.
7. When the end of the file is reached, send a message to the receiver with the size field set to 0. This will signal to the receiver that the sender will send no more.
8. Close the file, detach shared memory, and exit.

- **receiver:** this program shall implement the process that receives files from the sender process. It shall perform the following sequence of steps:

1. The program shall be invoked as ./recv where recv is the name of the executable.
2. The program shall setup a chunk of shared memory and a message queue.
3. The program shall wait on a message queue to receive a message from the sender program. When the message is received, the message shall contain a field called size denoting the number of bytes the sender has saved in the shared memory chunk.
4. If size is not 0, then the receiver reads size number of bytes from shared memory, saves them to the file (always called recvfile), sends message to the sender acknowl- edging successful reception and saving of data, and finally goes back to step 3.
5. Otherwise, if size field is 0, then the program closes the file, detaches the shared memory, deallocates shared memory and message queues, and exits.

- **When user presses** Control-C in order to terminate the receiver, the receiver shall de-allocate memory and the message queue and then exit. This can be implemented by setting up a signal handler for the SIGINT signal. Sample file illustrating how to do this have been provided (signaldemo.cpp).

Technical Details

The skeleton codes for sender and receiver can be found on Titanium. The files are as follows:

- sender.cpp: the skeleton code for the sender (see the TODO: comments in order to find out what to fill in)
- recv.cpp: the skeleton code for the receiver (see the TODO: comments in order to find out what to fill in).
- msg.h: the header file used by both the sender and the receiver

  It contains the struct of the message relayed through message queues). The struct contains two fields:
  - long mtype: represents the message type.
  - int size: the number of bytes written to the shared memory.

  In addition to the structure, msg.h defines macros representing two different message types:
  - SENDER_DATA_TYPE: macro representing the message sent from sender to receiver. It's type is 1.
  - RECV_DONE_TYPE: macro representing the message sent from receiver to the sender acknowledging successful reception and saving of data.

- **NOTE: both message types have the same structure. The difference is how the** mtype **field is set. Also, the messages of type** RECV_DONE_TYPE do **not make**

use of the size **field.**

- Makefile: enables you to build both sender and receiver by simply typing make at the command line.
- signaldemo.cpp: a program illustrating how to install a signal handler for SIGSTP signal sent to the process when user presses Control-C.

Please note: by default the skeleton programs will give you errors when you run them. This is because they are accessing unallocated, unattached regions of shared memory. It's your job to fill in the appropriate functionality in the skeleton, de-noted by the TODO comments, in order to make the programs work.

The following links provide additional documentation about shared memory and message queues:

- Message Queues: http://beej.us/guide/bgipc/output/html/multipage/mq.html
- Shared Memory: http://beej.us/guide/bgipc/output/html/multipage/shm.html

Bonus

Implement ***separate*** versions of sender and receiver which instead of relying on message queues to signal events, rely purely on signals. Hint: use handlers for SIGUSR1 and SIGUSR2. Hint: how can you tell the receiver the number of bytes in the shared memory?

Submission Guidelines:

- **This assignment MUST be completed using C or C++ on Linux.**
- You may work in groups of 3.
- Please hand in your source code electronically (do not submit .o or executable code)
- You must make sure that the code compiles and runs correctly.
- Write a README file (text file, do not submit a .doc file) which contains
  - Your Section#, Name and Email Address.
  - The programming language you used (i.e. C or C++).
  - How to execute your program.
  - Whether you implemented the extra credit.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as p2-[userid] for assignment 2, e.g. p2-ytian).
- Tar the contents of this directory using the following command. tar cvf [directory_name].tar

[directory name] E.g. tar -cvf p2-ytian.tar p2-ytian/

- Use TITANIUM to upload the tared file you created above.

Grading Guideline:

- Design of your program (write in a document called "Design of Assignment1"): 10'
- Program compiles: 5'
- Correct use of message queues: 25'
- Correct use of shared memory: 25'
- Program deallocates shared memory and message queues after exiting: 10'.
- Correct file transfer: 5'
- Correct signal handling: 5'
- All system calls are error-checked: 5'
- README file: 10'
    (1) List your team members' Names and Emails
    (2) How to run your program (platform, commands in each steps and a screenshot of one testing)
    (3) How did your team collaborate on your projects? Illustrate each member's contribution.
- Bonus: 10'
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

Academic Honesty: All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.