CSC 456 Homework #3 – Due before end of day 5/1/2015

The third programming assignment is to write an OS simulation program. This is a group project!

Required elements:

- Middleware shared memory utility. This block of shared memory will be available for storage and interprocess communication. The shell will setup the blocks of shared memory and the required semaphores to control access to the shared memory. Since multiple processes will have access to the shared memory, the standard shared resource problems will arise. You must resolve the multiple access (or starvation and deadlock) issues that can occur.
- Mailbox initialization
  - Usage: `mboxinit <number of mailboxes> <size of mailbox in kbytes>`
  - Create *k+1* blocks
  - *k* = number of mailboxes requested
  - block size = size of the mailbox
  - The first block of shared memory will contain the mailbox information for all to access.
  - The next k blocks will act as the mailboxes
- Mailbox removal
  - Usage: `mboxdel`
  - Clean up semaphores and shared memory. This must be done by the creating shell on exit if not before.
- Write data
  - Usage: `mboxwrite <boxnumber>`
  - ... data to be written ...
  - `ctrl-d` to end
  - Truncate if data is larger than the mailbox and flag user.
- Read data
  - Usage: `mboxread <boxnumber>`
  - Will read in the text until end of the string or the end of the mailbox is found.
- Copy data
  - Usage: `mboxcopy <boxnumber1> <boxnumber2>`
  - Copy the contents of one box to another.
  -
- Simulation of the process scheduler with Round Robin, Priority, Shortest Job First.
- Simulation of the memory management unit, TLB, page tables and paging.
- Simulation of the page replacement algorithms FIFO, Optimal, LRU, LFU, Second Chance, Clock.
  - Use both reference and distance strings.
  - Note: for bit strings - focus only on left entry, right shift.
- GUI interface is optional but will be not be frowned upon.

Submission contents:

- Documentation
  - Complete description of the implementation.
    - You must document the author of each code segment.
    - You must document the percentages of effort and these must add up to 100%
  - Complete description of the testing.
  - Complete description of what the system requirements are, how to build, how to run and how to use the GUI.
  - Write up should discuss how this is used as simulation and as a teaching tool.
- Main program.
- Any required external functions `*.c`.
- Any include files you use (other than the standard ones).
- You will need to produce a Makefile to build the executable:

Solution method:

- Get your group assembled
- Grab the shared memory code example and experiment with it. Modify the code so that it can allocate a several blocks and you know how to write and read from it.
- Test this code in two regular shells, one which allocates, attaches and writes; and the other which attaches and reads.
- Break the code into the required functions.
- Add to the initialization code the initialization of the required semaphores (semaphore example code).
- Place the semaphore code into the new functions.
- GSL - Gnu Scientific Library has good random number generation.
- Complete assignment.
- Hand in by: 5/1/15

Testing:

- Try to break the code.
- Open two shells and try to write to the same block.
- Just because human interaction is so slow and the chance of actual synchronization problems is very very low, does not mean that there are no problems. You must write this as if actual processes were running some mix of producer-consumer and reader-writer.

Notes and hints:

- Debugging `shm` and `sem` can be done from the command line with the `ipcs` command. Using `ipcs -m or ipcs -s` will provide information on existing `shm` or `sem` components. `ipcrm shm n` can be used to remove shm number n, similarly `ipcrm sem n` can be used for sem n.
- Fixed key values are ok.
- You may setup multiple shared memory blocks or a single one and partition it for the mailboxes.

Grading approach:

- copy from the submit directory to my grading directory
- `tar -xzf prog3.tgz`
- `cd prog3`
- `make`
- verify code
- Program demo
- assign a grade
- `cd ..`
- `rm -rf prog3`

Submission method: You will use the submit page to submit your program. The files need to be tarred and gzipped prior to submission. Please empty the directory prior to tar/zip. [You will tar up the directory containing the files: `tar -czf prog3.tgz  prog3` .]

Sample Makefile to build "crash" (your version will be slightly different):

```
#-----------------------------------------------------------------------
      # Use the GNU C/C++ compiler:
      CC = gcc
      CPP = g++

      # COMPILER OPTIONS:

      CFLAGS = -c

      #OBJECT FILES
      OBJS = crash.o foo.o crunch.o


      crash: ${OBJS}
            ${CC} -lm ${OBJS} -o crash

      crunch.o: crunch.c
            ${CC} -c -lm crunch.c

      crash.o: crash.c
      foo.o: foo.c
#-----------------------------------------------------------------------
```

Note: indents are tabs - NOT SPACES. Tab is a command (yes, horrible design to have whitespace commands).