CST 8244, Winter 2012, Assignment 2

Date: 4 March 2012 **Due**: 30 March 2012, Week 11

Objective: An academic exercise to assess usage of programming elements in QNX.

Background: Write two programs. Call them `producer` and `user.` A data file consisting of 1,000 pairs of floating point numbers is provided. `producer` reads the file, `user` computes arithmetic operations on the numbers and writes them back to the location.

Submission: 1. Demonstrate a working program.

2. Email all source code, makefile and output of the program to `terais@algonquincollege.com` . Subject line of email should be "`CST 8244, W12, Asgn 2`". Your submission could be lost if (i) the subject line is incorrect and/or (ii) if you are not using Algonquin live to send the email.

3. Upload all source code to Blackboard before the due date. Send and upload only `.zip` files. Do not submit `.rar` or other formats.

Documentation: Document your code with the header and footer provided earlier in a lab. Document all segments of your code where the required feature is implemented, for instance, reading the structure from shared memory, advancing the pointer, computing values, writing output to a file.

Requirements: Name the project **`ARITH2`**. The data file provided contains two floating point numbers on each line separated by a comma. Create an array of 1,000 structures in shared memory, structure is provided in the file `arith.h`. You may add additional data elements to the existing structure to meet the requirements. Refer to the Data Flow Diagram in the file titled `DFD-A2-W12.pdf`. The numbers in the data flow are not required for understanding the flow.

`producer`. Producer reads the data file and writes the two numbers in the array of structures created in shared memory. After populating the array producer sends a pulse to the user as an indication that the data has been loaded.

`user`. User receives a pulse from the `producer`. Indicating values have been loaded in shared memory. It creates four threads; **`sum`** thread computes the sum of the two

numbers, **`diff`** thread computes the difference between the two numbers, `operand1 - operand2`, **`mult`** thread computes the product of two numbers, **`div`** thread divides `operand1` by `operand2`. Each thread updates the corresponding value in the structure in shared memory.
Important: Each thread copies only the two operands to its memory variables. It does not copy the entire array of 1,000 structures. Use `memcpy` to copy the structure, write only the relevant structure element to shared memory, `double sum`, `double difference`, `double product` and `double quotient`. Advance the pointer by the size of the array to read the next two operands.

After all arithmetic operations have been computed, two new threads are created **`average`** and **`sum`**. **`average`** thread computes the average of each of six of the 1,000 numbers in the structure, **`sum`** thread computes the sum of each of the six of the 1,000 numbers in the structure.

Priority and timers: The sum and difference threads have priorities and timers. **`sum`** thread runs at priority 20; it reads, computes and writes values when the timer fires at `0.2 second`. **`diff`** thread runs at priority 18; it reads, computes and writes values when the timer fires at `0.15 second`. **`product`** and **`div`** threads run at default priority with no timers.

Results: Save your results to a file titled `A2-output.txt`; sum and average of `operand1`, `operand2`, sum, difference, product and quotient.
Print the 1,000 item array as a comma separated value to a file `1000-array.txt`