CS 218 – Assignment #12

Purpose: Become more familiar with operating system interaction, threading, and race conditions.
Points: 100 50 for program and 50 for write-up
Scoring will include functionality, documentation, and coding style

Assignment:

In recreational number theory, a Narcissistic number[1] is a number that is the sum of its own digits each raised to the power of the number of digits. For example, given 8208 which has 4 digits, the sum of each number raised to 4 is 8208.

$$8208 = 8^4 + 2^4 + 0^4 + 8^4$$

Write an assembly language program provide a count of the Narcissistic numbers between 0 and some user provided limit. For example, between 1 and 10,000 there are exactly 17 Narcissistic numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407, 1634, 8208, 9474).

The Simpsons, Season 17 Episode 22

The first, 8,191 is a Mersenne prime, a prime number that can be divided only by itself and one. And, 8,128, is a perfect number, where the sum of proper divisors add up to it. Then, 8,208 is a narcissistic number, which has four digits and, if you multiply each one by itself four times, the result adds up to 8,208.

In order to improve performance, the program should use threads to perform computations in parallel. The program should read the thread count and number limit in septenary from the command line in the following format; "./narCounter -t <1|2|3|4|5|6> -l <septNumber>". For example:

```
./narCounter -t 4 -l 564355
```

The following routines should be created:

- A value returning function, ***getUserArgs()***, that reads and verifies the command line arguments. This includes error and range checking based on provided parameters (in the template). The error strings are predefined in the template. The function should call the ***aSept2int()*** function.
- Value returning function ***aSept2int()*** to convert an ASCII/septenary string to integer. If the passed string is a valid ASCII/septenary value the value should be returned (via reference) and the function should return true. If there is an error, the function should return false. *Note*, the integer being returned is a quad (64-bits) which is different from previous assignments.
- A void function, ***narcissisticNumberCounter()***, that will be called as a thread function and determine the count of narcissistic numbers. The thread must perform updates to the global variables in a critical section to avoid race conditions. See below sections for additional detail.

1 For more information, refer to: https://en.wikipedia.org/wiki/Narcissistic_number

<u>Thread Function:</u>
Create a thread function, ***narcissisticNumberCounter()*** that performs the following operations:

- Print the thread starting message (predefined).
- Each thread will obtain the next bock of numbers to check (via global variable, ***currentIndex***) and increment the global by BLOCK_SIZE (range of numbers to check) using the predefined constant.
  - Loop while the next number is ≤ ***limitValue*** (globally available) to check each number in the block.
  - If a number is narcissistic, atomically increment *narcissisticCount* variable.
  - When the range of numbers has been processed, a new block should be obtained.
  - If the limit value is exceeded the function should exit.

When obtaining the next block to check and incrementing the global ***currentIndex*** variable, these operations must be performed as a critical section[2] (i.e., locked and unlocked using the provided ***spinLock()*** and ***spinUnlock()*** functions). As the numbers are being checked, no locks are needed. Once a number has been determined to be narcissistic, the number should be placed in the *narcissisticNumbers[]* array and the narcissistic counter should incremented accordingly. These operations must also be performed as a critical section.

It is recommended to complete and test the program initially with only the single sequential thread before testing the parallel threads. In order to debug, you may wish to temporarily insert a direct call (non-threaded) to the ***narcissisticNumberCounter()*** function.

<u>Results Write-Up</u>
When the program is working, complete additional timing and testing actions as follows;

- Use the provided script file to execute and time the working program.

- Compute the speed-up[3] factor from the base sequential execution and the parallel execution times. Use the following formula to calculate the speed-up factor:

$$SpeedUp = \frac{ExecTime_{sequential}}{ExecTime_{parallel}}$$

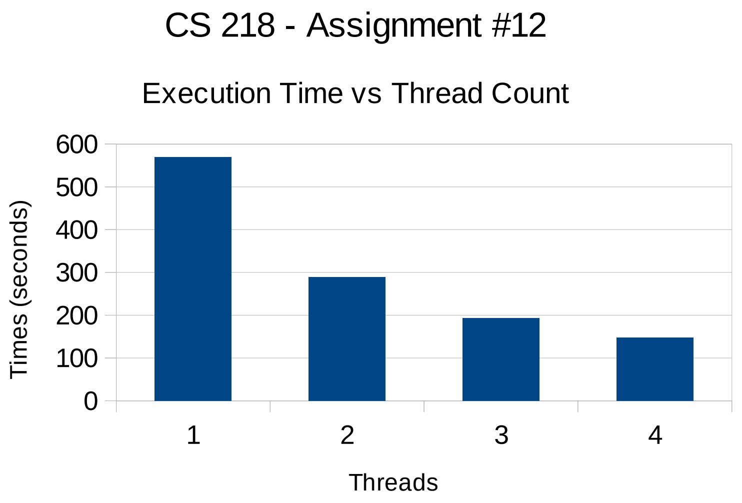
- Remove the locking calls (the ***spinLock()*** and ***spinUnlock()*** calls and the 'lock') and re-execute the program using a limit of 50,000,000 (1144664411_7) for 1, 2, 3, and then 4 threads. A provided timing script will help capture these results.
  - The Unix time command (as per asst #11B) should be used to obtain the execution times. Use the “real” time.

- Create a final write-up including a copy of the program output from the timing script file and an explanation of the results. The explanation must address:
  - the final results for 1, 2, 3, and 4 thread executions, including final results (list of perfect/abundant/deficient numbers) and the execution times for both each.
  - the speed-up factor from 1 thread to 2, 3, and 4 threads (via the provided formula)
  - simple chart plotting the execution time (in seconds) vs thread count (see example below).
  - the difference with and without the locking calls for the parallel execution
    - explain specifically what caused the difference

2 For more information, refer to: https://en.wikipedia.org/wiki/Critical_section
3 For more information, refer to: https://en.wikipedia.org/wiki/Speedup

The explanation part of the write-up (not including the output and timing data) should be less than ~300 words. Overly long explanations will be not be scored.

Below is an example of the type of chart to include



Such graphs are most easily done using a spreadsheet. An optional example spreadsheet is provided for reference.

Assignment #12 Timing Script

In order to obtain the times for the write-up a timing script is provided. After you download the script file, **`a12timer`**, set the execute permission as follows:

```
ed-vm$ chmod +x a12timer
ed-vm$ ./a12timer narCounter
```

The script file will expect the name of the executable as an argument (as shown).

The script may take a while to execute, but no interaction is required. The script will create an output file, **`a12times.txt`**, which will be included in the submission and the data be used create the write-up.

Submission:

- All source files must assemble and execute on Ubuntu and assemble with **`yasm`**.
- Submission three (3) files
  - Submit Source File
    - *Note*, only the functions file (**`a12procs.asm`**) will be submitted.
  - Submit Timing Script Output
    - Submit the **`a12times.txt`** file (created after executing the **`a12timer`** script).
  - Submit Write Up file (**`write_up.pdf`**)
    - Includes system description, speed-up amounts, and results explanation (per above).
- Once you submit, the system will score the code part of the project.
  - If the code does not work, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time (at a maximum rate of 5 submissions per hour).
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given assignment. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
;  Name: <your name>
;  NSHE ID: <your id>
;  Section: <section>
;  Assignment: <assignment number>
;  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a loss of up to 3%.

Scoring Rubric

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Assemble	-	Failure to assemble will result in a score of 0.
Program Header	3%	Must include header block in the required format (see above).
General Comments	7%	Must include an appropriate level of program documentation.
Program Functionality	40%	Program must meet the functional requirements as outlined in the assignment.
Timing Script Output	10%	Output from timing script showing results.
Write-Up	40%	Write-up includes required section, percentage change is appropriate for the machine description, and the explanation is complete.

Example Execution:
The following is an example execution for the sequential version:

```
ed-vm% time ./narCounter -t 1 -l 1144664411
------------------------------------------------------------
CS 218 - Assignment #12

Narcissistic Numbers Program

Hardware Cores: 12
Thread Count: 1
Numbers Limit: 50000000

   Start Counting...
 ...Thread starting...

Results:
--------
Narcissistic Count: 27
Narcissistic Numbers:
     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
     153
     370
     371
     407
     1634
     8208
     9474
     54748
     92727
     93084
     548834
     1741725
     4210818
     9800817
     9926315
     24678050
     24678051

real      0m5.402s
user      0m5.397s
sys       0m0.004s
ed-vm%
ed-vm%
```

```
ed-vm%
ed-vm% time ./narCounter -t 4 -l 1144664411
-----------------------------------------------------------
CS 218 - Assignment #12

Narcissistic Numbers Program

Hardware Cores: 12
Thread Count: 4
Numbers Limit: 50000000

   Start Counting...
 ...Thread starting...
 ...Thread starting...
 ...Thread starting...
 ...Thread starting...

Results:
--------
Narcissistic Count: 27
Narcissistic Numbers:
     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
     153
     370
     371
     407
     1634
     8208
     9474
     54748
     92727
     93084
     548834
     9800817
     9926315
     1741725
     4210818
     24678050
     24678051

real	0m1.459s
user	0m5.545s
sys	0m0.000s
ed-vm%
```

Note, the timing shown is for one specific machine. Actual mileage may vary.

The following are some addition examples of the applicable error messages.

```
ed-vm%
ed-vm% ./narCounter
Usgae: ./narCounter -t <1|2|3|4|5|6> -l <septNumber>
ed-vm%
ed-vm% ./narCounter -t 0 -lm 1564355
Error, thread count out of range.
ed-vm%
ed-vm% ./narCounter -th 3 -l 1564355
Error, invalid thread count specifier.
ed-vm%
ed-vm% ./narCounter -t 4 l 1564355
Error, invalid limit specifier.
ed-vm%
ed-vm% ./narCounter -t 4 -l 1564x355
Error, limit out of range.
ed-vm%
ed-vm%
ed-vm%
ed-vm%
```