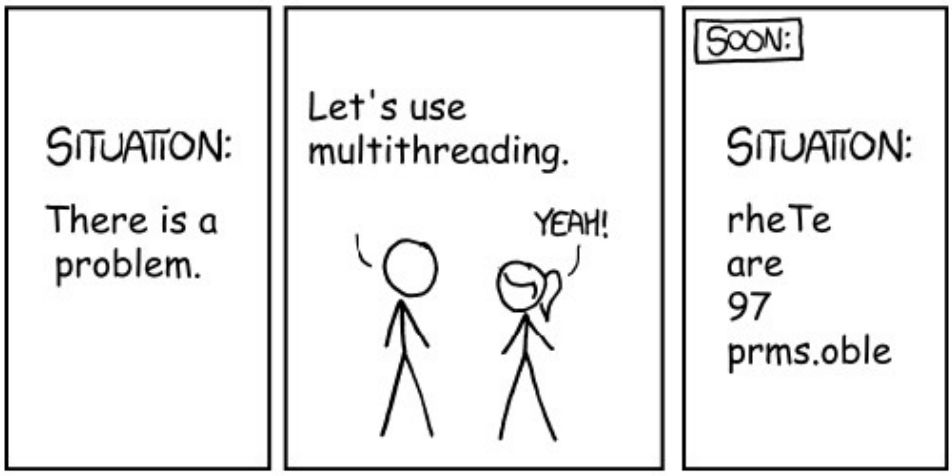
CS 218 – Assignment #12

Purpose: Become more familiar with operating system interaction, threading, and race conditions.
Points: 80 40 for program and 40 for write-up
Scoring will include functionality, documentation, coding style, and write-up

Assignment



In recreational number theory, a Smith Number[1] is a composite number whose sum of digits is equal to the sum of digits in its prime factorization[2]. For example, the number 666 would be (6 + 6 + 6) = 18 and the prime factorization of 666[3] is 2, 3, 3, and 37 which is (2+3+3+(3+7)) = 18. Thus, 666 is said to be a Smith Number.

Write an assembly language program to find the count of Smith numbers between 1 and some user provided limit. In order to improve performance, the program should use threads to perform some of the computations in parallel. The program should read the thread count option and number limit in senary from the command line in the following format; "./smithNums -t <senaryNumber> -l <senaryNumber>". For example, for 4 threads and a limit of 3000:

```
./smithNums -t 4 -l 21520
```

The following routines, called by the provided main, should be created:

- A value returning function, ***getArgs()***, 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 ***sSenary2int()*** function.
- Value returning function ***aSenary2int()*** to convert an ASCII/senary string to quad sized integer. If the passed string is a valid ASCII/senary 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, ***findSmithNumberCount()***, that will be called as a thread function and determine the count of Smith numbers. The thread must perform updates to the global variables in a critical section and avoid race conditions. See below sections for additional detail.

1 For more information, refer to: https://en.wikipedia.org/wiki/Smith_number
2 For more information, refer to: https://simple.wikipedia.org/wiki/Prime_factorization
3 For a worked example, refer to: https://factorization.info/prime-factors/0/prime-factors-of-666.html

Thread Function

The general flow of the ***findSmithNumberCount()*** thread function would be as follows:

- Display the one-time starting thread message (provided).
- Obtain the next number to check (via global variable, ***currentIndex***)
  - increment the global counter
- While the next number is ≤ ***userLimit*** (globally available);
  - Check if the number is a Smith number
    - If the number is a Smith number, increment the smith number count

A more detailed summary of the Smith number check algorithm is as follows:

```
int findSumOfDigits(int n) {
      int sum = 0;

      while (n > 0) {
            sumDigits = sumDigits + (n % 10);
            n = n / 10;
      }
      return  sumDigits;
}

bool isPrime(int n) {
      if (n <= 1)
            return false;

      for (int i = 2; i <= n/2; i++)
            if (n % i == 0)
                  return false;

      return true;
}

int findSumPrimeFactors(int n) {
      int i = 2, sum = 0;

      while (n > 1) {
            if ((n % i) == 0) {
                  sum = sum + findSumOfDigit(i);
                  n = n / i;
            } else {
                  do {
                        i++;
                  } while (!isPrime(i));
            } // end if
      } // end while

      return sum;
}
```

```
bool isSmithNum(int num) {
      if (isPrime(num)) {
            return false;
      }

      int a = findSumOfDigits(num);
      int b = findSumPrimeFactors(num);

      if (a==b)
            return true;
      else
            return false
}
```

Incrementing, obtaining, and checking the next number, ***currentIndex*** variable, must be performed as a critical section (i.e., locked and unlocked using the provided *spinLock()* and *spinUnlock()* functions).

Once an Smith number has been found, the counter should be incremented. The increment for the ***smithNumberCount*** should use the lock prefix to ensure an atomic increment. For example:

```
lock  inc   qword [smithNumberCount]
```

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 ***findSmithNumberCount()*** function.

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 smithNums
```

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.

Results Write-Up

When the program is working, complete additional timing and testing actions as follows;

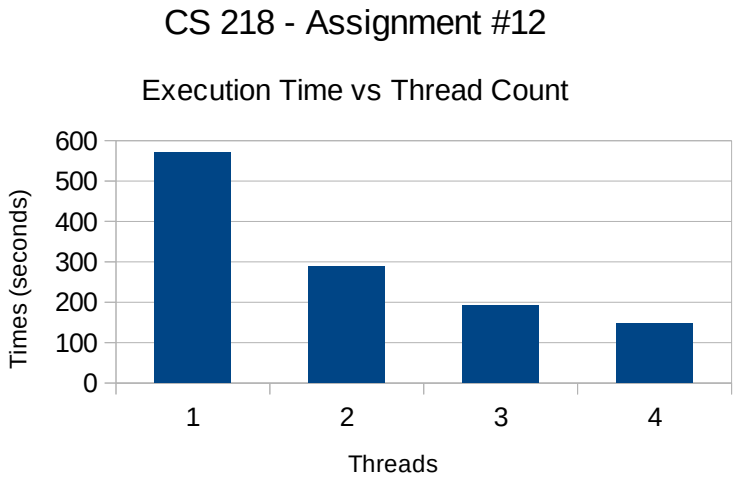
- Use the provided script file to execute and time the working program.
  - The script will execute the program with 1, 2, 3, and 4 threads and create
    - the final results (in **`a12results.txt`**)
    - the execution times (in **`a12times.txt`**)
- Compute the speed-up[4] factor from the base sequential execution and the parallel execution times. Use the following formula to calculate the speed-up factor:

4 For more information, refer to: https://en.wikipedia.org/wiki/Speedup

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

- 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
  - Include the final results for 1, 2, 3, and 4 thread executions, including final results (in **`a12results.txt`**) and the execution times (in **`a12times.txt`**).
  - 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).
  - Explain what might happen if the *spinLock()* calls were removed.

- 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.



Example Execution

The following are some example executions:

```
ed-vm%
ed-vm% ./smithNums -t 4 -l
Error, invalid command line options.
ed-vm%
ed-vm% ./smithNums -tx 2 -l 14414452
Error, invalid thread count specifier.
ed-vm%
ed-vm% ./smithNums -t 51 -l 14414452
Error, thread count out of range.
ed-vm%
ed-vm% ./smithNums -t d -l 14414452
Error, invalid thread count value.
ed-vm%
ed-vm% ./smithNums -t 2 -lmm 14414452
Error, invalid limit specifier.
ed-vm%
ed-vm% ./smithNums -t 3 -l 144x4452
```

```
Error, invalid limit value.
ed-vm%
ed-vm% ./smithNums -t 2 -l 1
Error, limit out of range.
ed-vm%
ed-vm% ./smithNums
Usage: ./smithNums -t <senaryNumber> -l <senaryNumber>
ed-vm%
ed-vm% ./smithNums -t 3 -l 2050544
------------------------------------------------------------
CS 218 - Assignment #12

Smith Numbers Counting Program

Thread Count: 3
Numbers Limit: 100000

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

Results:
--------
Smith Number Count: 3294
ed-vm%
ed-vm%
```

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: <4-digit-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.