Assignment #3:
PowerShell (Multithreading)

Notice

This is version #1 of this document. Be aware that revisions may be made to add details and/or requirements. If a revision is made, it will be posted to Lea. No revision will be made later than one week before the due date. You are responsible for meeting the latest requirement.

This document has a ton of detail. Read it from top-to-bottom and then re-read at least twice to make sure you don't miss anything.

Preamble

We've discussed the advantages of multi-threading (at length) and in this assignment we'll use multiple threads to increment a buffer variable. The focus of this assignment will be to highlight some of the headaches that multithreading can cause (specifically race conditions).

To complete this assignment, you will need to implement multi-threading (via PowerShell ***runspaces***) along with some type of *locking* mechanism (mutexes, semaphores, monitors, etc).

Part #1 Requirements

For this assignment, our threads will be implementing the following pseudo-code:

```
while(buffer < max){
   buffer++
}
```

You will be submitting several versions of the same script for this assignment, the first will contain a race condition, while the 2nd will fix the condition using a lock (see here or here for ideas).

The buffer itself is simply an integer, object property or hash element (your choice of lock may somewhat dictate the type of buffer you use).

Each thread should sleep for 1 second prior to doing anything (simply to allow all the other threads to also start; i.e. avoid the situation where 2-3 threads do all the work)

Input

Your program should take two parameters (using the **Param** keyword and the two names below). BOTH Parameters should be mandatory! (hint: this involves some code)

```
PS C:\Users\Jim> .\Desktop\asg3.ps1 -max 100 -numThreads 8
```

Output

Your program output should resemble the following as closely as possible (the following uses 8 threads with max = 100000):

```
Creating thread #1
Creating thread #2
...
Creating thread #8
VERBOSE: Pid: 63324, Thread: 81048 starting, initial buffer value: 1
VERBOSE: Pid: 63324, Thread: 81280 starting, initial buffer value: 362
...
VERBOSE: Pid: 63324, Thread: 81336 starting, initial buffer value: 19462
VERBOSE: Pid: 63324, Thread: 64364 starting, initial buffer value: 19460
VERBOSE: Pid: 63324, Thread: 25496 ending, current buffer value: 100001
VERBOSE: Pid: 63324, Thread: 64364 ending, current buffer value: 100003
...
VERBOSE: Pid: 63324, Thread: 59900 ending, current buffer value: 100005
Value of buffer at end of invocation: 100005
```

Hints

- You will need to review how to implement runspaces. The posted solution to assignment #2 should be sufficient.

Part #2

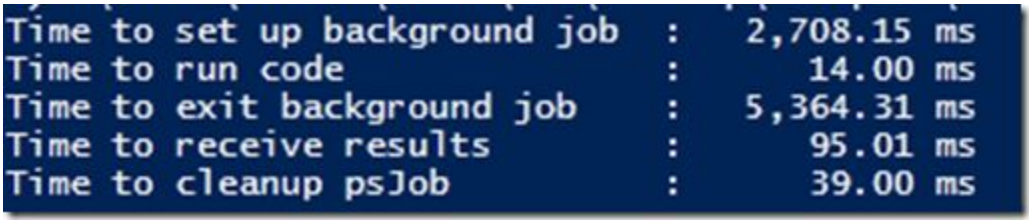
NOTE: The code is provided to you in the link below, you simply need to integrate it.

Create two copies of your corrected script (i.e. with locking), modify one of them to use PSJobs (`start-job`, `receive-job`, etc). PSJobs used multi-processing (as opposed to multi-threading).

To share memory across processes, use a file in the current directory, ie.

```
"this is output" >> $fileName
$thisIsInput = get-content $fileName
```

The goal here is to prove that threads (which runspaces uses) are much lighter-weight than processes (which PSJobs uses). Recall that each process has its own heap (threads share the heap with their parent, i.e. the process) in the PCB which is significantly heavier weight. Read this article and reproduce the timing output using your modified script.



A Note about PowerShell

- In addition to using strict mode, you should consider setting your script to die on the first error it encounters – this might save you some frustration:

```
$ErrorActionPreference = 'Stop'
```

- Should your script crash and/or exit abnormally, it's possible that some threads remain in the background. It is possible to essentially crash your PC if background tasks do stupid things (endless loops, huge memory allocations, etc). To kill background tasks, do this:

```
Get-Runspace | where {$_.id -ne 1} | %{$_.Dispose()} # kill child threads
```

Important details

- If your script uses filenames (to load or write files), you must use the $PWD environment variable. The penalty for failing to do so will be harsh! (10%)
- You should test your script under PowerShell 5 (run `$PSVersionTable.PSVersion.Major` to check). Our computer labs run PowerShell 5. If you have an older version installed at home, simply update it (PowerShell is a free download: https://www.microsoft.com/en-us/download/details.aspx?id=50395).
    - Using an older version is going to make your life much harder; there are numerous features in PowerShell 5 that aren't in early versions.
- Use Visual Studio Code with the PowerShell extension, you need a debugger!
    - When using the debugger, make sure the debug mode is set to "PowerShell Launch Current File in Temporary Console", otherwise your **`Set-Variable`** constant declarations will fail between runs (you did use constants, right?!)
- Your script must use strict mode!

```
Set-StrictMode -version latest
```

- We may build off of this assignment in the future, it is important that you complete it so as to not jeopardize your future grades.
- While programming style is not the primary focus of this class, you are expected to produce quality code. Please use common sense (i.e. functions where appropriate, liberal use of whitespace, consistent indentation, comments where needed, but not excessive, etc.). Grades will be deducted for inconsistency and/or incompetence re: coding style. See handout re: coding style.

Submission

A zip file containing:

- Your .ps1 script files:
    1. Multi-threaded with a race condition (call the file **asg3-racecondition.ps1**)
    2. Multi-threaded with the race condition fixed using a lock (call the file **asg3-locking.ps1**)
    3. Multi-threaded with the race condition fixed using a lock and a timer installed (call the file **asg3-timed_multi.ps1**)
    4. Parallel process with the race condition fixed using a lock and a timer installed (call the file **asg3-timed_parallel.ps1**)
- Screenshots showing the results of running scripts 1-4. Use a buffer size of **100000** and **8** threads (if your script takes more than 30 seconds to run, you probably have a bug).

Grading Rubric

% of Grade	Description
40%	Demonstrated mastery of introductory PowerShell (conditionals, loops, data structures, threading, locking, parallel processing).
20%	Correct result
20%	Code quality: KISS principle respected, commenting where appropriate, proper and consistent formatting (indenting, constants, etc.), **code re-use**.
20%	Complete working submission (four scripts, screenshots)

NOTES:

1. Code that does not work will clearly violate several of the above categories (e.g. broken code is clearly not of good quality, probably does not return the correct result, etc).

Late Penalty

10% per day to a maximum of three days late. If you have a legitimate reason, ask for an extension (a few days in advance) rather than losing grades for nothing.

My office hours are posted on my office door. Drop by if you're having trouble with anything!

Test Questions to Ponder

1. How are mutexes, monitors and semaphores different from each other?
2. What are the advantages of parallel jobs over multithreading?