CMPUT 379, Assignment 3, Winter 2018

(Topics: working set, process profiling)

Objective

You will implement a simulator to study the working set behavior of a running process by examining the output of `valgrind` for different programs and different sizes of instances.

The command line invocation

The simulator is invoked as follows:

`valws379` [`-i`] *pgsize windowsize*

Where *pgsize* is the assumed page size (always a power of 2 from 16 to 65,536), *windowsize* is the working set window in terms of memory references (a positive integer), and `-i` is an optional parameter which instructs `valws379` to ignore the memory references that are instruction fetches (type `I` in the output of `valgrind`).

`valws379` must work as a pipe. Its input is the output of `valgrind` and specifically it should be able to deal with the output produced by the following style `valgrind` invocation (and only for this style):

```
valgrind --tool=lackey --trace-mem=yes ...program...
```

The output of `valws379` should be a stream of integers which are the working set size as it evolves over time. You will get this output and produce visual representations and statistics about the working set size. Specifically, you can use Unix utilities (like `gnuplot`) to plot the time series of the working set size and the frequency histogram of the working set sizes, while also reporting its mean and variance.

Variations on implementation quality

The minimum implementation of `valws379` should treat *windowsize* as successive, non-overlapping, sequences of references. Hence the output will be one value for each *windowsize* worth of references. If the final set of references is less than *windowsize* you do not report it, i.e., you only report "full" *windowsize* references. Note that when the option `-i` is provided, the eliminated references are completely ignored from the count building up to *windowsize*

The better (higher quality) implementation, produces a continuous stream of values for the size of the working set, based on a "sliding window" interpretation of *windowsize* and in this case you get as many values of the working set size as references (again discounting the ones eliminated if it ran with `-i`).

Independently of whether you implement the minimum or higher quality version, an additional indicator of quality is to avoid reading the entire trace of memory references before processing it. That is, you should be generating the output values of the working set, in a "streaming" fashion, i.e., producing them gradually as the input (output of `valgrind`) is delivered to your `valws379` process.

Front-end considerations

Exceptionally, in this assignment, you are allowed to write the `valws379` as a bash shell script which acts as a wrapper to the actual binary executable (`valws379.bin`). It is the binary where the working set dynamics need to be calculated. In other words, you are given some leeway on how you want to deal with various (e.g., input validation, and pipe plumbing) steps without writing them in `C` but as a shell script wrapper instead. If you choose to implement everything in `C` then `valws379` can be simply a symbolic link to `valws379.bin`.

Performance Study

The purpose of the implementation is to perform your own analysis of the performance of (a) a *few example programs* for, (b), different *input instance sizes*, (c), with or without the *instruction fetches* included, and for various combinations of (d) *windowsize* and (e) *pagesize*. You will be given the source code of the example programs, which you can compile and execute under `valgrind` and you are expected to explain, using the performance results you got, why is it that their working sets

behave in particular ways. You should use the plots of the time series of working set size evolution and the frequency histogram to support your analysis. The programs that will be provided to you are simple in that they will not involve any significant amount of I/O, i.e., they will be CPU-bound. Nevertheless you are free to write additional test cases by modifying the given programs or by writing completely different ones -- if this would help you explain the performance you see.

Deliverables

The language of implementation is C except for the (optional) wrapper as outlined above, which is a bash script.

Your assignment should be submitted as a single compressed (`zip` or `tar.gz`) archive file. The archive should contain all necessary source files as well as a `Makefile`. Calling `make` without any arguments should generate the `valws379.bin` executable. If there is no bash shell script wrapper, then the symbolic link `valws379` to `valws379.bin` must also be created. If there is a shell script wrapper, then it should be included in your submission archive.

Your submission must include the `DESIGN.md` (plain text or markdown format) document addressing how you addressed the quality issues mentioned above and what kinds of tools (and how) you used them to generate the performance result plots. A short file `README.md` (plain text or markdown markup) included with the archive to describe how you balanced the work across the two group members. Also indicate whether your code was tested (and running) on the VM or on the physical hosts in lab CSC 219.

Finally, your submission must include a performance study report as a `REPORT.pdf` file. Note that since you will know the source code of the example programs, you are expected to be insightful in the explanation of the working set performance you are observing.

You are expected to deliver good quality, efficient, code. That is, **the quality of your code will be marked**.

Monday, March 12, 2018