CT420
Assignment 2 – POSIX.4 Programming and Benchmarking

Outline:

In this assignment you will explore and benchmark some elements of the POSIX.4 real-time extension in a Linux environment and determine its limitations when used for a hard real-time system. Your experiments should be conducted on your own computer. Details about the POSIX API calls you are going to use can be found in the lecture notes on Canvas.

The assignment can be broken down into the following steps:

1. **Determine your host environment**
   Unless you already have a Linux system at hand (e.g., via CT420 assignment 1), you need to install a hypervisor on your computer. Popular choices for Windows users include Virtualbox and Hyper-V, while Mac users can avail of various other options (see https://machow2.com/best-virtual-machine-mac/). Note there is a free version of VMware Fusion that works with Apple's M processors.
   Create a single-vCPU VM with a minimum amount of RAM (i.e., 512Mbyte or smaller).
   In your report, document your host environment setup.

2. **Decide on Linux distribution**
   Assuming you have no Linux system at hand, pick a compact Linux OS for your experiments (you may adjust the size of VM RAM to meet minimum OS requirements). You may pick a distribution that implements the PREEMPT_RT patches (see lecture notes), which are designed to further enhance the real-time capabilities of the Linux kernel.
   In your report, please state if your chosen system implements these patches.
   Also, create a user / administrator account with your **full name** as user id, so that your full name appears on all terminal shell screenshots you will add to your report.
   Finally, validate your setup by retrieving the POSIX Clock Resolution (see lecture notes for example code).

3. **Determine CPU and data-intensive applications**
   In your experiments you will probe the real-time characteristics of your Linux system under different load conditions. Therefore, you need to develop some simple CPU- and data-intensive applications (implemented in any language or your choice), while also deciding on a resource monitoring tool that shows the overall utilisation (i.e., CPU load, RAM usage, memory swapping) of your system.
   In your report, add source code listings for the above applications (e.g., code that generates large prime numbers, or processes / sorts very large data arrays) and outline your chosen resource monitoring tool.

4. **Review and enhance the provided code**
   a. The provided source code files bm1.c and bm2.c allow the benchmarking of
      - signal handling,
      - interval timer,
      - the nanosleep() function.

      Enhance the source code, so that the usleep() function is benchmarked as well.
      Study, comment and possibly refactor / merge the source code (to simplify part 4.b. and step 5).

b. Enhance the source code so that all individual jitter / latency measurements are stored as a list in a CSV file **after** the completion of an experiment.
c. Determine how Excel (or any other spreadsheet software) can be used import a CSV file, to plot the data, and to calculate min, max, average and standard deviation.

5. **Plan and execute the benchmarking experiments**
The goal of this task is to determine the correlation between system performance / responsiveness and system load, via the following steps:
    a. Using your code in 3., configure background processes that result in the following experiments:
        - Low system load and no additional memory swapping (i.e. no additional background processes)
        - Medium CPU load and no additional memory swapping
        - High CPU load and no additional memory swapping
        - Medium CPU load and extensive memory swapping
        - High CPU load and extensive memory swapping
    b. Using the above configurations, run benchmark experiments for signal handling, interval timer, nanosleep() and usleep(), collecting 10,000+ data points per experiment, and analyse the data as outlined in 4.c.
    c. Repeat your experiments, but this time with memory locking disabled, and analyse the data as outlined in 4.c.

6. **Data analysis**
    a. Contrast and compare the findings of your experiments in 5.b. and 5.c., i.e. real-time performance versus system load.
    b. Determine configurations / system loads that lead to the deterioration of the system's performance (in terms of jitter or latency) and potentially the observation of significant outliers, thereby distinguishing between signal handling, interval timer, nanosleep() and usleep() results.

7. **Report**
Summarise your work in a report that contains (in a suitable order)
    - details as outlined in steps 1., 2., and 3.,
    - screenshots of your code in action (showing your full name in the Linus shell, as outlined in 2.)
    - all source codes as outlined in steps 3., 4.a. and 4.b
    - an overview of your experiments as outlined in step 5.a.
    - your results in step 6.a. and 6.b.

8. **Submission**
Please submit:
    a. Your report in PDF format, but uncompressed
    b. All source code files in a compressed folder
    c. All data files in a compressed folder

Marking Scheme (out of 20 marks):
- 5 marks for well formatted / commented / refactored source code.
- 3 marks for outlining your experiments and design decisions (i.e., what system loads did you consider and how did you get there).
- 12 marks for a comprehensive data analysis & visualization, and discussion of the correlation between system performance / responsiveness and system load.