# Lab 6: Operating System Performance Measurement

**IUPUI CSCI 503 – Fall 2013**
**Assigned**: November 18, 2013
**Due time**: 11:59PM, December 6 (Friday), 2013

## 1. Introduction

You will use high-resolution hardware performance counters to measure the overhead of several common operating system activities. First, you need to go to http://icl.cs.utk.edu/papi/software, download and install the latest PAPI library (version 5.2.0 as I last time checked on 11/15/2013) under your home directory (on Linux machines). You will use the function `PAPI_get_real_usec()` to get the total real-time passed. Look at the online documentation to learn how to use the PAPI library and how to call the function.

After knowing how to make a PAPI call, you will write programs to measure the following four overheads on the Linux operating system (without using virtual machines):

1. **Function call overhead** in C. It is the minimum time to invoke a function. You may want to measure a dummy function that takes no argument and does nothing.
2. **System call overhead**. You may want to measure system call of either `getpid()` or `time()` to measure the overhead.
3. **Process context-switch overhead**
   Basic idea: You will create two processes: Parent and Child. The two processes (i.e., Parent and Child) will communicate through two pipes: one pipe for communication from Parent to Child, the other pipe for communication from Child to Parent. In your program, on the one hand, Parent writes a single-byte message to Child and then tries to read it back from Child. On the other hand, Child tries to read a message from Parent and sends it back to Parent immediately. You should repeat this process many times (e.g., 1000). Moreover, you need to think about how many context switches have happened during your experiment.
4. **Virtual-memory page fault overhead**
   Basic idea: (1) create two arrays with the same size as the physical memory (e.g., 4GB). (2) Initialize them with random numbers. (3) After initialization, read data from the first array. The distance between two consecutive reads are at least 4 KB (page size), and also the two consecutive reads should be far away in order to avoid data prefetching. (4) Try to read many pages (e.g, 500).

**Measurement methodology:**

- Make sure when you run your experiments, there is no other user using the same machine as you use (run "top" to check it)

- The measured or targeted overhead should be included in a loop. The start timer and stop timer are immediately before and after the loop. The result you reported should be an average value.
- At the end of each test, print out the measured overhead in terms of both time and cycles.

## 2. Summary of your measurements

You should run your codes on two different Linux machines and write down a summary of your measurement. In the summary, you first report the machine's name, CPU type, and memory size. Then report your performance results in 4 rows. Each row corresponds to one overhead. Each machine has a table.

| Overhead Name | Time (in milliseconds) | Number of cycles |
|---|---|---|
| --- | --- | --- |

## 3. Score distribution

- **10%**: coding style and error checking (as described in the "Labs Grading Policy")
- **20%**: able to use PAPI functions to measure elapsed time
- **10%**: function call overhead
- **10%**: system call overhead
- **20**%: context switch overhead
- **20%:** page fault overhead
- **10%:** measurement summary on two different Linux machines

If there is a "Segmentation Fault" when testing your code, then TA will deduct 20 points from your score.

The total score is 100 points.

## 4. Deliverable

You should submit source code in C, a Makefile, a performance summary, and a README. Make a tar ball and send it to the TA. The tar ball name may be "Lab6_your_name.tar". You might want to have four separate main programs for the four types of measurement.

## 5. How TA will grade

- Will check whether your program is using PAPI functions or not.
- Will run your program and check whether your measurement is reasonable or not. "Reasonable" means if the magnitude of your measurement is nearly correct (e.g., within 10 cycles, within 100 cycles, within 1000 cycles, and so on). If your code is running but does not measure correctly, you can get at most 2/3 of the assigned grades.
- Will check whether your summary of measurements provides complete information.