CS3500: Operating Systems

Tutorial for Lab 7

Date: 19th Sep 2019

In this week's and next week's labs we will focus on virtual memory. In this week's tutorial, you will learn some common ideas that will apply across both these labs.

1. As discussed in class, there are very few operating systems that support multiple page sizes due to the resultant complexity in address translation. Usually, all default pages have a constant size. Your first exercise is to find out the page size on your machine.
   Hint: Look for a shell command that provides this. Once you identify the shell command, look at the detailed `man` page to understand the available parameters.

2. In the previous labs, you had attempted to estimate elapsed time between certain code blocks using calls such as `gettimeofday()`. However, you had soon realised that the time resolution is poor. In particular, when you are interested in measuring the time taken by a few lines of assembly code, then introducing the overhead of system calls seems an overkill. Instead, use the following `rdtsc` function to measure time, before and after blocks of code which are to be profiled. Do some reading/thinking about this function. For example, why is it a static inline function, why the keyword volatile, what does the `rdtsc` instruction do?

```
#if defined(__i386__)
static __inline__ unsigned long long rdtsc(void)
{
    unsigned long long int x;
    __asm__ volatile("xorl %%eax,%%eax\n cpuid \n" ::: "%eax", "%ebx", "%
        ecx", "%edx"); // flush pipeline
    __asm__ volatile (".byte 0x0f, 0x31" : "=A" (x));
    __asm__ volatile("xorl %%eax,%%eax\n cpuid \n" ::: "%eax", "%ebx", "%
        ecx", "%edx"); // flush pipeline
    return x;
}
#elif defined(__x86_64__)
static __inline__ unsigned long long rdtsc(void)
{
    unsigned hi, lo;
    __asm__ __volatile__ ("xorl %%eax,%%eax\n cpuid \n" ::: "%eax", "%ebx
        ", "%ecx", "%edx"); // flush pipeline
    __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
    __asm__ __volatile__ ("xorl %%eax,%%eax\n cpuid \n" ::: "%eax", "%ebx
        ", "%ecx", "%edx"); // flush pipeline
    return ( (unsigned long long)lo)|( ((unsigned long long)hi)<<32 );
}
#endif
```

After the reading, do some profiling. Write any small piece of code and call the above function before and after it to compute the time taken.

3. In basic stats, you would have studied about the *central limit theorem*. If not, see the nice video about it on Khan academy. When profiling pieces of code, the time elapsed varies - due to numerous reasons. A natural question is how many times to run the measurements together to obtain a statistically robust estimate of time taken. Let us illustrate this with an example. Write a program that allocates 26 characters with `malloc`, then write the string "abcd...xyz" of 26 characters to the allocated space. Then free the allocated memory. Wrap this code in a `for` loop with N iterations. Wrap this `for` loop with `rdtsc` calls to measure the time taken by N calls. Now, we want to understand how the elpased time varies with N. Vary N in the range of 1 to 10,000 and plot the elapsed time in y-axis while N is in the x-axis. What do you observe? What conclusions would you draw on profiling this code?

   For a quick and neat way to plot, you can use TikZ:

```
\documentclass{standalone}
\usepackage{tikz,pgfplots,pgfplotstable}
\begin{document}
\pgfplotstableread[col sep = comma]{data.csv}\loadedtable
\begin{tikzpicture}
    \begin{axis}[xlabel=N, ylabel=Time]
            \addplot[scatter, only marks] table [x=column 1, y=column 2,
               col sep=comma] {\loadedtable};
    \end{axis}
\end{tikzpicture}
\end{document}
```

---

4. In class, we disussed briefly "hugepages". Let us experiment with them. Understand how to check if your system supports hugepages by looking at the file `/proc/meminfo`. What is the supported hugepage size for your system? What is the number of available hugepages in your system? Learn how to configure the kernel to modify the number of huge pages and then to reload the configuration (introducing `sysctl`).

5. We have already seen the `mmap` system call. Learn how to use this to create a chunk of 5 hugepages. After you have created the hugepages, check the number of free hugepages on your system.