CS 416 - Operating Systems Design

Project 0
Measuring the cost of OS invocations

September 9, 2016
Due September 23rd, 2016

1 Introduction

In this project, you will measure the cost (in time) of two common cases of operating system invocations: the **system call** and the **signal**.

A **system call** is an invocation of the operating system via a special **trap**, or software interrupt. Examples of common system calls are **read()**, **write()** and **getpid()**.

A **signal** is a mechanism for delivering an asynchronous event or notification to a process. Common reasons for a process to receive a signal are memory protection violations (**SIGSEGV**), divide by zero error (**SIGFPE**), and an expired timer (**SIGALRM**). Signals interrupt the normal flow of execution of a program and are generally *handled* by a custom signal handler function.

2 Description

Both system calls and signals happen very quickly in wall clock time. Indeed, they happen so quickly that a typical computer's clock is not accurate enough enough to reliably measure the elapsed time. To overcome this issue, you should instead measure the time cost of a large number of invocations (e.g. 100,000), and divide this cost by the number of invocations to come to a result. You can use system call **gettimeofday()** to calculate the elapsed time.

2.1 System Call

Implement a C program called **time-syscall.c** that measures the time taken to invoke the system call **getpid()** 100,000 times. Use this information to calculate the mean system call time.

Report the results in *exactly* the following format:

```
Syscalls Performed: XXXX
Total Elapsed Time: XXXX ms
Average Time Per Syscall: XXXXX ms
```

2.2 Signal

Implement a C program called **time-signal.c** that measures the time taken to handle an exception and invoke a signal handler 100,000 times. You can accomplish this by using the **signal()** system call to register a handler function for the signal **SIGFPE**, then causing a divide by zero exception.

A handler for **SIGFPE** can be registered in the following manner:

```
#include <signal.h>

void handle_sigfpe(int signum)
{
    // Handler code goes here
}

int main(int argc, char **argv)
{
    int x = 5;
    int y = 0;
    int z = 0;

    signal(SIGFPE, handle_sigfpe);

    z = x / y;     // This causes the exception

    return 0;
}
```

Please note that if a signal handler function is allowed to return, the instruction that caused the signal will execute again, therefore causing the exception to occur again, *ad infinitum*. Your signal handler will need to detect when it has been invoked 100,000 times, then report the results and terminate the program. You can use a **static** variable for this purpose.

Report the results in *exactly* the following format:

```
Exceptions Occurred: XXXX
Total Elapsed Time: XXXX ms
Average Time Per Exception: XXXXX ms
```

3 Requirements

- The code has to be written in **C** language. No C++, no exceptions.
- You should also implement a **Makefile** that compiles your two programs.
- To submit, create a tarball containing your two programs as well as your **Makefile**, and submit it via **Sakai**.
- Do not copy the solution from other students. Use **Sakai** to discuss ideas of how to implement it.
- Submit a report in PDF form on **Sakai** detailing what you accomplished for this project, including issues you encountered.