

## Assignment 4

ED. KAZMIERCZAK

SECOND SEMESTER, 2003

### Introduction

Software whose performance relies heavily on some factor involving time needs to be developed with reasonable estimates of the timing involved in the various operations which comprise the system. For hard real-time systems this time factor is often more critical.

The main focus of this assignment will be to understand time measurement experimentally. The experiment will demonstrate measuring time by software to reinforce the concept of time and the important features that need to be understood about measuring time in applications. Finally, it will provide pointers to other methods that can be used to measure the time and performance of applications.

### Aims

The aim is to measure the uncertainty with which time can be estimated in a system. For the purpose of this assignment, you will use the **system call** `gettimeofday()`. The `gettimeofday()` call is a system call originating in the BSD4.3 version of UNIX. (The **man** command can be used to obtain more information on *system calls*). It returns a `timeval` structure:

```
struct timeval
{
    int tv_sec;
    int tv_usec;
};
```

which contains the number of seconds and microseconds that have elapsed since January 1, 1970 at 00:00 (the time that is considered the birth of UNIX).

From the format of the `timeval` structure it is apparent that this system call will be no more accurate than 1 microsecond. However, it would be nice to know just how accurate this call is.

### Background

If you examine the code in the sample program for determining the resolution of the `gettimeofday()` call, several interesting points can be noted.

- In order to obtain the most accurate results, the only item performed in the first for loop is the `gettimeofday()` call.
- Given the uncertainty of software timing, whenever we perform a measurement such as this, it is important to perform only the minimum amount of calculations necessary to make the measurement (in this case, the incrementing of the for loop) in order to keep from affecting the value that is being measured.

Next, two values are calculated:

- The first is the average time that it takes to complete the call, found by taking the sum total of all the times and dividing it by the number of calls.
- The second is the maximum time taken to complete the call, found by looking at each value and comparing it the previous maximum value to determine which is higher, the highest value being stored.

The displayed values are only given in microsecond precision, because the `gettimeofday()` call only returns a value of microseconds, so anything beyond that is bound to be *uncertain*.

The **load** of a UNIX system is defined as the number of processes waiting to run at any given time. Although a UNIX system may have many processes running at once, most of these will be waiting for user input or network traffic and only a small number will actually be competing for CPU time. A load of 1 indicates that only one process was waiting for CPU time last time the scheduler checked. A load of 2.5 means that on average between two and three processes are waiting for CPU time.

UNIX utilities such as `top` and `uptime` can report the average system for the last one, five and fifteen minutes - you will need to use these utilities to record the system load when you execute the program provided for this assignment.

Information about the CSSE server machines can be found at [www.cs.mu.oz.au/systems/hosts.html](http://www.cs.mu.oz.au/systems/hosts.html).

## Questions and Tasks

### Working With the Sample Program

The first part of the assignment is to compile and execute the program on at least two different operating systems and machines. Run the program on a Windows machine which should have a relatively light load and should provide consistent results. Then run the program on one or two of the UNIX student machines. These machines have a heavy load and will probably have very inconsistent results. In addition the "worst case" will probably be far from the "average." Record your results in a chart as follows:

CPU Type and Speed	Operating System	Number of Users	Load	Mean	Worst

## Task 1

*2 hours* Run the sample program on two Windows machines (one with Windows 98 and another with Windows 2000) and at least four different UNIX (including Linux) machines and record the results. The program will report the average time taken to call `gettimeofday()` and the maximum time taken - you should run the program three times on each machine and record each result. Your final table of results should have at least 18 rows.

## Task 2

*1 hour* Explain why times vary on a single machine and between machines. Explain the effects of:

- Operating System
- Processor
- System Load
- Network Traffic
- Other non-deterministic events.

## Additional Information

There are several other ways to measure the performance of applications. Many UNIX's provide a method by which to compile information into the actual program which can then be used to keep performance statistics. This is called **profiling** and can be found in the UNIX documentation (see the man pages for `pixie` or `prof`).

If you wish you can also compute the standard deviation for the timing.