Introduction

It is sometimes desirable to gather experimental evidence about the running time of a program and/or one or more components of a program.

Timing The Entire Program

We can use the `time` command to obtain the amount of time it takes a program to run to completion. For example, to measure the amount of time it takes for `myprogram` to run to completion, give the command:

```
    time ./myprogram
```

or, to suppress output from the program:

```
    time ./myprogram > /dev/null
```

Timing One Component

Timing the execution of an entire program is simple, but it doesn’t let use compare two different functions within the same program or separate out any initialization time from the execution of an algorithm (such as reading input from a file then sorting). This is something that would be of interest when comparing two different algorithms.

To obtain this finer control over the timing, we can use the `clock()` function provided with Linux. Its use is documented in section 3 of the Linux manual—and is reproduced in the appendix of this document for convenience.

The `clock()` function provides a mechanism for measuring elapsed time similar to the clocks we find from ordinary life. To measure how long it takes to do a task:

- jot down the current time as shown on the clock
- perform the task
- note the current time
- calculate the elapsed time by forming the difference of the two observed times

We can make this easier to use by encapsulating these ideas in the `Stopwatch` class. This class allows us to use the paradigm of a stopwatch to measure the duration of an event: reset the stopwatch at the moment the event begins, then stop the stopwatch at the end of the event.

Refer to the appendix for coding details and an example of the use of the `Stopwatch` class.
Appendix

Contents of stopwatch.h

```c++
#ifndef STOPWATCH_H
#define STOPWATCH_H

#include <ctime>

class Stopwatch {
public:
    Stopwatch(); // Create a stopwatch object with the current time
    void reset(); // Reset the stopwatch to the current time

private:
    clock_t initialObservedTime;
};
#endif // STOPWATCH_H
```

Contents of stopwatch.cpp

```c++
#include "stopwatch.h"
#include <ctime>

// Refer to the manual page for "clock" (section 3) for more details
// about the clock() function.

Stopwatch::Stopwatch () {
    // initialize a new Stopwatch object,
    // setting its value to the current
    // time, as reported by the system clock.
    initialObservedTime = clock();
}

void Stopwatch::reset () {
    // reset the stopwatch to the current time
    initialObservedTime = clock();
}

float Stopwatch::elapsedTime () {
    // determine the amount of CPU time since the last observation,
    // then convert to seconds.
    return ((float) (clock() - initialObservedTime))/CLOCKS_PER_SEC;
}
Contents of test-stopwatch.cpp

```cpp
// Demonstration of the stopwatch class

#include <iostream>
#include <iomanip>
#include <cmath>
#include "stopwatch.h"

using namespace std;

void DoSomeWork(int N) {
    // Perform N iterations
    for (int i=0; i < N; i++)
        float x = sqrt((float)i);
    return;
}

int main() {
    const int LowBound = 500000;
    const int HighBound = 512*HighBound;
    Stopwatch s;
    for (int i = LowBound; i <= HighBound; i *= 2) {
        // start the stopwatch
        s.reset();
        // perform i iterations of finding a square root
        DoSomeWork(i);
        // stop the stopwatch and report the time
        float t = s.elapsedTime();
        cout << "Elapsed time for " << setw(9) << i << " iterations is "
             << setw(6) << setprecision(2) << setiosflags(ios::fixed) << t
             << " seconds" << endl;
    }
    return 0;
}
```

Results of test-stopwatch (450 Mhz Pentium III)

```
[cfwas@localhost stopwatch]$ ./test-stopwatch
Elapsed time for 500000 iterations is 0.16 seconds
Elapsed time for 1000000 iterations is 0.32 seconds
Elapsed time for 2000000 iterations is 0.63 seconds
Elapsed time for 4000000 iterations is 1.25 seconds
Elapsed time for 8000000 iterations is 2.50 seconds
Elapsed time for 16000000 iterations is 5.00 seconds
Elapsed time for 32000000 iterations is 10.09 seconds
Elapsed time for 64000000 iterations is 20.25 seconds
Elapsed time for 128000000 iterations is 40.26 seconds
```
NAME
clock − Determine processor time

SYNOPSIS
#include <time.h>

clock_t clock(void);

DESCRIPTION
The clock() function returns an approximation of processor time used by the program.

RETURN VALUE
The value returned is the CPU time used so far as a clock_t; to get the number of seconds used, divide by
CLOCKS_PER_SEC. If the processor time used is not available or its value cannot be represented, the
function returns the value (clock_t)-1.

CONFORMING TO
ANSI C. POSIX requires that CLOCKS_PER_SEC equals 1000000 independent of the actual resolution.

NOTES
The C standard allows for arbitrary values at the start of the program; subtract the value returned from a call
to clock() at the start of the program to get maximum portability.

Note that the time can wrap around. On a 32bit system where CLOCKS_PER_SEC equals 1000000 this
function will return the same value approximately every 72 minutes.

On several other implementations, the value returned by clock() also includes the times of any children
whose status has been collected via wait() (or another wait-type call). Linux does not include the times of
waited-for children in the value returned by clock(). The times() function, which explicitly returns (sepa-
rate) information about the caller and its children, may be preferable.

SEE ALSO
getrusage(2), times(2)