Timing Code

Sometimes it is useful to time the operation of code. This allows testing of different algorithms.

There are two easy ways to time operations in the Linux/Unix world. The `time` and `date` commands can be used. The `date` command is typically too coarse for exact timing.

Using `time` and `date`

```
$ time cmd

$ time ls

listing

real  0m0.004s
user  0m0.001s
sys   0m0.002s
```

Note that `time` shows real, user, and system time elapsed.

```
$ date ; cmd ; date

$ date ; ls ; date
Sat Nov  7 08:27:01 PST 2020

listing

Sat Nov  7 08:27:01 PST 2020
```

Check date before and after the command to be timed.

Timing Inside Code

The simplest way to actually time the operation of a section of code is to use the `clock()` function. To time the code, `clock()` is called before and after the section to be timed, then the difference is calculated and displayed.

Using `clock()`

```
/* timer0.c

 Reference:
   https://en.cppreference.com/w/c/chrono/clock_t
 */

#include <stdio.h>
#include <time.h>

volatile unsigned sink;

int main(void)
{
  clock_t start = clock();

  for( size_t i = 0 ; i < 10000000 ; ++i )
     sink++;

  clock_t end = clock();
  double cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

  printf("for loop took %lf seconds to execute \n", cpu_time_used);

  return 0;
}
```

Note: `size_t` is the unsigned integer type of the result of `sizeof`.
https://en.cppreference.com/w/c/types/size_t

Sample Output

```
$ time ./a.out
for loop took 0.020778 seconds to execute

real    0m0.025s
user    0m0.022s
sys     0m0.002s
```

Note that the *user* time is approximately the same as the time consumed by the `for` loop, since that's about all that is done during execution.

`clock()`

The C library function `clock_t clock(void)` returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, you will need to divide by `CLOCKS_PER_SEC`.

On a 32 bit system where `CLOCKS_PER_SEC` equals 1000000 this function will return the same value approximately every 72 minutes.

Reference
https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm

Another example using `clock_gettime()`

```
/* timer1.c

 Reference:
  https://man7.org/linux/man-pages/man3/clock_gettime.3.html
 */

#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include <limits.h>


int main()
{
    struct timespec elapsed_from_boot;

    printf( "LONG_MAX: %ld\n", LONG_MAX );   // Value?  in limits.h
    SecsToDays( nSecs );

    clock_gettime(CLOCK_MONOTONIC, &elapsed_from_boot);   // macOS
    //clock_gettime(CLOCK_BOOTTIME, &elapsed_from_boot);  // Linux

    printf( "%ld - seconds elapsed from boot\n", elapsed_from_boot.tv_sec );
    //printf( "%d - seconds elapsed from boot\n", elapsed_from_boot.tv_sec );
    long nSecs = elapsed_from_boot.tv_sec;
    SecsToDays( nSecs );

    return 0;
}
```

Sample Output

```
$ ./a.out
LONG_MAX: 9223372036854775807
579025220 days 15:30: 7
677014 - seconds elapsed from boot
  7 days 20: 3:34
```

Using the `chrono` class

C++ has a class, **chrono**, for better timing. See

http://www.cplusplus.com/reference/chrono/

```
/* chrono.cpp

  Three clocks:
   o system_clock
   o steady_clock
   o high_resolution_clock
 */

#include <iostream>

using namespace std;
```

```
int main()
{
      // compare the clocks

   cout << "chrono::system_clock::period" << endl;
   cout << chrono::system_clock::period::num << "/" <<
           chrono::system_clock::period::den << endl;

   cout << "chrono::steady_clock::period" << endl;
   cout << chrono::steady_clock::period::num << "/" <<
           chrono::steady_clock::period::den << endl;

   cout << "chrono::high_resolution_clock::period" << endl;
   cout << chrono::high_resolution_clock::period::num << "/" <<
           chrono::high_resolution_clock::period::den << endl;

   chrono::microseconds muSec(5000);
   chrono::nanoseconds  nSec = muSec;
   chrono::milliseconds mSec =
   chrono::duration_cast<chrono::milliseconds>(muSec);

   cout << "nSec:  " << nSec.count()  << endl;
   cout << "muSec: " << muSec.count() << endl;
   cout << "mSec:  " << mSec.count()  << endl;

   chrono::system_clock::time_point tp = chrono::system_clock::now();
   cout << "tp.time_since_epoch().count(): ";
   cout << tp.time_since_epoch().count() << endl;

   tp = tp + chrono::seconds(1);  // one second later

   cout << "tp.time_since_epoch().count(): ";
   cout << tp.time_since_epoch().count() << endl;

   return 0;
}
```

Sample Output

```
$ ./a.out
chrono::system_clock::period
1/1000000
chrono::steady_clock::period
1/1000000000
chrono::high_resolution_clock::period
1/1000000000
nSec:  5000000
muSec: 5000
mSec:  5
tp.time_since_epoch().count(): 1604770762340881
tp.time_since_epoch().count(): 1604770763340881
```

Note that the `2` changes to a `3` for one second – high resolution! – *good enough* for our usage.