Accuracy of Run Time Measurement

Contents

- Properties of digital clocks
- Errors of digital clocks
- Error compensation techniques
- Timing of short programs

Resource Consumption

- Processor Time
  - Processor utilization ($U=U_0+U_1$, $0 \leq U \leq 1$ or 100%)
    - Activity for user (U_1)
    - Activity for operating system (U_0)
  - Processor idling (1-U)
- Memory Space
- Disk Space
- Communication Bandwidth

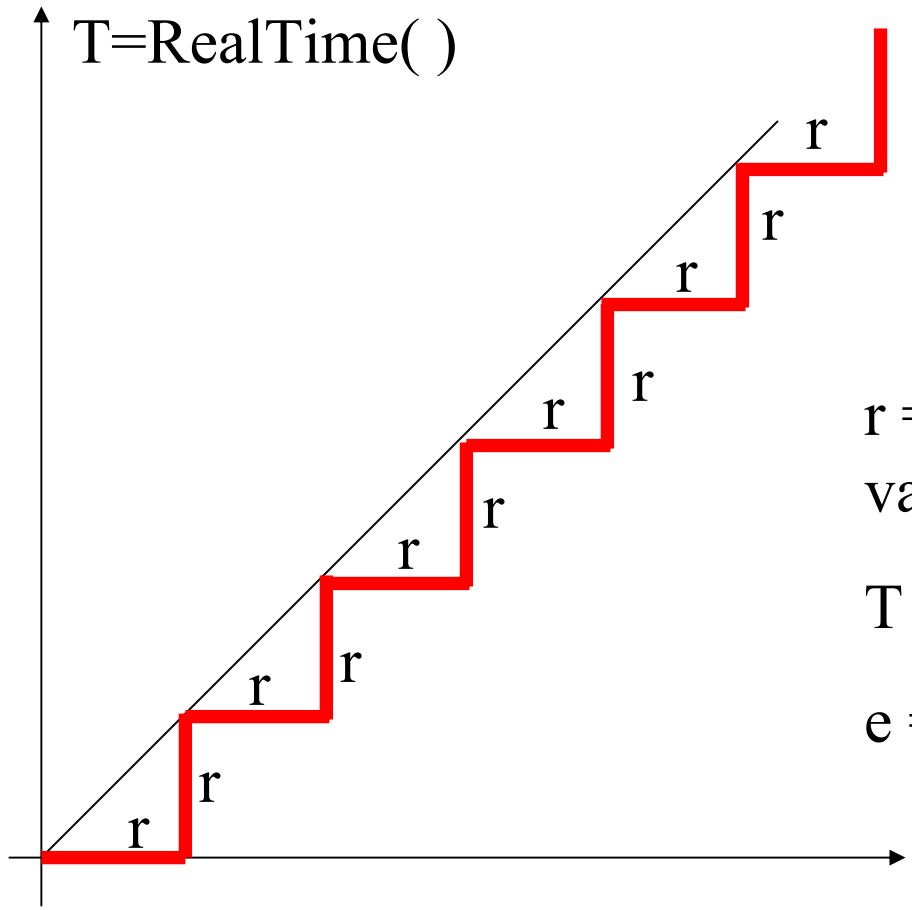
Measurement of Time

- Based on hardware device (Real Time Clock, RTC)
- RTC is incremented each *r* time units
- *r* is called clock resolution
- Reading RTC is realized by a special library function that will be denoted RealTime( )

Time Measurement Functions

- RealTime( ) = function that returns physical time (in seconds) from some moment in the past (or from the moment of starting computer)
- CPUtime( ) = function that returns the processor time accumulated by a process (program in execution)
- RealTime and CPUtime are generic names; actual names depend on language.

Output of Real Time Clock



r = clock resolution (typical value: r = 10 msec)

$T = t - e, \quad 0 \le e < r$

e = error

Program for Measuring *r*

```
#include <iostream.h>
#include <time.h>
#define N 1000

double sec(clock_t& c)
{return double(c=clock())/CLOCKS_PER_SEC;}

void main(void)
{ double T1, T2, r=0., c, cave=0., cmin=999999., cmax=0.;
  clock_t c1, c2;
  for(int i=1; i<=N; i++)
  {
     T1 = T2 = sec(c2); c1 = c2;
     while(T2 == T1) T2 = sec(c2);      // Wait for the change of clock
     r += (T2-T1) ;  cave += (c=c2-c1);
     if(c>cmax) cmax=c; else if(c<cmin) cmin=c;
     if(N-i<11) cout << "\nr = " << T2-T1 << " sec  Clock increment = "<< c;
  }
  r /= N;  cave /= N;
```

Program for Measuring *r* (Cont.)

```
  cout << "\n\nMS VC++6.0, WinXP results for N = " << N
       << " iterations\nCLOCKS_PER_SEC = " << CLOCKS_PER_SEC
       << "\nClock increment min,ave,max = "
       << cmin << " , " << cave << " , " << cmax
       << "\nr = " << r << " sec,  1/r = "
       << 1./r << " increments per sec\n";
}
```

Results for 1.7 GHz Pentium – Win XP

```
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.016 sec  Clock increment = 16
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.016 sec  Clock increment = 16
r = 0.015 sec  Clock increment = 15
r = 0.016 sec  Clock increment = 16
r = 0.016 sec  Clock increment = 16
MS VC++6.0, WinXP results for N = 1000 iterations
CLOCKS_PER_SEC = 1000
Clock increment min,ave,max = 15 , 15.625 , 16
r = 0.015625 sec,  1/r = 64 increments per sec
```

Results for 1.7 GHz Pentium – Linux

```
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
r = 0.01 sec  Clock increment = 10000
Linux results for N = 1000 iterations
CLOCKS_PER_SEC = 1000000
Clock increment min,ave,max = 10000 , 10000 , 10000
r = 0.01 sec,  1/r = 100 increments per sec
```

Results for 200MHz Pentium – Win 98

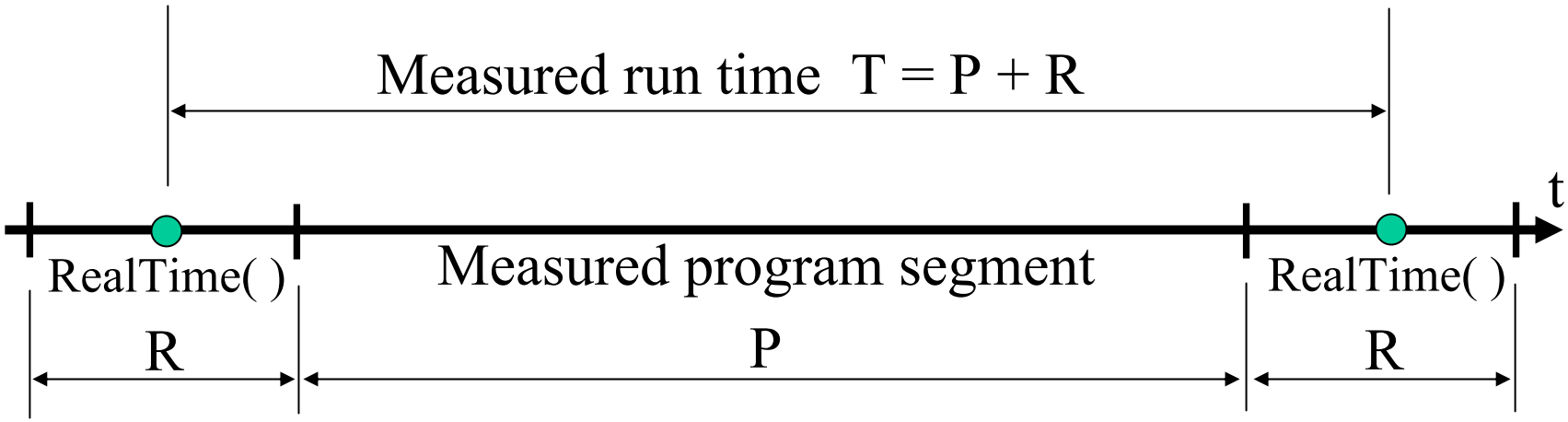
```
r = 0.06 sec  Clock increment = 60
r = 0.05 sec  Clock increment = 50
r = 0.06 sec  Clock increment = 60
r = 0.05 sec  Clock increment = 50
r = 0.06 sec  Clock increment = 60
r = 0.05 sec  Clock increment = 50
r = 0.06 sec  Clock increment = 60
r = 0.05 sec  Clock increment = 50
r = 0.06 sec  Clock increment = 60
r = 0.05 sec  Clock increment = 50
r = 0.06 sec  Clock increment = 60
```

```
MS VC++6.0, Win98 results for N = 1000 iterations
CLOCKS_PER_SEC = 1000
Clock increment min,ave,max = 50 , 54.93 , 60
r = 0.05493 sec,  1/r = 18.205 increments per sec
```

Measurement of Run Time for Relatively Long Programs

```
double T1, T2, T;
T1 = RealTime( );
      // Program segment that is being measured
T2 = RealTime( );
T = T2 – T1;
cout << “\nRun time = “ << T;
// T is sufficiently accurate only if T >> run time of RealTime( )
```

Measurement Error



RealTime( ) reads the contents of a RTC register at some moment during the execution of this function. Since T=P+R the measurement error is e = 100 (T-P)/P = R/P [%].

For small values of P the error can be rather large.

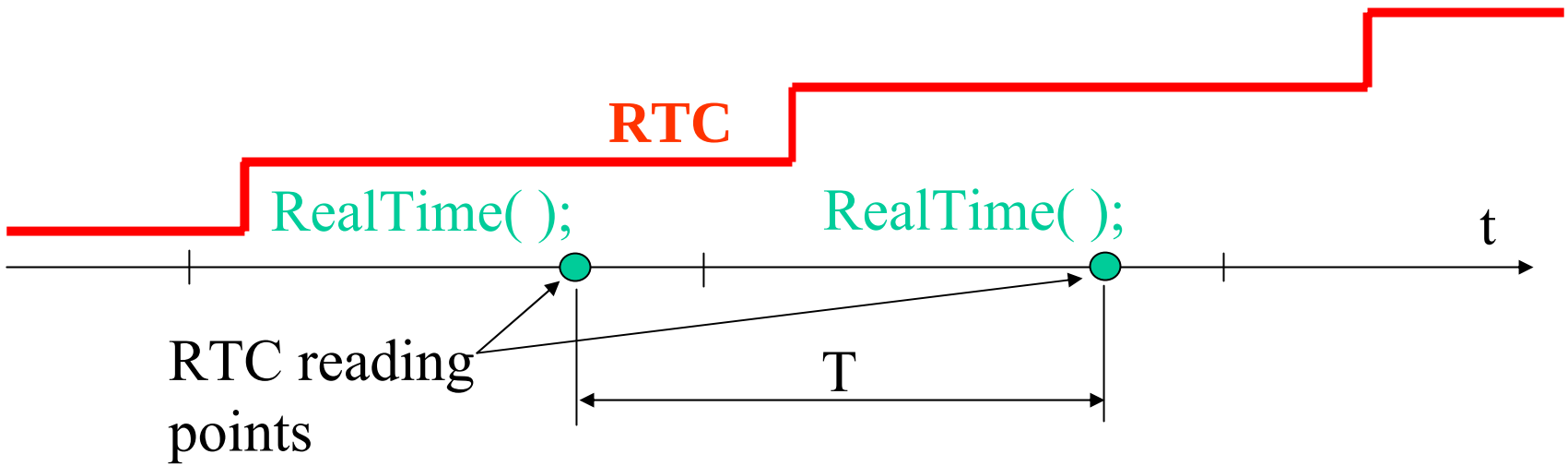
Measurement of Run Time for Very Short Programs

As the clock rate of processors increase it becomes more and more important to be able to measure the run time of very short programs. These are programs having run time shorter than *r*.

As an example of a very short program let us measure the run time of the RunTime( ) function.

Run Time of Function RealTime( )

```
double T1, T2 ;
T1 = RealTime( );
T2 = RealTime( );
cout << "\nRun time = " << (T2 – T1);    // R is either 0 or r
```



The measured value is mostly 0 and infrequently *r*. If the measurement is repeated many times inside a loop, then the resulting mean value will be close to the correct value.

```
double T1, T2, R=0.;
long N = 100000000;
while(N--) {
        T1 = RealTime( );
        T2 = RealTime( );
        R += T2 – T1;
}
cout << R/N;
```

Repetition Method

The idea behind this method is simple: to accurately measure very short programs they must be repeated many times, as follows:

```
T1 = RealTime( );
for(i=1; i<N; i++)
{
   T2 = RealTime( );
}
T2 = RealTime( );
cout << (T2 – T1)/N ;
```

Is (T2 – T1)/N an accurate value of the run time of the statement T2 = RealTime( )?

No.

(T2 – T1)/N reflects the run time of T2 = RealTime( ) plus the time necessary to service the for loop. This would be acceptable only if the loop time is negligible.

What Is the Difference Between RunTime1 and RunTime2?

```
T1 = RealTime( );
for(i=1; i<N; i++)
{
      T2 = RealTime( );
}
T2 = RealTime( );
RunTime1 = T2 – T1 ;
```

```
T2 = RealTime( );
for(i=1; i<N; i++)
{
      T3 = RealTime( );
      T3 = RealTime( );
 }
 T3 = RealTime( );
 RunTime2 = T3 – T2 ;
```

RunTime2 – RunTime1 = time to execute N times the statement T3=RealTime( );

We can use these two program segments as two parts of the same program and compute the run time of the statement T3=RealTime( ); as follows:

$$T = (RunTime2 - RunTime1) / N$$

$$= ((T3 - T2) - (T2 - T1)) / N$$

$$= (T3 - 2*T2 + T1) / N$$

Is this result now perfectly accurate?

$(T3 - 2*T2 + T1) / N$ would be perfectly accurate only if T1, T2, and T3 were perfectly accurate. However, due to the resolution of RTC, these values can be less than the correct physical times t1, t2, and t3. Therefore,

$$T1 = t1 - e1 \qquad T2 = t2 - e2 \qquad T3 = t3 - e3$$

The errors e1, e2, and e3 are between 0 and the RTC resolution *r* :

$0 \le e1 < r$, $0 \le e2 < r$, $0 \le e3 < r$. Therefore,

$$T = (T3 - 2*T2 + T1) / N = (t3 - 2*t2 + t1) / N - (e3 - 2*e2 + e1) / N$$

The accurate value is $t = (t3 - 2*t2 + t1) / N$ and the corresponding error is $e = (e3 - 2*e2 + e1) / N$. The maximum possible error emax=2r/N is obtained if e1=e3=r and e2=0, or if e1=e3=0 and e2=r. Therefore, the maximum relative error is approximately **E = 2r/NT** .

Why "approximately"? Because we use T as an approximation of t.

Example: Find N so that the error is less than 0.1% for r = 10 msec

To achieve **2r/NT < Emax** we must use N > **2r/TEmax** .

To measure T we must know N and to select N we must know T. Chicken-and-egg problem? Yes – to solve this problem we first select a value of N and compute the error E. If E>Emax we must increase N, to achieve accuracy better than Emax. Suppose that we found sufficient N and that the resulting time is T=100 μsec. Then,

$$2\,r / T\,Emax = 0.02 / 0.0001 \times 0.001 = 200000$$

Therefore, to achieve error that is less than 0.1% we must use

$$N > 200000$$

The Cost of Desired Accuracy Is an Expanded Measurement Time

In the case of the expanded repetition method the run time of the measurement program is

$$Tm = N[3 \times T + 2 \times (\text{loop service time})]$$

$$= N[3 \times T + 2 \times ((T2-T1)/N - T)]$$

If T = 100 μsec, N > 200000, and loop service time of 10 μsec, the the total measurement time is

$$Tm = 200000[0.0003 + 0.00002] = \underline{64 \text{ sec}}$$

Uncertainty Principle: Exact Monitoring of Computer Performance Using Software Monitors Is Not Possible Because The Act of Observing Interferes With the Observation

```
T1 = RealTime( );
for(i=1; i<N; i++)
{
      T2 = RealTime( );
}
T2 = RealTime( );
RunTime1 = T2 – T1 ;
```

To measure the CPU time we need additional CPU time. Disk monitoring causes slower disk accesses. In addition, measurement of time is based on clocks that usually have rather poor accuracy.

Indirect measurement techniques are necessary to eliminate noise, compensate errors, and extract desired values with appropriate accuracy.