

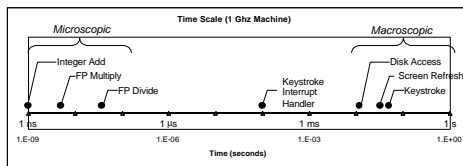
Time Measurement

These slides based on some provided by the authors of our text.

Topics

- Time scales
- Interval counting
- Cycle counters
- K-best measurement scheme

Computer Time Scales



Two Fundamental Time Scales

- Processor: $\sim 10^{-9}$ seconds
- External events: $\sim 10^{-2}$ seconds
 - Keyboard input
 - Disk seek
 - Screen refresh

Implication

- Can execute many instructions while waiting for external event to occur
- Can alternate among processes without anyone noticing

CompOrg Fall 2002 - Time Measurement

2

Measurement Challenge

How Much Time Does Program X Require?

- CPU time
 - How many total seconds are used when executing X?
 - Measure used for most applications
 - Small dependence on other system activities
- Actual ("Wall") Time
 - How many seconds elapse between the start and the completion of X?
 - Depends on system load, I/O times, etc.

Confounding Factors

- How does time get measured?
- Many processes share computing resources
 - Transient effects when switching from one process to another
 - Suddenly, the effects of alternating among processes become noticeable

CompOrg Fall 2002 - Time Measurement

3

"Time" on a Computer System



real (wall clock) time

= user time (time executing instructing instructions in the user process)

= system time (time executing instructing instructions in kernel on behalf of user process)

= some other user's time (time executing instructing instructions in different user's process)

+ + = real (wall clock) time

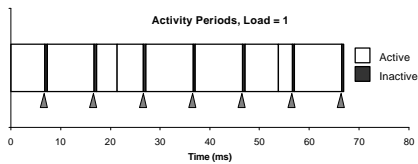
We will use the word "time" to refer to user time.

cumulative user time

CompOrg Fall 2002 - Time Measurement

4

Activity Periods: Light Load



- Most of the time spent executing one process

- Periodic interrupts every 10ms

- Interval timer
- Keep system from executing one process to exclusion of others

- Other interrupts

- Due to I/O activity

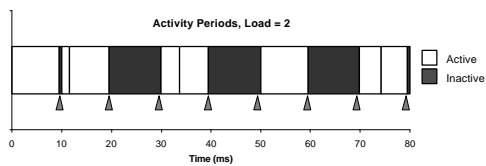
- Inactivity periods

- System time spent processing interrupts
- ~250,000 clock cycles

CompOrg Fall 2002 - Time Measurement

5

Activity Periods: Heavy Load



- Sharing processor with one other active process

- Periodic interrupts every 10ms

- Interval timer
- Keep system from executing one process to exclusion of others

- Other interrupts

- Due to I/O activity

- Inactivity periods

- System time spent processing interrupts
- Periods when other process executes

CompOrg Fall 2002 - Time Measurement

6

Cycle Counters

- Most modern systems have built in registers that are incremented every clock cycle
 - Very fine grained
 - Maintained as part of process state
 - » In Linux, counts elapsed global time
- Special assembly code instruction to access
- On (recent model) Intel machines:
 - 64 bit counter.
 - RDTSQ instruction sets `%edx` to high order 32-bits, `%eax` to low order 32-bits

Wrap Around Times for 550 MHz machine

- Low order 32-bits wrap around every $2^{32} / (550 * 10^6) = 7.8$ seconds
- High order 64-bits wrap around every $2^{64} / (550 * 10^6) = 33539534679$ seconds
 - 1065.3 years

CompOrg Fall 2002 - Time Measurement

10

Measuring with Cycle Counter

Idea

- Get current value of cycle counter
 - store as pair of unsigned's `cyc_hi` and `cyc_lo`
- Compute something
- Get new value of cycle counter
- Perform double precision subtraction to get elapsed cycles

```
/* Keep track of most recent reading of cycle counter */
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

void start_counter()
{
    /* Get current value of cycle counter */
    access_counter(&cyc_hi, &cyc_lo);
}
```

CompOrg Fall 2002 - Time Measurement

11

Accessing the Cycle Counter (cont.)

- GCC allows inline assembly code with mechanism for matching registers with program variables
- Code only works on x86 machine compiling with GCC

```
void access_counter(unsigned *hi, unsigned *lo)
{
    /* Get cycle counter */
    asm("rdtsq; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        : /* No input */
        : "%edx", "%eax");
}
```

- Emit assembly with `rdtsq` and two `movl` instructions
- Code generates two outputs:
 - Symbolic register `%0` should be used for `*hi`
 - Symbolic register `%1` should be used for `*lo`
- Code has no inputs
- Registers `%eax` and `%edx` will be overwritten

CompOrg Fall 2002 - Time Measurement

12

Completing Measurement

- Get new value of cycle counter
- Perform double precision subtraction to get elapsed cycles
- Express as double to avoid overflow problems

```
double get_counter()
{
    unsigned nyc_hi, nyc_lo;
    unsigned hi, lo, borrow;
    /* Get cycle counter */
    access_counter(&nyc_hi, &nyc_lo);
    /* Do double precision subtraction */
    lo = nyc_lo - cyc_lo;
    borrow = lo > nyc_lo;
    hi = nyc_hi - cyc_hi - borrow;
    return (double) hi * (1 << 30) * 4 + lo;
}
```

CompOrg Fall 2002 - Time Measurement

13

Timing With Cycle Counter

Determine Clock Rate of Processor

- Count number of cycles required for some fixed number of seconds

```
double MHZ;
int sleep_time = 10;
start_counter();
sleep(sleep_time);
MHZ = get_counter() / (sleep_time * 1e6);
```

Time Function P

- First attempt: Simply count cycles for one execution of P

```
double tsecs;
start_counter();
P();
tsecs = get_counter() / (MHZ * 1e6);
```

CompOrg Fall 2002 - Time Measurement

14

Measurement Pitfalls

Overhead

- Calling `get_counter()` incurs small amount of overhead
- Want to measure long enough code sequence to compensate

Unexpected Cache Effects

- artificial hits or misses
- e.g. these measurements were taken with the Alpha cycle counter:
`foo1(array1, array2, array3);` /* 68,829 cycles */
`foo2(array1, array2, array3);` /* 23,337 cycles */
vs.
`foo2(array1, array2, array3);` /* 70,513 cycles */
`foo1(array1, array2, array3);` /* 23,203 cycles */

CompOrg Fall 2002 - Time Measurement

15

Dealing with Overhead & Cache Effects

- Always execute function once to “warm up” cache
- Keep doubling number of times execute P() until reach some threshold
 - Used CMIN = 50000

```
int cnt = 1;
double cmeas = 0;
double cycles;
do
{
    int c = cnt;
    P();
    get_counter();
    while (c-- > 0)
        P();
    cmeas = get_counter();
    cycles = cmeas / cnt;
    cnt += cnt;
} while (cmeas < CMIN); /* Make sure have enough */
return cycles / (1e6 * MHZ);
```

Multitasking Effects

Cycle Counter Measures Elapsed Time

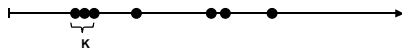
- Keeps accumulating during periods of inactivity
 - System activity
 - Running other processes

Key Observation

- Cycle counter never underestimates program run time
- Possibly overestimates by large amount

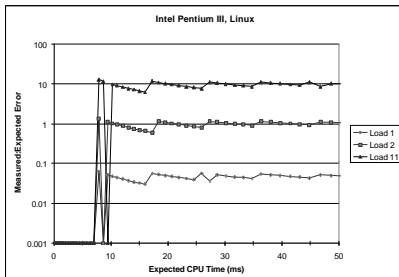
K-Best Measurement Scheme

- Perform up to N (e.g., 20) measurements of function
- See if fastest K (e.g., 3) within some relative factor ϵ (e.g., 0.001)



K-Best Validation

$K = 3, \epsilon = 0.001$



Very good accuracy for < 8ms

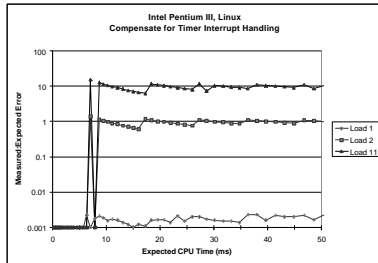
- Within one timer interval
- Even when heavily loaded

Less accurate of > 10ms

- Light load: -4% error
 - Interval clock interrupt handling
- Heavy load: Very high error

Compensate For Timer Overhead

$K = 3, \epsilon = 0.001$



Subtract Timer Overhead

- Estimate overhead of single interrupt by measuring periods of inactivity
- Call interval timer to determine number of interrupts that have occurred

Better Accuracy for > 10ms

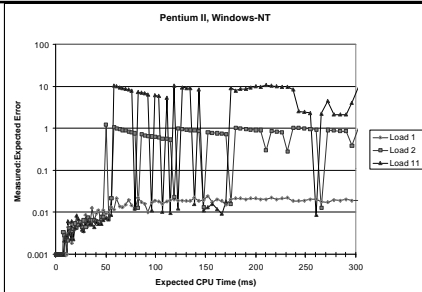
- Light load: 0.2% error
- Heavy load: Still very high error

CompOrg Fall 2002 - Time Measurement

19

K-Best on NT

$K = 3, \epsilon = 0.001$



Acceptable accuracy for < 50ms

- Scheduler allows process to run multiple intervals

Less accurate of > 10ms

- Light load: 2% error
- Heavy load: Generally very high error

CompOrg Fall 2002 - Time Measurement

20

Time of Day Clock

- Unix `gettimeofday()` function
- Return elapsed time since reference time (Jan 1, 1970)
- Implementation
 - Uses interval counting on some machines
 - » Coarse grained
 - Uses cycle counter on others
 - » Fine grained, but significant overhead and only 1 microsecond resolution

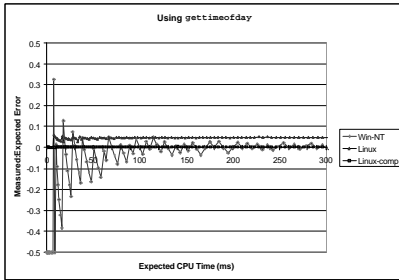
```
#include <sys/time.h>
#include <unistd.h>

struct timeval tstart, tfinish;
double tsecs;
gettimeofday(&tstart, NULL);
P();
gettimeofday(&tfinish, NULL);
tsecs = (tfinish.tv_sec - tstart.tv_sec) +
        1e6 * (tfinish.tv_usec - tstart.tv_usec);
```

CompOrg Fall 2002 - Time Measurement

21

K-Best Using gettimeofday



Linux

- As good as using cycle counter

- For times > 10 microseconds

Windows

- Implemented by interval counting

- Too coarse-grained

CompOrg Fall 2002 - Time Measurement

22

Measurement Summary

Timing is highly case and system dependent

- What is overall duration being measured?
 - > 1 second: interval counting is OK
 - << 1 second: must use cycle counters
- On what hardware / OS / OS version?
 - Accessing counters
 - » How gettimeofday is implemented
 - Timer interrupt overhead
 - Scheduling policy

Devising a Measurement Method

- Long durations: use Unix timing functions
- Short durations
 - If possible, use gettimeofday
 - Otherwise must work with cycle counters
 - K-best scheme most successful

CompOrg Fall 2002 - Time Measurement

23
