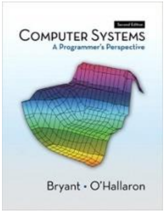
System Level Programming

Unit 10   Performance Measure

李靓 (Li Jing)   lijing712@scu.edu.cn
School of Computer Science / Software Engineering

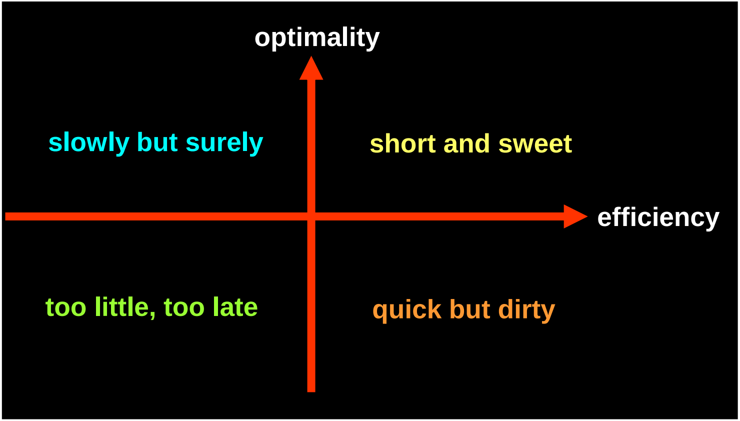


Performance Measure

- 10.1 Rationale (逻辑依据) for this unit
- 10.2 Performance Principles (法则)
- 10.3 Performance Measurement
- 10.4 VC Profiler (评测器)

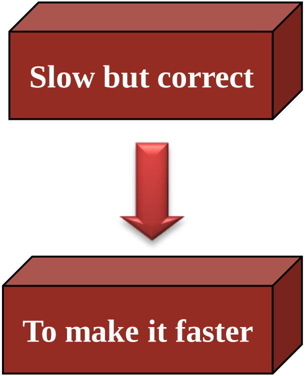
10.1 Rationale for this unit

- Attribute of a program



10.1 Rationale for this unit

- Procedure of developing a program



10.1 Rationale for this unit

- Some issues need to be considered

This unit

- Performance measure – Does this program run fast?
- Optimizing program – How to make program run fast

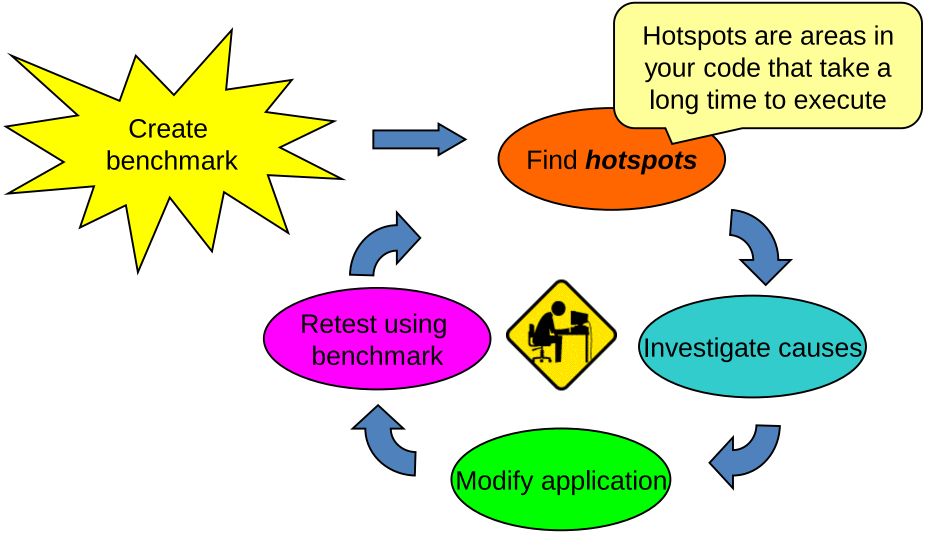
- Does this program run fast?

Bottlenecks / Hot spots

- We must find what should be optimized

10.1 Rationale for this unit

- The Software Optimization Process



Performance Measure

- 10.1 Rationale (逻辑依据) for this unit
- 10.2 Performance Principles (法则)
- 10.3 Performance Measurement
- 10.4 VC Profiler (评测器)

10.2 Performance principles

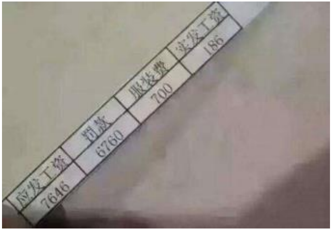
- 80/20 rule (Pareto Principle)

In the 1800's an Italian economist, Vilfredo Pareto, identified that 80% of the wealth of the country was owned by 20% of the people.

10.2 Performance principles

➢ 80/20 rule seems to apply to lots of things in life

20% of your stored phone numbers – you call 80% of the time
You see 20% of your family and friends – 80% of the time
80% of your monthly pay check vanishes in 20% of the month!!



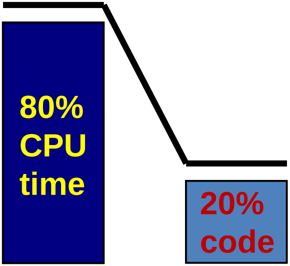
10.2 Performance principles

- 80/20 rule
  - 80% of CPU time is spent in 20% of the program

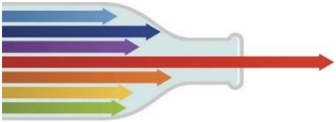
10.2 Performance principles

- 80/20 rule
  - You can have better performance by focusing on this 20%

You focus 80% of your time on the vital 20% of things that really matter

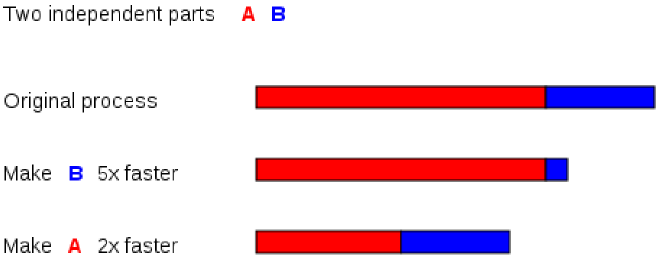


定性
Qualitative



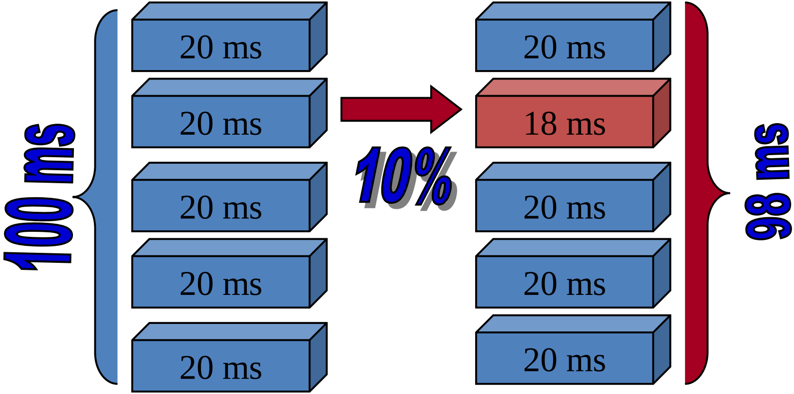
10.2 Performance principles

- Amdahl's Law / Argument (阿姆达尔法则)
  - Named after computer architect Gene Amdahl
  - Used to find the maximum expected improvement to an overall system when only part of the system is improved



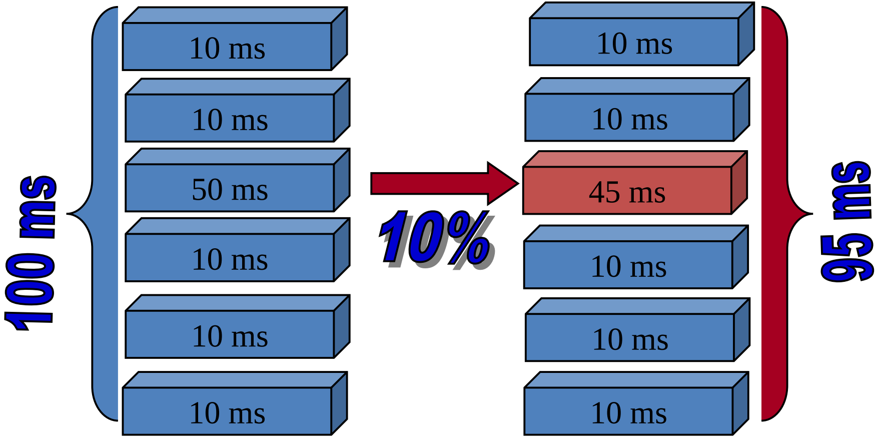
10.2 Performance principles

- 10% on one module means 2% as a whole



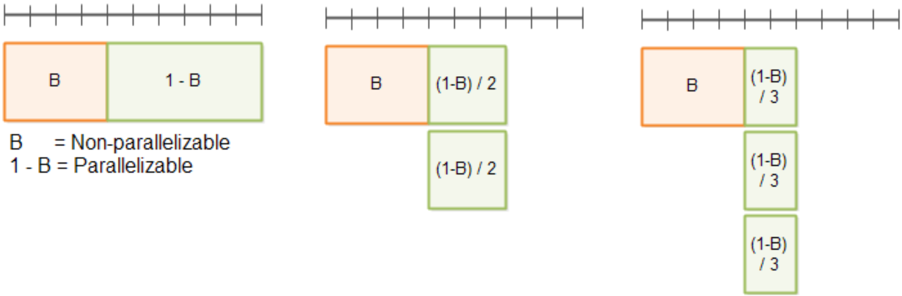
10.2 Performance principles

- 10% on one module means 5% as a whole
- Conclusion: focus on module with more CPU time



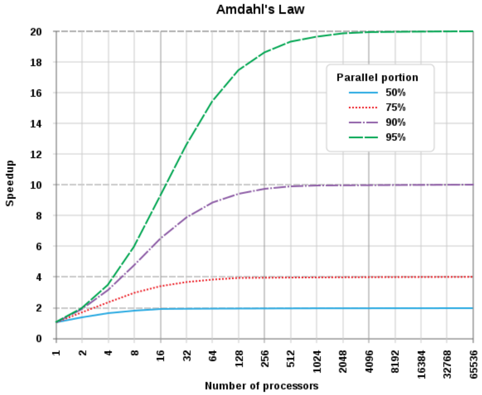
10.2 Performance principles

- Amdahl's Law / Argument (阿姆达尔法则)
  - Used in parallel computing to predict the theoretical maximum speedup



10.2 Performance principles

- Amdahl's Law / Argument (阿姆达尔法则)
  - Used in parallel computing to predict the theoretical maximum speedup



10.2 Performance principles

- Suppose the enhancement E accelerates a fraction P of one task by a factor S and the remainder of the task unaffected

$$ExcuteTime(Without\ E) = 1$$

$$ExcuteTime(With\ E) = (1 - P) + \frac{P}{S}$$

$$Speedup(E) = \frac{ExcuteTime(Without\ E)}{ExcuteTime(With\ E)} = \frac{1}{\left\{(1 - P) + \frac{P}{S}\right\}}$$

10.2 Performance principles

- Quiz
  - We are considering an enhancement to the processor of a web server.
  - The new CPU is 20 times faster on search queries than the old processor.
  - The old processor is busy with search queries 70% of the time.
  - What is the speedup gained by integrating the enhanced CPU?

Performance Measure

- 10.1 Rationale (逻辑依据) for this unit
- 10.2 Performance Principles (法则)
- 10.3 Performance Measurement
- 10.4 VC Profiler (评测器)

10.3 Performance measurement

- 10.3.1 What to measure
- 10.3.2 Timing mechanisms
- 10.3.3 Statistical sampling (统计抽样) / Profiling (分析)

10.3.1 What to measure

- The most common thing to measure is time

10.3 Performance measurement

- 10.3.1 What to measure
- 10.3.2 Timing mechanisms
- 10.3.3 Statistical sampling (统计抽样) / Profiling (分析)

10.3.2 Timing mechanisms

- 10.3.2.1 Introduction
- 10.3.2.2 Timer in hardware
- 10.3.3 Timer in OS
- 10.3.4 Timer in C/C++

10.3.2.1 Introduction

- What is time? ——1D quantity

"Do you love life? Then don't waste time, because life is made of time."—— Benjamin Franklin

- Time is used to

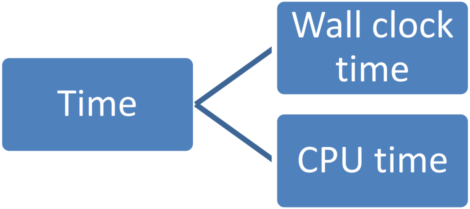
Measure events sequence
Quantify the durations of events and the intervals

上午	第01节课	08:15-09:00
	第02节课	09:10-09:55
	第03节课	10:15-11:00
	第04节课	11:10-11:55
下午	第05节课	13:50-14:35
	第06节课	14:45-15:30
	第07节课	15:40-16:25
	第08节课	16:45-17:30
	第09节课	17:40-18:25

10.3.2.1 Introduction

- Why we need time in CS?

Hardware need time and timer
OS need time and timer
Measuring program performance, e.g. execution time • How fast programs run on machine?



The time an ordinary clock on the wall or a wrist watch shows

10.3.2.1 Introduction

- Wall clock time - The overall time needed to run a particular program or solve a problem.

real (i.e. wall clock) time

= **User Time**: time spent executing instructions in the **user process**

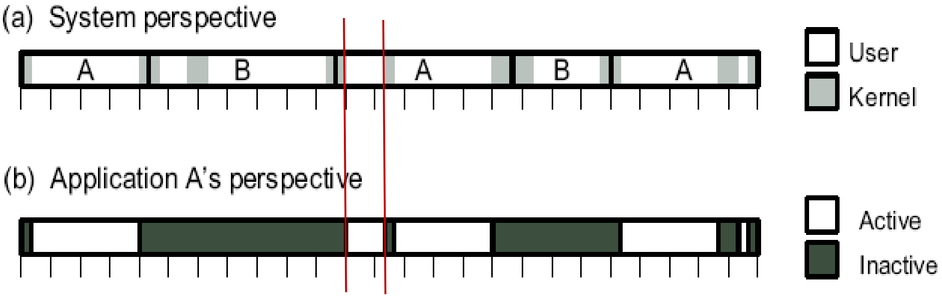
= **System Time:** time spent executing instructions in the ***kernel*** on behalf of the user process

= **all other time** (either idle or I/O waiting time or executing instructions unrelated to the user process)

10.3.2.1 Introduction

- CPU time

User CPU time + System CPU time = CPU time



10.3.2.1 Introduction

- How to use time in CS?

By using many kinds of timer (定时器)

- What is timer

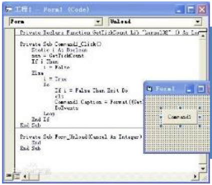
A component in Computer System/CS, as a hardware or software, which can provide the ability of measuring time in some degree.

10.3.2.1 Introduction

➢ Different kinds of timer

Timer in hardware

Intel Architecture, 32/X86, 8253



Timer in OS

Windows, GetTickCount() | Linux, jiffies

Timer in C/C++

<time.h>

10.3.2 Timing mechanisms

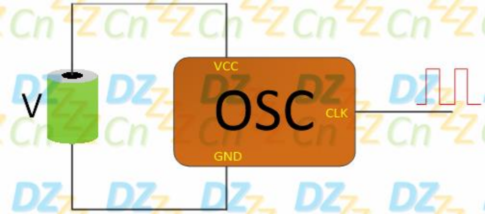
- 10.3.2.1 Introduction
- 10.3.2.2 Timer in hardware
- 10.3.3 Timer in OS
- 10.3.4 Timer in C/C++

10.3.2.2 Timer in hardware

- Real Time CMOS Clock 实时时钟 RTC
  - CMOS RAM: store time & configurations
  - Low powered battery
  - For the boot-up process

10.3.2.2 Timer in hardware

- Real Time CMOS Clock 实时时钟 RTC
  - The RTC keeps updating time in the background
  - Source: crystal oscillator produces the original clock frequency



10.3.2.2 Timer in hardware

- Main frequency | Frequency multiplication

时钟周期/振荡周期

Clock Cycle = seconds per cycle

$$Clock\ Frequency = \frac{1}{Clock\ Cycle}$$

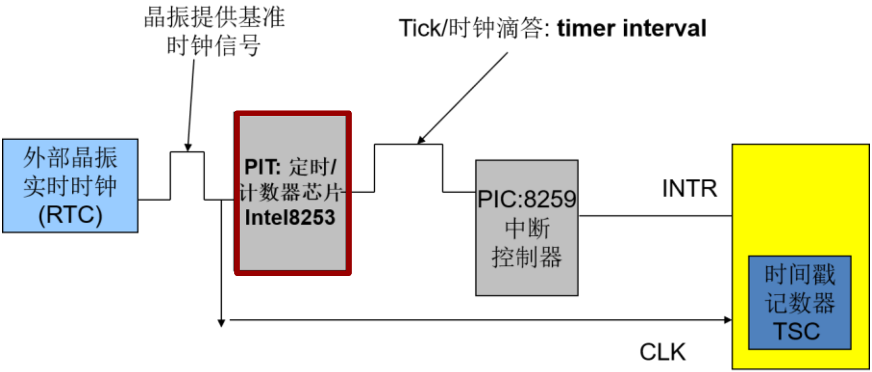
时钟频率 (Hz.= cycle/sec)

Clock Frequency = cycles per second

i.e. 1.4GHz

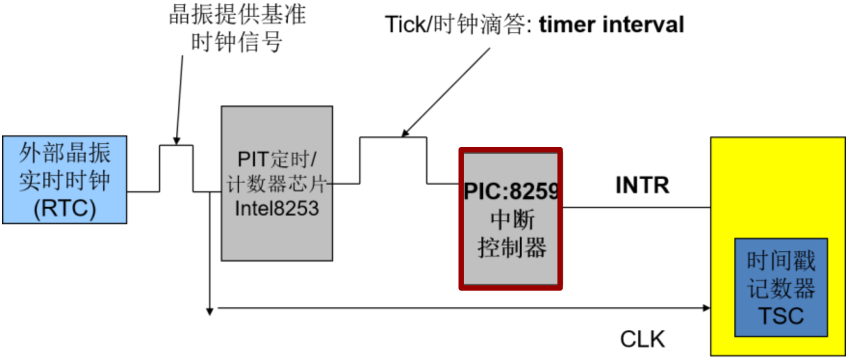
10.3.2.2 Timer in hardware

- PIT (Programmable Interval Timer 可编程间隔定时器)
  - A counter that generates output signal and (may) then trigger an interrupt



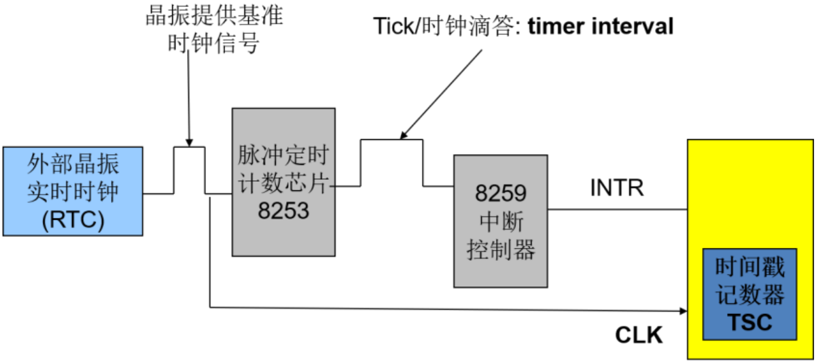
10.3.2.2 Timer in hardware

- PIC (Programmable Interrupt Controller)



10.3.2.2 Timer in hardware

- TSC (Time Stamp Counter 时间戳计数器)
  - TSC receives the CLK signal from RTC
  - A 64-bit register, count the number of cycles



10.3.2.2 Timer in hardware

- TSC: Time Stamp Counter
  - Intel Pentium processors (among others) have a very high-speed internal 64-bit counter
  - Be accessed using **RDTSC** (in nanoseconds)
    - Reads TSC value from registers

- Windows operating systems have an interface to access high-resolution performance counter (in microseconds)

QueryPerformanceFrequency
QueryPerformanceCounter

10.3.2.2 Timer in hardware

- High-precision timer function – Win interface

  - The QueryPerformanceFrequency() function retrieves the frequency of the high-resolution performance counter

```
BOOL QueryPerformanceFrequency (
            // address of current frequency
           LARGE_INTEGER* lpFrequency );
```

10.3.2.2 Timer in hardware

- High-resolution timer function – Win interface
  - The QueryPerformanceCounter() function retrieves the current value of the high-resolution performance counter

```
BOOL QueryPerformanceCounter (
        // Pointer to counter value
        LARGE_INTEGER* lpPerformanceCount );
```

10.3.2.2 Timer in hardware

- LARGE_INTEGER

Be used to represent a 64-bit signed integer value.
For a 64-bit compiler, use **QuadPart** to store 64-bit integer.
Otherwise, use **LowPart** and **HighPart** to store the 64-bit integer.

```
typedef union _LARGE_INTEGER {
        struct
        {
                DWORD LowPart;
                LONG HighPart;
        };
        LONGLONG QuadPart;
} LARGE_INTEGER;
```

10.3.2.2 Timer in hardware

- LARGE_INTEGER example

```
LARGE_INTEGER litmp;
LONGLONG QPart1, QPart2;
double dfFreq;
double dfMinus, dfTim;

QueryPerformanceFrequency(&litmp);
dfFreq = (double)litmp.QuadPart; // 获得计数器的时钟频率

QueryPerformanceCounter(&litmp);
QPart1 = litmp.QuadPart; // 获得初始值

Sleep(100);

QueryPerformanceCounter(&litmp);
QPart2 = litmp.QuadPart; //获得中止值

dfMinus = (double)(QPart2-QPart1);
dfTim = dfMinus / dfFreq; // 获得对应的时间值，单位为秒
```

10.3.2.2 Timer in hardware

- Example code
  - Using the high precision timer under Windows can be found in the .zip file precise.zip

void precise_start()

Begin timing

double precise_stop()

To get the elapsed time in seconds

10.3.2 Timing mechanisms

- 10.3.2.1 Introduction
- 10.3.2.2 Timer in hardware
- 10.3.3 Timer in OS
- 10.3.4 Timer in C/C++

10.3.2.3 Timer in OS

- Windows System Time

- The number of milliseconds elapsed since the system was last started
- DWORD GetTickCount(void); //Cycles every 49.7 days
- Read from RTC and then convert

SYSTEMTIME
- GetLocalTime(), 1ms
- GetSystemTime() - UTC, Universal Time Coordinated

FILETIME
- GetSystemTimeAsFileTime(), $100ns = 0.1\mu m$
- Win - 1601.1.1 | Linux – 1970.1.1

10.3.2.3 Timer in OS

```
typedef struct _SYSTEMTIME {
    WORD wYear;  // The valid values are 1601 through 30827.
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

```
typedef struct _FILETIME {
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
} FILETIME, *PFILETIME;
```

10.3.2.3 Timer in OS

```
#include <windows.h>
#include <stdio.h>

void main()  {
   SYSTEMTIME st, lt;

   GetSystemTime(&st);
   GetLocalTime(&lt);

   printf("The system time is: %02d:%02d\n", st.wHour, st.wMinute);
   printf("  The local time is: %02d:%02d\n", lt.wHour, lt.wMinute);
}
```

```
// Sample output
The system time is: 14:34
 The local time is: 22:34
```

10.3.2.3 Timer in OS

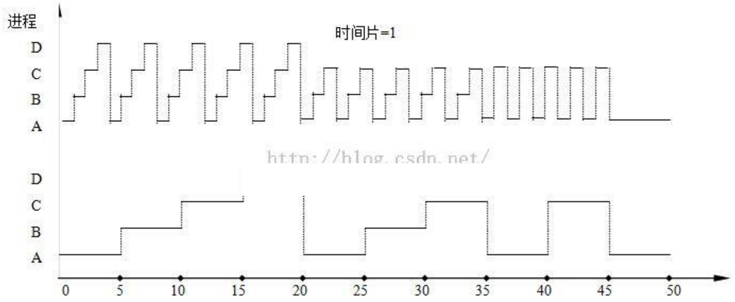
```
int main()
{
    ULARGE_INTEGER  uli;
    FILETIME        ft;
    SYSTEMTIME      st;

    GetSystemTimeAsFileTime(&ft);
    uli.LowPart = ft.dwLowDateTime;
    uli.HighPart = ft.dwHighDateTime;
    printf("System File Time: %I64u\n", uli.QuadPart);

    FileTimeToSystemTime(&ft, &st);
    printf("System Time (YYYY-MM-DD HH:MM:SS): %d-%d-%d %d:%d:%d\n",
        st.wYear, st.wMonth, st.wDay, st.wHour, st.wMinute, st.wSecond);

    return EXIT_SUCCESS;
}
```

```
System File Time: 132176695123942571
System Time (YYYY-MM-DD HH:MM:SS): 2019-11-8 6:51:52
Press any key to continue
```

10.3.2.3 Timer in OS

- Time slice

A specific number of clock ticks (时钟单元) before process gets moved to another state



10.3.2 Timing mechanisms

- 10.3.2.1 Introduction
- 10.3.2.2 Timer in hardware
- 10.3.3 Timer in OS
- 10.3.4 Timer in C/C++

10.3.2.4 Timer in C/C++

clock_t clock()

```
#include <time.h>
void elapsed_time()
{
    printf( "%d ms\n", clock());
    printf( "%d s\n", clock()/CLOCKS_PER_SEC);
}
```

How long the calling process has spent.

clock() The processor time that have elapsed

1 clock tick = 1/CLOCKS_PER_SEC

10.3.2.4 Timer in C/C++

- Macro宏 CLOCKS_PER_SEC
- Data types clock_t

```
// time.h
#define CLOCK_PER_SECOND((clock_t)1000)

#ifndef _CLOCK_T_DEFINED
typedef long clock_t;
#define _CLOCK_T_DEFINED
#endif
```

10.3.2.4 Timer in C/C++

- clock() in C/C++

```
Time for 1000000 iterations: 0.069s
Precision is 1000 clocks per second.
Press any key to continue
```

```
#include <stdlib.h>
#include <time.h>
#include <iostream.h>
void my_subroutine(long n) {
        // timing a subroutine call:
        char s[16];
        for (long i = 0; i < n; i++) {
                _itoa(i, s, sizeof(s));
        }
}
```

```
int main(int argc, char* argv[]) {
        long n = 1000000;
        clock_t start = clock();
        my_subroutine(n);
        clock_t finish = clock();
        double duration = (double)(finish - start) / CLOCKS_PER_SEC;
        cout << "Time for " << n << " iterations: " << duration << "s"
                <<endl<< “Precision is " << CLOCKS_PER_SEC
                << " clocks per second." << endl;
        return 0;
}
```

10.3.2.4 Timer in C/C++

time_t time() Calendar time

```
int main(void)
{
    time_t lt;
    lt = time(NULL);
    printf("The Calendar Time now is %d\n",lt);
    return 0;
}
```

```
The Calendar Time now is: 1573198707
Press any key to continue
```

从1970年1月1日0时0分0秒到现在的秒数

10.3.2.4 Timer in C/C++

Data types	time_t	struct tm
Function	time()	localtime() gmtime()

```
// time.h
#ifndef _TIME_T_DEFINED
typedef long time_t;
#define _TIME_T_DEFINED
#endif
```

10.3.2.4 Timer in C/C++

- **structure** tm

```
#ifndef _TM_DEFINED
struct tm {
        int tm_sec;
        int tm_min;
        int tm_hour;
        int tm_mday;
        int tm_mon;
        int tm_year;
        int tm_wday;
        int tm_yday;
        int tm_isdst;
};
#define _ TM_DEFINED
#endif
```

10.3.2.4 Timer in C/C++

- **structure** tm

```
Fri Nov 08 16:00:35 2019
Fri Nov 08 16:00:35 2019
Fri Nov 08 08:00:35 2019
Press any key to continue
```

```
int main(void)
{
    time_t  lt;
    lt = time(NULL); // calendar time
    printf("%s\n", asctime(&lt)); // convert lt to string

    struct tm *ptr;
    ptr = localtime(&lt); // convert calendar time to local tm
    printf("%s\n", asctime(ptr)); // convert ptr to string

    ptr = gmtime(&lt); // convert calendar time to greenwich tm
    printf("%s\n", asctime(ptr)); // convert ptr to string
    return 0;
}
```

10.3.2.4 Timer in C/C++

- Functions

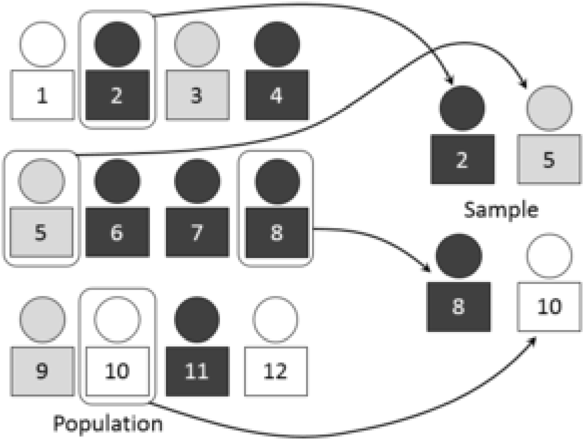
	Functions	Description
clock_t	clock()	Process time
time_t	time() difftime() gmtime() localtime()	Get time Time difference To GMT time To local time
struct tm	mktime() strftime()	To time_t Format

10.3 Performance measurement

- 10.3.1 What to measure
- 10.3.2 Timing mechanisms
- 10.3.3 Statistical sampling (统计抽样) / Profiling (分析)

10.3.3 Statistical sampling / Profiling

- Statistical Sampling 统计抽样法



10.3.3 Statistical sampling / Profiling

- Statistical Sampling 统计抽样法

  - Any sampling procedure that uses the laws of probability to measure status of the program.

A timer periodically interrupts the program and records the program counter
Estimate where time is spent in the program
Check if the program spends most of its time in a few places

10.3.3 Statistical sampling / Profiling

- Why

  - It saves time and involves less cost
  - In some cases, it might not be possible to check 100% (risk)

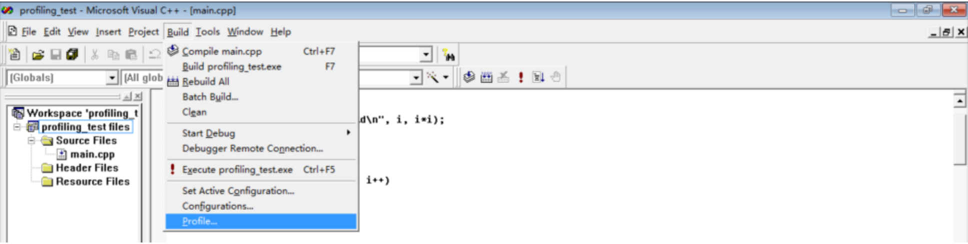
10.3.3 Statistical sampling / Profiling

- Software profiler
  - A program that benchmarks the execution of one or more pieces of procedural code to help the user understand where the time is being spent in terms of code execution.

  - GNU Gprof – Linux
  - Vtune - Intel
  - Visual C++ profiler
  - Valgrind for different platforms

10.3.3 Statistical sampling / Profiling

- VC profiler
  - In Visual C++, the **Profile...** entry in the **Build menu** gives instructions on obtaining a profile.
  - Profiling is only available in the Professional and Enterprise editions of Visual C++.

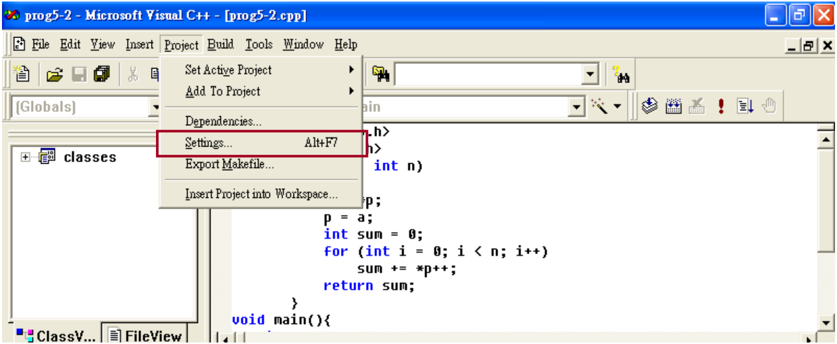


Performance Measure

- 10.1 Rationale (逻辑依据) for this unit
- 10.2 Performance Principles (法则)
- 10.3 Performance Measurement
- 10.4 VC Profiler (评测器)

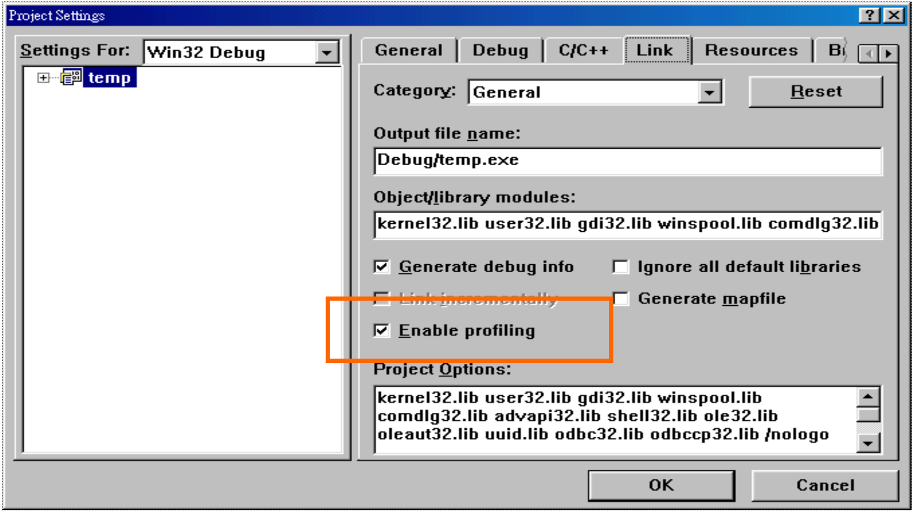
10.4 VC profiler

- Procedure (1) – setting



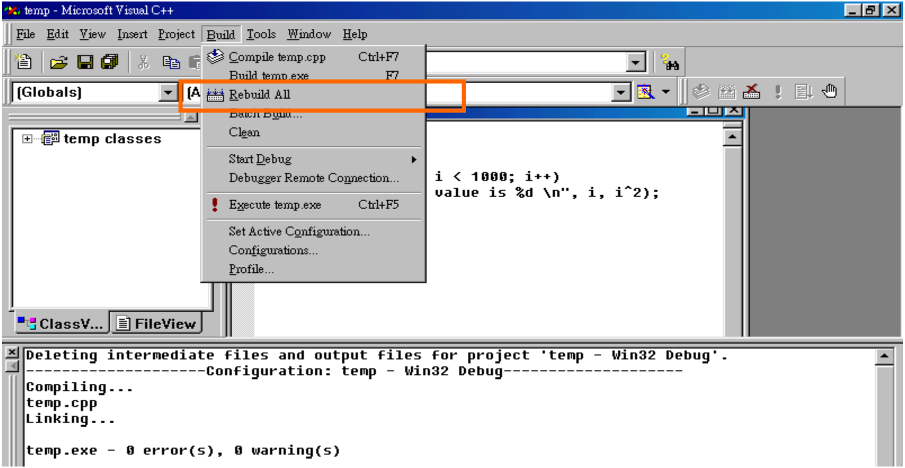
10.4 VC profiler

- Procedure (2) – enable profiling



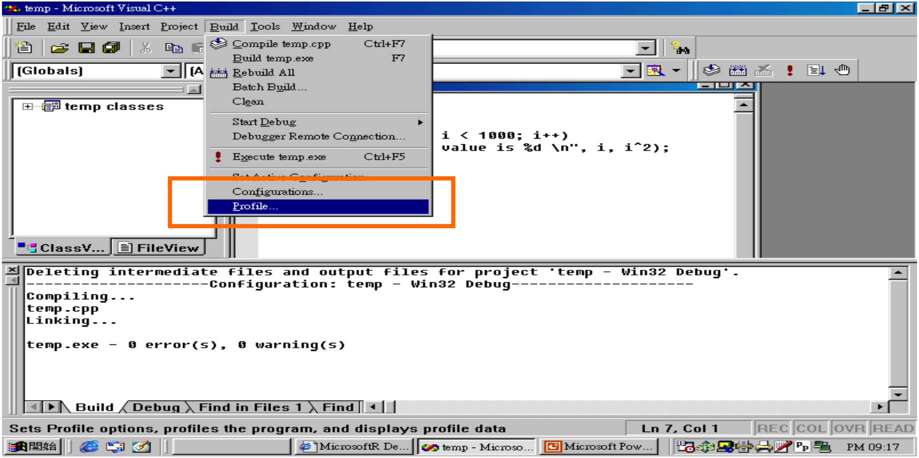
10.4 VC profiler

- Procedure (3) – rebuild



10.4 VC profiler

- Procedure (4) – run with profiling



10.4 VC profiler

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int i;
    for (i = 0; i < 1000; i++)
        printf("The value is %d\t %d \n", i, i*i);
}
```

10.4 VC profiler

```
Workspace 'test': 1 project
test files
```

```c
#include <stdio.h>
#include <stdlib.h>
void main()
{
     int i;
     for (i = 0; i < 1000; i++)
          printf("The value is %d\t %d \n", i, i*i);
}
```

Result of a simple for loop – total time is 248 ms

```
Time in module: 248.623 millisecond
Percent of time in module: 100.0%
Functions in module: 1
Hits in module: 1
Module function coverage: 100.0%

     Func            Func+Child            Hit
     Time     %         Time        %      Count   Function
---------------------------------------------------------
  248.623 100.0      248.623 100.0         1 _main (test.obj)
```

10.4 VC profiler

```
Workspace 'test': 1 project
test files
ClassView  FileView
```

```c
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int i;
    for (i = 0; i < 1000; i++)
        printf("The value is %d\t %d \n", i, i*i*i);
}
```

Result of a simple for loop – total time is 268 ms

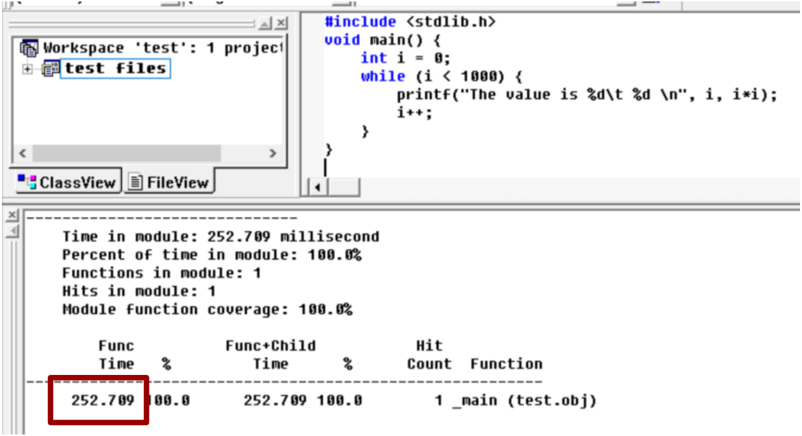
```
Module Statistics for test.exe
------------------------------
    Time in module: 268.222 millisecond
    Percent of time in module: 100.0%
    Functions in module: 1
    Hits in module: 1
    Module function coverage: 100.0%

        Func          Func+Child           Hit
        Time    %        Time      %      Count  Function
---------------------------------------------------------
     268.222 100.0     268.222 100.0         1 _main (test.obj)
```

10.4 VC profiler

```
#include <stdio.h>
#include <stdlib.h>
void main() {
	int i = 0;
	while (i < 1000) {
		printf("The value is %d\t %d \n", i, i*i);
		i++;
	}
}
```

10.4 VC profiler

- Example – result in millisecond second



10.4 VC profiler

- Example with a sub-routine

```
Workspace 'test': 1 project
test files
```

```c
#include <stdio.h>
#include <stdlib.h>

int sum(int a[], int n)
{
     int *p, i;
     p = a;
     int sum = 0;
     for (i=0; i<n; i++)
          sum += *p++;
     return sum;
}

void main() {
     int z[7] = {1,2,3,4,5,6,7};
     int i=7;
     int sum1 = sum(z, i);
     printf("The value of sum is %d\n", sum1);
}
```

```
    Percent of time in module: 100.0%
    Functions in module: 2
    Hits in module: 2
    Module function coverage: 100.0%

        Func          Func+Child           Hit
        Time     %         Time      %     Count  Function
---------------------------------------------------------
       0.097   99.8       0.098  100.0        1 _main (test.obj)
       0.000    0.2       0.000    0.2        1 sum(int * const,int) (test.obj)
```

Main()

Subroutine

10.4 VC profiler

- A program that can be used to determine million floating point operations (MFLOPS)

```
// This is matrix multiplication
#include <stdio.h>
#include <stdlib.h>
void main(){
        float a[250][250], b[250][250], c[250][250];
        int i, j, k;
        for (i = 0; i< 250; i++)
            for (j = 0; j < 250; j++)
                for (k =0; k <250; k++)
                    // matrix multiplication
                    c[i][j] += a[i][k] * b[k][j];
}
```

10.4 VC profiler

```
Program Statistics
------------------
    Command line at 2019 Nov 08 16:43: "D:\work\SLP\2019\week11\test\Debug\test"
    Total time: 110.217 millisecond
    Time outside of functions: 3.509 millisecond
    Call depth: 1
    Total functions: 1
    Total hits: 1
    Function coverage: 100.0%
    Overhead Calculated 1
    Overhead Average 1

Module Statistics for test.exe
------------------------------
    Time in module: 106.708 millisecond
    Percent of time in module: 100.0%
    Functions in module: 1
    Hits in module: 1
    Module function coverage: 100.0%

        Func          Func+Child           Hit
        Time   %         Time      %      Count  Function
---------------------------------------------------------
     106.708 100.0     106.708 100.0        1 _main (test.obj)
```

10.4 VC profiler

- Determination of Mega Flop

```
for (i = 0; i< 250; i++)
    for (j = 0; j < 250; j++)
        for (k =0; k <250; k++)
            // matrix multiplication
            c[i][j] += a[i][k] * b[k][j];
```

- 250 x 250 x 250 = 15625000 floating point operations
- The performance: 15625000/106ms = 15.625 x 10^6 /0.106 s = 147 MFLOPs (mega floating point operation).
- Try your computer

10.4 VC profiler

- Same output but change the program

```
// this program uses a temporary location t to store the value
void main() {
    float a[250][250], b[250][250], c[250][250];
    int i, j, k;
    float r = 0.0;
    for (i = 0; i< 250; i++) {
        for (j = 0; j < 250; j++) {
            for (k =0; k <250; k++)
                r += a[i][k] * b[k][j];  //this is matrix multiplication
               c[i][j] = r;
        }
    }
}
```

10.4 VC profiler

- Same machine – 90ms, why?

Workspace 'test': 1 project
test files

ClassView FileView

```
float r = 0.0;
for (i = 0; i< 250; i++) {
     for (j = 0; j < 250; j++) {
          for (k =0; k <250; k++)
               r += a[i][k] * b[k][j];  //th
          c[i][j] = r;
```

```
Program Statistics
------------------
    Command line at 2019 Nov 08 16:54: "D:\work\SLP\2019\week11\test\Debug\test"
    Total time: 93.231 millisecond
    Time outside of functions: 2.554 millisecond
    Call depth: 1
```

This is related to the cache memory effect, as the data is stored in cache.

```
Module Statistics for test.exe
------------------------------
    Time in module: 90.678 millisecond
    Percent of time in module: 100.0%
    Functions in module: 1
    Hits in module: 1
    Module function coverage: 100.0%

        Func          Func+Child           Hit
        Time   %        Time      %       Count  Function
---------------------------------------------------------
      90.678 100.0     90.678 100.0          1 _main (test.obj)
```

10.4 VC profiler

- A profiler is a great way to find where bottlenecks occur, so we can make our code more efficient, It can:

```
-----------------
    Command line at 2019 Nov 08 16:54: "D:\work\SLP\2019\week11\test\Debug\test"
    Total time: 93.231 millisecond
    Time outside of functions: 2.554 millisecond
    Call depth: 1
    Total functions: 1
    Total hits: 1
    Function coverage: 100.0%
    Overhead Calculated 0
    Overhead Average 0

Module Statistics for test.exe
------------------------------
    Time in module: 90.678 millisecond
    Percent of time in module: 100.0%
    Functions in module: 1
    Hits in module: 1
    Module function coverage: 100.0%

        Func          Func+Child          Hit
        Time    %        Time      %     Count  Function
---------------------------------------------------------
      90.678 100.0      90.678 100.0        1 _main (test.obj)
```

```
Source file: d:\work\slp\2019\week11\test\test.cpp

 Line Covered  Source
----------------------
   1:    *     void main() {
   2:              float a[250][250], b[250][250], c[250][250];
   3:              int i, j, k;
   4:    *         float r = 0.0;
   5:    *         for (i = 0; i< 250; i++) {
   6:    *             for (j = 0; j < 250; j++) {
   7:    *                 for (k =0; k <250; k++)
   8:    *                     r += a[i][k] * b[k][j];  //thi
   9:    *                     c[i][j] = r;
  10:    *             }
  11:    *         }
  11:          }
  12:    *
```

10.4 VC profiler

- Profiling is used to help programmers to identify

  - ✓ Which areas of the program are causing sluggish bottlenecks.
  - ✓ Which parts of the code are being called the most often.