Семинар 1. Измерение производительности компьютера.

1. Основные команды для работы в ОС Windows и ОС Unix.

1.1. Вход, выход, работа с файлами и программами.
1.2. Разработка и исполнение программ.

2. Функции измерения времени.

Функция clock

- Библиотечная функция определена в заголовочном файле time.h;
- Прототип clock_t clock();
- Возвращает время, прошедшее с момента запуска программы в единицах 1/CLK_TCK секунды;
- Используется в Windows и Linux;
- **Достоинства:** высокая платформенная независимость;
- **Недостатки:** низкая точность, при высокой загрузке процессора – неприемлемая точность, так как измеряется интервал времени, во время которого помимо процесса исследуемой программы исполнялись и другие процессы.

Пример использования (ex01a.c):

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
double f(double x); double f(double x){ return (4.0 / (1.0 + x*x)); }
double pi_calculate(const int n); double pi_calculate(const int n)
{
  int i; double x,h,sum=0.0;
  h = 1.0 / (double) n;
  for (i=1; i<=n; i++){ x = h * ((double)i - 0.5); sum += f(x); }
  return h*sum;
}
int main(int argc, char *argv[])
{
  long tick; double t, pi;
  tick = sysconf(_SC_CLK_TCK);
  t = (double)(clock()*0.0001/tick);
  pi = pi_calculate(100000000);
  t = (double)(clock()*0.0001/tick)-t;
  printf("Time: %lf sec Pi = %14.12lf\n",t,pi);
  return 0;
}
```

Функция gettimeofday

- Библиотечная функция определена в заголовочном файле sys\time.h;
- Прототип int gettimeofday(struct timeval* tv, struct timezone* tz);
- Время можно вычислить из структуры timeval;
- Используется в UNIX;
- **Достоинства:** высокая платформенная независимость в рамках UNIX систем;
- **Недостатки:** низкая точность, при высокой загрузке процессора – неприемлемая точность, так как измеряется интервал времени, во время которого помимо процесса исследуемой программы исполнялись и другие процессы.

Пример использования (ex01b.c):

```
#include <stdio.h>
#include <sys/time.h>
static struct timeval tv1,tv2,dtv; static struct timezone tz;
void time_start(); void time_start(){ gettimeofday(&tv1,&tz); }
double time_stop(); double time_stop()
{
  gettimeofday(&tv2, &tz);
  dtv.tv_sec  = tv2.tv_sec  - tv1.tv_sec;
  dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;
  if(dtv.tv_usec<0){ dtv.tv_sec--; dtv.tv_usec+=1000000; }
  return (double)(dtv.tv_sec) + (double)(dtv.tv_usec)*1e-6;
}
```

```
int main(int argc, char *argv[])
{
  double t, pi;
  time_start(); pi = pi_calculate(100000000); t = time_stop();
  printf("Time: %lf sec Pi = %14.12lf\n",t,pi);
  return 0;
}
```

Функция times

- Библиотечная функция определена в заголовочном файле sys/times.h
- Прототип clock_t times(struct tms *buf);
- Возвращает время, прошедшее с момента запуска программы в единицах 1/CLK_TCK секунды;
- Используется в UNIX;
- **Достоинства:** высокая точность (относительная независимость от других процессов системы);
- **Недостатки:** для малых интервалов результат зависит от интервала времени прерываний по таймеру.

Пример использования (ex01c.c):

```
#include <stdio.h>
#include <sys/times.h>
#include <unistd.h>
static struct tms tmsBegin, tmsEnd;
static long tick;
void time_tick(); void time_tick(){ tick = sysconf(_SC_CLK_TCK); }
void time_start(); void time_start(){ times(&tmsBegin); }
double time_stop(); double time_stop()
{
  times(&tmsEnd);
  return ((tmsEnd.tms_utime-tmsBegin.tms_utime)+
          (tmsEnd.tms_stime-tmsBegin.tms_stime))*1.0/tick;
}
int main(int argc, char *argv[])
{
  double t, pi; time_tick();
  time_start(); pi = pi_calculate(100000000); t = time_stop();
  printf("Time: %lf sec Pi = %14.12lf\n",t,pi);
  return 0;
}
```

Функция omp_get_wtime

- Библиотечная функция в рамках стандарта OpenMP, определена в заголовочном файле omp.h;
- Прототип double omp_get_wtime (void);
- Возвращает время, прошедшее с момента запуска программы в секундах с точностью до "тика", величину тика возвращает функция omp_get_wtick (1 тик = 1 мкс);
- Используется на всех платформах;
- **Достоинства:** высокая платформенная независимость, высокая точность;
- **Недостатки:** в некоторых реализациях возвращает время работы всех трэдов приложения.

Пример использования (ex01d.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
int main (int argc, char *argv[])
{
  double t, mypi;
  t = omp_get_wtime();
  mypi = pi_calculate(100000000);
  t = omp_get_wtime() - t;
  printf("Time: %lf sec Pi = %14.12lf\n",t,mypi);
  return 0;
}
```

Функция MPI_Wtime

- Библиотечная функция в рамках стандарта MPI, определена в заголовочном файле mpi.h;
- Прототип double MPI_Wtime (void);
- Возвращает время, прошедшее с момента запуска программы в секундах с точностью до "тика", величину тика возвращает функция MPI_Wtick (1 тик = 1 мкс);
- Используется на всех платформах;
- **Достоинства:** высокая платформенная независимость, высокая точность;
- **Недостатки:** в некоторых реализациях возвращает время работы всех трэдов приложения.

Пример использования (ex01e.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
int main (int argc, char *argv[])
{
  double t, mypi;
  t = MPI_Wtime();
  mypi = pi_calculate(100000000);
  t = MPI_Wtime() - t;
  printf("Time: %lf sec Pi = %14.12lf\n",t,mypi);
  return 0;
}
```

Функция QueryPerformanceCounter

- Библиотечная функция MS Windows определена в заголовочном файле windows.h
- Прототип bool WINAPI QueryPerformanceCounter(long long int *ticks);
- Возвращает время, прошедшее с момента запуска программы в секундах;
- Используется только в MS Windows;
- **Достоинства:** высокая точность (относительная независимость от других процессов системы);
- **Недостатки:** для малых интервалов результат зависит от интервала времени прерываний по таймеру, непереносимость на другие платформы.

Пример использования (ex01f.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <windows.h>
double PCFreq = 0.0;
__int64 CounterStart = 0;
void StartCounter()
{
    LARGE_INTEGER li;
    if (!QueryPerformanceFrequency(&li))
    printf("QueryPerformanceFrequency failed!\n");
    PCFreq = ((double)(li.QuadPart))/1000.0;
    QueryPerformanceCounter(&li);
    CounterStart = li.QuadPart;
}
double GetCounter()
{
    LARGE_INTEGER li;
    QueryPerformanceCounter(&li);
    return (double)((li.QuadPart-CounterStart)/PCFreq);
}
int main()
{
  double t, mypi, sum=0;
  StartCounter();
  mypi = pi_calculate(1000000000);
  sum += mypi;
  t = 1e-3 * GetCounter();
  printf("Time: %le sec mypi=%14.12lf sum=%14.12lf\n", t, mypi, sum);
```

```
  return 0;
}
```
Трансляция и результаты выполнения:
```
>mpicc -o ex01a.px -O2 ex01a.c -lm
>mpicc -o ex01b.px -O2 ex01b.c -lm
>mpicc -o ex01c.px -O2 ex01c.c -lm
>mpicc -o ex01d.px -O2 -fopenmp ex01d.c -lm
>mpicc -o ex01e.px -O2 ex01e.c -lm
>ex01a.px
Time: 1.413742 sec mypi = 3.141592653590
>ex01b.px
Time: 1.407399 sec mypi = 3.141592653590
>ex01c.px
Time: 1.400000 sec mypi = 3.141592653590
>ex01d.px
Time: 1.409396 sec mypi = 3.141592653590
>ex01e.px
Time: 1.496606 sec mypi = 3.141592653590
```

3. Измерение времени отдельных операций.

3.1. Два способа измерения: прямое и косвенное.

Прямое измерение	Косвенное измерение

void main () {
int i, nop=1000000;
double t1; double a=1.234, b=1.567, c=0;
t1 = mytime();
for (i=0; i<nop; i++) c += (a*i+b)/(a+b*i);
t1 = mytime() - t1;
dt = t1 / (5*nop); printf("dt=%lf\n",dt);
}
	void main () {
	int i, nop=1000000;
	double t1, t2; double a=1.234, b=1.567, c=0, d=0;
	t1 = mytime();
	for (i=0; i<nop; i++) c += (a*i+b)/(a+b*i);
	t1 = mytime() - t1;
	t2 = mytime();
	for (i=0; i<nop; i++) d += c*(a*i+b)/(a+b*i);
	t2 = mytime() - t2;
	dt = (t2 - t1)/nop; printf("dt=%lf\n",dt);
	}

3.2. Простейшие операции: сложение, умножение, деление.

3.3. Функции: pow, exp, sin и др.

Пример 1 (ex02a.c):
```
#include <stdio.h>
#include <sys/time.h>
#include "mycom.h"
double f1(double x); double f1(double x) { return (4.0 / (1.0 + x*x));}
double f2(double x); double f2(double x) { return (4.0 / (1.0 + x + x*x));}
double f3(double x); double f3(double x) { return (4.0 / (1.0 + 3.0*x*x));}
double f4(double x); double f4(double x) { return (4.0 / (1.0 + x*x/3.0));}
int main(int argc, char *argv[])
{
  int nc=1000000000; double t1, t2, sum;
  t1 = mytime(0); sum = integrate(f1,0.0,1.0,nc); t1 = mytime(1);
  printf("Time: %lf sec sum = %14.12lf\n",t1,sum);
  t2 = mytime(0); sum = integrate(f2,0.0,1.0,nc); t2 = mytime(1);
  printf("Time: %lf sec sum = %14.12lf diff: %lf\n",t2,sum,t2-t1);
  t2 = mytime(0); sum = integrate(f3,0.0,1.0,nc); t2 = mytime(1);
  printf("Time: %lf sec sum = %14.12lf diff: %lf\n",t2,sum,t2-t1);
  t2 = mytime(0); sum = integrate(f4,0.0,1.0,nc); t2 = mytime(1);
  printf("Time: %lf sec sum = %14.12lf diff: %lf\n",t2,sum,t2-t1);
  return 0;
}
```
Трансляция:
```
>mpicc -o ex02a.px -O2 ex02a.c mycom.c -lm
```
Результат работы:
```
>ex02a.px
Time: 2.764401 sec sum = 3.141592651590
Time: 2.736200 sec sum = 2.418399150979 diff: -0.028201
Time: 2.736536 sec sum = 2.418399151313 diff: -0.027865
Time: 2.740000 sec sum = 3.627598725468 diff: -0.024401
```

Пример 2 (ex02b.c):

```
#include <stdio.h> #include <sys/time.h>
#include "mycom.h"
double f1(double x); double f1(double x){ return (4.0/(1.0 + x*(x+exp(-x))));}
double f2(double x); double f2(double x){ return (4.0/(1.0 + x*x/(x+exp(-x))));}
double f3(double x); double f3(double x){ return (4.0/(1.1 + exp(x) + sin(x)));}
double f4(double x); double f4(double x){ return (4.0/(1.1 + 2.2*exp(x) + 3.3*sin(x)));}
int main(int argc, char *argv[])
{
  int nc=1000000000;
  double t1, t2, dt, sum;

  t1 = mytime(0); sum = integrate(f1,0.0,1.0,nc); t1 = mytime(1);
  t2 = mytime(0); sum = integrate(f2,0.0,1.0,nc); t2 = mytime(1);
  dt = 1.0/dabs(t2-t1);
  printf("Time: %lf %lf sec Div. perf.: %le GFlops\n",t1,t2,dt);

  t1 = mytime(0); sum = integrate(f3,0.0,1.0,nc); t1 = mytime(1);
  t2 = mytime(0); sum = integrate(f4,0.0,1.0,nc); t2 = mytime(1);
  dt = 2.0/dabs(t2-t1);
  printf("Time: %lf %lf sec Mult. perf.: %le GFlops\n",t1,t2,dt);

  return 0;
}
```

Трансляция:

```
>mpicc -o ex02b.px -O2 ex02b.c mycom.c -lm
```

Результат работы:

```
>ex02b.px
Time:  9.514120 10.723616 sec  Div. perf.: 8.267907e-01 GFlops
Time: 17.255178 17.936890 sec Mult. perf.: 2.933790e+00 GFlops
```