НГТУ ФПМИ
Кафедра параллельных вычислительных технологий

# Инструментальные средства разработки программ, измерение времени выполнения программ

2018

# План лекции

- задание 1;
- платформа;
- инструменты разработчика;
- измерение времени.

# Задание 1

1. Написать на Си или Си++ программу умножения двух квадратных матриц.

2. Проверить правильность работы программы на нескольких тестовых наборах входных данных.

3. Измерить время работы подпрограммы умножения матриц для матриц различных размеров. Минимальный размер матриц выбирается такой, при котором время счета приблизительно равно 0.1-0.5 сек. Максимальный размер матриц выбирается так, чтобы время счета было приблизительно равно 30-50 сек.

4. Повторить п. 3 для двух-трех различных уровней оптимизации компилятора. По п.п. 3-4 построить графики зависимости времени счета от размера матриц.

5. Модифицировать программу из п.1 следующим образом. Матрицу, обход которой при умножении происходит по столбцам, необходимо транспонировать до умножения. Т.е. при умножении элементы обеих матриц будут считываться последовательно, по строкам. Для модифицированной программы выполнить п.п. 2-4.

6. Оформить отчет о проделанной работе.

# Разделы отчета

1. Титульный лист.
2. Цель работы.
3. Исходные тексты программ умножения из п.п.1 и 5 со включенным кодом, реализующим замер времени.
4. Графики из п. 4. Оси графиков должны быть подписаны.
5. Вывод по результатам работы.

Работу можно выполнять как в ОС Linux, так и в Windows. Предпочтительный набор инструментов разработчика при реализации работы в ОС Linux - компилятор GCC, отладчик GDB, профайлер GPROF. При реализации в Windows можно использовать Microsoft Visual Studio, CYGWIN+GCC или MinGW+GCC.

# Инструменты разработчика

# Инструменты разработчика

- компилятор;
- интерпретатор;
- библиотекарь и компоновщик (linker);
- отладчик;
- профилировщик;
- верификатор;
- генератор документации;
- генератор исходного текста;
- ...

# Фазы компиляции

- препроцессирование;
- лексический анализ;
- синтаксический анализ;
- генерация кода;
- оптимизация кода.

# Оптимизация кода

- критерии оптимизации;
- виды оптимизационных преобразований;
- уровни в компиляторе GCC.

# Критерии оптимизации

- **Минимизация потребляемых ресурсов:**
  - **память (расход памяти / space cost):**
    - **оперативная или виртуальная, внешняя;**
  - **время (расход времени / time cost):**
    - **процессорное, астрономическое;**
  - **сетевой трафик;**
  - **ресурс внешних устройств, расход энергии, ресурса расходных материалов:**
    - **флэш, эл. энергия.**

# Виды оптимизационных преобразований

- **Оптимизация размера программы:**
  - **удаление мертвого кода;**
  - **свертка выражений.**
- **Оптимизация времени выполнения программы:**
  - **хранение локальных переменных в регистрах;**
  - **раскрутка цикла.**

...

- понижение сложности операции;
- встраивание (inlining);
- оптимизация условного оператора;
- слияние циклов;
- (Loop collapsing);
- вынесение операций из цикла.

# Уровни оптимизации в GCC

-O0 – без оптимизации;

-O1 – простая оптимизация, быстрая компиляция;

-O2 – полная оптимизация, медленная компиляция;

-O3 – O2 + агрессивные оптимизации с циклами и inlining;

-Os – оптимизация размера.

# Форматы файлов

- **объектные файлы:**
  - ***Obj, o;***
- **статические библиотеки:**
  - ***Lib, a;***
- **исполняемые файлы и динамические библиотеки:**
  - ***Exe/mz, a.out, COFF, ELF.***

# Формат ELF

- заголовок
- таблица заголовков программы
- секция .text
- секция .rodata
- ...
- секция .data
- таблица заголовков секции

# Инструменты разработчика

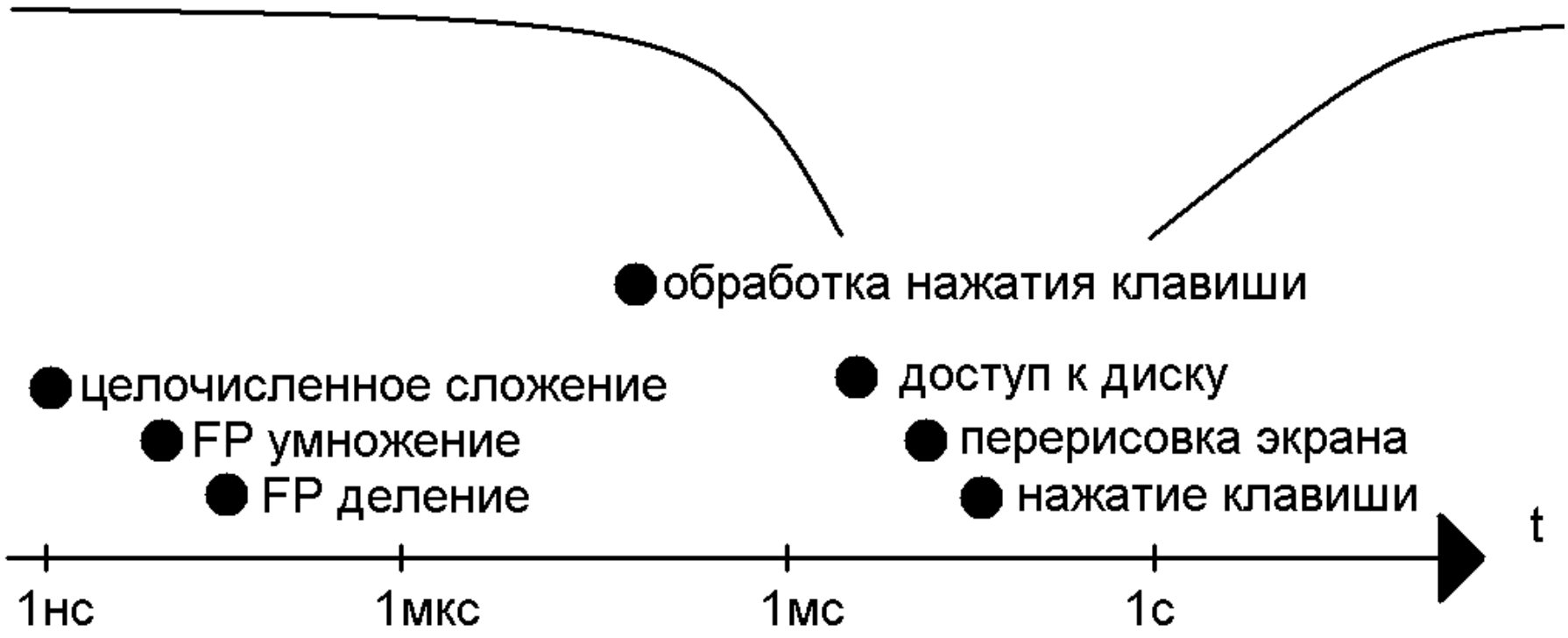
- компилятор;
- интерпретатор;
- библиотекарь и компоновщик (linker);
- отладчик;
- профилировщик;
- верификатор;
- генератор документации;
- генератор исходного текста;
- ...

# Измерение времени в ЭВМ

# Цели измерения времени

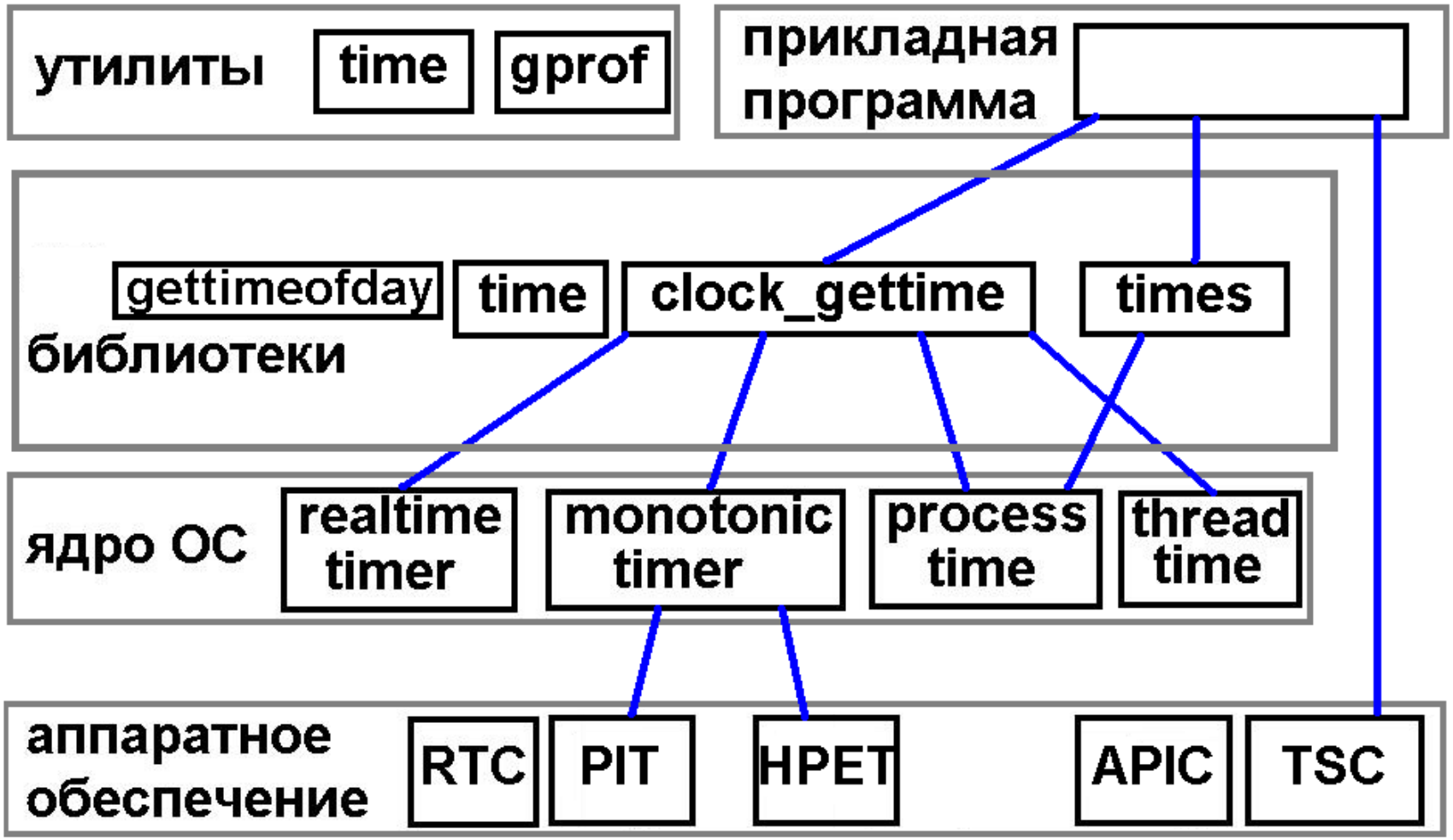
- оценка производительности ЭВМ на тестовых задачах;
- оценка эффективности программы;
- выявление фрагментов программы, подлежащих оптимизации;
- прочее (работа с устройствами ввода/вывода, организация многозадачности, работа с мультимедиа, системы реального времени).

# Временная шкала событий в ЭВМ




**(для системы с тактовой частотой 1 ГГц)**

# Основные способы измерения времени в ОС Linux



# Аппаратное обеспечение в архитектуре x86 для измерения времени

- RTC – Real Time Clock;
- PIT – Programmable Interrupt Controller;
- HPET – High Precision Event Timer;
- APIC – Advanced Programmable Interrupt Controller;
- ACPI Power Management Timer;
- GPS receiver.

# RTC – часы реального времени

- **Отсчет времени даже когда компьютер выключен.**
- **Периодическая (2 – 8192 Гц) или однократная генерация прерывания RTC (IRQ8).**
- **Доступ к функциям RTC через порты ввода/вывода 0x70, 0x71.**

# PIT – программируемый интервальный таймер

- Частота осциллятора около 1МГц.
- 3 счетчика – доступен ОС только один (два других заняты для обновления памяти и управления динамиком).
- Периодическая (100 – 1000 Гц в Linux) генерация прерывания таймера (IRQ0).
- Доступ к функциям PIT через порты ввода/вывода 0x40 - 0x43.

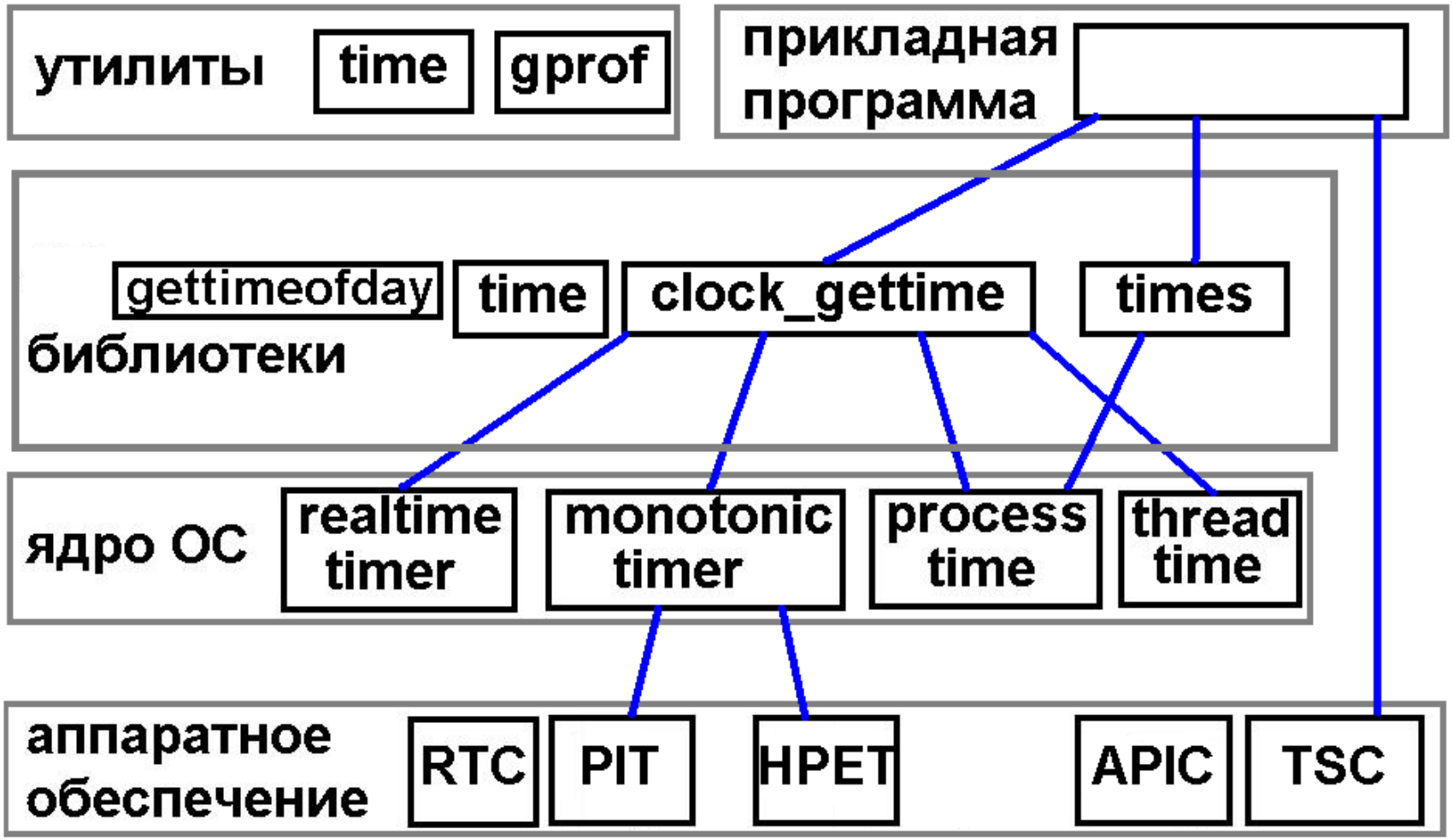
# HPET – высокоточный таймер событий

- **Частота осциллятора – 10МГц.**
- **Периодическая (до 10 МГц) или однократная генерация прерывания таймера;**
- **до 8 счетчиков с собственной частотой;**
- **до 32 таймеров на каждый счетчик.**

# TSC – счетчик тактов

- точный для измерения малых промежутков времени до 10 мс.;
- зависит от архитектуры, есть не для всех архитектур;
- в SMP нужна привязка процессов;
- в современных процессорах с переменной тактовой частотой пользоваться счетчиком затруднительно.

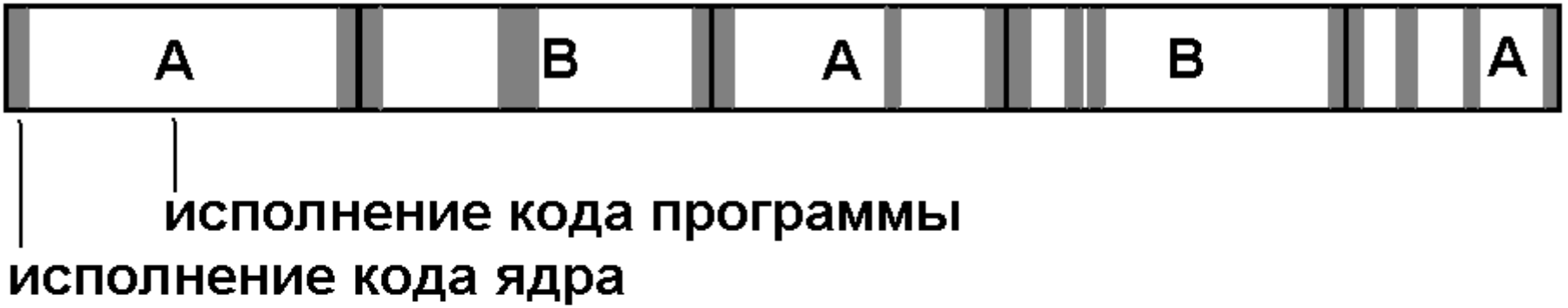
# Основные способы измерения времени в ОС Linux

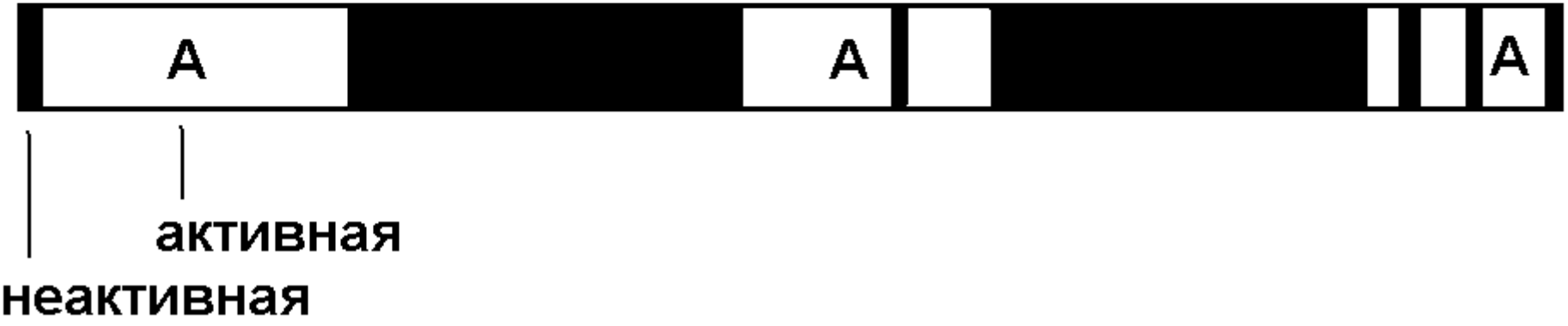


# Таймеры в ядре ОС

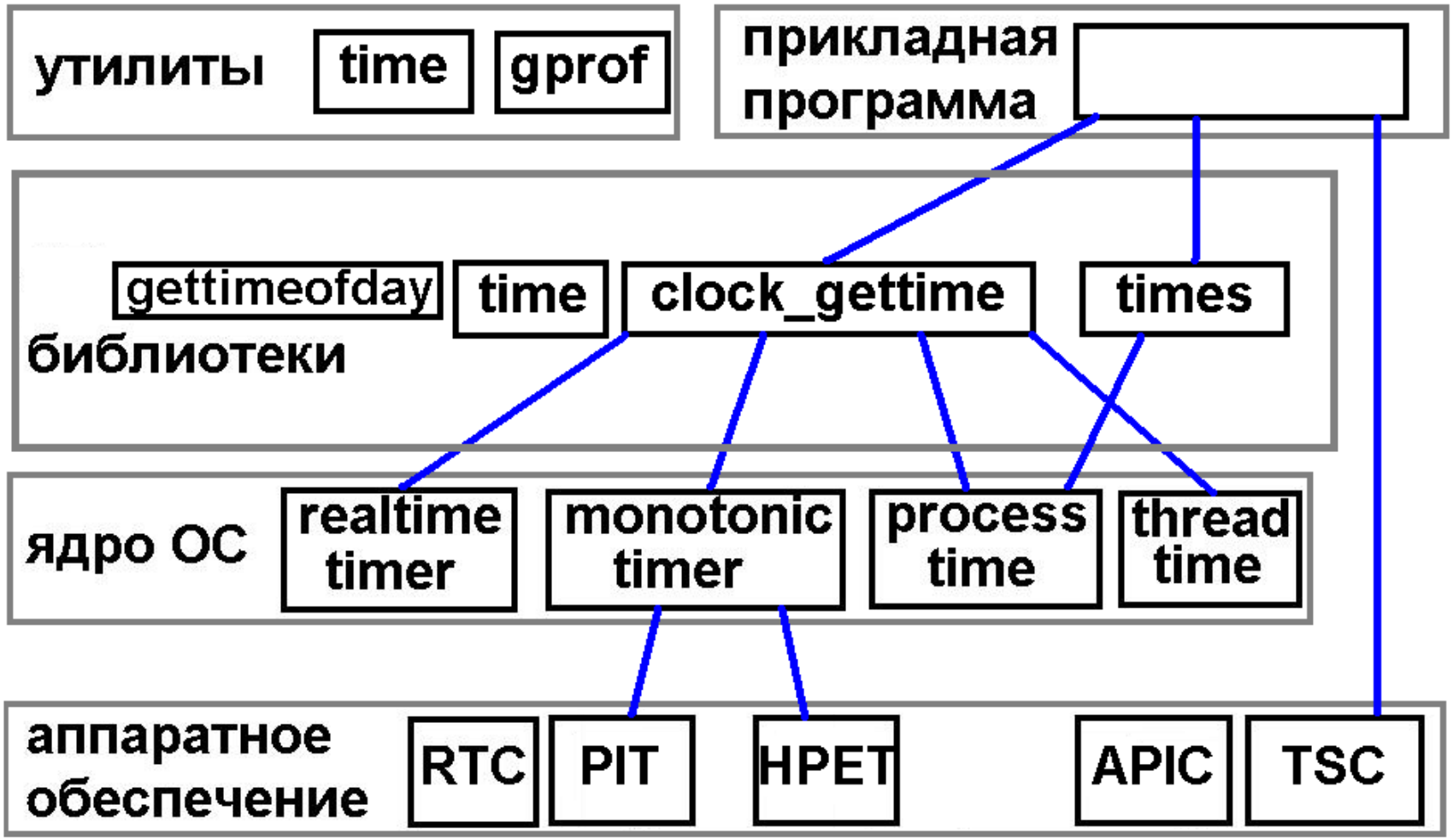
- realtime timer
  – астрономическое время
  (одинаково для всех процессов, запущенных на компьютере)
- monotonic timer;
- process time;
- thread time.

# Измерение времени в системах с разделением времени



# Основные способы измерения времени в ОС Linux



# Уровень библиотек

**Windows:**

- ***GetSystemTime(), GetTickCount(),***
- ***time(), clock();***

**Linux:**

- ***gettimeofday(),***
- ***time(), clock(),***
- ***clock_gettime().***

# Измерение системного времени

## Счетчик системного времени

**Вычислительная система имеет несколько программных и аппаратных счетчиков, отражающих течение времени с различных точек зрения. Необходимо различать следующие счетчики:**

**Счетчик системного времени (system time, wall-clock time) – программный счетчик, который отражает течение времени с точки зрения операционной системы и, как правило, соответствует реальному течению времени. Значение системного времени в каждый момент одинаково для всех программ, работающих на данном компьютере.**

Функции для измерения системного времени:

- Windows: `GetSystemTime()`, `()`, `time()`, `clock()`,
- Linux: `gettimeofday()`, `time()`, `clock()`, `clock_gettime`.

# Измерение системного времени

Функция
`int gettimeofday(struct timeval *tv, struct timezone *tz)`
возвращает в полях tv_sec и tv_usec переменной tv количество секунд и микросекунд, прошедших с полуночи 1 января 1970 года.

**Пример использования:**

```
#include <sys/time.h>
struct timeval tv1, tv2, dtv;
struct timezone tz;
//функция для начала замера времени, сохраняющая в переменной tv1 время начала измерений
void time_start()
{ gettimeofday(&tv1, &tz); }
//функция для окончания замера времени, возвращающая количество миллисекунд, прошедших с начала замера времени
long time_stop()
{ gettimeofday(&tv2, &tz);
  dtv.tv_sec= tv2.tv_sec  - tv1.tv_sec;   //разница секунд
  dtv.tv_usec=tv2.tv_usec - tv1.tv_usec;  //разница микросекунд
  if (dtv.tv_usec<0) { dtv.tv_sec--; dtv.tv_usec+=1000000; }
  //возвращение количества прошедших миллисекунд
  return dtv.tv_sec*1000 + dtv.tv_usec/1000;
}
```

# Измерение времени процесса

## Счетчик времени процесса

**Счетчик времени процесса (process time, CPU time) – программный счетчик, который отражает использование процессорного времени только конкретным процессом. Шаг изменения этого счетчика относительно велик, поэтому его не следует использовать для измерения малых промежутков времени.**

**Функции для получения времени процесса:**

- **Windows: *GetThreadTimes(),GetProcessTimes();***
- **Linux: *times().***

# Измерение времени процесса

**Функция**
**clock_t times (tms *buffer)**

**Возвращает в поле tms_utime переменной buffer количество тактов, потраченных на исполнение инструкций пользовательского процесса с момента начала его исполнения; в поле tms_stime переменной buffer количество тактов, затраченных на исполнение системных вызовов инициированных процессом.**

**Использование функции times. Перевод тактов в миллисекунды производится так же, как и в примере для функции clock.**

```
#include <sys/times.h>
#include <time.h>
struct tms tmsBegin,tmsEnd;
void time_start() { times(&tmsBegin); }
long time_stop()
{ times(&tmsEnd);
  return ((tmsEnd.tms_utime - tmsBegin.tms_utime)+
          (tmsEnd.tms_stime - tmsBegin.tms_stime))*
      1000/CLOCKS_PER_SEC;
}
```

# Счетчик тактов процессора

**rdtsc - ассемблерная инструкция для платформы x86, читающая счётчик TSC (Time Stamp Counter) и возвращающая его в регистрах EDX:EAX 64-битное количество тактов с момента последнего сброса процессора.**

**Пример использования инструкции rdtsc в ОС Linux:**

```
#include <time.h>
long long TimeValue=0;
//функция, возвращающая количество тактов, прошедших с момента последнего сброса процессора
unsigned long long time_RDTSC()
{ union ticks
  { unsigned long long tx;
    struct dblword { long tl,th; } dw;
  } t;
  asm("rdtsc\n": "=a"(t.dw.tl),"=d"(t.dw.th));
  return t.tx;
}
void time_start() { TimeValue=time_RDTSC(); }
long long time_stop() {
    return (time_RDTSC()-TimeValue)*1000/CLOCKS_PER_SEC;
}
```

# Счетчик тактов процессора

**Пример использования инструкции rdtsc в Windows, MS Visual C++:**

```
#include <intrin.h>

unsigned __int64 TimeValue=0;

unsigned __int64 rdtsc(void)
{
   return __rdtsc();
}

void time_start() { TimeValue = rdtsc(); }

long long time_stop()
{
     return (rdtsc() - TimeValue) * 1000 / CLOCKS_PER_SEC;
}
```

# Идентификатор таймера в clock_gettime

**CLOCK_REALTIME**

System-wide realtime clock.

**CLOCK_MONOTONIC _RAW**

Clock that cannot be set and represents monotonic time since some unspecified starting point.

**CLOCK_PROCESS_CPUTIME_ID**

High-resolution per-process timer from the CPU.

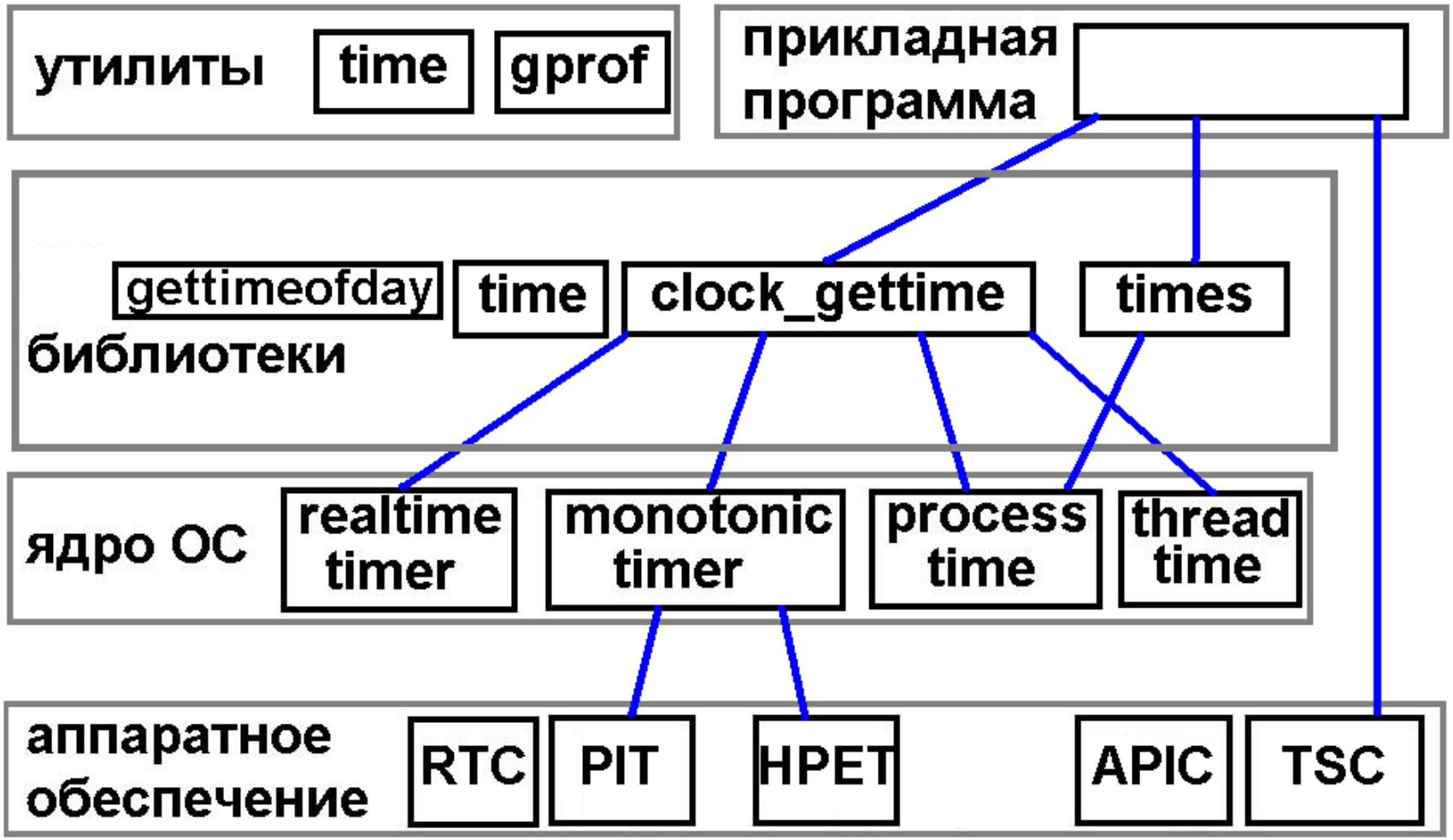
**CLOCK_THREAD_CPUTIME_ID**

Thread-specific CPU-time clock.

# Пример программы измерения времени монотонным таймером

```
int main( int argc, char **argv ){
   struct timespec start, stop;    double accum;

   if( clock_gettime( CLOCK_MONOTONIC_RAW, &start) == -1 ) {
     perror( "clock gettime" ); exit( EXIT_FAILURE );
   }

   system( argv[1] );

   if( clock_gettime( CLOCK_MONOTONIC_RAW, &stop) == -1 ) {
     perror( "clock gettime" ); exit( EXIT_FAILURE );
   }

   accum = ( stop.tv_sec - start.tv_sec )  +
         ( stop.tv_nsec - start.tv_nsec ) / BILLION;
   printf( "%lf\n", accum );
   return( EXIT_SUCCESS );
 }
```

# Основные способы измерения времени в ОС Linux



# Уровень утилит

- **Утилиты измерения времени выполнения программы – *time:***
  - ***real*** **– общее время работы программы согласно системному таймеру;**
  - ***user*** **– время, которое работал процесс (кроме выполнения системных вызовов);**
  - ***sys*** **– время, затраченное процессом на выполнение системных вызовов программы.**
- **Профилировщики:**
  - **GNU Profiler (gprof).**

# Факторы, вносящие искажения в измерение времени и способы их устранения

| Факторы | Способы устранения |
| --- | --- |
| Другие процессы в многозадачных операционных системах | • Остановить другие приложения.<br>• Оставить активными только необходимые сервисы или демоны операционных систем.<br>• Сделать несколько замеров, взять минимальное значение. |
| Виртуальная память, дисковый кэш | • Запустить сброс дискового кэша (sync в Linux) перед каждым запуском программы.<br>• Сделать несколько замеров, взять минимальное значение. |
| Разрешающая способность способа измерения времени | • Подобрать способ в соответствии с априорной оценкой продолжительности. |