НГТУ ФПМИ
Кафедра параллельного программирования

Измерение времени в ЭВМ

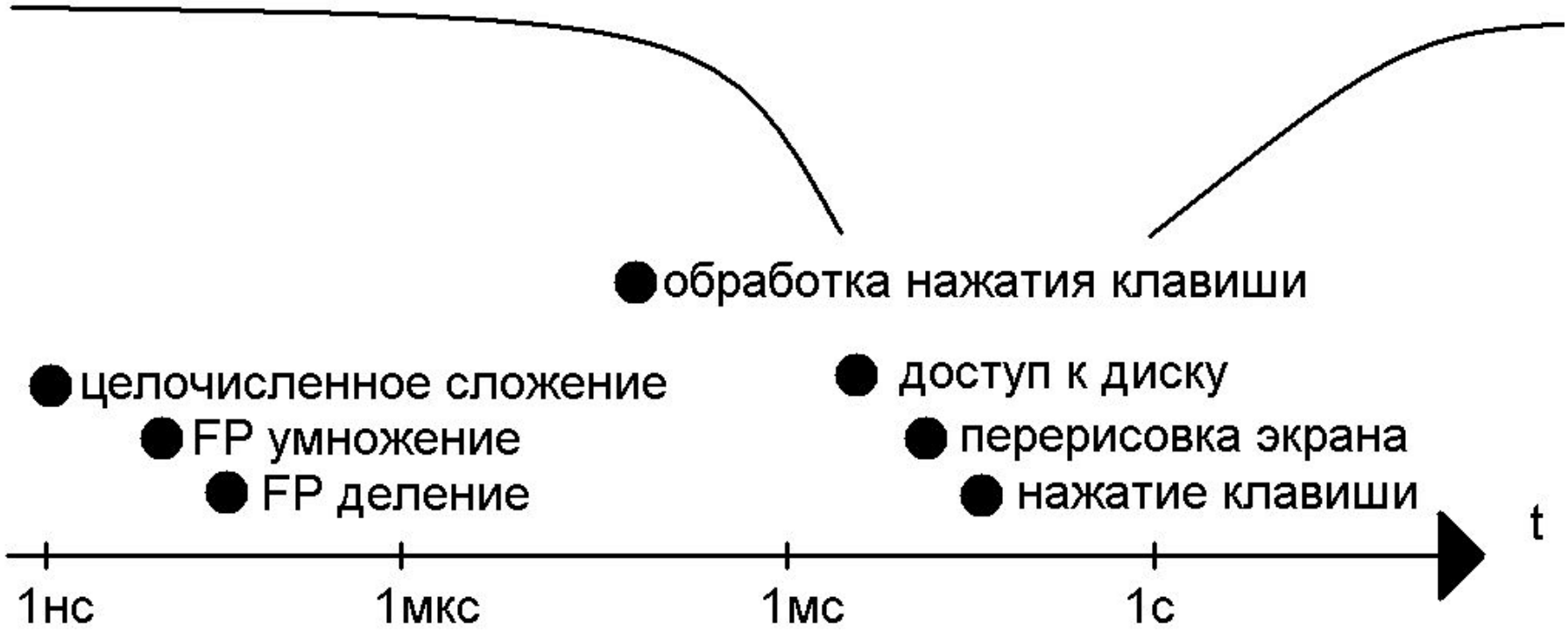
В.П. Маркова, М.Б. Остапкевич

Новосибирск
2014

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

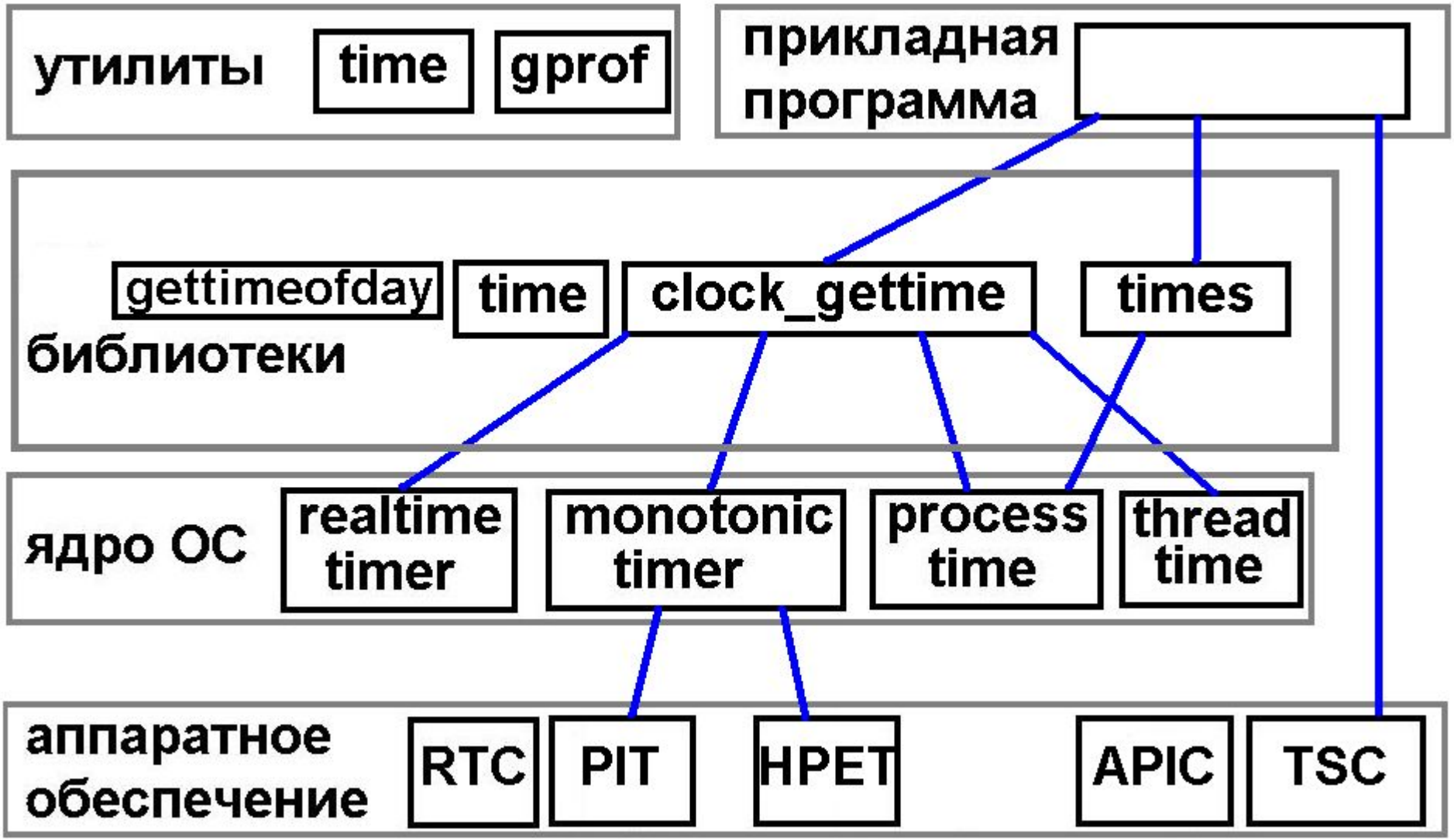


(для системы с тактовой частотой 1 ГГц)

Уровни средств измерения времени

- утилиты;
- библиотеки подпрограмм;
- ядро ОС;
- аппаратное обеспечение.

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

- Периодическая (100 – 1000 Гц в Linux) генерация прерывания таймера (IRQ0)
- Доступ к функциям PIT через порты ввода/вывода 0x40 - 0x43

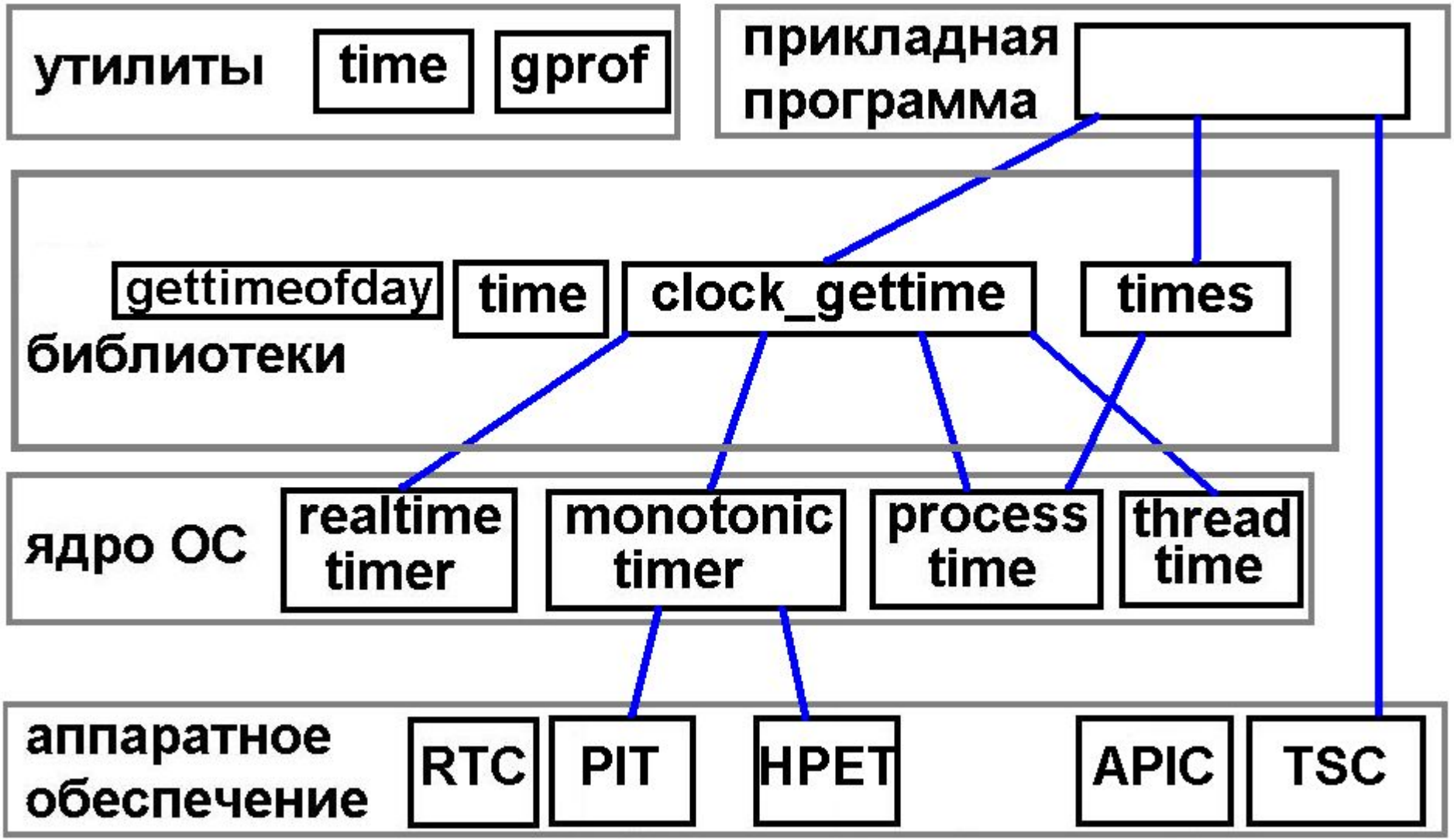
HPET – высокоточный таймер событий

- 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 мсек.;
- 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

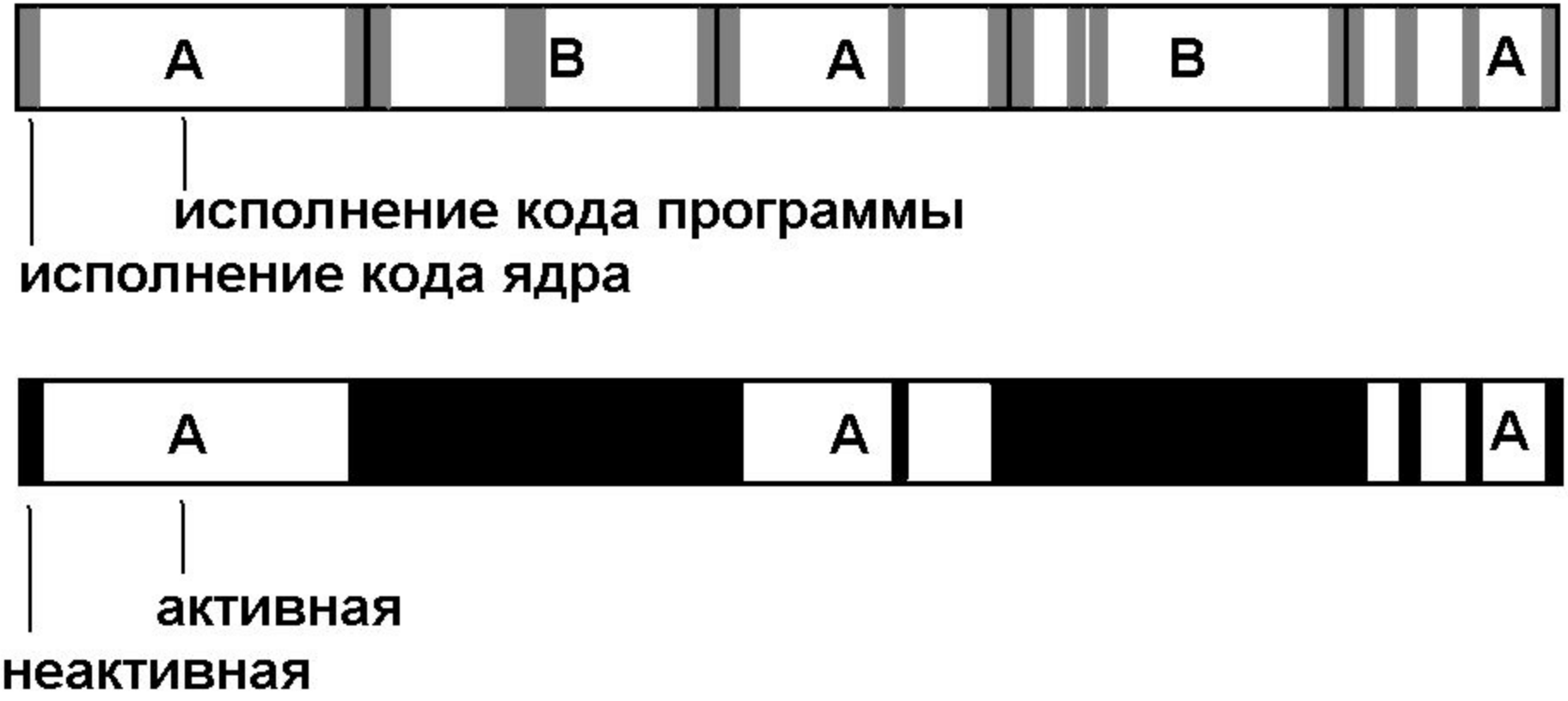
- realtime timer
  – астрономическое время
  (одинаково для всех процессов, запущенных на компьютере)
- monotonic timer;
- process time;
- thread time.

Измерение временных интервалов в языке Си

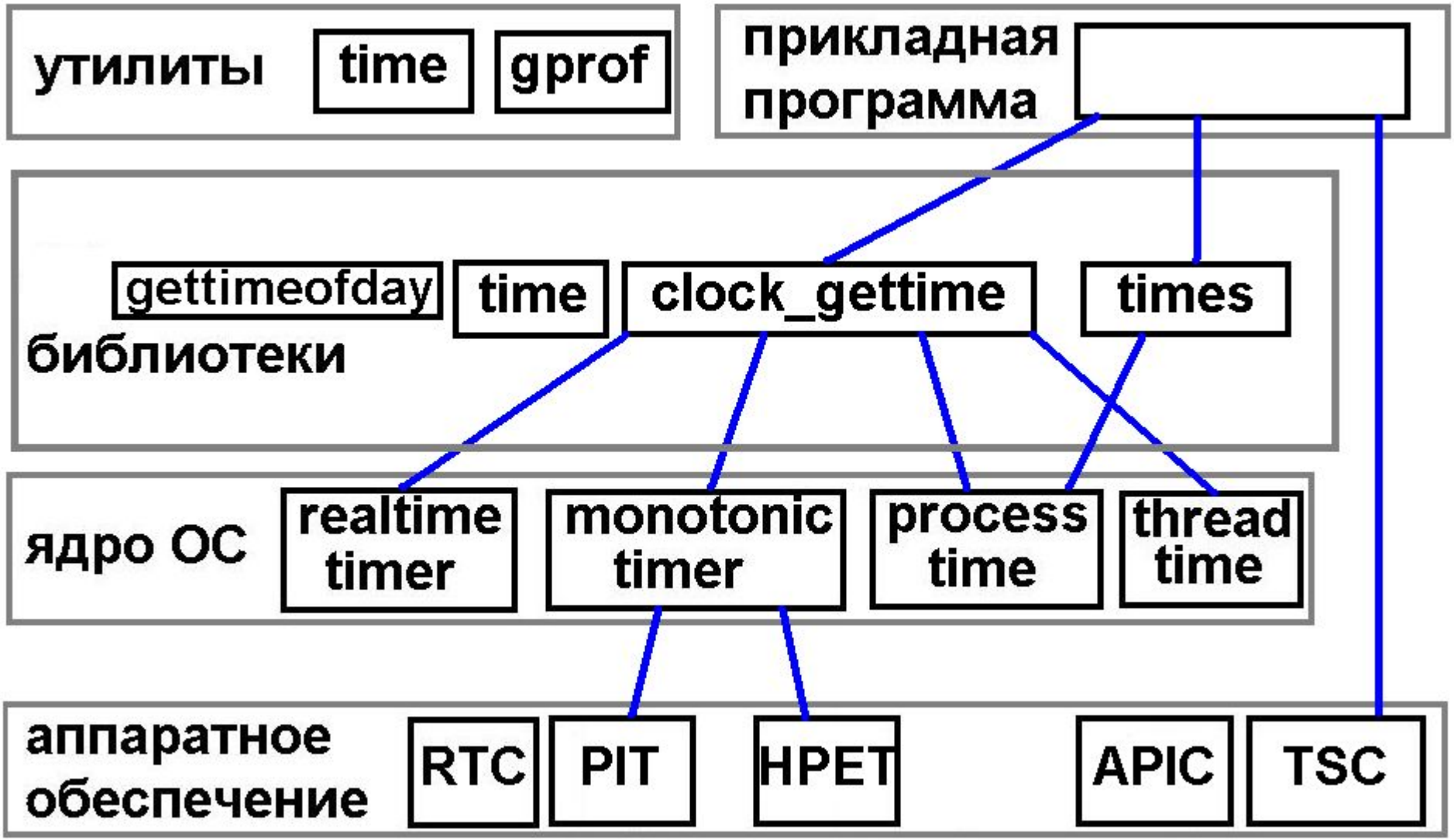
Измерение времени работы программы в многозадачной операционной системе (Windows, Linux, …) имеет свои особенности. В такой системе каждый процессор (ядро) всегда выполняет несколько процессов (программ) в режиме разделения времени.

Операционная система выделяет каждому процессу квант времени процессора и управляет переключением процессора с одного процесса на другой. Таким образом, если замерить время работы некоторой программы внешним хронометром, отражающим реальное течение времени, то этот интервал попадет и время работы каких-то других процессов.

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

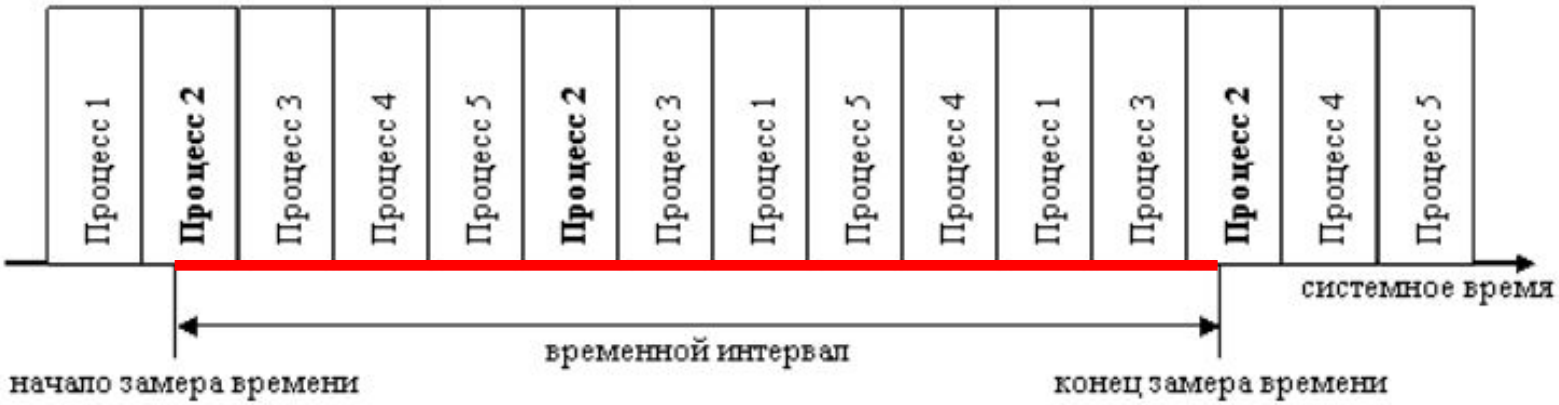
Счетчик системного времени

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(), GetTickCount(), 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 системного времени

Функция
`clock_t clock()`
возвращает количество тактов, прошедших с момента запуска программы

```
#include <time.h>
clock_t tm;
void time_start() { tm = clock(); }
long time_stop()
{
    return (clock() –
tm)*1000/CLOCKS_PER_SEC;
}
```

`CLOCKS_PER_SEC` – константа, значение которой равно количеству тактов исполняемых за секунду. Таким образом,
`(clock() – tm)*1000/CLOCKS_PER_SEC` – количество миллисекунд, прошедших с «момента времени» tm.

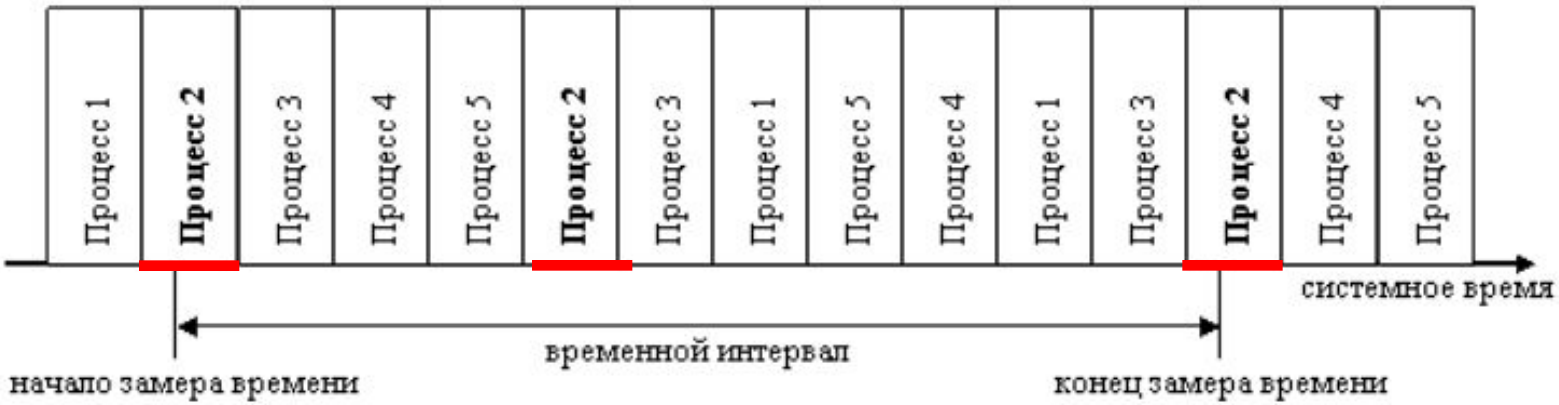
Измерение времени процесса

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;
}
```

Измерение времени процесса

Счетчик тактов процессора

Счетчик тактов процессора (CPU time stamp counter) – аппаратный счетчик, значение которого увеличивается на каждом такте процессора. Такт процессора – самый малый интервал времени в вычислительной системе, который теоретически может быть замерен. Поэтому счетчик тактов позволяет с большой точностью измерять малые промежутки времени (вплоть до нескольких команд процессора).

Счетчик тактов процессора имеет смысл использовать только для измерения интервалов времени меньших кванта времени, выделяемого процессу операционной системой. Для получения значения счетчика тактов используются специальные команды процессора, свои для каждой архитектуры:

- x86/x86-64: `rdtsc` (Read Time Stamp Counter)
- Alpha: `rpcc`,
- Itanium: `ar.itc`,
- PowerPC: `mftb`, `mftbu`.

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. Setting this clock requires appropriate privileges.

CLOCK_MONOTONIC _RAW

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

CLOCK_PROCESS_CPUTIME_ID

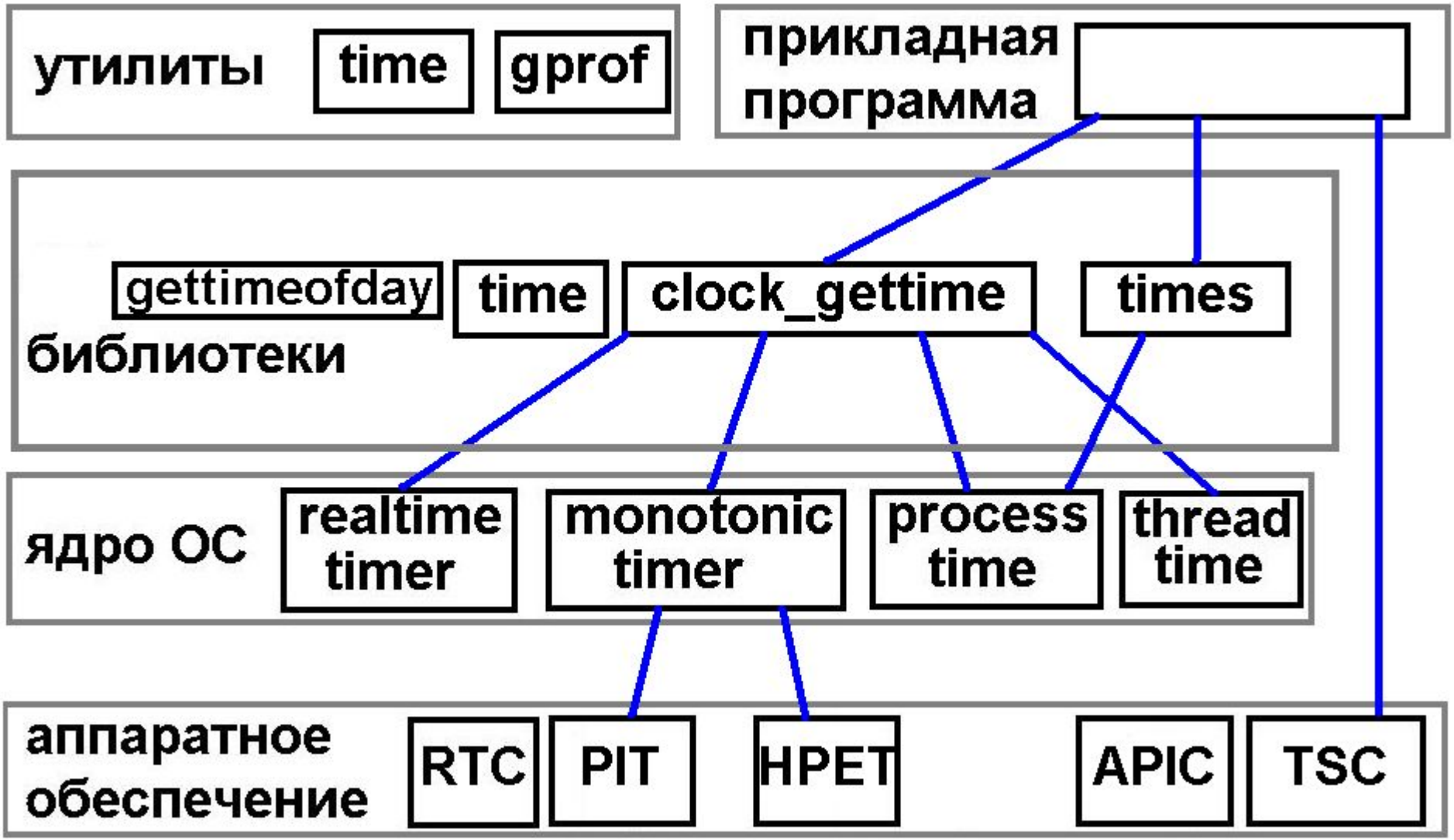
High-resolution per-process timer from the CPU.

CLOCK_THREAD_CPUTIME_ID

Thread-specific CPU-time clock.

```
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 );
 }
```

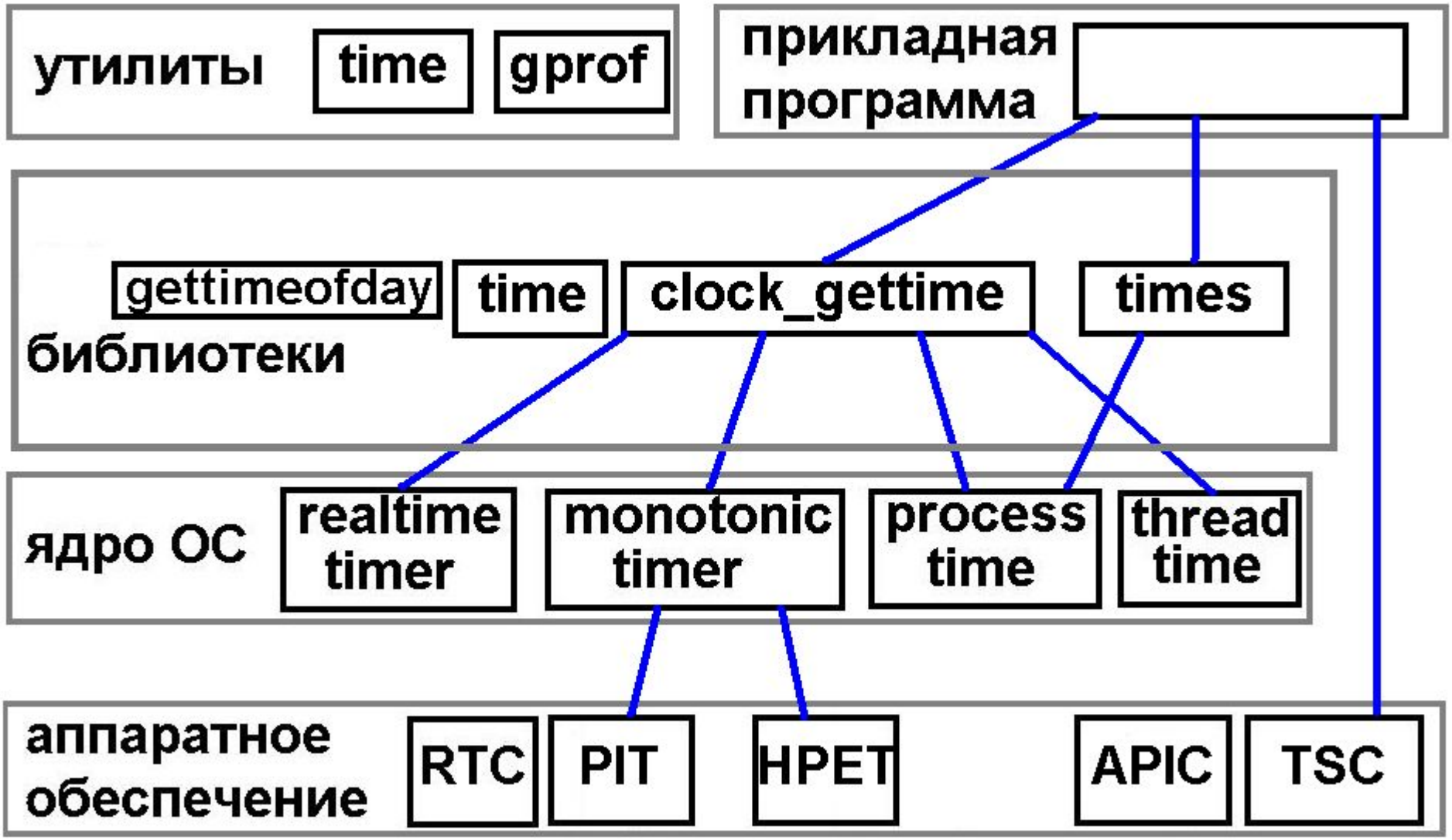
Основные способы измерения времени в ОС Linux



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)**

Основные способы измерения времени в ОС Linux



Факторы, вносящие искажения в измерение интервалов времени

- Исполняемые процессы многозадачной ОС
- Влияние кода, измеряющего время
- Состояние ЭВМ перед началом измеряемого интервала (например, что хранится в кэш-памяти, дисковом кэше)

Пути уменьшения влияния факторов, вносящих искажения

- Остановка лишних процессов
- Многократное повторение измерений
- Сброс дискового кэша
- Уменьшение числа обращений к таймерам
- Многократное повторение счета, увеличивающее время измеряемого интервала

Пути уменьшения влияния факторов, вносящих искажения

- **Другие процессы в многозадачных операционных системах**
  - **остановить другие приложения, оставить активными только необходимые сервисы или демоны ОС,**
  - **сделать несколько замеров, взять минимальное значение.**
- **Виртуальная память, дисковый кэш**
  - **запускать сброс дискового кэша (sync в Linux) перед каждым запуском программы,**
  - **сделать несколько замеров, взять минимальное значение.**
- **Разрешающая способность способа измерения времени**
  - **Подобрать способ в соответствии с априорной оценкой продолжительности.**