Embedded Systems Programming - PA8001

`http://bit.ly/15mmqf7`

Lecture 8

Mohammad Mousavi
`m.r.mousavi@hh.se`

HALMSTAD UNIVERSITY
For the Development of Organisations, Products and Quality of Life

Center for Research on Embedded Systems
School of Information Science, Computer and Electrical Engineering

Real Time

Real Time and a program

- An external process to sample (did that!)
- An external process to react to (did that: remember `AFTER`?)
- An external process to be constrained by.

Constrained by time

Do something before a certain point in time.

Difficult

There is a limit to how fast a processor can work . . .

Execution speed

Fast enough in sequential programs

- use a sufficiently efficient algorithm
- running it on a sufficiently fast computer

Execution time . . .

the time from program start to program stop

. . . depends on input data

So . . . the real issue is whether the Worst Case Execution Time (WCET) for a program on a platform is small enough!

Obtaining WCET

By meassurement

Deal with data dependencies by testing the program on every possible combination of input data.

Usually not feasible! Must find instead a representative subset of all cases!

By analysis

Deal with data dependencies using semantic information and conservative approximations.

Exact analysis is usually no more feasible than exhaustive testing!

WCET by meassurements

Generate test cases automaticaly?

```
int g(int in1, int in2){
  if((in1*in2)%in2==3831)
  // do something that takes 300ms
  else
  // do something that takes 5ms
}
```

How likely is it that it generates data that finds the worst case?

WCET by meassurements

Test all cases?

For one 16-bit integer as input there are 65536 cases.

Test all cases?

For two 16-bit integer as input there are 4 294 967 296 cases.

WCET through analysis

Example

```
for(i=1;i<=10;i++){
  if(E)
  // do something
  // that takes 300ms
  else
  // do something
  // that takes 5ms
}
```

A conservative approximation

Each turn takes 300 ms and so WCET $= 10*300$ ms!

Assume the worst, err on the safe side!

Using semantic information

Suppose `E` is `i<3`. The test is true at most 2 turns, WCET is $2*300+8*5 = 640$ms!

Obtaining WCET

Testing

is likely to find the typical execution times, but finding the worst case is much harder.

Analysis

can always find a safe WCET approximation but comming close to the real WCET is much harder

There is a lot of research about how to obtain WCET, it is beyond the scope of this course dealing with programming techniques.

In this course

We will assume that for any sequential program fragment a safe WCET can be obtained either by meassurement or by analysis or both!

Scheduling

If 2 tasks share a single processor, there are 2 ways of running one before the other

If 3 tasks share a single processor, there are 3*2 ways of running them in series

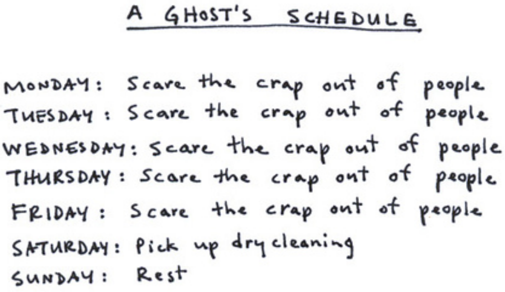
If n tasks share a single processor, there are n! ways of running them.

Interleaving

Moreover, if tasks can be split into arbitrarily small fragments, there are infinitely many ways of running the fragments of even just 2 tasks!

Scheduling

The schedule
is a major factor
in real-time
behaviour of
concurrent tasks!



A GHOST'S SCHEDULE

MONDAY: Scare the crap out of people
TUESDAY: Scare the crap out of people
WEDNESDAY: Scare the crap out of people
THURSDAY: Scare the crap out of people
FRIDAY: Scare the crap out of people
SATURDAY: Pick up dry cleaning
SUNDAY: Rest

Three issues

Deadlines

How do we express the real-time constraints a program must meet?

How do we construct a scheduler that ensures that those constraints are met if at all possible?

Priority scheduling!

Schedulability analysis

How do we tell whether scheduling is impossible? Ahead of time or only when it is too late? (next lecture)

Deadlines

A point in time when some work must be finished is called a deadline.

A deadline is often meassured relative to the occurrence of some event:

- When the bill arrives, pay it whithin 10 days
- At 9am, complete the exam in 5 hours
- When a MIDI note-on message arrives, start emitting a tone within 15 milliseconds

Deadlines

Meeting a deadline

Generate some specific response before the specified time

- Signal level must reach 10mV before ...
- Letter must be post-stamped no later than ...

Deadlines for reactive objects

A point in time when the reaction to an event mut be completed!

Deadlines are naturally meassured relative to the baseline of the current event.

Example 1

When a `SIG_PIN_CHANGE` interrupt occurs, react within 15ms from the time of the interrupt (i.e. the newly defined baseline)

Example 2

When a timer signals that a future baseline is due, react within 200ms from the new baseline

Deadlines for reactive objects

What should qualify as a response to an event?

What must actually be done in order to meet a deadline?

Begin execution?

Does that mean completing the first assembler instruction? Is that observable?

Complete the observable instructions?

For example port writes . . . But not all methods write to ports!

Complete all instructions?

Plausible. But then what about messages a method generates itself?

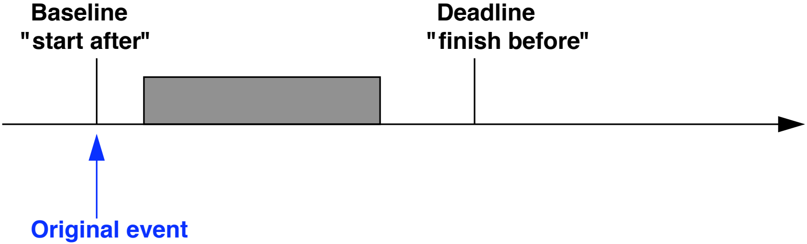
Deadlines for reactive objects

- A `SYNC` message is really executed by the caller . . .
- An `ASYNC` message is just a delegation from one task to another!

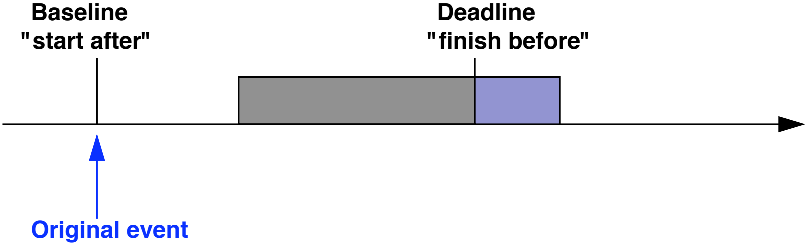
Conclusion

All instructions should be completed before the deadline for all messages of a chain-reaction.

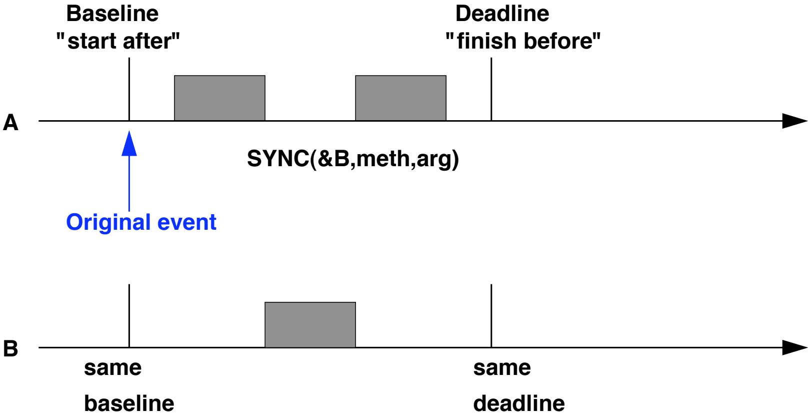
Timely reaction



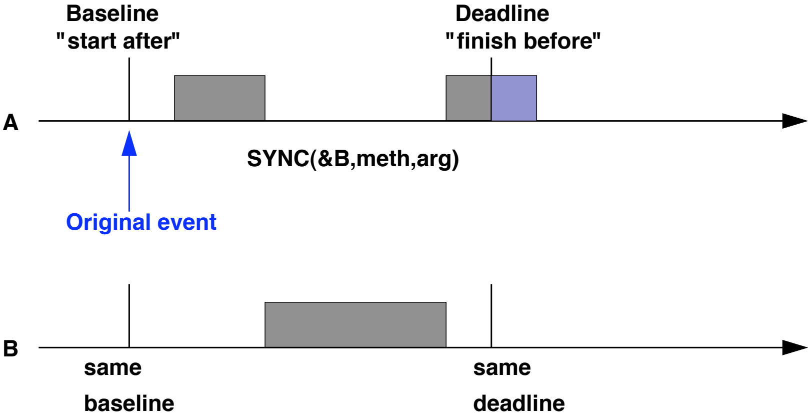
Late reaction



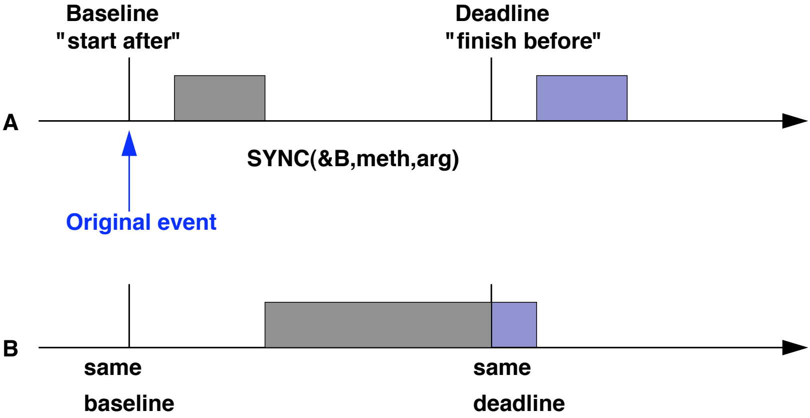
Timely reaction



Late reaction



Late reaction



Baseline
"start after"

Deadline
"finish before"

A

SYNC(&B,meth,arg)

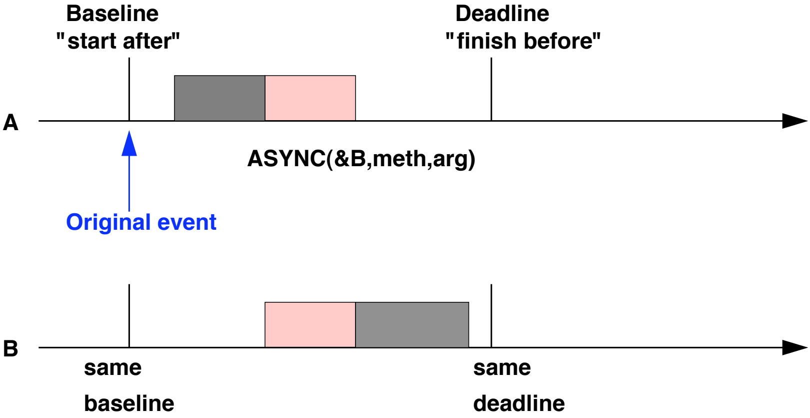
Original event

B

same
baseline

same
deadline

Timely reaction



Baseline
"start after"

Deadline
"finish before"

A

ASYNC(&B,meth,arg)

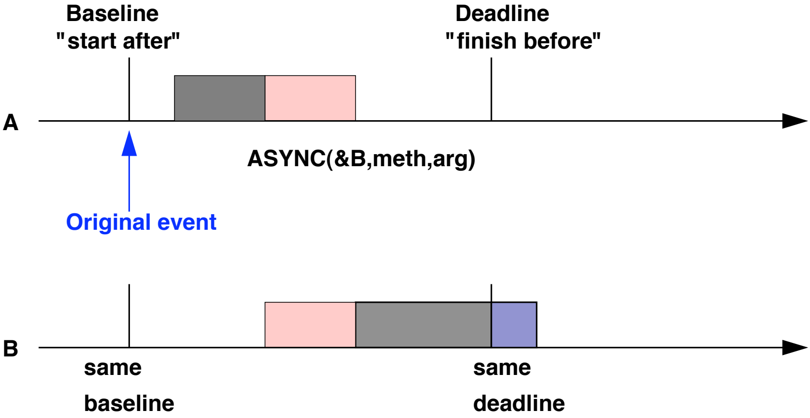
Original event

B

same
baseline

same
deadline

Late reaction



Baseline
"start after"

Deadline
"finish before"

A

ASYNC(&B,meth,arg)

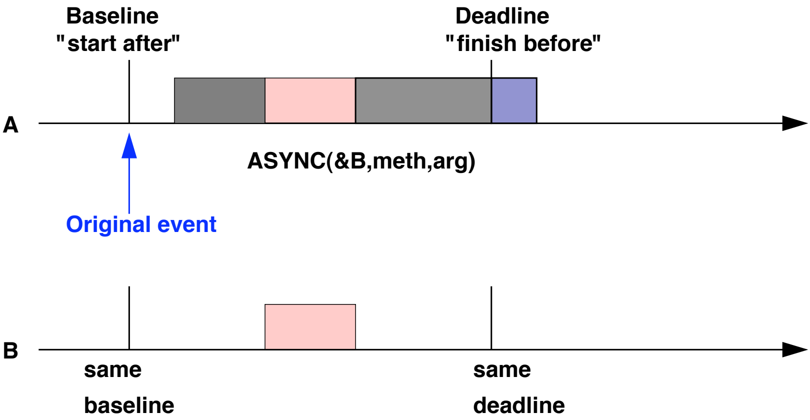
Original event

B

same
baseline

same
deadline

Late reaction



Priorities

Task or Thread or Message priorities are integer values that denote the relative importance of each task.

Quite often the priority scale is reversed!

Low priority values = high priority!

Priority scheduler

Always run the task with the highest priority! *(tasks with the same prio are sorted according to some secondary scheme, e.g. FIFO)*

A task can only run after all tasks considered more important have terminated or are blocked.

Terminology

Static vs. dynamic priorities

- A system where the programmer assigns the priorities of each task is said to use static (or fixed) priorities.
- A system where priorities are automaticaly derived from some other run-time value is using dynamic priorities.

Terminology

Preemptivness

- A system where the scheduler is run only when a task calls the kernel (or terminate) is non-preemptive.
- A system where it also runs as the result of interrupts is called preemptive.

The common case

Preemptive scheduling based on static prios

totally dominates the field of real-time programming.

in OS

Supported by real-time operating systems like QNX, VxWorks, RTLinux, Lynx and standards like POSIX (pthreads)

in Languages

The basis of real-time languages like Ada and Real-time Java

This course

- Preemptive scheduling (`dispatch` might be called within interrupt handlers).
- Static as well as dynamic priorities.

Implementing priority scheduling

```
static void enqueueByPriority(Msg p, Msg *queue){
  Msg prev = NULL;
  Msg q = *queue;
  while(q && (q->priority <= p->priority) ){
    prev=q;
    q=q->next;
  }
  p->next=q;
  if(prev==NULL)
     *queue=p;
  else
     prev->next=p;
}
```

Replace calls to enqueue by calls to `enqueueByPriority`. `Msg` has an extra field! See the reversed scale?

Setting the priority

Could be done like this (but TinyTimber does differently!)

```
void async(Time offset, int prio ,
           Object *to, Method meth, int arg){
  Msg m = dequeue(&msgPool);
  m->to   = to;
  m->meth = meth;
  m->arg  = arg;
  m->baseline = MAX(TIMERGET(),current->baseline+offset);
  m->priority = prio;
  ...
}
```

We discuss TinyTimber later!

Using priorities

Static priorities offer a way of assigning a relative importance to each task/thread/message.

The highest priority task is offered the whole processor.

Any cycles not used by this task are offered to the second but highest priority task.

A task that consumes whatever cycles it is given will effectively disable all lower priority tasks.

Using priorities

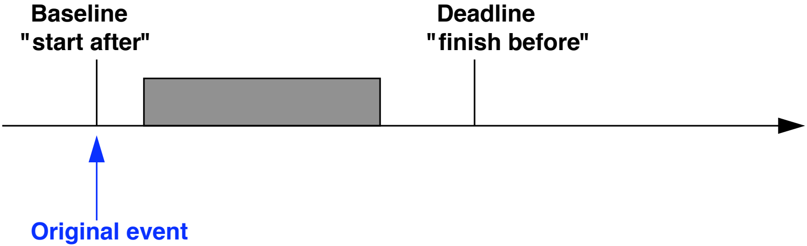
With static priorities, the relative importance of each task must be such that its active execution time is less than the deadline of every task of less importance!

Then all possibilities of interference by several high priority tasks must be taken into account!

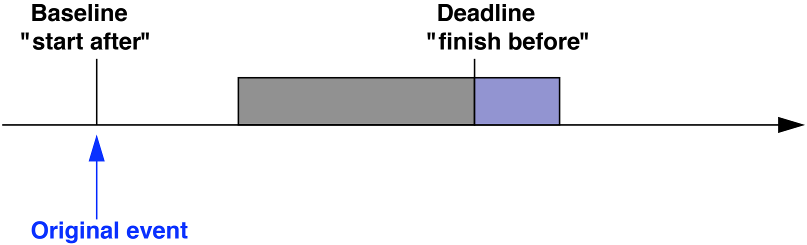
Depends on detailed knowledge (or assumptions) about external event patterns!

Requires means to connect the priority settings to deadline constraints, as well as sophisticated analysis techniques.

Timely reaction



Late reaction

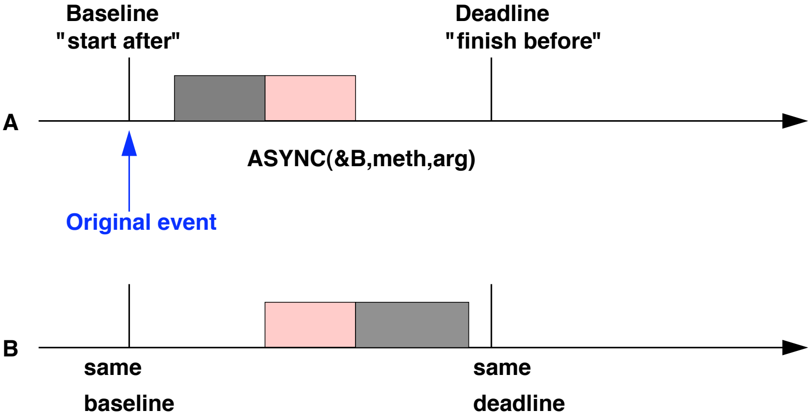


Where will this reaction deadline be defined?

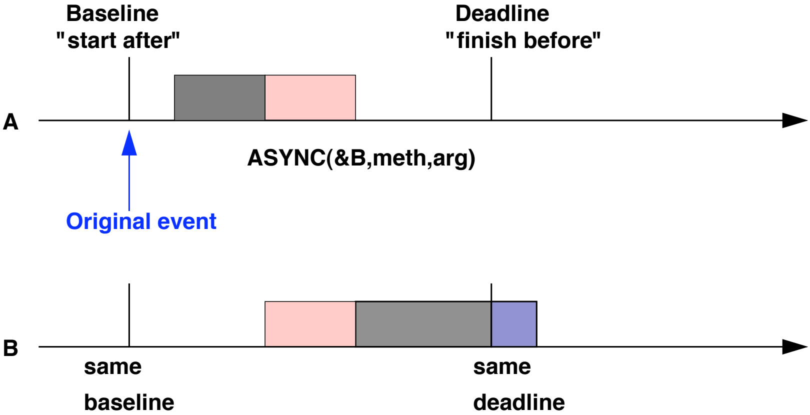
In informal comments only?

Or in concrete source code?

Timely reaction

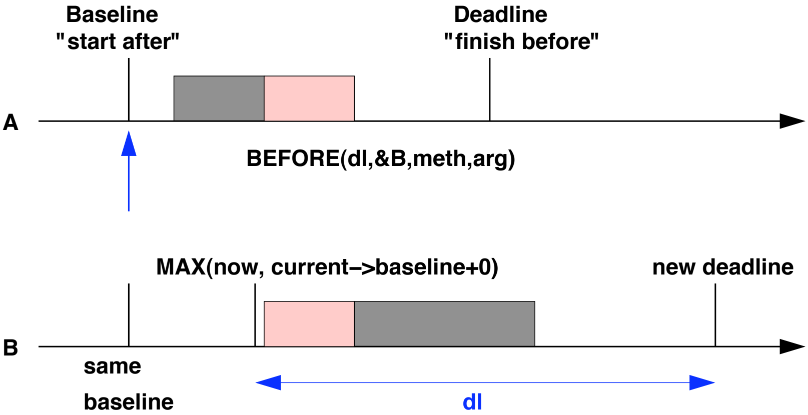


Late reaction

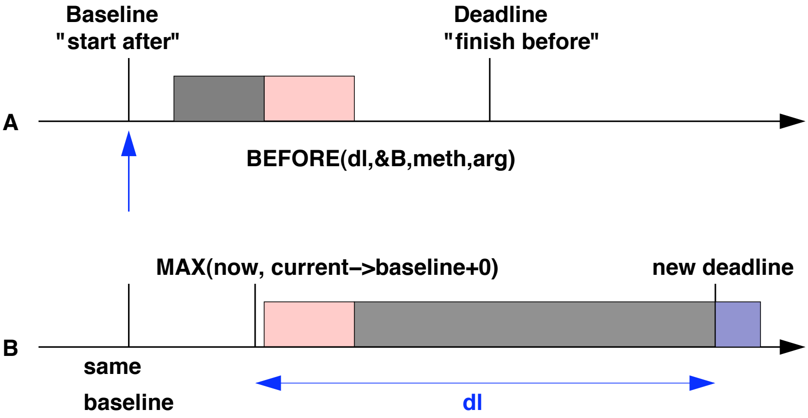


But what if B actually needs a deadline of its own?

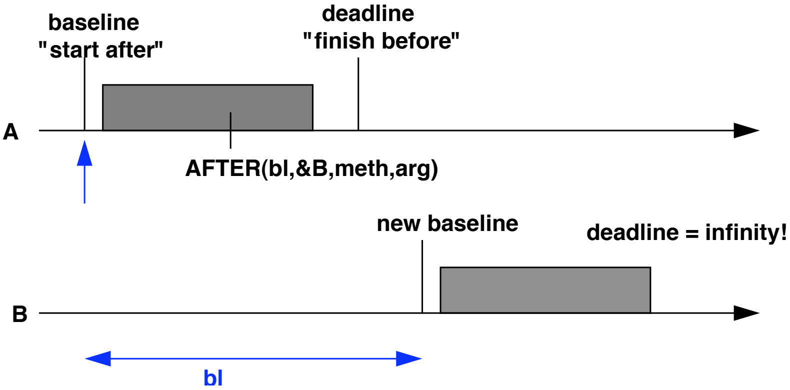
Adjusted deadlines



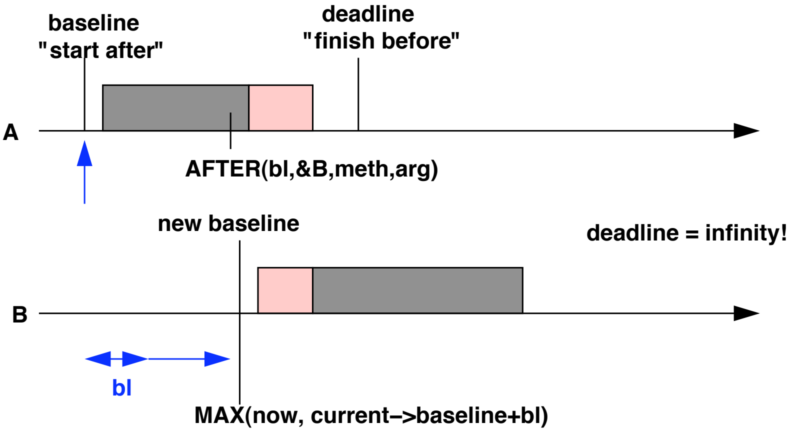
Late reaction



Deadlines and `AFTER`



Deadlines and `AFTER`



baseline
"start after"

deadline
"finish before"

A

AFTER(bl,&B,meth,arg)

new baseline

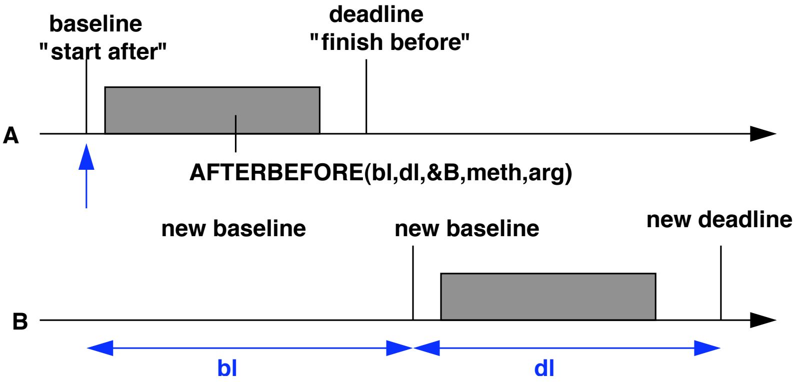
deadline = infinity!

B

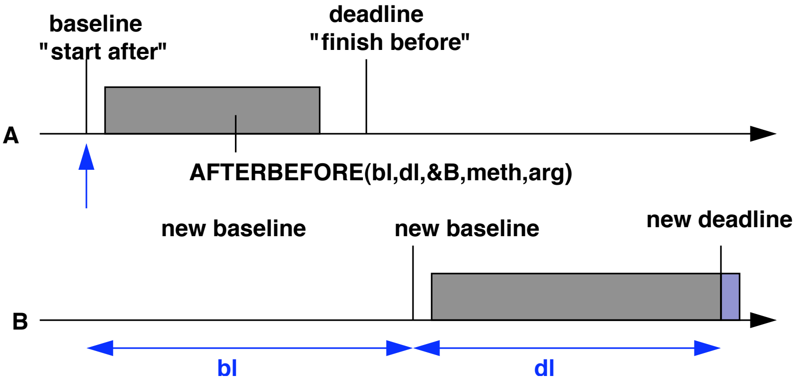
bl

MAX(now, current–>baseline+bl)

Deadlines and `AFTER`



Late reaction



Interrupt handler deadline

Note

Interrupt handlers are scheduled by the CPU hardware, i.e. they will run as fast as possible without regard to any deadline.

Expressing deadlines

In `TinyTimber.h`

```
#define BEFORE(dl, to, meth, arg) \
    SEND(0, dl, to, meth, arg);

#define AFTER(bl, to, meth, arg) \
    SEND(bl, 0, to, meth, arg);

#define ASYNC(to, meth, arg) \
    SEND(0, 0, to, meth, arg);

#define SEND(bl, dl, to, meth, arg) \
    async(bl, dl, to, meth, arg);
```

Defaults for interrupt handlers

`baseline = timestamp` and `deadline = infinity` (0).

Deadlines and priorities

In the application

Using `BEFORE`, we can both define the deadline for a chain of reactions to an external interrupt, and fork off a new chain of reactions with its own deadline at any point.

Inside the kernel

The priorities used will determine in which order messages are scheduled, and hence affect the time when a reaction is able to complete.

Core question

What will be the preferred relation between deadlines and priorities?

Priority assignment

Question

How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

It is possible to get by if we concentrate on programs of a restricted form.