Embedded Systems Programming - PA8001

http://goo.gl/cu8OOH

Lecture 6

Mohammad Mousavi

m.r.mousavi@hh.se

HALMSTAD UNIVERSITY

Center for Research on Embedded Systems

School of Information Science, Computer and Electrical Engineering

Real Time?

In what ways can a program be related to time in the environment (the *real time*)?

Salvador Dali, The Persistence of Memory.

Real Time

An external process to . . .

- Sample: reading a clock,
- React: a handler for an interrupt clock, and
- Constraint: a deadline to respect.

Real Time

An external process to . . .

- Sample: reading a clock,
- React: a handler for an interrupt clock, and
- Constraint: a deadline to respect.

Real Time

An external process to . . .

- Sample: reading a clock,
- React: a handler for an interrupt clock, and
- Constraint: a deadline to respect.

Sampling the time

Requires a hardware clock (read as an external device)

Multitude of alternatives

- Units? Seconds? Milliseconds? CPU cycles?
- Since when? Program start? System boot? Jan 1, 1970?
- Real time? Time stops when: other threads are running? when CPU sleeps? Time that cannot be set and always increases?

Sampling the time

Requires a hardware clock (read as an external device)

Multitude of alternatives

- Units? Seconds? Milliseconds? CPU cycles?
- Since when? Program start? System boot? Jan 1, 1970?
- Real time? Time stops when: other threads are running? when CPU sleeps? Time that cannot be set and always increases?

Sampling the time

Requires a hardware clock (read as an external device)

Multitude of alternatives

- Units? Seconds? Milliseconds? CPU cycles?
- Since when? Program start? System boot? Jan 1, 1970?
- Real time? Time stops when: other threads are running? when CPU sleeps? Time that cannot be set and always increases?

Sampling the time

Requires a hardware clock (read as an external device)

Multitude of alternatives

- Units? Seconds? Milliseconds? CPU cycles?
- Since when? Program start? System boot? Jan 1, 1970?
- Real time? Time stops when: other threads are running? when CPU sleeps? Time that cannot be set and always increases?

Sampling the time

Requires a hardware clock (read as an external device)

Multitude of alternatives

- Units? Seconds? Milliseconds? CPU cycles?
- Since when? Program start? System boot? Jan 1, 1970?
- Real time? Time stops when: other threads are running? when CPU sleeps? Time that cannot be set and always increases?

Timestamps

Relative timing: prevalent in reactive systems, reactions are relative to events

Example

Teacher left 15 min. after the start of the lecture.

In embedded programming, time-stamping an event: reading performed around the event detection.

Timestamps

Relative timing: prevalent in reactive systems, reactions are relative to events

Example

Teacher left 15 min. after the start of the lecture.

In embedded programming, time-stamping an event: reading performed around the event detection.

Timestamps

Relative timing: prevalent in reactive systems, reactions are relative to events

Example

Teacher left 15 min. after the start of the lecture.

In embedded programming, time-stamping an event: reading performed around the event detection.

Time spans

The difference between two time-stamps: a time span independent of the nominal clock values (modulo clock resolution).

The meaning of time-stamp

- The time of some arbitrary program instruction?
- The beginning or end of a function call?
- The time of sending or receiving an asynchronous message?

Too much program dependent!

Time spans

The difference between two time-stamps: a time span independent of the nominal clock values (modulo clock resolution).

The meaning of time-stamp

- The time of some arbitrary program instruction?
- The beginning or end of a function call?
- The time of sending or receiving an asynchronous message?

Too much program dependent!

Time spans

The difference between two time-stamps: a time span independent of the nominal clock values (modulo clock resolution).

The meaning of time-stamp

- The time of some arbitrary program instruction?
- The beginning or end of a function call?
- The time of sending or receiving an asynchronous message?

Too much program dependent!

Time spans

The difference between two time-stamps: a time span independent of the nominal clock values (modulo clock resolution).

The meaning of time-stamp

- The time of some arbitrary program instruction?
- The beginning or end of a function call?
- The time of sending or receiving an asynchronous message?

Too much program dependent!

Time spans

The difference between two time-stamps: a time span independent of the nominal clock values (modulo clock resolution).

The meaning of time-stamp

- The time of some arbitrary program instruction?
- The beginning or end of a function call?
- The time of sending or receiving an asynchronous message?

Too much program dependent!

Time spans

The difference between two time-stamps: a time span independent of the nominal clock values (modulo clock resolution).

The meaning of time-stamp

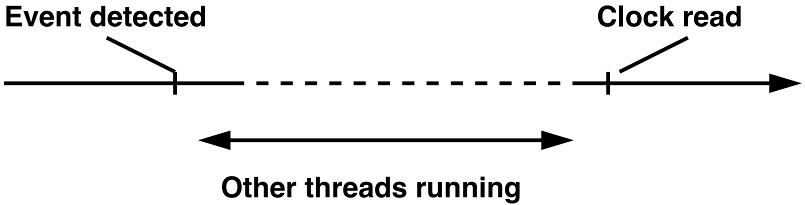
- The time of some arbitrary program instruction?
- The beginning or end of a function call?
- The time of sending or receiving an asynchronous message?

Too much program dependent!

In a scheduled system

What looks like . . .

might very well be . . .

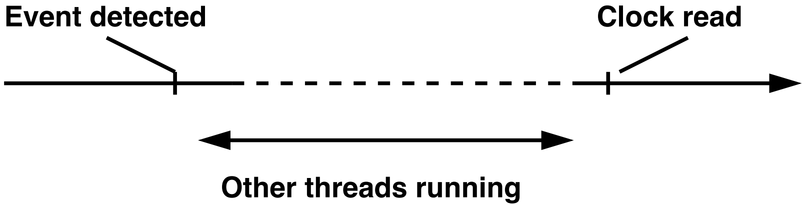


Close proximity is not the same as subsequent statements!

In a scheduled system

What looks like . . .

might very well be . . .



Close proximity is not the same as subsequent statements!

Time-stamping events

Goal: to time-stamp events that *drive* a system

Idea!

Read the clock in the interrupt handler detecting the event

- Disable other interrupts, hence no threads might interfere
- Tight predictable upper bound on the time-stamp error

Time-stamping events

Goal: to time-stamp events that *drive* a system

Idea!

Read the clock in the interrupt handler detecting the event

- Disable other interrupts, hence no threads might interfere
- Tight predictable upper bound on the time-stamp error

Time-stamping events

Goal: to time-stamp events that *drive* a system

Idea!

Read the clock in the interrupt handler detecting the event

- Disable other interrupts, hence no threads might interfere
- Tight predictable upper bound on the time-stamp error

Time-stamping events

Goal: to time-stamp events that *drive* a system

Idea!

Read the clock in the interrupt handler detecting the event

- Disable other interrupts, hence no threads might interfere
- Tight predictable upper bound on the time-stamp error

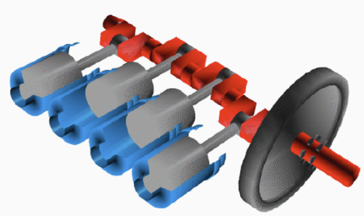
Real-time events to react to

So far: how to sample the real-time clock to know about time

Now: how to take action after a certain amount of time

Example

The wheel is an engine crankshaft and we have to emit ignition signals to each cylinder



How to postpone program execution until certain time

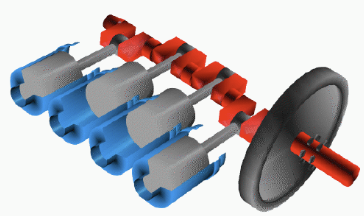
Real-time events to react to

So far: how to sample the real-time clock to know about time

Now: how to take action after a certain amount of time

Example

The wheel is an engine crankshaft and we have to emit ignition signals to each cylinder



How to postpone program execution until certain time

Reacting to real time events

Very poor man's solution

Consume a fixed amount of CPU cycles in a (silly) loop

```
int i;
for(i=0;i<N;i++);  // wait
do_future_action();
```

Problems

1. Determine `N` by testing!
2. `N` will be highly platform dependent!
3. A lot of CPU cycles will simply be wasted!

Reacting to real time events

Very poor man's solution

Consume a fixed amount of CPU cycles in a (silly) loop

```
int i;
for(i=0;i<N;i++);  // wait
do_future_action();
```

Problems

1. Determine N by testing!
2. N will be highly platform dependent!
3. A lot of CPU cycles will simply be wasted!

Reacting to real time events

Very poor man's solution

Consume a fixed amount of CPU cycles in a (silly) loop

```
int i;
for(i=0;i<N;i++);  // wait
do_future_action();
```

Problems

1. Determine N by testing!
2. N will be highly platform dependent!
3. A lot of CPU cycles will simply be wasted!

Reacting to real time events

Very poor man's solution

Consume a fixed amount of CPU cycles in a (silly) loop

```
int i;
for(i=0;i<N;i++);  // wait
do_future_action();
```

Problems

1. Determine `N` by testing!
2. `N` will be highly platform dependent!
3. A lot of CPU cycles will simply be wasted!

Reacting to real time events

Very poor man's solution

Consume a fixed amount of CPU cycles in a (silly) loop

```
int i;
for(i=0;i<N;i++);   // wait
do_future_action();
```

Problems

1. Determine `N` by testing!
2. `N` will be highly platform dependent!
3. A lot of CPU cycles will simply be wasted!

Reacting to real time events

The nearly as poor man's solution

Configure a timer/counter with a known clock speed, and busy-wait for a suitable time increment

```
unsigned int i = TCNT1+N;
while(TCNT1<i); // wait
do_future_action();
```

Problems

1. Determine N by calculation
2. Still a lot of wasted CPU!

Reacting to real time events

The nearly as poor man's solution

Configure a timer/counter with a known clock speed, and busy-wait for a suitable time increment

```
unsigned int i = TCNT1+N;
while(TCNT1<i); // wait
do_future_action();
```

Problems

1. Determine `N` by calculation
2. Still a lot of wasted CPU!

Reacting to real time events

The nearly as poor man's solution

Configure a timer/counter with a known clock speed, and busy-wait for a suitable time increment

```
unsigned int i = TCNT1+N;
while(TCNT1<i); // wait
do_future_action();
```

Problems

1. Determine `N` by calculation
2. Still a lot of wasted CPU!

Reacting to real time events

The nearly as poor man's solution

Configure a timer/counter with a known clock speed, and busy-wait for a suitable time increment

```
unsigned int i = TCNT1+N;
while(TCNT1<i); // wait
do_future_action();
```

Problems

1. Determine `N` by calculation
2. Still a lot of wasted CPU!

Reacting to real time events

The standard solution

Use the OS to *fake* busy-waiting

```
delay(N);    // wait (blocking OS call)
do_future_action();
```

- No platform dependency!
- No wasted CPU cycles (at the expense of a complex OS)

Still a problem ...

... common to all solutions ...

Reacting to real time events

The standard solution

Use the OS to *fake* busy-waiting

```
delay(N);   // wait (blocking OS call)
do_future_action();
```

- No platform dependency!
- No wasted CPU cycles (at the expense of a complex OS)

Still a problem ...

... common to all solutions ...

Reacting to real time events

The standard solution

Use the OS to *fake* busy-waiting

```
delay(N);   // wait (blocking OS call)
do_future_action();
```

- No platform dependency!
- No wasted CPU cycles (at the expense of a complex OS)

Still a problem ...

...common to all solutions ...

Reacting to real time events

The standard solution

Use the OS to *fake* busy-waiting

```
delay(N);   // wait (blocking OS call)
do_future_action();
```

- No platform dependency!
- No wasted CPU cycles (at the expense of a complex OS)

Still a problem ...

. . . common to all solutions . . .

Reacting to real time events

The standard solution

Use the OS to *fake* busy-waiting

```
delay(N);   // wait (blocking OS call)
do_future_action();
```

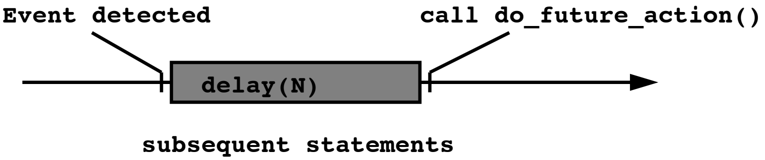
- No platform dependency!
- No wasted CPU cycles (at the expense of a complex OS)

Still a problem . . .

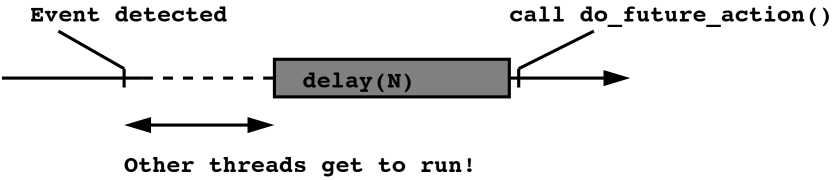
. . . common to all solutions . . .

In a scheduled system

What looks like . . .



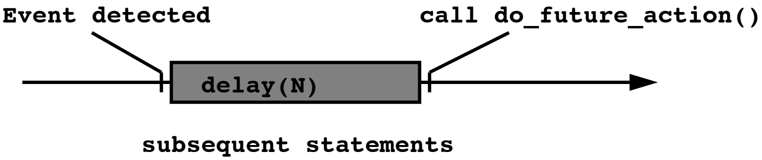
might very well be . . .



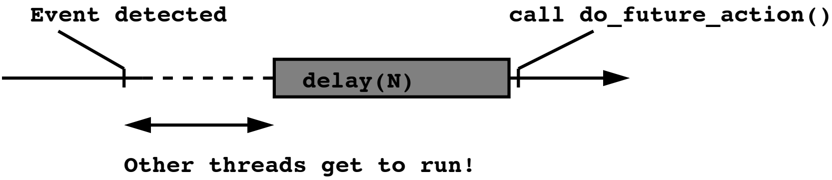
Had we known the scheduler's choice, a smaller N had been used!

In a scheduled system

What looks like ...



might very well be ...



Had we known the scheduler's choice, a smaller N had been used!

Relative delays

The problem: relative time without fixed references:

- The constructed real-time event will occur at after `N` units from *now*.
- What is *now*?!

Other common OS services share this problem: `sleep`, `usleep` and `nanosleep`.

We are not going to use OS services in the course.

Relative delays

The problem: relative time without fixed references:

- The constructed real-time event will occur at after `N` units from *now*.
- What is *now*?!

Other common OS services share this problem: `sleep`, `usleep` and `nanosleep`.

We are not going to use OS services in the course.

Relative delays

The problem: relative time without fixed references:

- The constructed real-time event will occur at after `N` units from *now*.
- What is *now*?!

Other common OS services share this problem: `sleep`, `usleep` and `nanosleep`.

We are not going to use OS services in the course.

Relative delays

The problem: relative time without fixed references:

- The constructed real-time event will occur at after `N` units from *now*.
- What is *now*?!

Other common OS services share this problem: `sleep`, `usleep` and `nanosleep`.

We are not going to use OS services in the course.

Relative delays

The problem: relative time without fixed references:

- The constructed real-time event will occur at after `N` units from *now*.
- What is *now*?!

Other common OS services share this problem: `sleep`, `usleep` and `nanosleep`.

We are not going to use OS services in the course.

Yet another problem

Threads and interleaving make it worse

Example

Consider a task running a CPU-heavy function `do_work()` every 100 millisecods. The naive implementation sing `delay()`:

```
while(1){
  do_work();
  delay(100);
}
```

Yet another problem

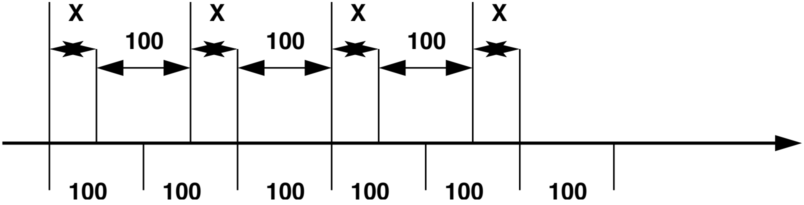
Threads and interleaving make it worse

Example

Consider a task running a CPU-heavy function `do_work()` every 100 millisecods. The naive implementation sing `delay()`:

```
while(1){
  do_work();
  delay(100);
}
```

Accumulating drift

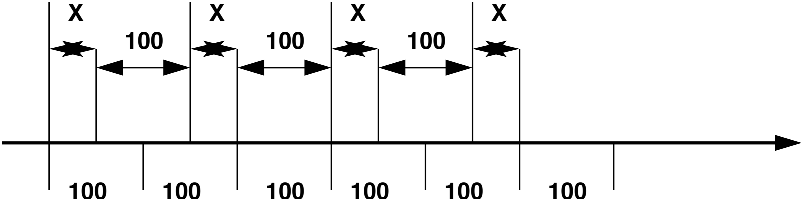


X is the time take to `do_work`

Each turn takes at least 100+X milliseconds.

A drift of X milliseconds will accumulate every turn!

Accumulating drift

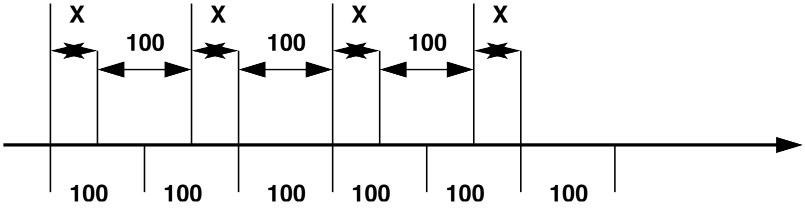


X is the time take to do_work

Each turn takes at least 100+X milliseconds.

A drift of X milliseconds will accumulate every turn!

Accumulating drift

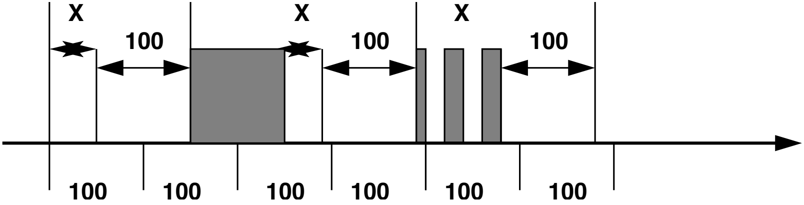


X is the time take to do_work

Each turn takes at least 100+X milliseconds.

A drift of X milliseconds will accumulate every turn!

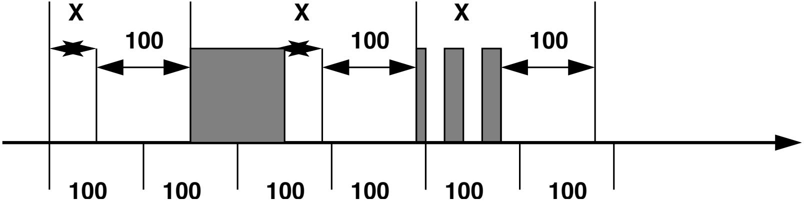
Accumulating drift



With threads and interleaving, the bad scenario gets worse!

Even with a known X, delay time is not predictable.

Accumulating drift



With threads and interleaving, the bad scenario gets worse!

Even with a known `X`, delay time is not predictable.

A stable reference

What we need is a stable time reference to use as a basis whenever we specify a relative time (instead of now).

Baselines

We introduce the baseline of a message to mean the earliest time a message is allowed to start.

Time stamps of interrupts!

The baseline of an event is its time-stamp:

A stable reference

What we need is a stable time reference to use as a basis whenever we specify a relative time (instead of now).

Baselines

We introduce the baseline of a message to mean the earliest time a message is allowed to start.

Time stamps of interrupts!

The baseline of an event is its time-stamp:

A stable reference

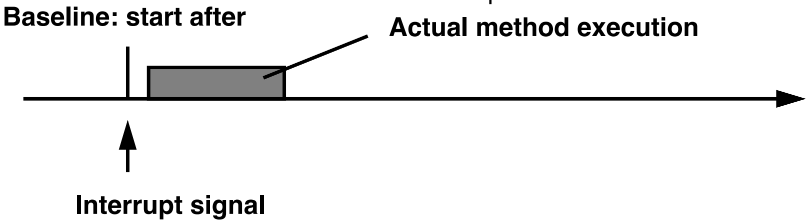
What we need is a stable time reference to use as a basis whenever we specify a relative time (instead of now).

Baselines

We introduce the baseline of a message to mean the earliest time a message is allowed to start.

Time stamps of interrupts!

The baseline of an event is its time-stamp:



Bonus Questions

What are the issues with time in a distributed system? Find out what Lamport Clocks are and explain them (in your own words) in a few lines.
(Please send your answers by email before 17:00 tomorrow.)

Real Time

Real Time and a program

- An external process to sample (did that!)
- An external process to react to (postponed...)
- 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 ...

Real Time

Real Time and a program

- An external process to sample (did that!)
- An external process to react to (postponed...)
- 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 . . .

Real Time

Real Time and a program

- An external process to sample (did that!)
- An external process to react to (postponed...)
- 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 . . .

Real Time

Real Time and a program

- An external process to sample (did that!)
- An external process to react to (postponed...)
- 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!

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!

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!

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!

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!

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!

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!

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!

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!

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 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 = 640ms!

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 = 640ms!

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 = 640ms!

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 = 640ms!

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!

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!

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!

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!

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

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

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

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

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

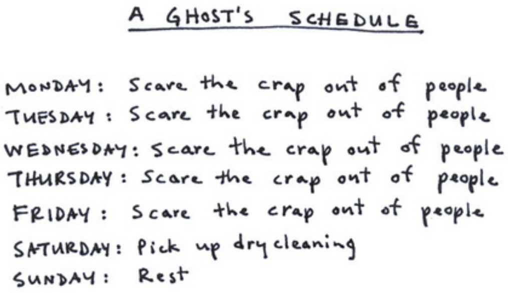
If n tasks share a single processor, there are n! ways of running them.

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!



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)

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)

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)

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

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

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

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

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

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 ...

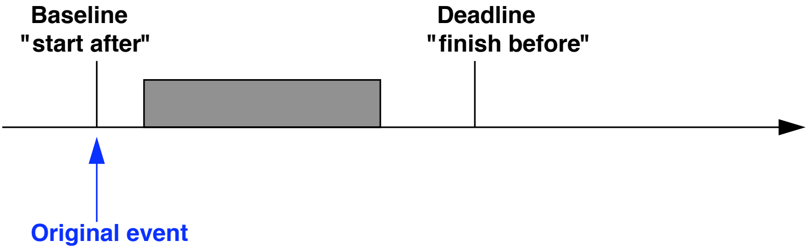
Deadlines

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 ...

Timely 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.

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.

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.

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

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

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.

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.

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.

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.

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.

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?

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?

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

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

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

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.

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.

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.

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.