Scheduling

Goal: assign priorities so that deadlines met.

Outline:

Rate monotonic priority assignment.

Hand priority assignment.

Static scheduling for a *cyclic executive*.

Source

Burns & Wellings, “Real-Time Systems and Programming Languages,” second edition. New York: Addison-Wesley, 1997, chapter 13, pp. 399–440.

Definitions

Scheduling is said to be *effective* if it guarantees deadlines will be met.

A system is called *pure periodic* if

- all events are periodic
- all events’ deadlines are equal to their period
- worst-case execution times are available for all event handlers.

A *distinct priority assignment* is one in which no two events have same priority.

Rate Monotonic Priority Assignment (RMPA)

Method for assigning priorities with goal of meeting deadlines.

Rate monotonic priority assignment does <u>not</u> guarantee deadlines will be met.

A pure periodic system has a *rate monotonic priority assignment* when

- each event triggers an interrupt at a distinct strong priority level
- priority order is the same as frequency order
  (highest priority has shortest period, etc.).

Rate Monotonic Priority Assignment Example

Assign priorities using RMPA for the pure-periodic events described in the table below:

Event Name	Handler Run Time	Event Period
A	$5\,\mu$s	$30\,\mu$s
B	$4\,\mu$s	$22\,\mu$s
C	$30\,\mu$s	$100\,\mu$s

Rate Monotonic Priority Assignment:

Event Name	Handler Run Time	Event Period	Strong Priority
A	$5\,\mu$s	$30\,\mu$s	2
B	$4\,\mu$s	$22\,\mu$s	3
C	$30\,\mu$s	$100\,\mu$s	1

Effectiveness of Rate Monotonic Priority Assignment

RMPA is not effective on all pure periodic systems.

Two results useful for determining effectiveness:

RMPA is effective iff there exists an effective distinct strong-priority assignment.

That is, if RMPA is not effective . . .
. . . then neither is any other assignment of distinct strong priorities.

Safe-Load Test: RMPA is effective if the following relation holds:

$$\sum_{e\in\mathcal{E}} \frac{t_{\mathrm{h}}(e)}{t_{\mathrm{b}}(e)} < |\mathcal{E}| \left(2^{\frac{1}{|\mathcal{E}|}} - 1\right),$$

where $\mathcal{E}$ is the set of event names (*e.g.*, $\mathcal{E} = \{\, A, B, C \,\}$),

$|\mathcal{E}|$ is the number of events (*e.g.*, $|E| = 3$, called N in class),

$t_{\mathrm{h}}(e)$ is the handler run time for event e,

and $t_{\mathrm{b}}(e)$ is the period of event e.

Effectiveness Test Examples

Determine if the RMPA scheduling for the pure-periodic events described in the table below is effective.

Event Name	Handler Run Time	Event Period	Priority	Load
A	$5\,\mu$s	$10\,\mu$s	3	0.5000
B	$4\,\mu$s	$12\,\mu$s	2	0.3333
C	$2\,\mu$s	$15\,\mu$s	1	0.1333

Applying the safe-load test: $\dfrac{29}{30} \approx 0.9667 \overset{?}{<} 3(2^{1/3} - 1) \approx 0.7798$

The relation does not hold, therefore RMPA *may* not be effective in this case.

To determine if it is effective, compute response time by hand:

Response time of C is $20\,\mu$s, including two A's and two B's.

Since response time exceeds period (its assumed deadline), scheduling not effective.

Because RMPA scheduling is not effective here, no other priority scheme effective.

Rate Monotonic Priority Assignment Example II

Determine if the RMPA scheduling for the pure-periodic events described in the table below is effective.

Event Name	Handler Run Time	Event Period	Priority
A	$5\,\mu$s	$10\,\mu$s	3
B	$4\,\mu$s	$15\,\mu$s	2
C	$6\,\mu$s	$30\,\mu$s	1

Applying the safe-load test:

$$\frac{29}{30} \approx 0.9667 \overset{?}{<} 3(2^{1/3} - 1) \approx 0.7798$$

As before test fails, meaning must compute response times to determine effectiveness.

Response time for C is $29\,\mu$s, 3 A's plus 2 B's.

Response time for B is $9\,\mu$s.

Response time for A is $5\,\mu$s.

Since all deadlines met, scheduling effective.

Rate Monotonic Priority Assignment Example III

Determine if the RMPA scheduling for the pure-periodic events described in the table below is effective.

Event Name	Handler Run Time	Event Period	Priority
A	$4\,\mu$s	$10\,\mu$s	3
B	$3\,\mu$s	$15\,\mu$s	2
C	$5\,\mu$s	$30\,\mu$s	1

Applying the safe-load test:

$$\frac{23}{30} \approx 0.7667 \overset{?}{<} 3(2^{1/3} - 1) \approx 0.7798$$

Test passes, so there is no need to compute response times.

 17-9

Manual Priority Assignment

Theorem below shows efficient method to search for priority assignments.

Let $\mathcal{E}$ be a set of pure periodic events, and $L \in \mathcal{E}$. Consider all possible distinct strong priority assignments in which L has the lowest priority. Either L meets its deadlines in all of these assignments or L meets its deadlines in none of these assignments.

In other words, ...
... the response time of the lowest-priority event ...
... does not change if the other priorities are rearranged.

Application

When assigning priorities by hand, assign lowest priority first.

The event will not affect higher priority events' handlers ...
... and assignment of higher priorities can ignore lowest.

 17-10

Static Scheduling

Idea: determine run times in advance.

Static schedule is non-reactive (not reacting to external event).

Plan schedule so that preemption not necessary (maybe not possible).

Result:

Table of handler start times.

Table covers a period of time called a *major cycle*.

OS starts handlers based on table.

Major cycle designed to repeat.

 17-11

Static Scheduling Method

Express periods as integers. (Possibly clock ticks.)

Set table length to least-common multiple (LCM) of periods.[1]

Put handler start times in table so that deadlines met.

If LCM of periods too large then, if possible, adjust periods ...
... or use dynamic scheduling.

[1] The LCM of a set of integers is the smallest integer that is a positive multiple of all the integers. For example, $\mathrm{LCM}\{10, 15, 20\} = 60 = 6 \times 10 + 4 \times 15 + 3 \times 20$.

 17-12

Deadlines and Static Scheduling

Deadline in a dynamically scheduled system based on event time.

No explicit event time in static system.

For problems in class use an assumed event time:

event e with period $t_{\mathrm{b}}(e)$ will occur with period $t_{\mathrm{b}}(e)$ ...

... but with *whatever* phase needed to ensure that deadlines met.

For example, let $t_{\mathrm{b}}(A) = 10\,\mu\mathrm{s}$.

It might occur at $t = 0, 10\,\mu\mathrm{s}, 20\,\mu\mathrm{s}, \ldots$ or $t = 1, 11\,\mu\mathrm{s}, 21\,\mu\mathrm{s}, \ldots$...

... or any other phase that would allow deadlines to be met.

This timing assumption **is not** applied to dynamically scheduled systems ...
... because they *can* react to external events.

Static Scheduling Example

Compute a static schedule for the following system:

Event Name	Handler Run Time	Event Period
A	$4\,\mu$s	$10\,\mu$s
B	$3\,\mu$s	$15\,\mu$s
C	$5\,\mu$s	$30\,\mu$s

LCM = 30, so table covers $30\,\mu$s.

Table:

Time	Action
$0\,\mu$s	Start A
$4\,\mu$s	Start B
$10\,\mu$s	Start A
$17\,\mu$s	Start B ($2\,\mu$s early)
$24\,\mu$s	Start C

Note that the second occurrence of B is $2\,\mu$s early.

Static Scheduling Using a Cyclic Executive

Possible disadvantages of static scheduling as described above:

Large number of timer expirations (specified in table).

A *cyclic executive* reduces the number of timer interrupts . . .

. . . by running handlers in bunches called *bins*.

Cyclic Executive Bins

Bin: Code (maybe handler or daemon) that calls event-specific handlers.

These perform function of handlers in earlier problems.

Handlers within a bin run one after the other (without pause).

First handler in bin runs when bin starts, second when first ends, etc.

Notation:

Bin 1: $\mathcal{B}_1 = (A, B, C, A)$.
Indicates that handlers for A, B, C, and A (again) will run when $\mathcal{B}_1$ runs.

Bin 2: $\mathcal{B}_2 = (A, D, A, C)$.

Indicates that handlers for A, D, A (again), and C will run when $\mathcal{B}_2$ runs.

Cyclic Executive Bins

Bin Timing

Bin starts at fixed interval. (Based on OS timer).

Execution of bin called *minor cycle*.

Time between bin starts also called *minor cycle*.

Different bins may run in consecutive minor cycles, some may repeat.

For example: $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_1, \mathcal{B}_3$ (note that $\mathcal{B}_1$ used twice.)

Time period in which sequence repeats called a major cycle (as with static schedule).

Cyclic Executive Example

Set up a cyclic executive for the pure periodic events described in table below:

Event Name	Handler Run Time	Event Period
A	$4\,\mu$s	$10\,\mu$s
B	$3\,\mu$s	$15\,\mu$s
C	$5\,\mu$s	$30\,\mu$s

Set major cycle to $30\,\mu$s, set minor cycle to $15\,\mu$s.

$\mathcal{B}_1 = (A, B, A)$ and $\mathcal{B}_2 = (B, A, C)$.

The timing above would meet deadlines if the events occurred in the following way:

Event A: $t = -3\,\mu\text{s}, 7\,\mu\text{s}, 17\,\mu\text{s}, 27\,\mu\text{s}, \ldots$.

Event B: $t = 0\,\mu\text{s}, 15\,\mu\text{s}, 30\,\mu\text{s}, \ldots$.

Event C: $t = 22\,\mu\text{s}, 52\,\mu\text{s}, \ldots$.

Cyclic Executive Design

No special method. Use guidelines below.

Minor cycle:

Typically of fixed size (which must divide major cycle).

Longer than longest handler. (May need to divide handlers into parts.)

Try to set minor cycle to greatest common divisor of longer periods.

If major cycle chosen correctly, minor cycle multiple of shorter periods.

Cyclic Executive Tradeoffs

Advantages of Cyclic Executive

- Easier to assure timing than dynamic scheduling.
- Fewer interrupts or other scheduler actions needed than ordinary static scheduling.

Disadvantages of Cyclic Executive

Not useful when periods vary widely. (*E.g.*, $1\,\mu$s, 3 ms.).

Not reactive, must assume phase of periodic events.

Difficult to achieve exact start times for all handlers. (*E.g.*, when bin has more than one handler.)

Cannot be used with non-periodic events.

Cannot easily be used with long running handlers.