

Package: governor (via r-universe)

August 26, 2024

Type Package

Title Speed Limiter to Control Rate of Execution of Loops

Version 0.1.3

Maintainer Mike Cheng <mikefc@coolbutuseless.com>

Description It can be necessary to limit the rate of execution of a loop or repeated function call e.g. to show or gather data only at particular intervals. This package includes two methods for limiting this execution rate; speed governors and timers. A speed governor will insert pauses during execution to meet a user-specified loop time. Timers are alarm clocks which will indicate whether a certain time has passed. These mechanisms are implemented in 'C' to minimize processing overhead.

License MIT + file LICENSE

Encoding UTF-8

URL <https://github.com/coolbutuseless/governor>

BugReports <https://github.com/coolbutuseless/governor/issues>

RoxygenNote 7.3.2

Suggests knitr, rmarkdown

VignetteBuilder knitr

Repository <https://coolbutuseless.r-universe.dev>

RemoteUrl <https://github.com/coolbutuseless/governor>

RemoteRef HEAD

RemoteSha 1021c77c6bdb8999410e7f17988b6b3e2b73ccb8

Contents

gov_enable	2
gov_init	2
gov_wait	3
timer_check	4
timer_enable	5
timer_init	5

Index[7](#)

gov_enable	<i>Disable/enable a governor</i>
------------	----------------------------------

Description

When disabled a governor always returns immediately without any waiting

Usage

```
gov_enable(gov)
```

```
gov_disable(gov)
```

Arguments

gov	object created by gov_init()
-----	------------------------------

Value

None.

Examples

```
gov <- gov_init(1/30)
gov_disable(gov)
gov_enable(gov)
```

gov_init	<i>Initialize a governor to control the speed of a loop</i>
----------	---

Description

Initialize a governor to control the speed of a loop

Usage

```
gov_init(interval, alpha = 0.4, alpha_decay = 0.95, alpha_target = 0.05)
```

Arguments

interval	desired interval in seconds E.g. interval = 1.5 sets the time-per-loop to 1.5 seconds. E.g. interval = 1/30 sets the loop to run at 30 times per second
alpha	initial learning rate used to adjust wait time. Default: 0.4
alpha_decay	rate at which alpha decays. Default: 0.95 i.e. 5 each iteration
alpha_target	the baseline alpha to reach when running long term. default: 0.05

Value

gov object to be used with gov_wait()

Examples

```
# This loop should take approx 1 second as the governor will limit
# the loop to run every thirtieth of a second.
gov <- gov_init(1/30)
system.time({
  for (i in 1:30) {
    gov_wait(gov)
  }
})
```

gov_wait

Wait an appropriate amount of time within a for-loop to match the desired interval set in gov

Description

Wait an appropriate amount of time within a for-loop to match the desired interval set in gov

Usage

```
gov_wait(gov)
```

Arguments

gov object created by gov_init()

Value

Logical value. If TRUE then the governor is suggesting that the work for next loop be skipped if the loop interval is to be maintained in the long term.

Examples

```
# This loop should take approx 1 second
gov <- gov_init(1/30) # interval = 0.0333 seconds
system.time({
  for (i in 1:30) {
    Sys.sleep(0.01) # Work done in loop
    gov_wait(gov) # Compensate to keep interval loop time
  }
})
```

timer_check	<i>Check the status of a timer</i>
-------------	------------------------------------

Description

Check the status of a timer

Usage

```
timer_check(timer)
```

Arguments

timer timer object returned by timer_init()

Value

logical indicating if timer was triggered. If TRUE, then the internal state of the timer will be reset (so that it will trigger again after another duration has elapsed)

Examples

```
# Run two timers in a tight while loop
# The short timer should trigger every 0.1 seconds
# The long timer will trigger after 1 second
# Note that the timers will reset every time they trigger (after returning TRUE)
long_timer <- timer_init(1)
short_timer <- timer_init(0.1)
counter <- 0L
while(TRUE) {
  if (timer_check(long_timer)) {
    cat("\nLong timer fired at count: ", counter, "\n")
    break;
  }
  if (timer_check(short_timer)) {
    cat("Short timer fired at count: ", counter, "\n")
  }
  counter <- counter + 1L
}
```

timer_enable	<i>Disable/enable a timer</i>
--------------	-------------------------------

Description

When disabled a timer always immediately returns FALSE

Usage

```
timer_enable(timer)

timer_disable(timer)
```

Arguments

timer	timer object returned by timer_init()
-------	---------------------------------------

Value

None.

Examples

```
timer <- timer_init(1)
timer_disable(timer)
timer_enable(timer)
```

timer_init	<i>Create a timer object which will return TRUE when checked and the given duration has elapsed.</i>
------------	--

Description

The timer will automatically reset any time it is checked (via check_timer()) and it returns TRUE.

Usage

```
timer_init(duration, reset_mode = "checked")
```

Arguments

duration	timer duration in seconds
reset_mode	one of 'checked' (default) or 'created' . If 'checked', then when the timer is reset the next alarm will be set to duration seconds after timer_check() last returned TRUE. If 'created', then the time is reset to the next increment of N * duration after the timestamp when the timer was created

Value

a timer object to used with timer_check()

Examples

```
# Run two timers in a tight 'while' loop
# The short timer should trigger every 0.1 seconds
# The long timer will trigger after 1 second
# Note that the timers will reset every time they trigger (after returning TRUE)
long_timer <- timer_init(1)
short_timer <- timer_init(0.1)
counter <- 0L
while(TRUE) {
  if (timer_check(long_timer)) {
    cat("\nLong timer fired at count: ", counter, "\n")
    break;
  }
  if (timer_check(short_timer)) {
    cat("Short timer fired at count: ", counter, "\n")
  }
  counter <- counter + 1L
}
```

Index

gov_disable (gov_enable), 2

gov_enable, 2

gov_init, 2

gov_wait, 3

timer_check, 4

timer_disable (timer_enable), 5

timer_enable, 5

timer_init, 5