# BuguRTOS

0.7.0

Generated by Doxygen 1.8.1.2

Sat May 17 2014 21:12:46

# Contents

## 1 Main Page

The BuguRTOS is a RTOS kernel. It is written by anonimous JUST FOR FUN.

**Warning**

BuguRTOS license is modifyed GPLv3, look at exception.txt for more info.

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

# 3 File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# 4 Data Structure Documentation

## 4.1 _gitem_t Struct Reference

A grouped list item.

`#include "include/gitem.h"`

### Data Fields

- item_t parent
- group_t * group
- group_t grp

#### 4.1.1 Detailed Description

A grouped list item.

Such an item is allways in some group (see group_t). Every gitem_t object has a group_t field, that field is a group that holds this gitem_t object at the begining.

#### 4.1.2 Field Documentation

##### 4.1.2.1 item_t parent

A tarent type is item_t.

##### 4.1.2.2 group_t* group

A pointer to current group.

##### 4.1.2.3 group_t grp

A group field, used as initial group.

The documentation for this struct was generated from the following file:

- include/gitem.h

## 4.2 _group_t Struct Reference

A group of gitem_t objects.

`#include "include/gitem.h"`

#### Data Fields

- void * link
- prio_t prio
- count_t el_num

### 4.2.1 Detailed Description

A group of gitem_t objects.

A gitem_t object, stores priority information of grouped gitem_t objects, and apointer to an xlist_t container, which holds these gitem_t objects.

Every gitem_t object has group_t field, this field is initial group for this object. When gitem_t object is transfered to gxlist_t container gitem->group may be pushed to local pool of that container.

### 4.2.2 Field Documentation

#### 4.2.2.1 void* link

Stores an information about container, or about next group in a pool.

#### 4.2.2.2 prio_t prio

A group priority.

#### 4.2.2.3 count_t el_num

The number of gitem_t objects in group, link counter.

The documentation for this struct was generated from the following file:

- include/gitem.h

## 4.3 _gxlist_t Struct Reference

#### Data Fields

- xlist_t parent
- pool_t pool

### 4.3.1 Field Documentation

#### 4.3.1.1 xlist_t parent

Parent type is xlist_t.

#### 4.3.1.2 pool_t pool

Unused group_t objects container.

The documentation for this struct was generated from the following file:

- include/gitem.h

## 4.4 _item_t Struct Reference

A list item.

```
#include "include/item.h"
```

### Data Fields

- item_t * next
- item_t * prev

### 4.4.1 Detailed Description

A list item.

All structures, that must be listed, will inherit item_t properties and methods.

### 4.4.2 Field Documentation

#### 4.4.2.1 item_t* next

Next item in a list.

#### 4.4.2.2 item_t* prev

Previous item in a list.

The documentation for this struct was generated from the following file:

- include/item.h

## 4.5 _kernel_t Struct Reference

A BuguRTOS kernel structure.

```
#include "include/kernel.h"
```

### Data Fields

- sched_t sched
- proc_t idle
- timer_t timer
- void(* timer_tick )(void)

### 4.5.1 Detailed Description

A BuguRTOS kernel structure.

The kernel stores information about launched processes, system time and other important information.

### 4.5.2 Field Documentation

#### 4.5.2.1 sched_t sched

The scheduler.

#### 4.5.2.2 proc_t idle

The IDLE process.

#### 4.5.2.3 timer_t timer

The system timer.

#### 4.5.2.4 void(* timer_tick)(void)

The system timer tick hook pointer.

The documentation for this struct was generated from the following file:

- include/kernel.h

## 4.6 _mutex_t Struct Reference

A mutex.

```
#include "include/mutex.h"
```

### Data Fields

- xlist_t wait
- bool_t free
- proc_t * owner
- count_t dirty

### 4.6.1 Detailed Description

A mutex.

Mutexes are used to control an access to common data. If your code needs yo use some common data for a long time, then you should use mutex instead of critical section. Mutex nesting is supported.

#### Warning

Only a process can lock or free a mutex!
Locked mutex can be freeed only by a locker process!

### 4.6.2 Field Documentation

#### 4.6.2.1 xlist_t wait

A list of waiting processes.

#### 4.6.2.2 bool_t free

This flag is 1 when mutex is free and 0 when mutex is locked.

#### 4.6.2.3 proc_t* owner

A pointer to a process, that holds a mutex.

#### 4.6.2.4 count_t dirty

Dirty priority inheritanse transaction counter.

The documentation for this struct was generated from the following file:

- include/mutex.h

## 4.7 _pcounter_t Struct Reference

A locked resource counter.

`#include "include/pcounter.h"`

### Data Fields

- count_t counter [BITS_IN_INDEX_T]
- index_t index

### 4.7.1 Detailed Description

A locked resource counter.

pcounter_t objects are used to count mutex controled resources locked by processes when CONFIG_USE_HIGHEST_LOCKER is defined.

### 4.7.2 Field Documentation

#### 4.7.2.1 count_t counter[BITS_IN_INDEX_T]

A counter array.

#### 4.7.2.2 index_t index

An index to speedup search.

The documentation for this struct was generated from the following file:

- include/pcounter.h

## 4.8 _pool_t Struct Reference

Pool of groups;.

`#include "include/gitem.h"`

### Data Fields

- group_t * top
- group_t * bot

### 4.8.1 Detailed Description

Pool of groups;.

Pool is a stack container for group_t objects.

### 4.8.2 Field Documentation

#### 4.8.2.1 group_t* top

Top of pool.

#### 4.8.2.2 group_t* bot

Bottom of pool.

The documentation for this struct was generated from the following file:

- include/gitem.h

## 4.9 _proc_t Struct Reference

A process.

```
#include "include/proc.h"
```

### Data Fields

- gitem_t parent
- flag_t flags
- prio_t base_prio
- pcounter_t lres
- timer_t time_quant
- timer_t timer
- void * buf
- code_t pmain
- code_t sv_hook
- code_t rs_hook
- void * arg
- stack_t * sstart
- stack_t * spointer

### 4.9.1 Detailed Description

A process.

There are many OSes, so It may be called a process, a thread, a task etc. The point of all these names is: independent sequence of CPU instructions.

So a process is a part of your program, that has its own "main" routine (stored in pmain field of proc_t object). A process "main" routine can be written in a way as if there were no other processes!

It's possible to use one "main" routine for many processes, as differents processes are independent, but you have to remember one thing about static variables in such "main" routine.

#### Warning

Be carefull with static variables, these variables are common for all processes sharing one routine! You must access such static variables using process synchronization facilities.

### 4.9.2 Field Documentation

#### 4.9.2.1 gitem_t parent

A parent is gitem_t.

#### 4.9.2.2 flag_t flags

Process state flags (to treat process state quickly).

#### 4.9.2.3 prio_t base_prio

A base process priority.

#### 4.9.2.4 pcounter_t lres

A locked resource counter.

#### 4.9.2.5 timer_t time_quant

A process time slice.

#### 4.9.2.6 timer_t timer

A process timer, it is used as watchdog for real time processes

#### 4.9.2.7 void* buf

Apointer to process IPC data storage.

#### 4.9.2.8 code_t pmain

A pointer to a process "main" routine.

#### 4.9.2.9 code_t sv_hook

A context save hook, it is run after saving a process context.

#### 4.9.2.10 code_t rs_hook

A context restore hook, it is run before restoring a process context.

#### 4.9.2.11 void* arg

An argument for pmain, sv_hook, rs_hook, may be used to store process local data.

#### 4.9.2.12 stack_t* sstart

A process stack bottom pointer.

#### 4.9.2.13 stack_t* spointer

A process stack top pointer.

The documentation for this struct was generated from the following file:

- include/proc.h

## 4.10 _sched_t Struct Reference

A scheduler.

```
#include "include/sched.h"
```

### Data Fields

- proc_t * current_proc
- xlist_t * ready
- xlist_t * expired
- xlist_t plst [2]
- count_t nested_crit_sec

#### 4.10.1 Detailed Description

A scheduler.

A scheduler oject contains an information about processes, running on some CPU core.

#### 4.10.2 Field Documentation

##### 4.10.2.1 proc_t* current_proc

A currently running process.

##### 4.10.2.2 xlist_t* ready

A pointer to a ready process list.

##### 4.10.2.3 xlist_t* expired

Apointer to an expired process list.

##### 4.10.2.4 xlist_t plst[2]

A storage for a ready and for an expired process lists.

##### 4.10.2.5 count_t nested_crit_sec

A critical section nesting count.

The documentation for this struct was generated from the following file:

- include/sched.h

### 4.11 _sem_t Struct Reference

A counting semaphore.

`#include "include/sem.h"`

#### Data Fields

- xlist_t wait
- count_t counter

#### 4.11.1 Detailed Description

A counting semaphore.

Counting semaphores are used for process synchronization. It is not recomended to use them in common data access control, because priority inversion is possible. A counting semaphore can be locked by one process and freeed by another.

#### 4.11.2 Field Documentation

##### 4.11.2.1 xlist_t wait

xlist_t is parrent type.

#### 4.11.2.2 count_t counter

A counter.

The documentation for this struct was generated from the following file:

- include/sem.h

## 4.12 _sig_t Struct Reference

A signal.

`#include "include/sig.h"`

### Data Fields

- gxlist_t wait
- gxlist_t wakeup

### 4.12.1 Detailed Description

A signal.

Signals are used for process-event synchronization. A process can wait for a signal. Other process or interrupt handler can fire a signal and launch one or all processes waiting for that signal.

### 4.12.2 Field Documentation

#### 4.12.2.1 gxlist_t wait

A waiting process list.

#### 4.12.2.2 gxlist_t wakeup

A list of processes to wake up.

The documentation for this struct was generated from the following file:

- include/sig.h

## 4.13 _xlist_t Struct Reference

A prioritized list.

`#include "include/xlist.h"`

### Data Fields

- item_t * item [BITS_IN_INDEX_T]
- index_t index

### 4.13.1 Detailed Description

A prioritized list.

A container type, xlist_t objects store lists of item_t objects. In fact these containers store lists of #pitem_t or other compatible objects.

#### 4.13.2 Field Documentation

##### 4.13.2.1 item_t* item[BITS_IN_INDEX_T]

An array of list head pointers.

##### 4.13.2.2 index_t index

Index for fast search.

The documentation for this struct was generated from the following file:

- include/xlist.h

### 4.14 ipc_exchange_arg_t Struct Reference

An argument structure for SYSCALL_IPC_EXCHANGE.

#### Data Fields

- ipc_send_arg_t send
- ipc_data_t * receive

#### 4.14.1 Detailed Description

An argument structure for SYSCALL_IPC_EXCHANGE.

#### 4.14.2 Field Documentation

##### 4.14.2.1 ipc_send_arg_t send

A parent.

##### 4.14.2.2 ipc_data_t* receive

Apointer to storage for data to receive.

The documentation for this struct was generated from the following file:

- kernel/ipc.c

### 4.15 ipc_send_arg_t Struct Reference

An argument structure for SYSCALL_IPC_SEND.

#### Data Fields

- proc_t * proc
- bool_t ret
- ipc_data_t ipc_data

#### 4.15.1 Detailed Description

An argument structure for SYSCALL_IPC_SEND.

#### 4.15.2 Field Documentation

##### 4.15.2.1 proc_t* proc

A pointer to a destignation process.

##### 4.15.2.2 bool_t ret

A storage for a result.

##### 4.15.2.3 ipc_data_t ipc_data

A data to send.

The documentation for this struct was generated from the following file:

- kernel/ipc.c

### 4.16 mutex_lock_arg_t Struct Reference

An argument structure for SYSCALL_MUTEX_LOCK and SYSCALL_MUTEX_TRY_LOCK.

#### Data Fields

- mutex_t * mutex
- bool_t ret

#### 4.16.1 Detailed Description

An argument structure for SYSCALL_MUTEX_LOCK and SYSCALL_MUTEX_TRY_LOCK.

#### 4.16.2 Field Documentation

##### 4.16.2.1 mutex_t* mutex

A pointer to a mutex.

##### 4.16.2.2 bool_t ret

A storage for a result.

The documentation for this struct was generated from the following file:

- kernel/mutex.c

### 4.17 proc_runtime_arg_t Struct Reference

An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

#### Data Fields

- proc_t * proc
- bool_t ret

#### 4.17.1 Detailed Description

An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

#### 4.17.2 Field Documentation

##### 4.17.2.1 proc_t* proc

A pointer to a process.

##### 4.17.2.2 bool_t ret

A result storage.

The documentation for this struct was generated from the following file:

- kernel/proc.c

## 4.18 proc_set_prio_arg_t Struct Reference

An argument for system call SYSCALL_PROC_SET_PRIO.

#### Data Fields

- proc_t * proc
- prio_t prio

#### 4.18.1 Detailed Description

An argument for system call SYSCALL_PROC_SET_PRIO.

#### 4.18.2 Field Documentation

##### 4.18.2.1 proc_t* proc

A pointer to a process.

##### 4.18.2.2 prio_t prio

Priority.

The documentation for this struct was generated from the following file:

- kernel/proc.c

## 4.19 sem_lock_arg_t Struct Reference

An argument structure for SYSCALL_SEM_LOCK and SYSCALL_SEM_TRY_LOCK.

#### Data Fields

- sem_t * sem
- bool_t ret

#### 4.19.1 Detailed Description

An argument structure for SYSCALL_SEM_LOCK and SYSCALL_SEM_TRY_LOCK.

#### 4.19.2 Field Documentation

##### 4.19.2.1 sem_t* sem

A pointer to a semaphore.

##### 4.19.2.2 bool_t ret

A storage for a result.

The documentation for this struct was generated from the following file:

- kernel/sem.c

# 5 File Documentation

## 5.1 include/bugurt.h File Reference

The top header file.

```
#include "index.h"
#include "item.h"
#include "xlist.h"
#include "gitem.h"
#include "pcounter.h"
#include "crit_sec.h"
#include "proc.h"
#include "sched.h"
#include "kernel.h"
#include "sig.h"
#include "sem.h"
#include "mutex.h"
#include "ipc.h"
#include "timer.h"
#include "syscall.h"
```

### Macros

- #define SPIN_INIT(arg)

  *Wrapper macro.*
- #define SPIN_LOCK(arg)

  *Wrapper macro.*
- #define SPIN_FREE(arg)

  *Wrapper macro.*
- #define RESCHED_PROC(proc) resched()

  *Wrapper macro.*

### Typedefs

- typedef void(∗ code_t )(void ∗)

  *Executable code.*

### Functions

- void resched (void)

  *Rescheduling.*
- void disable_interrupts (void)

  *Interrupt disable.*
- void enable_interrupts (void)

  *Interrupt enable.*
- proc_t ∗ current_proc (void)

  *Current process.*
- stack_t ∗ proc_stack_init (stack_t ∗sstart, code_t pmain, void ∗arg, void(∗return_address)(void))

  *A process stack initialization.*
- void init_bugurt (void)

  *The Kernel initiation.*
- void start_bugurt (void)

  *The OSstart.*
- void syscall_bugurt (syscall_t num, void ∗arg)

  *A system call.*

#### 5.1.1 Detailed Description

The top header file. All other BuguRTOS headers are included here. On the other hand all BuguRTOSsource files include this file.

#### 5.1.2 Macro Definition Documentation

##### 5.1.2.1 #define SPIN_INIT( *arg* )

Wrapper macro.

Initialization wrapper for arg->lock spinlock. Emty macro in single core system.

##### 5.1.2.2 #define SPIN_LOCK( *arg* )

Wrapper macro.

Lock wrapper for arg->lock spinlock. Emty macro in single core system.

##### 5.1.2.3 #define SPIN_FREE( *arg* )

Wrapper macro.

Lock wrapper for arg->lock spinlock. Emty macro in single core system.

##### 5.1.2.4 #define RESCHED_PROC( *proc* ) resched()

Wrapper macro.

A wrapper for resched function.

### 5.1.3 Typedef Documentation

#### 5.1.3.1 typedef void(∗ code_t)(void ∗)

Executable code.

A pointer to a void function, that takes void pointer as argument.

### 5.1.4 Function Documentation

#### 5.1.4.1 void resched ( void )

Rescheduling.

Launces a reschedule sequence.

#### 5.1.4.2 void disable_interrupts ( void )

Interrupt disable.

Disables interrupts globally.

#### 5.1.4.3 void enable_interrupts ( void )

Interrupt enable.

Enables interrupts globally.

#### 5.1.4.4 proc_t∗ current_proc ( void )

Current process.

Current process.

**Returns**

a pointer to a current process on a local processor core.

#### 5.1.4.5 stack_t∗ proc_stack_init ( stack_t ∗ *sstart,* code_t *pmain,* void ∗ *arg,* void(∗)(void) *return_address* )

A process stack initialization.

This function prepares a process stack for running a process. It treats a pocess stack in such a way that pmain(arg) is called when a process context is restored from a process stack.

**Parameters**

| | |
|---|---|
| *sstart* | a process stack bottom. |
| *pmain* | a poiter to a function to call. |
| *arg* | an argument to a function to call. |
| *return_address* | an adress to return from pmain. |

**Returns**

a pointer to a prepared process stack top.

#### 5.1.4.6 void init_bugurt ( void )

The Kernel initiation.

Initiates the Kernel before the OSstart.

#### 5.1.4.7 void start_bugurt ( void )

The OSstart.

The OSstart. It is not necessary to write any code after call of this function, because such a code won't be run normally.

#### 5.1.4.8 void syscall_bugurt ( syscall_t *num*, void * *arg* )

A system call.

This function switches a processor core from a process context to the kernel context. The kernel code is allways run in the kernel context. This is done to save memory in process stacks. A system calls are done on every operations with processes, mutexes, semaphores and signals. The Kernel does all of this job.

**Parameters**

| | |
|---|---|
| *num* | a number of a system call (what is going to be done). |
| *arg* | a system call argument (a pointer to an object to be processed). |

## 5.2 include/crit_sec.h File Reference

A critical section header.

### Macros

- #define ENTER_CRIT_SEC() enter_crit_sec()

  *A wraper macro.*
- #define EXIT_CRIT_SEC() exit_crit_sec()

  *A wraper macro.*

### Functions

- void enter_crit_sec (void)
- void exit_crit_sec (void)

### 5.2.1 Detailed Description

A critical section header. A critical section is a part of a code where interrupts are disabled. Critical sections are used when a common data are used for a short time. Critical sections may be nested, in this case interrupts get enabled on exit from all critical sections.

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 #define ENTER_CRIT_SEC( ) enter_crit_sec()

A wraper macro.

A critical section start.

**Warning**

Must be used on a start of a code block!
All local variables must be declared before ENTER_CRIT_SEC, and all executable code must be below it.

#### 5.2.2.2 #define EXIT_CRIT_SEC( ) exit_crit_sec()

A wraper macro.

A critical section end.

**Warning**

Must be used at the end of a code block.

### 5.2.3 Function Documentation

#### 5.2.3.1 void enter_crit_sec ( void )

A critical section start.

#### 5.2.3.2 void exit_crit_sec ( void )

A critical section end.

## 5.3 include/gitem.h File Reference

A grouped list and item header. Data types: group_t, pool_t, gitem_t, gxlist_t.

**Data Structures**

- struct _group_t
  *A group of gitem_t objects.*
- struct _pool_t
  *Pool of groups;.*
- struct _gitem_t
  *A grouped list item.*
- struct _gxlist_t

**Macros**

- #define INIT_POOL_T() { (group_t *)0, (group_t *)0 }
- #define INIT_GROUP_T(p) { (void *)0, (prio_t)p, (count_t)1 }
- #define INIT_G_ITEM_T(a, p) { INIT_ITEM_T(a), &a.grp, INIT_GROUP_T(p) }

**Typedefs**

- typedef struct _group_t group_t
- typedef struct _pool_t pool_t
- typedef struct _gitem_t gitem_t
- typedef struct _gxlist_t gxlist_t

**Functions**

- void pool_init (pool_t *pool)
- void pool_merge (pool_t *src, pool_t *dst)
- void group_init (group_t *group, prio_t prio)
  *A group_t object iniation.*
- void group_push (group_t *group, pool_t *pool)

*Pushes a group to a pool.*

- group_t * group_pop (pool_t *pool)

  *Pop a group from a pool.*

- void gitem_init (gitem_t *gitem, prio_t prio)

  *A gitem_t object initiation.*

- void gitem_insert (gitem_t *gitem, xlist_t *xlist)

  *Inserts gitem_t object to an xlist_t container.*

- void gitem_insert_group (gitem_t *gitem, gxlist_t *gxlist)

  *Inserts gitem_t object to a gxlist_t container.*

- void gitem_fast_cut (gitem_t *gitem)

  *Cuts a gitem_t object from its container.*

- void gitem_cut (gitem_t *gitem)

  *Cuts a gitem_t object from its container.*

- void gxlist_init (gxlist_t *gxlist)

  *A gxlist_t object initiation.*

- void gxlist_merge (gxlist_t *source, gxlist_t *destination)

  *Transfers all gitem_t obcets from one gxlist_t container to another.*

### 5.3.1 Detailed Description

A grouped list and item header. Data types: group_t, pool_t, gitem_t, gxlist_t.

### 5.3.2 Macro Definition Documentation

#### 5.3.2.1 #define INIT_POOL_T( ) { (group_t *)0, (group_t *)0 }

Static pool_t object initiation.

#### 5.3.2.2 #define INIT_GROUP_T( *p* ) { (void *)0, (prio_t)p, (count_t)1 }

Static group_t object initiation.

**Parameters**

| | |
|---|---|
| *p* | Priority. |

#### 5.3.2.3 #define INIT_G_ITEM_T( *a*, *p* ) { INIT_ITEM_T(a), &a.grp, INIT_GROUP_T(p) }

Static gitem_t object initiation.

**Parameters**

| | |
|---|---|
| *a* | - A gitem_t object name. |
| *p* | - A priority. |

### 5.3.3 Typedef Documentation

#### 5.3.3.1 typedef struct _group_t group_t

See _group_t;

#### 5.3.3.2 typedef struct _pool_t pool_t

See _pool_t;

#### 5.3.3.3 typedef struct _gitem_t gitem_t

Grouped priority list item

#### 5.3.3.4 typedef struct _gxlist_t gxlist_t

A grouped xlist (xlist_t extension).

### 5.3.4 Function Documentation

#### 5.3.4.1 void pool_init ( pool_t * pool )

A pool_t object initiation;

**Parameters**

| | |
|---|---|
| *pool* | A pool_t pointer. |

#### 5.3.4.2 void pool_merge ( pool_t * src, pool_t * dst )

Merges two pools.

**Parameters**

| | |
|---|---|
| *src* | Source pool_t object pointer. |
| *dst* | Destignation pool_t object pointer. |

#### 5.3.4.3 void group_init ( group_t * group, prio_t prio )

A group_t object iniation.

**Parameters**

| | |
|---|---|
| *group* | A group_t object pointer. |
| *prio* | Priority. |

#### 5.3.4.4 void group_push ( group_t * group, pool_t * pool )

Pushes a group to a pool.

**Parameters**

| | |
|---|---|
| *group* | A group_t object pointer. |
| *pool* | A pool_t container pointer. |

#### 5.3.4.5 group_t* group_pop ( pool_t * pool )

Pop a group from a pool.

**Parameters**

| | |
|---|---|
| *pool* | A pool_t container pointer. |

**Returns**

A group_t object pointer.

#### 5.3.4.6 void gitem_init ( gitem_t * gitem, prio_t prio )

A gitem_t object initiation.

**Parameters**

| | |
|---|---|
| *gitem* | A gitem_t pointer. |
| *prio* | A priority. |

#### 5.3.4.7 void gitem_insert ( gitem_t * gitem, xlist_t * xlist )

Inserts gitem_t object to an xlist_t container.

**Parameters**

| | |
|---|---|
| *gitem* | A gitem_t pointer. |
| *xlist* | An xlist_t container pointer. |

#### 5.3.4.8 void gitem_insert_group ( gitem_t * gitem, gxlist_t * gxlist )

Inserts gitem_t object to a gxlist_t container.

A gitem_t object is inserted to a list with prio = gitem->group->prio, object is also transfered to gxlist->parent.-item[prio]->group, its original gitem->group is pushed to local pool (see pool_t).

**Parameters**

| | |
|---|---|
| *gitem* | A gitem_t pointer. |
| *gxlist* | A gxlist_t container pointer. |

#### 5.3.4.9 void gitem_fast_cut ( gitem_t * gitem )

Cuts a gitem_t object from its container.

Cuts a gitem_t object, from its cintainer. If gitem_t object was in gxlist_t container, then it is cut from its group, new group for this object is poped from local pool. A gitem->group->link pointer is not cleared.

**Parameters**

| | |
|---|---|
| *gitem* | A gitem_t object pointer. |

#### 5.3.4.10 void gitem_cut ( gitem_t * gitem )

Cuts a gitem_t object from its container.

Calls gitem_fast_cut and then clears gitem->group->link pointer.

**Parameters**

| | |
|---|---|
| *gitem* | A gitem_t pointer. |

#### 5.3.4.11 void gxlist_init ( gxlist_t * gxlist )

A gxlist_t object initiation.

**Parameters**

| | |
|---|---|
| *gxlist* | A gxlist_t pointer. |

#### 5.3.4.12 void gxlist_merge ( gxlist_t * *source*, gxlist_t * *destignation* )

Transfers all gitem_t obcets from one gxlist_t container to another.

Has O(1) complexity.

**Parameters**

| | |
|---|---|
| *source* | Source container pointer. |
| *destignation* | Destignation container pointer. |

## 5.4 include/index.h File Reference

An index search header.

### Functions

- prio_t index_search (index_t index)

### 5.4.1 Detailed Description

An index search header.

### 5.4.2 Function Documentation

#### 5.4.2.1 prio_t index_search ( index_t *index* )

An index search.

**Parameters**

| | |
|---|---|
| *index* | An index. |

**Returns**

Highest priority of an index (with minimal value).

## 5.5 include/ipc.h File Reference

An IPC header.

### Functions

- void _ipc_wait (void *ipc_pointer)
  *Wait for IPC kernel part.*
- ipc_data_t ipc_wait (void)
  *Wait for IPC.*
- bool_t ipc_send (proc_t *proc, ipc_data_t ipc_data)
  *IPCdata transmition.*
- bool_t ipc_send_isr (proc_t *proc, ipc_data_t ipc_data)
  *IPCdata transmition for ISR usage.*
- bool_t _ipc_exchange (proc_t *proc, ipc_data_t send, ipc_data_t *receive)
  *An IPC data transmition with wait for answer via IPC kernel part.*
- bool_t ipc_exchange (proc_t *proc, ipc_data_t send, ipc_data_t *receive)
  *An IPC data transmition with wait for answer via IPC kernel part.*

#### 5.5.1 Detailed Description

An IPC header.

#### 5.5.2 Function Documentation

##### 5.5.2.1 void _ipc_wait ( void * *ipc_pointer* )

Wait for IPC kernel part.

**Warning**

For internal usage only!!!

**Parameters**

| | |
|---|---|
| *ipc_pointer* | A pointer to IPCdata storage. |

##### 5.5.2.2 ipc_data_t ipc_wait ( void )

Wait for IPC.

**Returns**

IPC data.

##### 5.5.2.3 bool_t ipc_send ( proc_t * *proc,* ipc_data_t *ipc_data* )

IPCdata transmition.

This function checks a destignation process state. If destignation process is waiting for IPC, then data gets transmited and destignation process gets launched.

**Parameters**

| | |
|---|---|
| *proc* | A ddestignation process pointer. |
| *ipc_data* | A data to transmit. |

**Returns**

1 - if data has been transmited, else 0.

##### 5.5.2.4 bool_t ipc_send_isr ( proc_t * *proc,* ipc_data_t *ipc_data* )

IPCdata transmition for ISR usage.

**Warning**

Use in interrupt service routines.

This function checks a destignation process state. If destignation process is waiting for IPC, then data gets transmited and destignation process gets launched.

**Parameters**

| | |
|---|---|
| *proc* | A ddestignation process pointer. |
| *ipc_data* | A data to transmit. |

#### Returns

1 - if data has been transmited, else 0.

#### 5.5.2.5 bool_t _ipc_exchange ( proc_t * *proc,* ipc_data_t *send,* ipc_data_t * *receive* )

An IPC data transmition with wait for answer via IPC kernel part.

#### Warning

For internal usage only!

This function checks a destignation process state. If destignation process is waiting for IPC, then data gets transmited and destignation process gets launched. If transmition has been sucessful then caller process wats for answer via IPC.

#### Parameters

| | |
|---|---|
| *proc* | A ddestignation process pointer. |
| *send* | A data to transmit. |
| *receive* | A pointer to receivedata storage. |

#### Returns

1 - if data has been transmited, else 0.

#### 5.5.2.6 bool_t ipc_exchange ( proc_t * *proc,* ipc_data_t *send,* ipc_data_t * *receive* )

An IPC data transmition with wait for answer via IPC kernel part.

This function checks a destignation process state. If destignation process is waiting for IPC, then data gets transmited and destignation process gets launched. If transmition has been sucessful then caller process wats for answer via IPC.

#### Parameters

| | |
|---|---|
| *proc* | A ddestignation process pointer. |
| *send* | A data to transmit. |
| *receive* | A pointer to receivedata storage. |

#### Returns

1 - if data has been transmited, else 0.

## 5.6 include/item.h File Reference

A list item header.

#### Data Structures

- struct _item_t

  *A list item.*

#### Macros

- #define INIT_ITEM_T(a) { (item_t *)&a, (item_t *)&a }

### Typedefs

- typedef struct _item_t item_t

### Functions

- void item_init (item_t *item)

  *An item_t object initiation.*
- void item_insert (item_t *item, item_t *head)

  *Insert an item to a list.*
- void item_cut (item_t *item)

  *Cut an item from a list.*

#### 5.6.1 Detailed Description

A list item header.

#### 5.6.2 Macro Definition Documentation

##### 5.6.2.1 #define INIT_ITEM_T( *a* ) { (item_t *)&a, (item_t *)&a }

Static item initiation.

**Parameters**

| | |
|---|---|
| *a* | An item_t variable name. |

#### 5.6.3 Typedef Documentation

##### 5.6.3.1 typedef struct _item_t item_t

See _item_t;

#### 5.6.4 Function Documentation

##### 5.6.4.1 void item_init ( item_t * *item* )

An item_t object initiation.

**Parameters**

| | |
|---|---|
| *item* | An item_t pointer. |

##### 5.6.4.2 void item_insert ( item_t * *item*, item_t * *head* )

Insert an item to a list.

**Parameters**

| | |
|---|---|
| *item* | A pointer to an item. |
| *head* | A pointer to a destignation list head. |

##### 5.6.4.3 void item_cut ( item_t * *item* )

Cut an item from a list.

**Parameters**

| | |
|---|---|
| *item* | A pointer to an item to cut. |

## 5.7 include/kernel.h File Reference

A kernel header.

### Data Structures

- struct _kernel_t

  *A BuguRTOS kernel structure.*

### Typedefs

- typedef struct _kernel_t kernel_t

### Functions

- void kernel_init (void)

  *The kernel initiation.*
- void idle_main (void *arg)

  *An IDLE process main function.*

### Variables

- kernel_t kernel

  *The BuguRTOSkernel.*

#### 5.7.1 Detailed Description

A kernel header.

#### 5.7.2 Typedef Documentation

##### 5.7.2.1 typedef struct _kernel_t kernel_t

See _kernel_t;

#### 5.7.3 Function Documentation

##### 5.7.3.1 void kernel_init ( void )

The kernel initiation.

This function prepares the kernel to work.

##### 5.7.3.2 void idle_main ( void * *arg* )

An IDLE process main function.

You can use builtin function, or you can write your own. IDLEprocess can work with timers, fire signals and FREE semaphores, SEND IPC data!

#### Warning

An idle_main sholud NOT return, lock mutexes or semaphores, wait for IPC or signals!!!

#### Parameters

| | |
|---|---|
| *arg* | An argument pointer. |

### 5.7.4 Variable Documentation

#### 5.7.4.1 kernel_t kernel

The BuguRTOSkernel.

It's the one for the entire system!

## 5.8 include/mutex.h File Reference

A mutex header.

#### Data Structures

- struct _mutex_t

  *A mutex.*

#### Macros

- #define MUTEX_PRIO(m) ((((xlist_t *)m)->index) ? index_search(((xlist_t *)m)->index) : PROC_PRIO_LOWEST)

#### Typedefs

- typedef struct _mutex_t mutex_t

#### Functions

- void mutex_init_isr (mutex_t *mutex)

  *A mutex initiation for usage in ISRs or in critical sections.*
- void mutex_init (mutex_t *mutex)

  *A mutex initiation.*
- bool_t mutex_lock (mutex_t *mutex)

  *Lock a mutex.*
- bool_t mutex_try_lock (mutex_t *mutex)

  *Try to lock a mutex.*
- void mutex_free (mutex_t *mutex)

  *Mutex free.*
- bool_t _mutex_lock (mutex_t *mutex)

  *Lock a mutex kernel part.*
- bool_t _mutex_try_lock (mutex_t *mutex)

  *Try to lock a mutex kernel part.*
- bool_t _mutex_free (mutex_t *mutex)

  *Mutex free kernel part.*

#### 5.8.1 Detailed Description

A mutex header.

#### 5.8.2 Macro Definition Documentation

##### 5.8.2.1 #define MUTEX_PRIO( *m* ) ((((xlist_t *)m)->index) ? index_search(((xlist_t *)m)->index) : PROC_PRIO_LOWEST)

Calculates a mutex priority

#### 5.8.3 Typedef Documentation

##### 5.8.3.1 typedef struct _mutex_t mutex_t

See _mutex_t;

#### 5.8.4 Function Documentation

##### 5.8.4.1 void mutex_init_isr ( mutex_t * *mutex* )

A mutex initiation for usage in ISRs or in critical sections.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

##### 5.8.4.2 void mutex_init ( mutex_t * *mutex* )

A mutex initiation.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

##### 5.8.4.3 bool_t mutex_lock ( mutex_t * *mutex* )

Lock a mutex.

If a mutex is free then caller process locks it and continues, else caller process stops and waits until mutex gets freeed.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

**Returns**

1 if mutex was locked without wait, else 0.

##### 5.8.4.4 bool_t mutex_try_lock ( mutex_t * *mutex* )

Try to lock a mutex.

If mutex is free then caller process locks it and continues, if not caller process continues without wait.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

**Returns**

1 - if mutex was succefully locked else - 0.

#### 5.8.4.5 void mutex_free ( mutex_t * mutex )

Mutex free.

If a mutex wait list is empty, then caller process frees a mutex, else mutex wait lish head gets launched.

**Parameters**

| | |
|---|---|
| *mutex* | . |

#### 5.8.4.6 bool_t _mutex_lock ( mutex_t * mutex )

Lock a mutex kernel part.

If a mutex is free then caller process locks it and continues, else caller process stops and waits until mutex gets freeed.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

**Returns**

1 if mutex was locked without wait, else 0.

#### 5.8.4.7 bool_t _mutex_try_lock ( mutex_t * mutex )

Try to lock a mutex kernel part.

If mutex is free then caller process locks it and continues, if not caller process continues without wait.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

**Returns**

1 - if mutex was succefully locked else - 0.

#### 5.8.4.8 bool_t _mutex_free ( mutex_t * mutex )

Mutex free kernel part.

If a mutex wait list is empty, then caller process frees a mutex, else mutex wait lish head gets launched.

**Parameters**

| | |
|---|---|
| *mutex* | A mutex pointer. |

## 5.9 include/pcounter.h File Reference

A locked resource counter header.

### Data Structures

- struct _pcounter_t

  *A locked resource counter.*

### Typedefs

- typedef struct _pcounter_t pcounter_t

### Functions

- void pcounter_init (pcounter_t *pcounter)

  *A pcounter_t object initiation.*
- void pcounter_inc (pcounter_t *pcounter, prio_t prio)

  *Increment counter.*
- index_t pcounter_dec (pcounter_t *pcounter, prio_t prio)

  *Decrement counter.*
- void pcounter_plus (pcounter_t *pcounter, prio_t prio, count_t count)

  *Increase counter by a number of steps.*
- index_t pcounter_minus (pcounter_t *pcounter, prio_t prio, count_t count)

  *Decrease counter by a number of steps;.*

#### 5.9.1 Detailed Description

A locked resource counter header.

#### 5.9.2 Typedef Documentation

##### 5.9.2.1 typedef struct _pcounter_t pcounter_t

See _pcounter_t;

#### 5.9.3 Function Documentation

##### 5.9.3.1 void pcounter_init ( pcounter_t * *pcounter* )

A pcounter_t object initiation.

**Parameters**

| | |
|---|---|
| *pcounter* | A pcounter_t pointer. |

##### 5.9.3.2 void pcounter_inc ( pcounter_t * *pcounter,* prio_t *prio* )

Increment counter.

**Parameters**

| | |
|---|---|
| *pcounter* | A pcounter_t pointer. |
| *prio* | A priority. |

#### 5.9.3.3 index_t pcounter_dec ( pcounter_t * *pcounter*, prio_t *prio* )

Decrement counter.

**Parameters**

| | |
|---|---|
| *pcounter* | A pcounter_t pointer. |
| *prio* | A priority. |

#### 5.9.3.4 void pcounter_plus ( pcounter_t * *pcounter*, prio_t *prio*, count_t *count* )

Increase counter by a number of steps.

**Parameters**

| | |
|---|---|
| *pcounter* | A pcounter_t pointer. |
| *prio* | A priority. |
| *count* | A number of increment steps. |

#### 5.9.3.5 index_t pcounter_minus ( pcounter_t * *pcounter*, prio_t *prio*, count_t *count* )

Decrease counter by a number of steps;.

**Parameters**

| | |
|---|---|
| *pcounter* | A pcounter_t pointer. |
| *prio* | A priority. |
| *count* | A number of decrement steps. |

**Returns**

0 if correspondent counter is nulled, not 0 else.

## 5.10 include/proc.h File Reference

A process header.

**Data Structures**

- struct _proc_t

  *A process.*

**Macros**

- #define PROC_LRES_INIT(a) pcounter_init(&a->lres)

  *Wrapper macro.*
- #define PROC_LRES_INC(a, b) pcounter_inc( &a->lres, b )

  *Wrapper macro.*
- #define PROC_LRES_DEC(a, b) pcounter_dec( &a->lres, b )

  *Wrapper macro.*

- #define PROC_FLG_RT ((flag_t)0x80)

  *A real time flag.*
- #define PROC_FLG_MUTEX ((flag_t)0x40)

  *A mutex lock flag.*
- #define PROC_FLG_SEM ((flag_t)0x20)

  *A semaphore lock flag.*
- #define PROC_FLG_PRE_STOP ((flag_t)0x10)

  *A proces stop preparation flag.*
- #define PROC_FLG_LOCK_MASK ((flag_t)(PROC_FLG_MUTEX|PROC_FLG_SEM))

  *A PROC_FLG_MUTEX or PROC_FLG_SEM mask.*
- #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

  *An execution state clear mask.*
- #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)

  *An execution state clear mask.*
- #define PROC_STATE_MASK ((flag_t)0x0F)

  *An execution state mask.*
- #define PROC_STATE_RESTART_MASK ((flag_t)0xC)

  *A process execution state check mask.*
- #define PROC_STATE_RUN_MASK ((flag_t)0x7)

  *A process execution state check mask.*
- #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

  *A process execution state check mask.*
- #define PROC_STATE_STOPED ((flag_t)0x0)
- #define PROC_STATE_END ((flag_t)0x1)
- #define PROC_STATE_W_WD_STOPED ((flag_t)0x2)
- #define PROC_STATE_WD_STOPED ((flag_t)0x3)
- #define PROC_STATE_DEAD ((flag_t)0x4)
- #define PROC_STATE_PCHANGE ((flag_t)0x5)
- #define PROC_STATE_READY ((flag_t)0x6)
- #define PROC_STATE_RUNNING ((flag_t)0x7)
- #define PROC_STATE_W_MUT ((flag_t)0x8)
- #define PROC_STATE_W_SEM ((flag_t)0x9)
- #define PROC_STATE_W_SIG ((flag_t)0xA)
- #define PROC_STATE_W_IPC ((flag_t)0xB)
- #define PROC_STATE_W_DEAD ((flag_t)0xC)
- #define PROC_STATE_W_PCHANGE ((flag_t)0xD)
- #define PROC_STATE_W_READY ((flag_t)0xE)
- #define PROC_STATE_W_RUNNING ((flag_t)0xF)
- #define PROC_PRE_STOP_TEST(a) ( ( a->flags & PROC_FLG_PRE_STOP ) && ( !( a->flags & PROC_FLG_LOCK_MASK ) ) )

  *A PROC_FLG_PRE_STOP condition test macro.*
- #define PROC_RUN_TEST(a) ( ( a->flags & PROC_STATE_RUN_MASK ) >= PROC_STATE_READY )

  *Check if process is ready or running.*
- #define PROC_GET_STATE(a) ( a->flags & PROC_STATE_MASK )

  *Reads a process state.*
- #define PROC_SET_STATE(a, b) ( a->flags &= PROC_STATE_CLEAR_MASK, proc->flags |= b )

  *Sets process state.*
- #define PROC_IPC_TEST(a) ( PROC_GET_STATE(a) == PROC_STATE_W_IPC )

  *Checks if process is waiting for IPC.*
- #define PROC_PRIO_LOWEST ((prio_t)BITS_IN_INDEX_T - (prio_t)1)

  *Lowest priority level.*

### Typedefs

- typedef struct _proc_t proc_t

### Functions

- void proc_init_isr (proc_t *proc, code_t pmain, code_t sv_hook, code_t rs_hook, void *arg, stack_t *sstart, prio_t prio, timer_t time_quant, bool_t is_rt)

  *A process initialization. Must be used in critical sections and interrupt service routines.*
- void proc_init (proc_t *proc, code_t pmain, code_t sv_hook, code_t rs_hook, void *arg, stack_t *sstart, prio_t prio, timer_t time_quant, bool_t is_rt)

  *A process initialization.*
- void proc_run_wrapper (proc_t *proc)

  *A wrapper for process "main" routines.*
- void proc_terminate (void)

  *A process termination routine called after proc->pmain return. Internal usage function.*
- void _proc_terminate (void)

  *A process termination routine called after proc->pmain return. Internal usage function.*
- bool_t proc_run (proc_t *proc)

  *A process launch routine.*
- bool_t proc_run_isr (proc_t *proc)

  *A process launch routine for usage in interrupt service routines and critical sections.*
- bool_t proc_restart (proc_t *proc)

  *Aprocess restart routine.*
- bool_t proc_restart_isr (proc_t *proc)

  *Aprocess restart routine for usage in interrupt service routines and critical sections.*
- bool_t proc_stop (proc_t *proc)

  *A process stop routine.*
- bool_t proc_stop_isr (proc_t *proc)

  *A process stop routine for usage in interrupts service routines and critical sections.*
- void proc_self_stop (void)

  *A process self stop routine.*
- void _proc_self_stop (void)

  *A process self stop routine (for internal usage only!).*
- void proc_reset_watchdog (void)

  *A watchdog reset routine for real time processes.*
- void _proc_reset_watchdog (void)

  *A watchdog reset routine for real time processes for internal usage.*
- void _proc_dont_stop (proc_t *proc, flag_t flags)

  *Run stoped process and set PROC_FLG_PRE_STOP. For internal usage.*
- void _proc_cut_and_run (proc_t *proc, flag_t state)

  *Cut the a process from wait list and run it. For internel usage.*
- void _proc_prio_propagate (proc_t *proc)

  *Propagation of priority through a blovked process chain. For internal usage.*
- void _proc_stop_flags_set (proc_t *proc, flag_t mask)

  *A low level process stop with flags set routine. For internal usage.*
- void _proc_flag_stop (flag_t mask)

  *A::PROC_FLG_PRE_STOP flag processing routine. For internal usage.*
- void proc_flag_stop (flag_t mask)

  *A::PROC_FLG_PRE_STOP flag processing routine.*
- void _proc_prio_control_stoped (proc_t *proc)

*A stopedprocess priority control routine.*

- void proc_set_prio (proc_t *proc, prio_t prio)

  *Set a priotity of a process.*

- void _proc_set_prio (proc_t *proc, prio_t prio)

  *Set a priotity of a process. For internel usage.*

### 5.10.1 Detailed Description

A process header.

### 5.10.2 Macro Definition Documentation

#### 5.10.2.1 #define PROC_LRES_INIT( *a* ) pcounter_init(&a->lres)

Wrapper macro.

Initiates proc->lres field of a process.

**Parameters**

| | |
|---|---|
| *a* | a pointer to a process. |

#### 5.10.2.2 #define PROC_LRES_INC( *a, b* ) pcounter_inc( &a->lres, b )

Wrapper macro.

An increment of locked mutex counter field of a process.

**Parameters**

| | |
|---|---|
| *a* | a pointer to a process. |
| *b* | a priority of a locked mutex for highest locker protocol. |

#### 5.10.2.3 #define PROC_LRES_DEC( *a, b* ) pcounter_dec( &a->lres, b )

Wrapper macro.

A decrement of locked mutex counter field of a process.

**Parameters**

| | |
|---|---|
| *a* | a pointer to a process. |
| *b* | a priority of a locked mutex for highest locker protocol. |

#### 5.10.2.4 #define PROC_FLG_RT ((flag_t)0x80)

A real time flag.

This flag enables real time process scheduling policy.

#### 5.10.2.5 #define PROC_FLG_MUTEX ((flag_t)0x40)

A mutex lock flag.

A process has locked some mutex controled resources.

#### 5.10.2.6 #define PROC_FLG_SEM ((flag_t)0x20)

A semaphore lock flag.

It is set on sem_lock call or on succesfull sem_try_lock call. It is necessary to clear this flag manually, when semaphore controled resource is released. Use proc_flag_stop call to clear this flag.

#### 5.10.2.7 #define PROC_FLG_PRE_STOP ((flag_t)0x10)

A proces stop preparation flag.

A process must be stoped, but it can't be stoped now. It'll be stoped when possible.

#### 5.10.2.8 #define PROC_FLG_LOCK_MASK ((flag_t)(PROC_FLG_MUTEX|PROC_FLG_SEM))

A PROC_FLG_MUTEX or PROC_FLG_SEM mask.

Used to test if a process has locked some resources.

#### 5.10.2.9 #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

An execution state clear mask.

Used clear execution state bitts in proc->flags.

#### 5.10.2.10 #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)

An execution state clear mask.

Used clear execution three LSBs state bitts in proc->flags.

#### 5.10.2.11 #define PROC_STATE_RESTART_MASK ((flag_t)0xC)

A process execution state check mask.

Used by proc_restart and proc_restart_isr to check for restart posibility.

#### 5.10.2.12 #define PROC_STATE_RUN_MASK ((flag_t)0x7)

A process execution state check mask.

Used to check if the process has been run.

#### 5.10.2.13 #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

A process execution state check mask.

Used to check if the process is waiting for semaphore, mutex, ipc or signal.

#### 5.10.2.14 #define PROC_STATE_STOPED ((flag_t)0x0)

Initial state, stoped.

#### 5.10.2.15 #define PROC_STATE_END ((flag_t)0x1)

Normal process termination.

#### 5.10.2.16 #define PROC_STATE_W_WD_STOPED ((flag_t)0x2)

Watchdog termination from W_RUNNING state.

#### 5.10.2.17 #define PROC_STATE_WD_STOPED ((flag_t)0x3)

Watchdog termination.

#### 5.10.2.18 #define PROC_STATE_DEAD ((flag_t)0x4)

Abnormal termination, terminated with resources locked.

#### 5.10.2.19 #define PROC_STATE_PCHANGE ((flag_t)0x5)

A process has been run during priority change

#### 5.10.2.20 #define PROC_STATE_READY ((flag_t)0x6)

Is ready to run.

#### 5.10.2.21 #define PROC_STATE_RUNNING ((flag_t)0x7)

Is running.

#### 5.10.2.22 #define PROC_STATE_W_MUT ((flag_t)0x8)

Is waiting for mutex.

#### 5.10.2.23 #define PROC_STATE_W_SEM ((flag_t)0x9)

Is waiting for semaphore.

#### 5.10.2.24 #define PROC_STATE_W_SIG ((flag_t)0xA)

Is waiting for signal.

#### 5.10.2.25 #define PROC_STATE_W_IPC ((flag_t)0xB)

Is waiting for IPC.

#### 5.10.2.26 #define PROC_STATE_W_DEAD ((flag_t)0xC)

Watchdog termination from W_RUNNING state with resources locked.

#### 5.10.2.27 #define PROC_STATE_W_PCHANGE ((flag_t)0xD)

A process is stoped for priority change.

#### 5.10.2.28 #define PROC_STATE_W_READY ((flag_t)0xE)

Is ready to run (special).

#### 5.10.2.29 #define PROC_STATE_W_RUNNING ((flag_t)0xF)

Is running (special).

#### 5.10.2.30 #define PROC_PRE_STOP_TEST( *a* ) ( ( a->flags & PROC_FLG_PRE_STOP ) && ( !( a->flags & PROC_FLG_LOCK_MASK ) ) )

A PROC_FLG_PRE_STOP condition test macro.

Used to test if a process can be stoped on PROC_FLG_PRE_STOP flag. A process should not have locked resources at a moment of a flag stop.

### 5.10.3 Typedef Documentation

#### 5.10.3.1 typedef struct _proc_t proc_t

See _proc_t;

### 5.10.4 Function Documentation

#### 5.10.4.1 void proc_init_isr ( proc_t * *proc*, code_t *pmain*, code_t *sv_hook*, code_t *rs_hook*, void * *arg*, stack_t * *sstart*, prio_t *prio*, timer_t *time_quant*, bool_t *is_rt* )

A process initialization. Must be used in critical sections and interrupt service routines.

**Parameters**

| | |
|---|---|
| *proc* | A ponter to a initialized process. |
| *pmain* | A pointer to a process "main" routine. |
| *sv_hook* | A context save hook pointer. |
| *rs_hook* | A context save hook pointer. |
| *arg* | An argument pointer. |
| *sstart* | Aprocess stack bottom pointer. |
| *prio* | A process priority. |
| *time_quant* | A process time slice. |
| *is_rt* | A real time flag. If frue, then a process is scheduled in a real time manner. |

#### 5.10.4.2 void proc_init ( proc_t * *proc*, code_t *pmain*, code_t *sv_hook*, code_t *rs_hook*, void * *arg*, stack_t * *sstart*, prio_t *prio*, timer_t *time_quant*, bool_t *is_rt* )

A process initialization.

**Parameters**

| | |
|---|---|
| *proc* | A ponter to a initialized process. |
| *pmain* | A pointer to a process "main" routine. |
| *sv_hook* | A context save hook pointer. |
| *rs_hook* | A context save hook pointer. |
| *arg* | An argument pointer. |
| *sstart* | Aprocess stack bottom pointer. |
| *prio* | A process priority. |
| *time_quant* | A process time slice. |
| *is_rt* | A real time flag. If frue, then a process is scheduled in a real time manner. |

#### 5.10.4.3 void proc_run_wrapper ( proc_t * *proc* )

A wrapper for process "main" routines.

This function calls proc->pmain(proc->arg), and if pmain returns, then proc_run_wrapper terminates process correctly.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to launch. |

#### 5.10.4.4 bool_t proc_run ( proc_t * *proc* )

A process launch routine.

This function schedules a process if possible.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to launch. |

**Returns**

1 - if a process has been scheduled, 0 in other cases.

#### 5.10.4.5 bool_t proc_run_isr ( proc_t * proc )

A process launch routine for usage in interrupt service routines and critical sections.

This function schedules a process if possible.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to launch. |

**Returns**

1 - if a process has been scheduled, 0 in other cases.

#### 5.10.4.6 bool_t proc_restart ( proc_t * proc )

Aprocess restart routine.

This function reinitializes a process and schedules it if possible.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to launch. |

**Returns**

1 - if a process has been scheduled, 0 in other cases.

#### 5.10.4.7 bool_t proc_restart_isr ( proc_t * proc )

Aprocess restart routine for usage in interrupt service routines and critical sections.

This function reinitializes a process and schedules it if possible.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to launch. |

**Returns**

1 - if a process has been scheduled, 0 in other cases.

#### 5.10.4.8 bool_t proc_stop ( proc_t * proc )

A process stop routine.

This function stops a process if possible.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to stop. |

**Returns**

1 - if a process has been stoped, 0 in other cases.

#### 5.10.4.9 bool_t proc_stop_isr ( proc_t * proc )

A process stop routine for usage in interrupts service routines and critical sections.

This function stops a process if possible.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process to stop. |

**Returns**

1 - if a process has been stoped, 0 in other cases.

#### 5.10.4.10 void proc_self_stop ( void )

A process self stop routine.

This function stops caller process.

#### 5.10.4.11 void _proc_self_stop ( void )

A process self stop routine (for internal usage only!).

This function stops caller process.

#### 5.10.4.12 void proc_reset_watchdog ( void )

A watchdog reset routine for real time processes.

If a caller process is real time, then this function resets its timer. If a real time process failes to reset its watchdog, then the scheduler stops such process and wakes up next ready process.

#### 5.10.4.13 void _proc_reset_watchdog ( void )

A watchdog reset routine for real time processes for internal usage.

If a caller process is real time, then this function resets its timer. If a real time process failes to reset its watchdog, then the scheduler stops such process and wakes up next ready process.

#### 5.10.4.14 void _proc_prio_control_stoped ( proc_t * *proc* )

A stopedprocess priority control routine.

Used with CONFIG_USE_HIGHEST_LOCKER option. A process must be stoped before call of the routine.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process. |

#### 5.10.4.15 void proc_set_prio ( proc_t * *proc*, prio_t *prio* )

Set a priotity of a process.

It sets a procees priority. A process current state doesn't matter.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process. |
| *prio* | - New process priority value. |

#### 5.10.4.16 void _proc_set_prio ( proc_t * *proc*, prio_t *prio* )

Set a priotity of a process. For internel usage.

It sets a procees priority. A process current state doesn't matter.

**Parameters**

| | |
|---|---|
| *proc* | - A pointer to a process. |
| *prio* | - New process priority value. |

## 5.11 include/sched.h File Reference

Ascheduler header.

### Data Structures

- struct _sched_t

  *A scheduler.*

### Macros

- #define _SCHED_INIT() ((sched_t *)&kernel.sched)

  *Wrapper macro.*

### Typedefs

- typedef struct _sched_t sched_t

### Functions

- void sched_init (sched_t *sched, proc_t *idle)

  *A scheduler initiation routine.*
- void sched_schedule (void)

  *A scheduler routine.*
- void sched_reschedule (void)

  *Recheduler routine.*
- void sched_proc_run (proc_t *proc, flag_t state)

  *A low level process run routine. For internal usage.*
- void sched_proc_stop (proc_t *proc)

  *A low level process stop routine. For internal usage.*
- bool_t _sched_proc_yeld (void)

  *Pass control to next ready process (for internal usage only!).*
- bool_t sched_proc_yeld (void)

  *Pass control to next ready process.*

### 5.11.1 Detailed Description

Ascheduler header.

**Warning**

All functions in this file are internel usage functins!!!

#### 5.11.2 Macro Definition Documentation

##### 5.11.2.1 #define _SCHED_INIT( ) ((sched_t *)&kernel.sched)

Wrapper macro.

Initialization wrapper for sched variable in sched_schedule and sched_reschedule functions.

#### 5.11.3 Typedef Documentation

##### 5.11.3.1 typedef struct _sched_t sched_t

See _sched_t;

#### 5.11.4 Function Documentation

##### 5.11.4.1 void sched_init ( sched_t * *sched,* proc_t * *idle* )

A scheduler initiation routine.

This function prepares a scheduler object for work.

**Parameters**

| | |
|---|---|
| *sched* | - A sceduler pointer. |
| *idle* | - An IDLE process pointer. |

##### 5.11.4.2 void sched_schedule ( void )

A scheduler routine.

This function switches processes in system timer interrupt handler.

##### 5.11.4.3 void sched_reschedule ( void )

Recheduler routine.

This function switches processes if needed.

##### 5.11.4.4 bool_t _sched_proc_yeld ( void )

Pass control to next ready process (for internal usage only!).

If there is another running process, this function passes control to it.

**Returns**

One if power saving mode can be used, zero in other cases.

##### 5.11.4.5 bool_t sched_proc_yeld ( void )

Pass control to next ready process.

If there is another running process, this function passes control to it.

#### Returns

One if power saving mode can be used, zero in other cases.

## 5.12 include/sem.h File Reference

A counting semaphores header.

#### Data Structures

- struct _sem_t
  *A counting semaphore.*

#### Typedefs

- typedef struct _sem_t sem_t

#### Functions

- void sem_init_isr (sem_t *sem, count_t count)
  *Semaphore initiation from ISR.*
- void sem_init (sem_t *sem, count_t count)
  *Semaphore initiation.*
- bool_t sem_lock (sem_t *sem)
  *A semaphore lock.*
- bool_t sem_try_lock (sem_t *sem)
  *Try to lock a semaphore.*
- void sem_free (sem_t *sem)
  *Semaphore free.*
- void sem_free_isr (sem_t *sem)
  *Semaphore free for ISRusage.*
- bool_t _sem_lock (sem_t *sem)
  *A semaphore lock kernel part.*
- bool_t _sem_try_lock (sem_t *sem)
  *Try to lock a semaphore kernel part.*

### 5.12.1 Detailed Description

A counting semaphores header.

### 5.12.2 Typedef Documentation

#### 5.12.2.1 typedef struct _sem_t sem_t

See _sem_t;

### 5.12.3 Function Documentation

#### 5.12.3.1 void sem_init_isr ( sem_t * *sem*, count_t *count* )

Semaphore initiation from ISR.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |
| *count* | A counter start value. |

#### 5.12.3.2 void sem_init ( sem_t * *sem*, count_t *count* )

Semaphore initiation.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |
| *count* | A counter start value. |

#### 5.12.3.3 bool_t sem_lock ( sem_t * *sem* )

A semaphore lock.

If semaphore counter greater than zero, then it will be decreased and caller process will continue, else caller process will stop and wait until semaphore get free.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |

**Returns**

1 if semaphore was locked without wait, else 0.

#### 5.12.3.4 bool_t sem_try_lock ( sem_t * *sem* )

Try to lock a semaphore.

If semaphore counter greater than zero, then it will be decreased and caller process will continue, else caller process will just continue.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |

**Returns**

1 if semaphore was succefully locked, else 0.

#### 5.12.3.5 void sem_free ( sem_t * *sem* )

Semaphore free.

If semaphore wait lisk is empty, then counter will be encreased, else semaphore wait list head will be launched.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |

#### 5.12.3.6 void sem_free_isr ( sem_t * sem )

Semaphore free for ISRusage.

If semaphore wait lisk is empty, then counter will be encreased, else semaphore wait list head will be launched.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |

#### 5.12.3.7 bool_t _sem_lock ( sem_t * sem )

A semaphore lock kernel part.

If semaphore counter greater than zero, then it will be decreased and caller process will continue, else caller process will stop and wait until semaphore get free.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |

**Returns**

1 if semaphore was locked without wait, else 0.

#### 5.12.3.8 bool_t _sem_try_lock ( sem_t * sem )

Try to lock a semaphore kernel part.

If semaphore counter greater than zero, then it will be decreased and caller process will continue, else caller process will just continue.

**Parameters**

| | |
|---|---|
| *sem* | A sem_t pointer. |

**Returns**

1 if semaphore was succefully locked, else 0.

## 5.13 include/sig.h File Reference

A signal header.

**Data Structures**

- struct _sig_t
  *A signal.*

**Typedefs**

- typedef struct _sig_t sig_t

**Functions**

- void sig_init_isr (sig_t *sig)
  *A signal initiation from ISR or critical section.*

- void sig_init (sig_t *sig)
  *Signal initiation.*
- void sig_wait (sig_t *sig)
  *Wait for a singnal.*
- void _sig_wait_prologue (sig_t *sig)
  *A signal wait prologue kernel part.*
- void _sig_wait_epilogue (void)
  *Sig wait epilogue. For intrnal usage.*
- void sig_signal (sig_t *sig)
  *Fire a signal, launch one waiting process.*
- void sig_broadcast (sig_t *sig)
  *Fire a signal, launch all waiting processes.*
- void sig_signal_isr (sig_t *sig)
  *Fire a signal from ISR, launch one waiting process.*
- void sig_broadcast_isr (sig_t *sig)
  *Fire a signal from ISR, launch all waiting processes.*

### 5.13.1 Detailed Description

A signal header.

### 5.13.2 Typedef Documentation

#### 5.13.2.1 typedef struct _sig_t sig_t

See _sig_t;

### 5.13.3 Function Documentation

#### 5.13.3.1 void sig_init_isr ( sig_t * *sig* )

A signal initiation from ISR or critical section.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.2 void sig_init ( sig_t * *sig* )

Signal initiation.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.3 void sig_wait ( sig_t * *sig* )

Wait for a singnal.

This function stops caller process and inserts it to signal wait list. On multicore system signal has one wait list per CPU core, so load prebalancing is done. After firing a signal process will be lounched PROC_FLG_PRE_STOP processing will be done.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.4 void _sig_wait_prologue ( sig_t * *sig* )

A signal wait prologue kernel part.

This function stops cureent running process and insert it to signal wait list.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.5 void _sig_wait_epilogue ( void )

Sig wait epilogue. For intrnal usage.

Wakes up next proces from sig->wakeup, if needed.

#### 5.13.3.6 void sig_signal ( sig_t * *sig* )

Fire a signal, launch one waiting process.

On multicore system: This functin finds most loaded signal wait list (using signal statistic array) and launches its head on the least loaded CPU core. On one coresystem: This function launches signal wait list head.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.7 void sig_broadcast ( sig_t * *sig* )

Fire a signal, launch all waiting processes.

This function launches all processes waiting for certain signal. This function is O(1), as #pitem_xlist_chain is used.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.8 void sig_signal_isr ( sig_t * *sig* )

Fire a signal from ISR, launch one waiting process.

On multicore system: This functin finds most loaded signal wait list (using signal statistic array) and launches its head on the least loaded CPU core. On one coresystem: This function launches signal wait list head.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

#### 5.13.3.9 void sig_broadcast_isr ( sig_t * *sig* )

Fire a signal from ISR, launch all waiting processes.

This function launches all processes waiting for certain signal. This function is O(1), as gxlist_merge is used.

**Parameters**

| | |
|---|---|
| *sig* | A sig_t pointer. |

## 5.14 include/syscall.h File Reference

System call header.

### Macros

- #define SYSCALL_PROC_RUN ((syscall_t)(1))
- #define SYSCALL_PROC_RESTART (SYSCALL_PROC_RUN + (syscall_t)(1))
- #define SYSCALL_PROC_STOP (SYSCALL_PROC_RESTART + (syscall_t)(1))
- #define SYSCALL_PROC_SELF_STOP (SYSCALL_PROC_STOP + (syscall_t)(1))
- #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))
- #define SYSCALL_PROC_FLAG_STOP (SYSCALL_PROC_TERMINATE + (syscall_t)(1))
- #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FLAG_STOP + (syscall_t)(1))
- #define SYSCALL_PROC_SET_PRIO (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))
- #define SYSCALL_SCHED_PROC_YELD (SYSCALL_PROC_SET_PRIO + (syscall_t)(1))
- #define SYSCALL_SIG_WAIT (SYSCALL_SCHED_PROC_YELD + (syscall_t)(1))
- #define SYSCALL_SIG_WAKEUP (SYSCALL_SIG_WAIT + (syscall_t)(1))
- #define SYSCALL_SIG_SIGNAL (SYSCALL_SIG_WAKEUP + (syscall_t)(1))
- #define SYSCALL_SIG_BROADCAST (SYSCALL_SIG_SIGNAL + (syscall_t)(1))
- #define SYSCALL_SEM_LOCK (SYSCALL_SIG_BROADCAST + (syscall_t)(1))
- #define SYSCALL_SEM_TRY_LOCK (SYSCALL_SEM_LOCK + (syscall_t)(1))
- #define SYSCALL_SEM_FREE (SYSCALL_SEM_TRY_LOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_LOCK (SYSCALL_SEM_FREE + (syscall_t)(1))
- #define SYSCALL_MUTEX_TRY_LOCK (SYSCALL_MUTEX_LOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_FREE (SYSCALL_MUTEX_TRY_LOCK + (syscall_t)(1))
- #define SYSCALL_IPC_WAIT (SYSCALL_MUTEX_FREE + (syscall_t)(1))
- #define SYSCALL_IPC_SEND (SYSCALL_IPC_WAIT + (syscall_t)(1))
- #define SYSCALL_IPC_EXCHANGE (SYSCALL_IPC_SEND + (syscall_t)(1))
- #define SYSCALL_USER (SYSCALL_IPC_EXCHANGE + (syscall_t)(1))

### Functions

- void do_syscall (void)

  *System call processing routine.*
- void scall_proc_run (void *arg)

  *A SYSCALL_PROC_RUN handler.*
- void scall_proc_restart (void *arg)

  *A SYSCALL_PROC_RESTART handler.*
- void scall_proc_stop (void *arg)

  *A SYSCALL_PROC_STOP handler.*
- void scall_proc_self_stop (void *arg)

  *A SYSCALL_PROC_SELF_STOP handler.*
- void scall_sched_proc_yeld (void *arg)

  *A #SYSCALL_PROC_YELD handler.*
- void scall_proc_terminate (void *arg)

  *A SYSCALL_PROC_TERMINATE handler.*
- void scall_proc_flag_stop (void *arg)

  *A SYSCALL_PROC_FLAG_STOP handler.*
- void scall_proc_reset_watchdog (void *arg)

  *A SYSCALL_PROC_RESET_WATCHDOG handler.*
- void scall_proc_set_prio (void *arg)

  *A SYSCALL_PROC_SET_PRIO handler.*

- void scall_sig_wait (void *arg)
  *A SYSCALL_SIG_WAIT hadnler.*
- void scall_sig_wakeup (void *arg)
  *A SYSCALL_SIG_WAKEUP hadnler.*
- void scall_sig_signal (void *arg)
  *A SYSCALL_SIG_SIGNAL handler.*
- void scall_sig_broadcast (void *arg)
  *A SYSCALL_SIG_BROADCAST handler.*
- void scall_sem_lock (void *arg)
  *A SYSCALL_SEM_LOCK handler.*
- void scall_sem_try_lock (void *arg)
  *A SYSCALL_SEM_TRY_LOCK handler.*
- void scall_sem_free (void *arg)
  *A SYSCALL_SEM_FREE handler.*
- void scall_mutex_lock (void *arg)
  *A SYSCALL_MUTEX_LOCK handler.*
- void scall_mutex_try_lock (void *arg)
  *A SYSCALL_MUTEX_TRY_LOCK handler.*
- void scall_mutex_free (void *arg)
  *A SYSCALL_MUTEX_FREE handler.*
- void scall_ipc_wait (void *arg)
  *A SYSCALL_IPC_WAIT handler.*
- void scall_ipc_send (void *arg)
  *A SYSCALL_IPC_SEND handler.*
- void scall_ipc_exchange (void *arg)
  *A SYSCALL_IPC_EXCHANGE handler.*
- void scall_user (void *arg)
  *A SYSCALL_USER handler.*

#### Variables

- syscall_t syscall_num
  *System call processing routine.*
- void * syscall_arg

### 5.14.1 Detailed Description

System call header.

### 5.14.2 Macro Definition Documentation

#### 5.14.2.1 #define SYSCALL_PROC_RUN ((syscall_t)(1))

A process launch.

#### 5.14.2.2 #define SYSCALL_PROC_RESTART (SYSCALL_PROC_RUN + (syscall_t)(1))

A Process restart.

#### 5.14.2.3 #define SYSCALL_PROC_STOP (SYSCALL_PROC_RESTART + (syscall_t)(1))

A process stop.

#### 5.14.2.4 #define SYSCALL_PROC_SELF_STOP (SYSCALL_PROC_STOP + (syscall_t)(1))

A process self stop.

#### 5.14.2.5 #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))

A process termination.

#### 5.14.2.6 #define SYSCALL_PROC_FLAG_STOP (SYSCALL_PROC_TERMINATE + (syscall_t)(1))

PROC_FLG_PRE_STOP flag processing.

#### 5.14.2.7 #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FLAG_STOP + (syscall_t)(1))

A real time process watchdog reset.

#### 5.14.2.8 #define SYSCALL_PROC_SET_PRIO (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))

Set a process priority.

#### 5.14.2.9 #define SYSCALL_SCHED_PROC_YELD (SYSCALL_PROC_SET_PRIO + (syscall_t)(1))

Transfer control to another process.

#### 5.14.2.10 #define SYSCALL_SIG_WAIT (SYSCALL_SCHED_PROC_YELD + (syscall_t)(1))

Wait for signal.

#### 5.14.2.11 #define SYSCALL_SIG_WAKEUP (SYSCALL_SIG_WAIT + (syscall_t)(1))

Signal wakeup processing.

#### 5.14.2.12 #define SYSCALL_SIG_SIGNAL (SYSCALL_SIG_WAKEUP + (syscall_t)(1))

Signal to one waiting process.

#### 5.14.2.13 #define SYSCALL_SIG_BROADCAST (SYSCALL_SIG_SIGNAL + (syscall_t)(1))

Signal to all waiting processes.

#### 5.14.2.14 #define SYSCALL_SEM_LOCK (SYSCALL_SIG_BROADCAST + (syscall_t)(1))

Lock a semaphore.

#### 5.14.2.15 #define SYSCALL_SEM_TRY_LOCK (SYSCALL_SEM_LOCK + (syscall_t)(1))

Try yo lock a semaphore.

#### 5.14.2.16 #define SYSCALL_SEM_FREE (SYSCALL_SEM_TRY_LOCK + (syscall_t)(1))

Unlock a semaphore.

#### 5.14.2.17 #define SYSCALL_MUTEX_LOCK (SYSCALL_SEM_FREE + (syscall_t)(1))

Lock a mutex.

#### 5.14.2.18 #define SYSCALL_MUTEX_TRY_LOCK (SYSCALL_MUTEX_LOCK + (syscall_t)(1))

Try to lock a mutex.

#### 5.14.2.19 #define SYSCALL_MUTEX_FREE (SYSCALL_MUTEX_TRY_LOCK + (syscall_t)(1))

Unlock a mutex.

#### 5.14.2.20 #define SYSCALL_IPC_WAIT (SYSCALL_MUTEX_FREE + (syscall_t)(1))

Wait for data (IPC).

#### 5.14.2.21 #define SYSCALL_IPC_SEND (SYSCALL_IPC_WAIT + (syscall_t)(1))

Send data via IPC.

#### 5.14.2.22 #define SYSCALL_IPC_EXCHANGE (SYSCALL_IPC_SEND + (syscall_t)(1))

Exchange data via IPC.

#### 5.14.2.23 #define SYSCALL_USER (SYSCALL_IPC_EXCHANGE + (syscall_t)(1))

A user syscall.

### 5.14.3 Function Documentation

#### 5.14.3.1 void do_syscall ( void )

System call processing routine.

This function calls system call handlers and passes arguments to them.

#### 5.14.3.2 void scall_proc_run ( void * *arg* )

A SYSCALL_PROC_RUN handler.

This function tries to launch a process by proc_run_isr call.

**Parameters**

| | |
|---|---|
| *arg* | A proc_runtime_arg_t pointer. |

#### 5.14.3.3 void scall_proc_restart ( void * *arg* )

A SYSCALL_PROC_RESTART handler.

This function tries to restart a process by proc_restart_isr call.

**Parameters**

| | |
|---|---|
| *arg* | A proc_runtime_arg_t pointer. |

#### 5.14.3.4 void scall_proc_stop ( void * *arg* )

A SYSCALL_PROC_STOP handler.

This function tries to stop a process by proc_stop_isr call.

**Parameters**

| | |
|---|---|
| *arg* | A proc_runtime_arg_t pointer. |

#### 5.14.3.5 void scall_proc_self_stop ( void * *arg* )

A SYSCALL_PROC_SELF_STOP handler.

This function stops calling process.

**Parameters**

| | |
|---|---|
| *arg* | Not used. |

#### 5.14.3.6 void scall_sched_proc_yeld ( void * *arg* )

A #SYSCALL_PROC_YELD handler.

Transfers control to another process.

**Parameters**

| | |
|---|---|
| *arg* | Not used. |

#### 5.14.3.7 void scall_proc_terminate ( void * *arg* )

A SYSCALL_PROC_TERMINATE handler.

This function terminates calling process after pmain return by _proc_terminate call.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a process. |

#### 5.14.3.8 void scall_proc_flag_stop ( void * *arg* )

A SYSCALL_PROC_FLAG_STOP handler.

This function process PROC_FLG_PRE_STOP of the calling process and clears masked flags of a calling process. It calls _proc_flag_stop.

**Parameters**

| | |
|---|---|
| *arg* | A poointer to a flag mask. |

#### 5.14.3.9 void scall_proc_reset_watchdog ( void * *arg* )

A SYSCALL_PROC_RESET_WATCHDOG handler.

This function calls _proc_reset_watchdog.

**Parameters**

| | |
|---|---|
| *arg* | Not used. |

#### 5.14.3.10 void scall_proc_set_prio ( void * *arg* )

A SYSCALL_PROC_SET_PRIO handler.

This function calls _proc_set_prio.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to proc_set_prio_arg_t object. |

#### 5.14.3.11 void scall_sig_wait ( void * *arg* )

A SYSCALL_SIG_WAIT hadnler.

Transfers a caller process in to signal wait state by _sig_wait_prologue call.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a signal. |

#### 5.14.3.12 void scall_sig_wakeup ( void * *arg* )

A SYSCALL_SIG_WAKEUP hadnler.

Calls _sig_wait_epilogue, handles PROC_FLG_PRE_STOP.

**Parameters**

| | |
|---|---|
| *arg* | . |

#### 5.14.3.13 void scall_sig_signal ( void * *arg* )

A SYSCALL_SIG_SIGNAL handler.

Wakes up one waiting process by sig_signal_isr call.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a signal. |

#### 5.14.3.14 void scall_sig_broadcast ( void * *arg* )

A SYSCALL_SIG_BROADCAST handler.

This function wakes up all waiting processes by sig_broadcast_isr call.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a signal. |

#### 5.14.3.15 void scall_sem_lock ( void * *arg* )

A SYSCALL_SEM_LOCK handler.

This function calls _sem_lock.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to an sem_lock_arg_t object. |

#### 5.14.3.16 void scall_sem_try_lock ( void * *arg* )

A SYSCALL_SEM_TRY_LOCK handler.

This function calls _sem_try_lock.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to an sem_lock_arg_t object. |

#### 5.14.3.17 void scall_sem_free ( void * *arg* )

A SYSCALL_SEM_FREE handler.

This function calls sem_free_isr.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a semaphore. |

#### 5.14.3.18 void scall_mutex_lock ( void * *arg* )

A SYSCALL_MUTEX_LOCK handler.

This function calls _mutex_lock.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to an mutex_lock_arg_t object. |

#### 5.14.3.19 void scall_mutex_try_lock ( void * *arg* )

A SYSCALL_MUTEX_TRY_LOCK handler.

This function calls _mutex_try_lock.

**Parameters**

| | |
|---|---|
| *arg* | A mutex_lock_arg_t pointer. |

#### 5.14.3.20 void scall_mutex_free ( void * *arg* )

A SYSCALL_MUTEX_FREE handler.

This function calls _mutex_free.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a mutex. |

#### 5.14.3.21 void scall_ipc_wait ( void * *arg* )

A SYSCALL_IPC_WAIT handler.

This funtion transfers a caller process to IPC wait state by _ipc_wait call.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to storage for data to receive. |

#### 5.14.3.22 void scall_ipc_send ( void * *arg* )

A SYSCALL_IPC_SEND handler.

This function tries to transfer data to waiting process by ipc_send_isr call.

**Parameters**

| | |
|---|---|
| *arg* | A ipc_send_arg_t pointer. |

#### 5.14.3.23 void scall_ipc_exchange ( void * *arg* )

A SYSCALL_IPC_EXCHANGE handler.

This function tries to transfer data to waiting process and on success transfers a caller process to IPCwait state. This function calls _ipc_exchange.

**Parameters**

| | |
|---|---|
| *arg* | A ipc_exchange_arg_t pointer. |

#### 5.14.3.24 void scall_user ( void * *arg* )

A SYSCALL_USER handler.

Calls user function.

**Parameters**

| | |
|---|---|
| *arg* | A pointer to a callee. |

**Warning**

Be carefull! Callee pointer is not checked before call!

### 5.14.4 Variable Documentation

#### 5.14.4.1 syscall_t syscall_num

System call processing routine.

This function calls system call handlers and passes arguments to them.

System call number.

#### 5.14.4.2 void* syscall_arg

System call argument.

## 5.15 include/timer.h File Reference

Asoftware timer headers.

**Macros**

- #define SPIN_LOCK_KERNEL_TIMER()
  *Wrapper macro.*
- #define SPIN_FREE_KERNEL_TIMER()
  *Wrapper macro.*
- #define CLEAR_TIMER(t) _clear_timer( (timer_t *)&t)
  *Reset software timer.*
- #define TIMER(t) (timer_t)_timer( (timer_t)t )
  *Get software timer value.*

**Functions**

- void wait_time (timer_t time)
  *Wait for certain time.*
- void _clear_timer (timer_t *t)
  *Clear software timer. For unternal usage.*
- timer_t _timer (timer_t t)
  *Get software timer. For internal usage.*

#### 5.15.1 Detailed Description

Asoftware timer headers. Software timers used for time-process synchronization.

**Warning**

Software timers can not be used for precision time interval measurement!

#### 5.15.2 Macro Definition Documentation

##### 5.15.2.1 #define SPIN_LOCK_KERNEL_TIMER( )

Wrapper macro.

A wrapper for kernel timer spin-lock, on single core system - empty macro.

##### 5.15.2.2 #define SPIN_FREE_KERNEL_TIMER( )

Wrapper macro.

A wrapper for kernel timer spin-free, on single core system - empty macro.

##### 5.15.2.3 #define CLEAR_TIMER( *t* ) _clear_timer( (timer_t *)&t)

Reset software timer.

**Parameters**

| | |
|---|---|
| *t* | A timer variable name. |

##### 5.15.2.4 #define TIMER( *t* ) (timer_t)_timer( (timer_t)t )

Get software timer value.

**Parameters**

| | |
|---|---|
| *t* | Software timer value. |

#### 5.15.3 Function Documentation

##### 5.15.3.1 void wait_time ( timer_t *time* )

Wait for certain time.

Caller process spins in a loop for a time.

**Parameters**

| | |
|---|---|
| *time* | Wait time. |

##### 5.15.3.2 void _clear_timer ( timer_t * *t* )

Clear software timer. For unternal usage.

**Parameters**

| | |
|---|---|
| *t* | A pointer to a timer. |

#### 5.15.3.3 timer_t _timer ( timer_t *t* )

Get software timer. For internal usage.

**Parameters**

| | |
|---|---|
| *t* | A timer value. |

## 5.16 include/xlist.h File Reference

A prioritized list header.

### Data Structures

- struct _xlist_t

  *A prioritized list.*

### Typedefs

- typedef struct _xlist_t xlist_t

### Functions

- void xlist_init (xlist_t *xlist)

  *An xlist_t object initiation.*
- item_t * xlist_head (xlist_t *xlist)

  *List head search.*
- void xlist_switch (xlist_t *xlist, prio_t prio)

  *Switch a head pointer.*

### 5.16.1 Detailed Description

A prioritized list header.

### 5.16.2 Typedef Documentation

#### 5.16.2.1 typedef struct _xlist_t xlist_t

See _xlist_t;

### 5.16.3 Function Documentation

#### 5.16.3.1 void xlist_init ( xlist_t * *xlist* )

An xlist_t object initiation.

**Parameters**

| | |
|---|---|
| *xlist* | An xlist_t pointer. |

#### 5.16.3.2 item_t* xlist_head ( xlist_t * *xlist* )

List head search.

**Parameters**

| | |
|---|---|
| *xlist* | An xlist_t pointer. |

**Returns**

The head pointer, wich is the most prioritized pointer in the list head pointer array.

#### 5.16.3.3 void xlist_switch ( xlist_t * *xlist*, prio_t *prio* )

Switch a head pointer.

Does xlist->item[prio] = xlist->item[prio]->next.

**Parameters**

| | |
|---|---|
| *xlist* | An xlist_t pointer. |
| *prio* | A priority to switch. |