BuguRTOS

0.6.1

Generated by Doxygen 1.6.3

Sun Jun 30 11:09:52 2013

Contents

1 Main Page

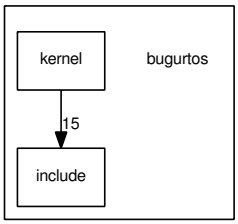
The BuguRTOS is a RTOS kernel. It is written by anonimous JUST FOR FUN.

Warning

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

2 Directory Documentation

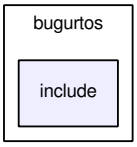
2.1 bugurtos/ Directory Reference



Directories

- directory include
- directory kernel

2.2 bugurtos/include/ Directory Reference



Files

- file bugurt.h
  The top header file.
- file crit_sec.h
  A critical section header.
- file index.h
  An index search header.
- file ipc.h
  An IPC header.
- file item.h
  A list item header.
- file kernel.h
  A kernel header.
- file mutex.h
  A mutex header.
- file pcounter.h
  A locked resource counter header.
- file pitem.h
  A prioritixed lis item header.
- file proc.h
  A process header.
- file sched.h
  Ascheduler header.
- file sem.h
  A counting semaphores header.
- file sig.h
  A signal header.

- file syscall.h

  System call header.

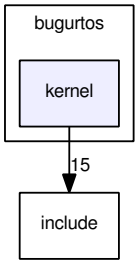
- file timer.h

  Asoftware timer headers.

- file xlist.h

  A prioritized list header.

2.3 bugurtos/kernel/ Directory Reference



Files

- file crit_sec.c
- file index.c
- file ipc.c
- file item.c
- file kernel.c
- file mutex.c
- file pcounter.c
- file pitem.c
- file proc.c
- file sched.c
- file sem.c
- file sig.c
- file syscall.c
- file timer.c
- file xlist.c

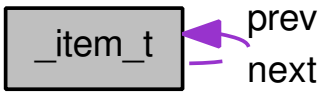
3 Data Structure Documentation

3.1 _item_t Struct Reference

A list item.

`#include "item.h"`

Collaboration diagram for _item_t:



Data Fields

- item_t * next
- item_t * prev

3.1.1 Detailed Description

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

3.1.2 Field Documentation

3.1.2.1 item_t* next

Next item in a list.

3.1.2.2 item_t* prev

Previous item in a list.

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

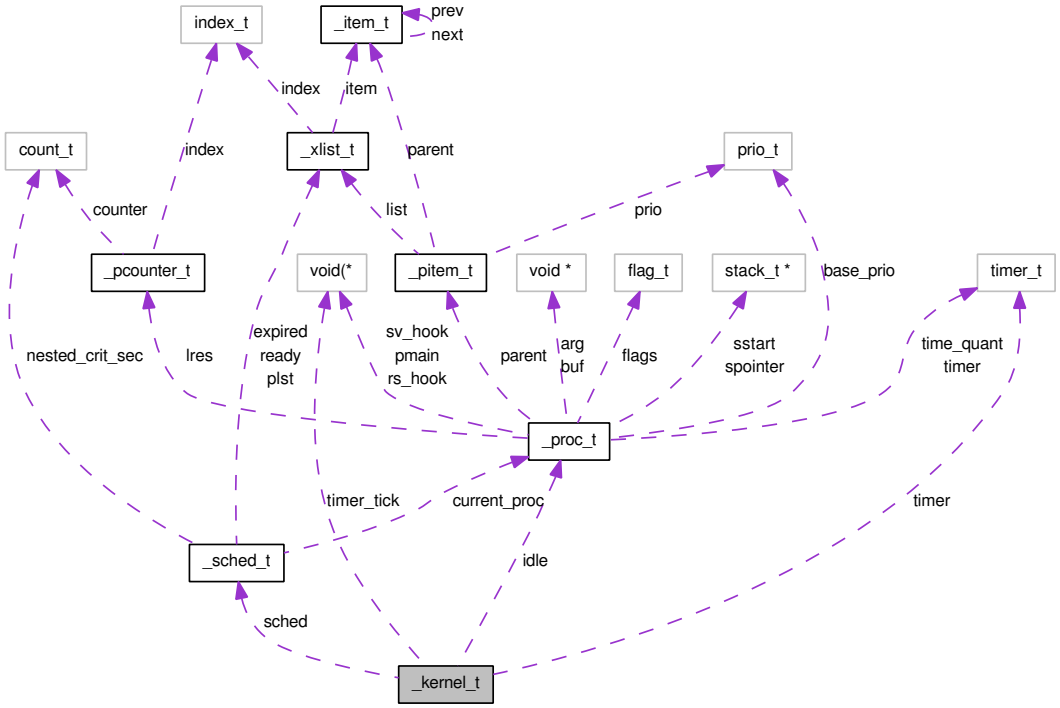
- bugurtos/include/item.h

3.2 _kernel_t Struct Reference

A BuguRTOS kernel structure.

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

Collaboration diagram for _kernel_t:



Data Fields

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

3.2.1 Detailed Description

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

3.2.2 Field Documentation

3.2.2.1 sched_t sched

The scheduler.

3.2.2.2 proc_t idle

The IDLE process.

3.2.2.3 timer_t timer

The system timer.

3.2.2.4 void(∗ timer_tick)(void)

The system timer tick hook pointer.

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

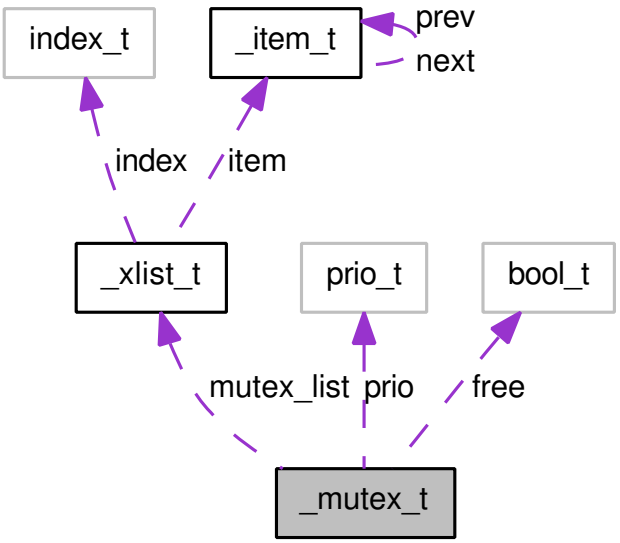
- bugurtos/include/kernel.h

3.3 _mutex_t Struct Reference

A mutex.

`#include "mutex.h"`

Collaboration diagram for _mutex_t:



Data Fields

- xlist_t mutex_list
- prio_t prio
- bool_t free

3.3.1 Detailed Description

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. Highest locker protocol is supported when CONFIG_USE_HIGHEST_LOCKER option is defined.

Warning

Only a process can lock or unlock a mutex!
Locked mutex can be unlocked only by a locker process!

3.3.2 Field Documentation

3.3.2.1 xlist_t mutex_list

A list of waiting processes.

3.3.2.2 prio_t prio

A priority of a mutex.

3.3.2.3 bool_t free

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

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

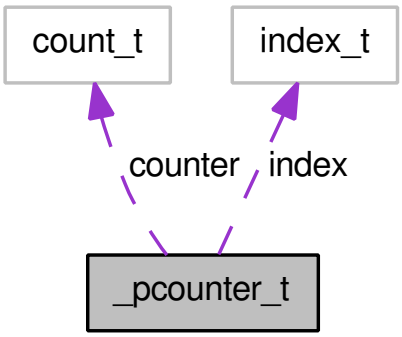
- bugurtos/include/mutex.h

3.4 _pcounter_t Struct Reference

A locked resource counter.

`#include "pcounter.h"`

Collaboration diagram for _pcounter_t:



Data Fields

- count_t counter [BITS_IN_INDEX_T]
- index_t index

3.4.1 Detailed Description

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

3.4.2 Field Documentation

3.4.2.1 count_t counter[BITS_IN_INDEX_T]

A counter array.

3.4.2.2 index_t index

An index to speedup search.

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

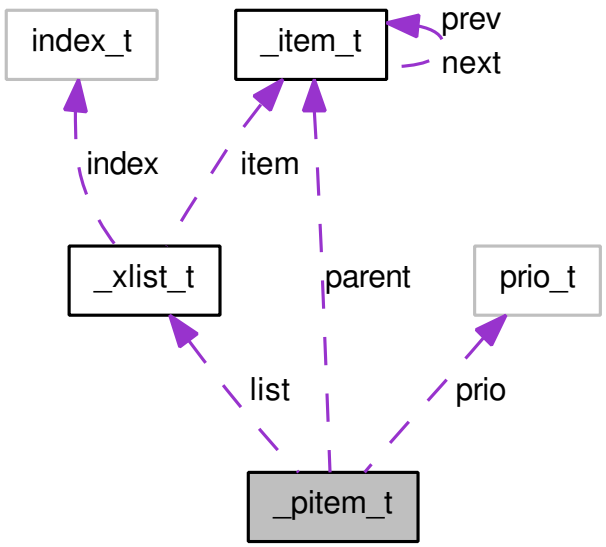
- bugurtos/include/pcounter.h

3.5 _pitem_t Struct Reference

A prioritized list item.

`#include "pitem.h"`

Collaboration diagram for _pitem_t:



Data Fields

- item_t parent
- xlist_t * list
- prio_t prio

3.5.1 Field Documentation

3.5.1.1 item_t parent

A perrent - item_t.

3.5.1.2 xlist_t* list

A pointer to an xlist_t object.

3.5.1.3 prio_t prio

A rpiority.

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

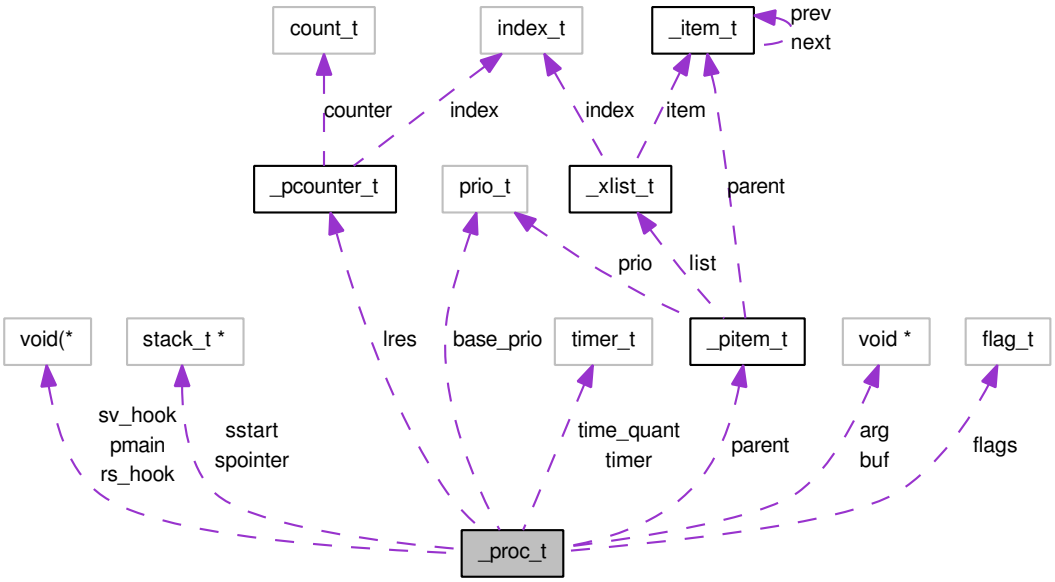
- bugurtos/include/pitem.h

3.6 _proc_t Struct Reference

A process.

`#include "proc.h"`

Collaboration diagram for _proc_t:



Data Fields

- pitem_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

3.6.1 Detailed Description

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.

3.6.2 Field Documentation

3.6.2.1 pitem_t parent

A parent is pitem_t.

3.6.2.2 flag_t flags

Process state flags (to treat process state quickly).

3.6.2.3 prio_t base_prio

A base process priority.

3.6.2.4 pcounter_t lres

A locked resource counter.

3.6.2.5 timer_t time_quant

A process time slice.

3.6.2.6 timer_t timer

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

3.6.2.7 void* buf

Apointer to process IPC data storage.

3.6.2.8 code_t pmain

A pointer to a process "main" routine.

3.6.2.9 code_t sv_hook

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

3.6.2.10 code_t rs_hook

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

3.6.2.11 void* arg

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

3.6.2.12 stack_t* sstart

A process stack bottom pointer.

3.6.2.13 stack_t* spointer

A process stack top pointer.

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

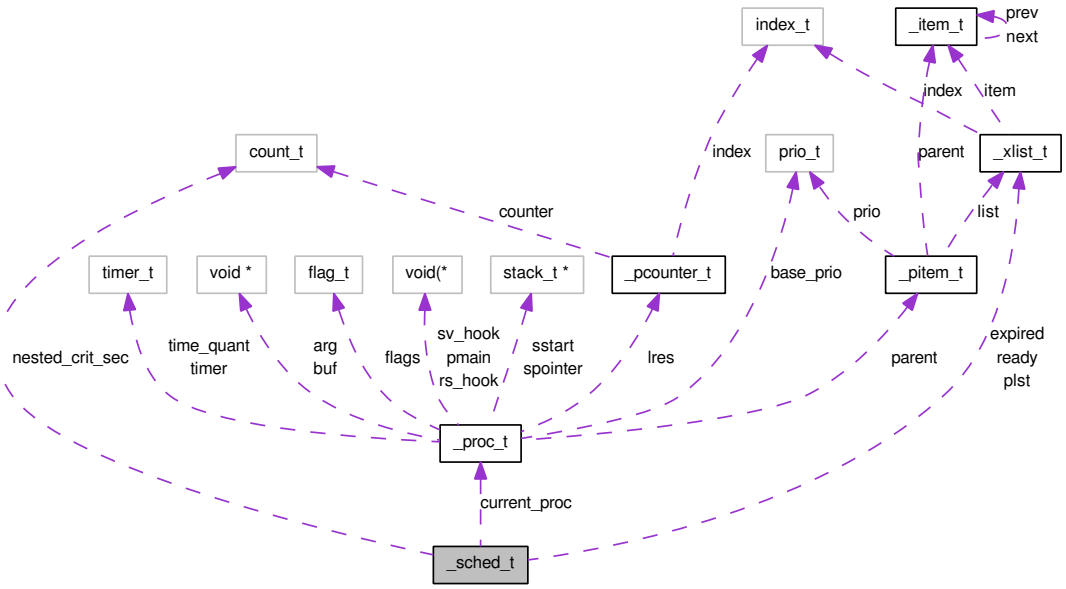
- bugurtos/include/proc.h

3.7 _sched_t Struct Reference

A scheduler.

`#include "sched.h"`

Collaboration diagram for _sched_t:



Data Fields

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

3.7.1 Detailed Description

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

3.7.2 Field Documentation

3.7.2.1 proc_t* current_proc

A currently running process.

3.7.2.2 xlist_t* ready

A pointer to a ready process list.

3.7.2.3 xlist_t* expired

Apointer to an expired process list.

3.7.2.4 xlist_t plst[2]

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

3.7.2.5 count_t nested_crit_sec

A critical section nesting count.

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

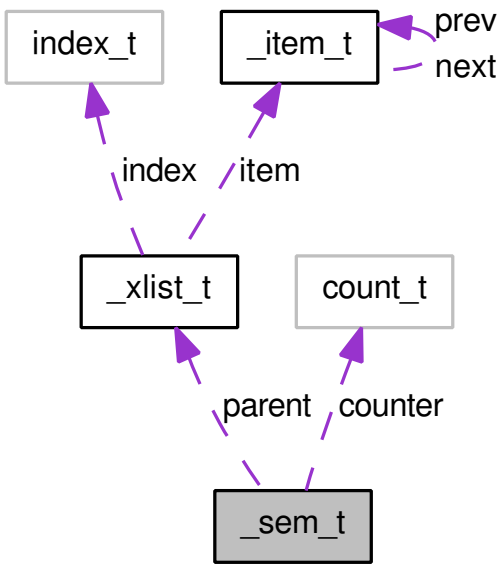
- bugurtos/include/sched.h

3.8 _sem_t Struct Reference

A counting semaphore.

`#include "sem.h"`

Collaboration diagram for _sem_t:



Data Fields

- xlist_t parent
- count_t counter

3.8.1 Detailed Description

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 unlocked by another.

3.8.2 Field Documentation

3.8.2.1 xlist_t parent

xlist_t is parrent type.

3.8.2.2 count_t counter

A counter.

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

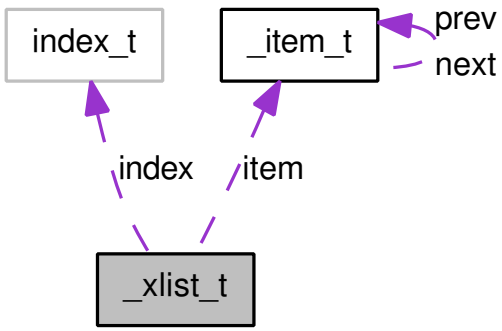
- bugurtos/include/sem.h

3.9 _xlist_t Struct Reference

A prioritized list.

`#include "xlist.h"`

Collaboration diagram for _xlist_t:



Data Fields

- item_t * item [BITS_IN_INDEX_T]
- index_t index

3.9.1 Detailed Description

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.

3.9.2 Field Documentation

3.9.2.1 item_t* item[BITS_IN_INDEX_T]

An array of list head pointers.

3.9.2.2 index_t index

Index for fast search.

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

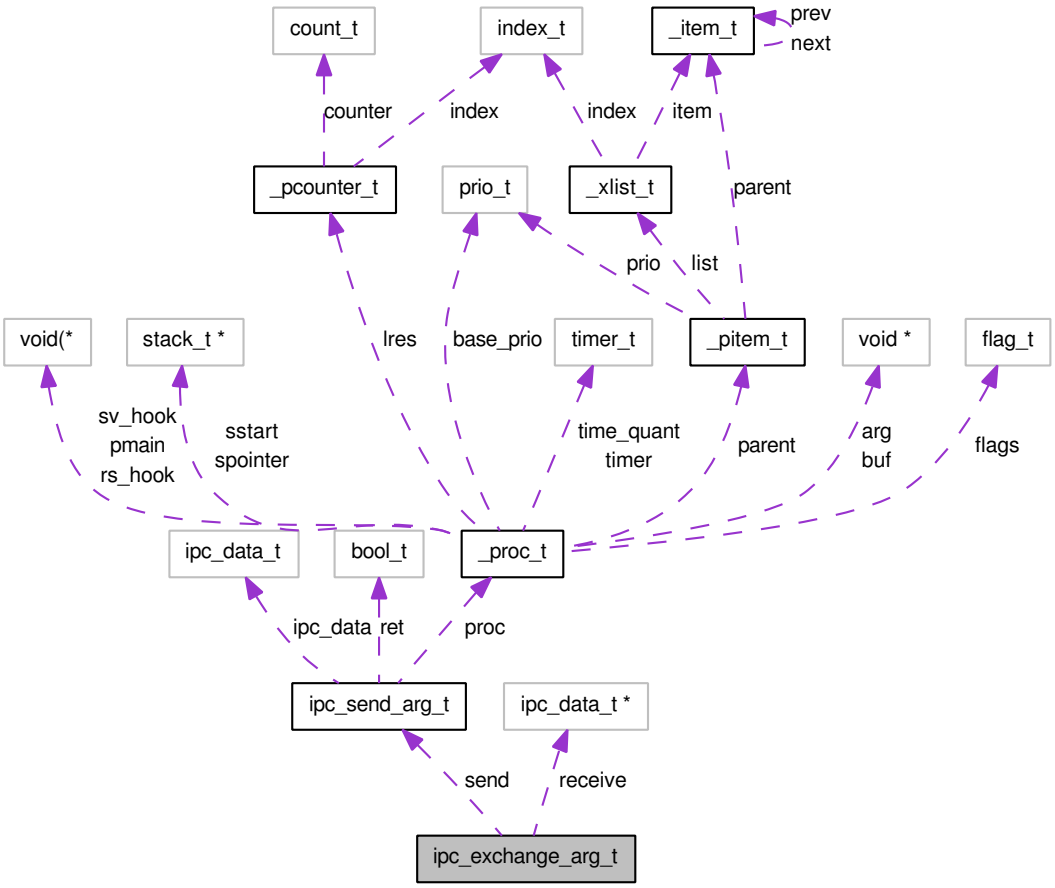
- bugurtos/include/xlist.h

3.10 ipc_exchange_arg_t Struct Reference

An argument structure for SYSCALL_IPC_EXCHANGE.

`#include "syscall.h"`

Collaboration diagram for ipc_exchange_arg_t:



Data Fields

- ipc_send_arg_t send
- ipc_data_t * receive

3.10.1 Field Documentation

3.10.1.1 ipc_send_arg_t send

A parent.

3.10.1.2 ipc_data_t* receive

Apointer to storage for data to receive.

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

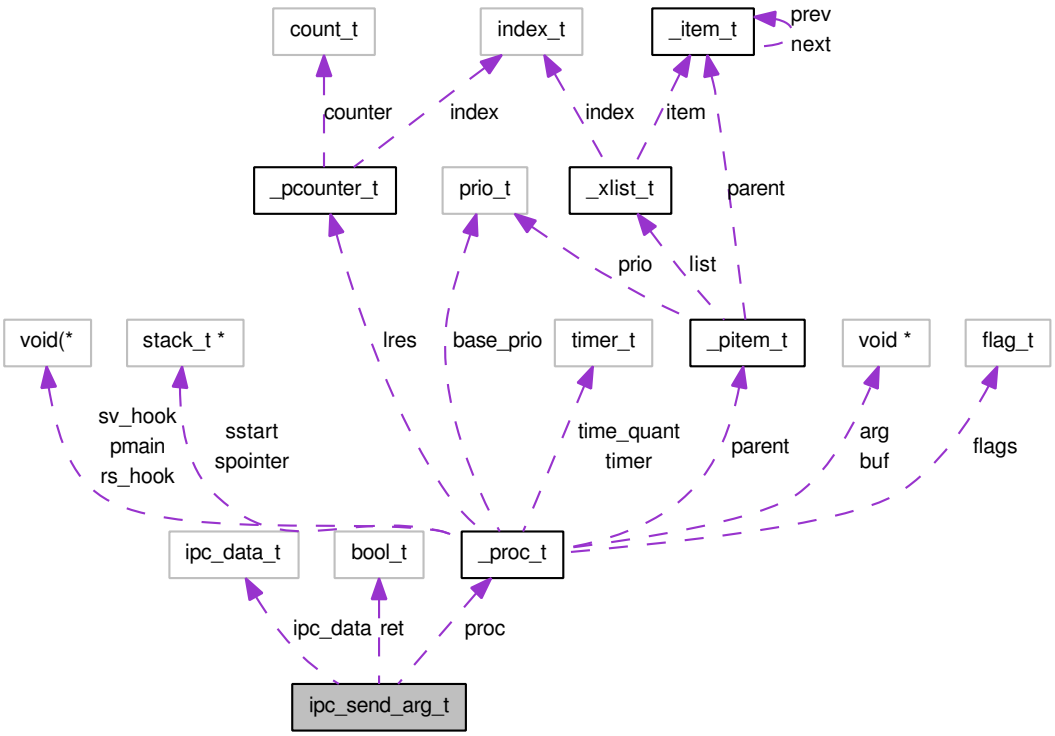
- bugurtos/include/syscall.h

3.11 ipc_send_arg_t Struct Reference

An argument structure for SYSCALL_IPC_SEND.

`#include "syscall.h"`

Collaboration diagram for ipc_send_arg_t:



Data Fields

- proc_t * proc
- bool_t ret
- ipc_data_t ipc_data

3.11.1 Field Documentation

3.11.1.1 proc_t* proc

A pointer to a destignation process.

3.11.1.2 bool_t ret

A storage for a result.

3.11.1.3 ipc_data_t ipc_data

A data to send.

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

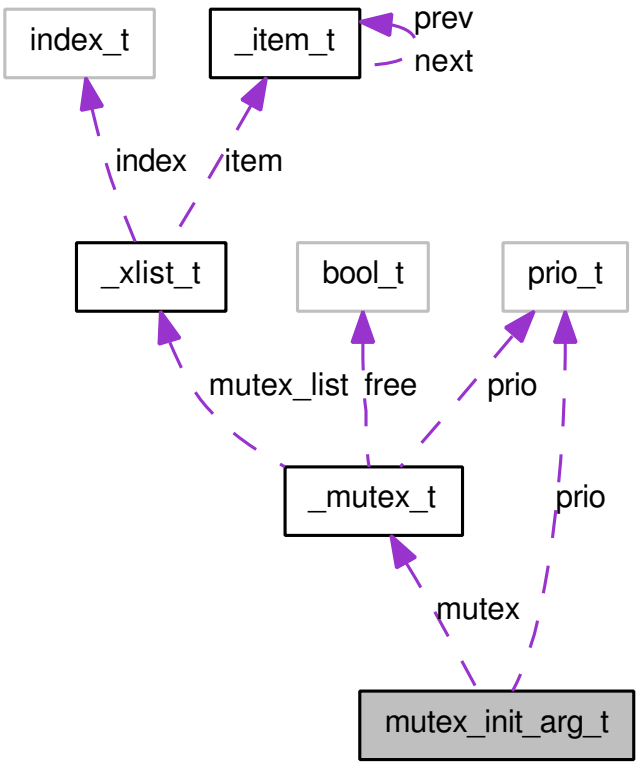
- bugurtos/include/syscall.h

3.12 mutex_init_arg_t Struct Reference

An argument structure for SYSCALL_MUTEX_INIT.

`#include "syscall.h"`

Collaboration diagram for mutex_init_arg_t:



Data Fields

- mutex_t * mutex
- prio_t prio

3.12.1 Field Documentation

3.12.1.1 mutex_t* mutex

A pointer to a mutex.

3.12.1.2 prio_t prio

A mutex priority.

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

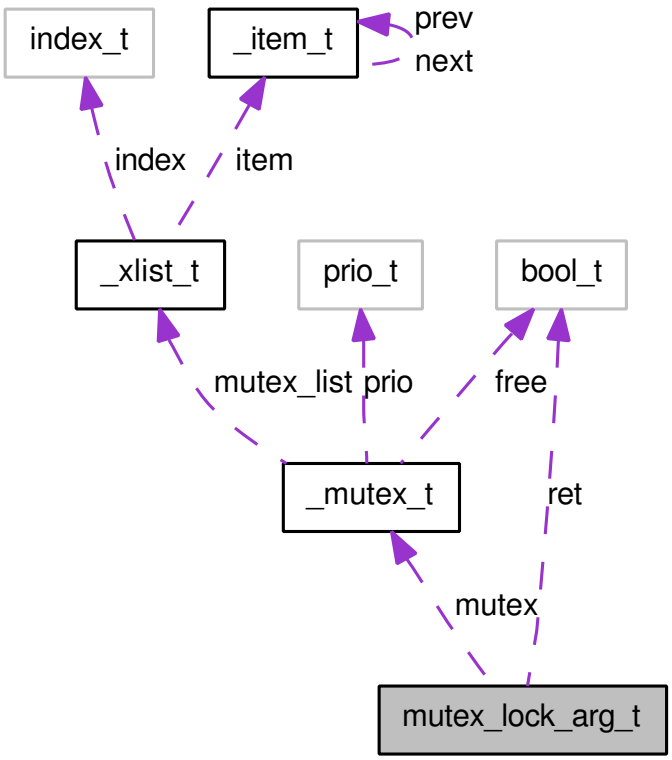
- bugurtos/include/syscall.h

3.13 mutex_lock_arg_t Struct Reference

An argument structure for SYSCALL_MUTEX_LOCK and SYSCALL_MUTEX_TRY_LOCK.

`#include "syscall.h"`

Collaboration diagram for mutex_lock_arg_t:



Data Fields

- mutex_t * mutex
- bool_t ret

3.13.1 Field Documentation

3.13.1.1 mutex_t* mutex

A pointer to a mutex.

3.13.1.2 bool_t ret

A storage for a result.

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

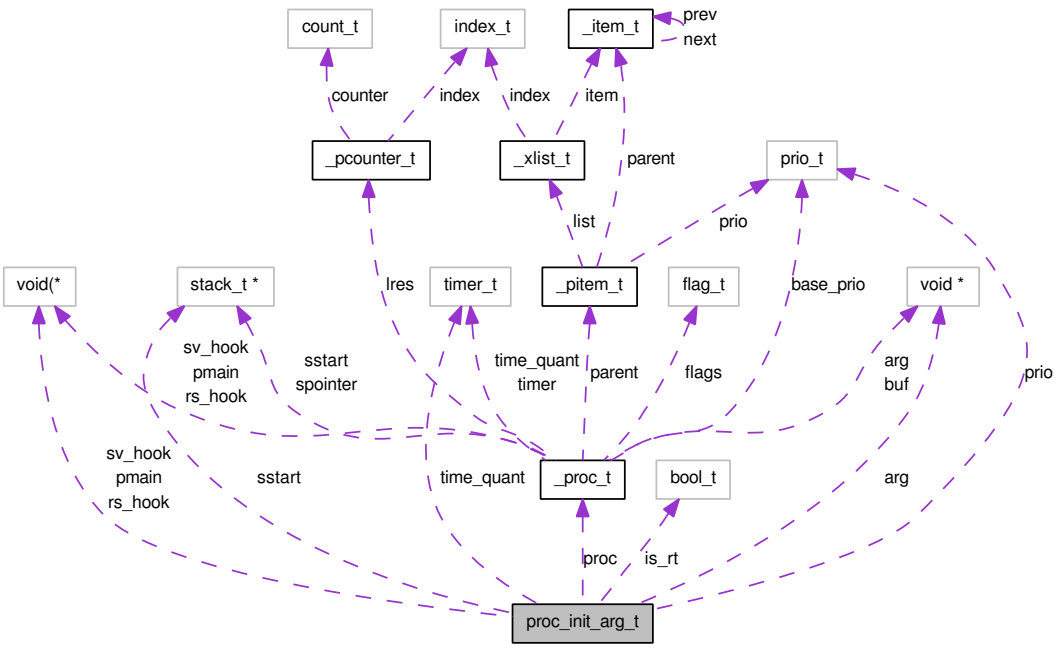
- bugurtos/include/syscall.h

3.14 proc_init_arg_t Struct Reference

An argument for SYSCALL_PROC_INIT.

`#include "syscall.h"`

Collaboration diagram for proc_init_arg_t:



Data Fields

- 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

3.14.1 Detailed Description

A process initialization structure. Contents an information about a process.

3.14.2 Field Documentation

3.14.2.1 proc_t* proc

A ponter to a initialized process.

3.14.2.2 code_t pmain

A pointer to a process "main" routine.

3.14.2.3 code_t sv_hook

A context save hook pointer.

3.14.2.4 code_t rs_hook

A context save hook pointer. , .

3.14.2.5 void* arg

An argument pointer.

3.14.2.6 stack_t* sstart

Aprocess stack bottom pointer.

3.14.2.7 prio_t prio

A process priority.

3.14.2.8 timer_t time_quant

A process time slice.

3.14.2.9 bool_t is_rt

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

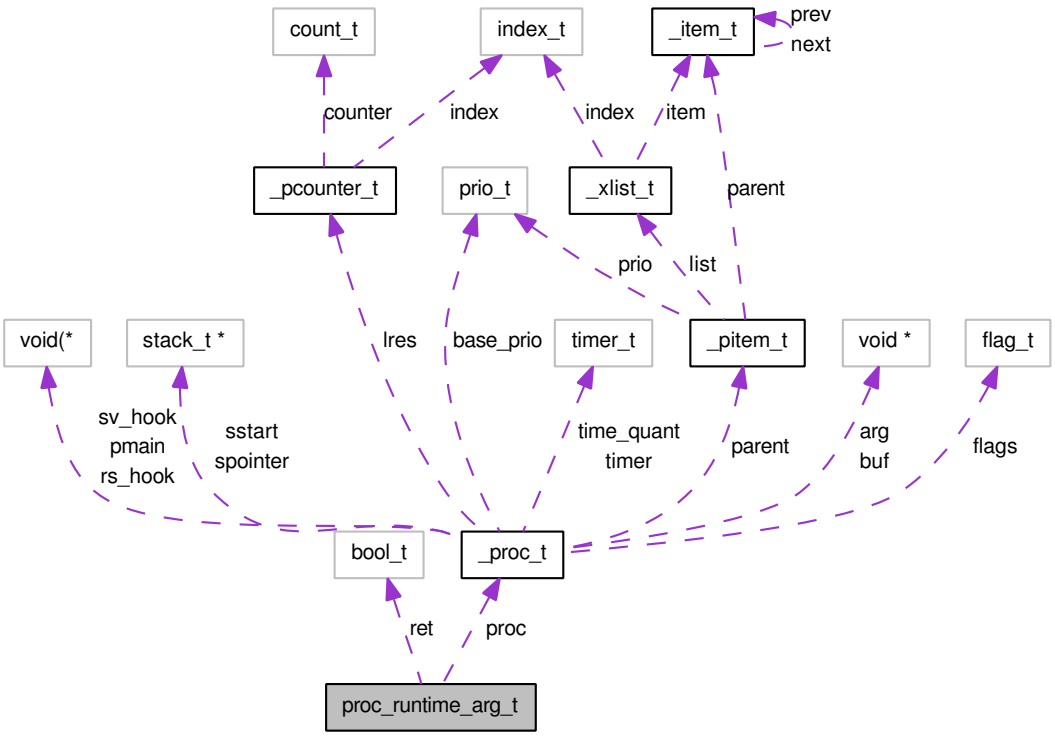
- bugurtos/include/syscall.h

3.15 proc_runtime_arg_t Struct Reference

An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_-PROC_STOP.

`#include "syscall.h"`

Collaboration diagram for proc_runtime_arg_t:



Data Fields

- proc_t * proc
- bool_t ret

3.15.1 Field Documentation

3.15.1.1 proc_t* proc

A pointer to a process.

3.15.1.2 bool_t ret

A result storage.

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

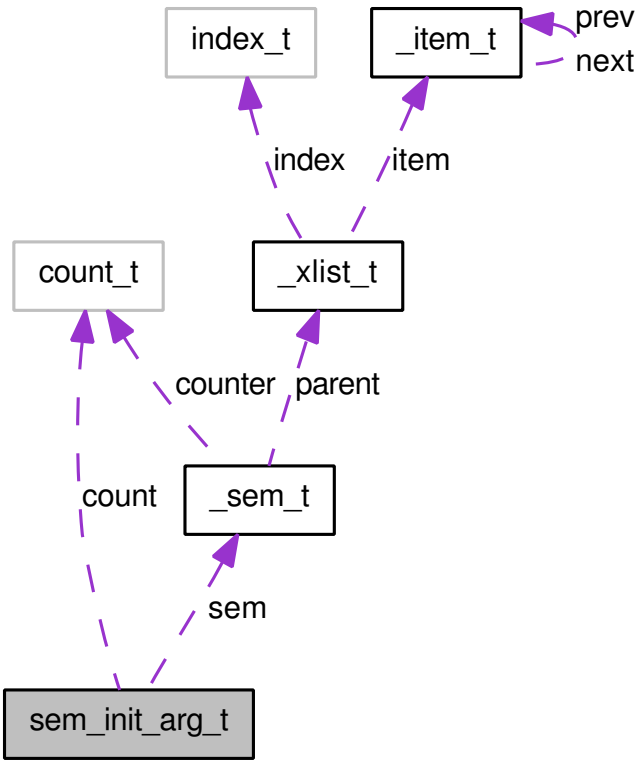
- bugurtos/include/syscall.h

3.16 sem_init_arg_t Struct Reference

A SYSCALL_SEM_INIT argument structure.

`#include "syscall.h"`

Collaboration diagram for sem_init_arg_t:



Data Fields

- sem_t * sem
- count_t count

3.16.1 Field Documentation

3.16.1.1 sem_t* sem

A pointer to a semaphore.

3.16.1.2 count_t count

A semaphore counter initial value.

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

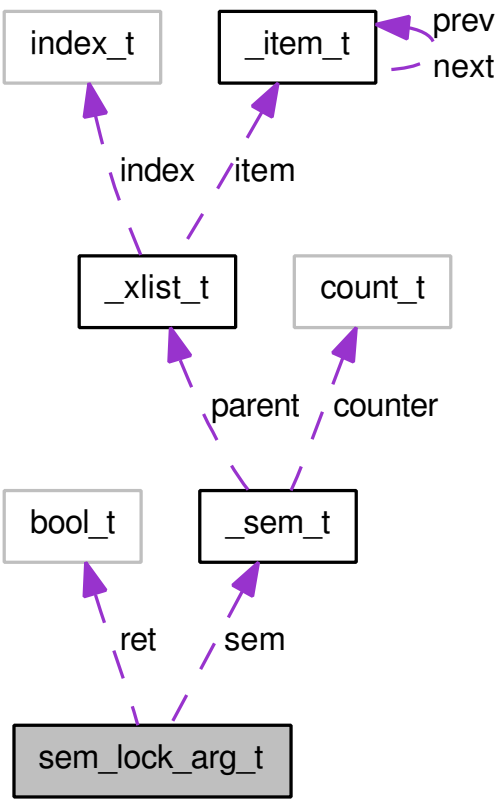
- bugurtos/include/syscall.h

3.17 sem_lock_arg_t Struct Reference

An argument structure for SYSCALL_SEM_LOCK and SYSCALL_SEM_TRY_LOCK.

`#include "syscall.h"`

Collaboration diagram for sem_lock_arg_t:



Data Fields

- sem_t * sem
- bool_t ret

3.17.1 Field Documentation

3.17.1.1 sem_t* sem

A pointer to a semaphore.

3.17.1.2 bool_t ret

A storage for a result.

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

- bugurtos/include/syscall.h

4 File Documentation

4.1 bugurtos/include/bugurt.h File Reference

The top header file.

```
#include "index.h"
```

Defines

- #define SPIN_INIT(arg)
  Wrapper macro.
- #define SPIN_LOCK(arg)
  Wrapper macro.
- #define SPIN_UNLOCK(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.

4.1.1 Detailed Description

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

4.1.2 Define Documentation

4.1.2.1 #define SPIN_INIT(arg)

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

4.1.2.2 #define SPIN_LOCK(arg)

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

4.1.2.3 #define SPIN_UNLOCK(arg)

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

4.1.2.4 #define RESCHED_PROC(proc) resched()

A wrapper for resched function.

4.1.3 Typedef Documentation

4.1.3.1 typedef void(* code_t)(void *)

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

4.1.4 Function Documentation

4.1.4.1 void resched (void)

Launces a reschedule sequence.

4.1.4.2 void disable_interrupts (void)

Disables interrupts globally.

4.1.4.3 void enable_interrupts (void)

Enables interrupts globally.

4.1.4.4 proc_t* current_proc (void)

Current process.

Returns

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

4.1.4.5 stack_t* proc_stack_init (stack_t * *sstart*, code_t *pmain*, void * *arg*, void(*)(void) *return_address*)

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.

Returns

a pointer to a prepared process stack top.

4.1.4.6 void init_bugurt (void)

Initiates the Kernel before the OSstart.

4.1.4.7 void start_bugurt (void)

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

4.1.4.8 void syscall_bugurt (syscall_t *num*, void * *arg*)

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).

4.2 bugurtos/include/crit_sec.h File Reference

A critical section header.

Defines

- #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)

4.2.1 Detailed Description

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.

4.2.2 Define Documentation

4.2.2.1 #define ENTER_CRIT_SEC() enter_crit_sec()

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.

4.2.2.2 #define EXIT_CRIT_SEC() exit_crit_sec()

A critical section end.

Warning

Must be used at the end of a code block.

4.2.3 Function Documentation

4.2.3.1 void enter_crit_sec (void)

A critical section start.

4.2.3.2 void exit_crit_sec (void)

A critical section end.

4.3 bugurtos/include/index.h File Reference

An index search header.

Functions

- prio_t index_search (index_t index)

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 prio_t index_search (index_t *index*)

4.4 bugurtos/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.

4.4.1 Detailed Description

4.4.2 Function Documentation

4.4.2.1 void _ipc_wait (void * *ipc_pointer*)

Warning

For internal usage only!!!

Parameters

ipc_pointer A pointer to IPCdata storage.

4.4.2.2 ipc_data_t ipc_wait (void)

Returns

IPC data.

4.4.2.3 bool_t ipc_send (proc_t * *proc*, ipc_data_t *ipc_data*)

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.

4.4.2.4 bool_t ipc_send_isr (proc_t * *proc*, ipc_data_t *ipc_data*)

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.

4.4.2.5 bool_t _ipc_exchange (proc_t * *proc*, ipc_data_t *send*, ipc_data_t * *receive*)

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.

4.4.2.6 bool_t ipc_exchange (proc_t * *proc*, ipc_data_t *send*, ipc_data_t * *receive*)

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.

4.5 bugurtos/include/item.h File Reference

A list item header.

Data Structures

- struct _item_t
  A list item.

Defines

- #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.

4.5.1 Detailed Description

4.5.2 Define Documentation

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

Static item initiation.

Parameters

a An item_t variable name.

4.5.3 Typedef Documentation

4.5.3.1 typedef struct _item_t item_t

4.5.4 Function Documentation

4.5.4.1 void item_init (item_t * *item*)

Parameters

item An item_t pointer.

4.5.4.2 void item_insert (item_t * *item*, item_t * *head*)

Parameters

item A pointer to an item.

head A pointer to a destignation list head.

4.5.4.3 void item_cut (item_t * *item*)

Parameters

item A pointer to an item to cut.

4.6 bugurtos/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.

4.6.1 Detailed Description

4.6.2 Typedef Documentation

4.6.2.1 typedef struct _kernel_t kernel_t

4.6.3 Function Documentation

4.6.3.1 void kernel_init (void)

This function prepares the kernel to work.

4.6.3.2 void idle_main (void * *arg*)

You can use builtin function, or you can write your own. IDLEprocess can work with timers, fire signals and UNLOCK 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.

4.6.4 Variable Documentation

4.6.4.1 kernel_t kernel

It's the one for the entire system!

4.7 bugurtos/include/mutex.h File Reference

A mutex header.

Data Structures

- struct _mutex_t

  A mutex.

Defines

- #define GET_PRIO(mutex) mutex->prio

Typedefs

- typedef struct _mutex_t mutex_t

Functions

- void mutex_init_isr (mutex_t *mutex, prio_t prio)
  A mutex initiation for usage in ISRs or in critical sections.
- void mutex_init (mutex_t *mutex, prio_t prio)
  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_unlock (mutex_t *mutex)
  Mutex unlock.
- 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.
- void _mutex_unlock (mutex_t *mutex)
  Mutex unlock kernel part.

4.7.1 Detailed Description

4.7.2 Define Documentation

4.7.2.1 #define GET_PRIO(mutex) mutex->prio

4.7.3 Typedef Documentation

4.7.3.1 typedef struct _mutex_t mutex_t

4.7.4 Function Documentation

4.7.4.1 void mutex_init_isr (mutex_t * *mutex*, prio_t *prio*)

Parameters

mutex A mutex pointer.
prio A mutex priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.7.4.2 void mutex_init (mutex_t * *mutex*, prio_t *prio*)

Parameters

mutex A mutex pointer.

prio A mutex priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.7.4.3 bool_t mutex_lock (mutex_t * *mutex*)

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

Parameters

mutex A mutex pointer.

Returns

1 if mutex was locked without wait, else 0.

4.7.4.4 bool_t mutex_try_lock (mutex_t * *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.

4.7.4.5 void mutex_unlock (mutex_t * *mutex*)

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

Parameters

mutex .

4.7.4.6 bool_t _mutex_lock (mutex_t * *mutex*)

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

Parameters

mutex A mutex pointer.

Returns

1 if mutex was locked without wait, else 0.

4.7.4.7 bool_t _mutex_try_lock (mutex_t * *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.

4.7.4.8 void _mutex_unlock (mutex_t * *mutex*)

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

Parameters

mutex A mutex pointer.

KERNEL_PREEMPT

4.8 bugurtos/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;.

4.8.1 Detailed Description

4.8.2 Typedef Documentation

4.8.2.1 typedef struct _pcounter_t pcounter_t

4.8.3 Function Documentation

4.8.3.1 void pcounter_init (pcounter_t * *pcounter*)

Parameters

pcounter A pcounter_t pointer.

4.8.3.2 void pcounter_inc (pcounter_t * *pcounter*, prio_t *prio*)

Parameters

pcounter A pcounter_t pointer.
prio A priority.

4.8.3.3 index_t pcounter_dec (pcounter_t * *pcounter*, prio_t *prio*)

Parameters

pcounter A pcounter_t pointer.
prio A priority.

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

Parameters

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

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

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.

4.9 bugurtos/include/pitem.h File Reference

A prioritixed lis item header.

Data Structures

- struct _pitem_t

  A prioritized list item.

Defines

- #define INIT_P_ITEM_T(a, p) { INIT_ITEM_T(a), (xlist_t *)0, (prio_t)p }

Typedefs

- typedef struct _pitem_t pitem_t

Functions

- void pitem_init (pitem_t *pitem, prio_t prio)

  A pitem_t object initiation.

- void pitem_insert (pitem_t *pitem, xlist_t *xlist)

  Insert pitem_t object to xlist_t container.

- void pitem_fast_cut (pitem_t *pitem)

  Fast cut pitem_t object from xlist_t container.

- void pitem_cut (pitem_t *pitem)

  Cut pitem_t object from xlist_t container.

- pitem_t * pitem_xlist_chain (xlist_t *src)

  "Chain" pitem_t objects from xlist_t container.

4.9.1 Detailed Description

4.9.2 Define Documentation

4.9.2.1 #define INIT_P_ITEM_T(a, p) { INIT_ITEM_T(a), (xlist_t *)0, (prio_t)p }

A static pitem_t object initiation.

Parameters

a A variable name.

p A rpiority.

4.9.3 Typedef Documentation

4.9.3.1 typedef struct _pitem_t pitem_t

4.9.4 Function Documentation

4.9.4.1 void pitem_init (pitem_t * *pitem*, prio_t *prio*)

Parameters

pitem A pitem_t pointer.

prio A priority.

4.9.4.2 void pitem_insert (pitem_t * *pitem*, xlist_t * *xlist*)

Parameters

pitem A pitem_t pointer.

xlist A pointer to destignation list.

4.9.4.3 void pitem_fast_cut (pitem_t * *pitem*)

This function cuts pitem_t object from xlist_t container without pitem->list field.

Parameters

pitem A pitem_t pointer.

4.9.4.4 void pitem_cut (pitem_t * *pitem*)

This function calls pitem_fast_cut and then nulls pitem->list field.

Parameters

pitem A pitem_t pointer.

4.9.4.5 pitem_t* pitem_xlist_chain (xlist_t * *src*)

Cut all pitem_t objects from xlist_t container and form an ordinary list from them.

Parameters

src A xlist_t pointer.

Returns

An ordinary doublelinked list head pointer.

4.10 bugurtos/include/proc.h File Reference

A process header.

Data Structures

- struct _proc_t
  A process.

Defines

- #define PROC_LRES_INIT(a) pcounter_init(&a->lres)
  Wrapper macro.
- #define PROC_LRES_INC(a, b) _proc_lres_inc(a,b)
  Wrapper macro.
- #define PROC_LRES_DEC(a, b) _proc_lres_dec(a,b)
  Wrapper macro.
- #define PROC_PRIO_CONTROL_STOPED(a) _proc_prio_control_stoped(a)
  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_READY ((flag_t)0x5)
- #define PROC_STATE_RESERVED_0x6 ((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_READY ((flag_t)0xD)
- #define PROC_STATE_RESERVED_0xE ((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_IPC_TEST(a) ((a->flags & PROC_STATE_MASK) == PROC_STATE_W_IPC)

  Checks if process is waiting for IPC.

- #define __proc_run(proc) pitem_insert((pitem_t *)proc, kernel.sched.ready)

  A routine that inserts a process to ready process list. For internal usage.

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!).

- index_t _proc_yeld (void)

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

- index_t proc_yeld (void)

  Pass control to next ready process.

- 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_run (proc_t *proc)

  A low level process run routine. For internal usage.

- void _proc_stop (proc_t *proc)

  A low level process stop routine. 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)

  APROC_FLG_PRE_STOP flag processing routine. For internal usage.

- void proc_flag_stop (flag_t mask)

  APROC_FLG_PRE_STOP flag processing routine.

- void _proc_lres_inc (proc_t *proc, prio_t prio)

  A locked resource counter increment routine. For internal usage.

- void _proc_lres_dec (proc_t *proc, prio_t prio)

  A locked resource counter decrement routine. For internal usage.

- void _proc_prio_control_stoped (proc_t *proc)

  A stopedprocess priority control routine.

- void _proc_prio_control_running (proc_t *proc)

  A runningprocess priority control routine.

4.10.1 Detailed Description

4.10.2 Define Documentation

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

Initiates proc->lres field of a process.

Parameters

a a pointer to a process.

4.10.2.2 #define PROC_LRES_INC(a, b) _proc_lres_inc(a,b)

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.

4.10.2.3 #define PROC_LRES_DEC(a, b) _proc_lres_dec(a,b)

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.

4.10.2.4 #define PROC_PRIO_CONTROL_STOPED(a) _proc_prio_control_stoped(a)

Stoped process priority control. If highest locker protocol is used, then this macro computes a proc->group->prio using proc->lres field, else this macro does nothing.

Parameters

a a pointer to a process.

4.10.2.5 #define PROC_FLG_RT ((flag_t)0x80)

This flag enables real time process scheduling policy.

4.10.2.6 #define PROC_FLG_MUTEX ((flag_t)0x40)

A process has locked some mutex controled resources.

4.10.2.7 #define PROC_FLG_SEM ((flag_t)0x20)

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.

4.10.2.8 #define PROC_FLG_PRE_STOP ((flag_t)0x10)

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

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

Used to test if a process has locked some resources.

4.10.2.10 #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

Used clear execution state bitts in proc->flags.

4.10.2.11 #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)

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

4.10.2.12 #define PROC_STATE_MASK ((flag_t)0x0F)

4.10.2.13 #define PROC_STATE_RESTART_MASK ((flag_t)0xC)

Used by proc_restart and proc_restart_isr to check for restart posibility.

4.10.2.14 #define PROC_STATE_RUN_MASK ((flag_t)0x7)

Used to check if the process has been run.

4.10.2.15 #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

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

4.10.2.16 #define PROC_STATE_STOPED ((flag_t)0x0)

Initial state, stoped.

4.10.2.17 #define PROC_STATE_END ((flag_t)0x1)

Normal process termination.

4.10.2.18 #define PROC_STATE_W_WD_STOPED ((flag_t)0x2)

Watchdog termination from W_RUNNING state.

4.10.2.19 #define PROC_STATE_WD_STOPED ((flag_t)0x3)

Watchdog termination.

4.10.2.20 #define PROC_STATE_DEAD ((flag_t)0x4)

Abnormal termination, terminated with resources locked.

4.10.2.21 #define PROC_STATE_READY ((flag_t)0x5)

Is ready to run.

4.10.2.22 #define PROC_STATE_RESERVED_0x6 ((flag_t)0x6)

Reserved.

4.10.2.23 #define PROC_STATE_RUNNING ((flag_t)0x7)

Is running.

4.10.2.24 #define PROC_STATE_W_MUT ((flag_t)0x8)

Is waiting for mutex.

4.10.2.25 #define PROC_STATE_W_SEM ((flag_t)0x9)

Is waiting for semaphore.

4.10.2.26 #define PROC_STATE_W_SIG ((flag_t)0xA)

Is waiting for signal.

4.10.2.27 #define PROC_STATE_W_IPC ((flag_t)0xB)

Is waiting for IPC.

4.10.2.28 #define PROC_STATE_W_DEAD ((flag_t)0xC)

Watchdog termination from W_RUNNING state with resources locked.

4.10.2.29 #define PROC_STATE_W_READY ((flag_t)0xD)

Is ready to run (special).

4.10.2.30 #define PROC_STATE_RESERVED_0xE ((flag_t)0xE)

Reserved.

4.10.2.31 #define PROC_STATE_W_RUNNING ((flag_t)0xF)

Is running (special).

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

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.

4.10.2.33 #define PROC_RUN_TEST(a) ((a->flags & PROC_STATE_RUN_MASK) >= PROC_STATE_READY)

4.10.2.34 #define PROC_IPC_TEST(a) ((a->flags & PROC_STATE_MASK) == PROC_STATE_W_IPC)

4.10.2.35 #define __proc_run(proc) pitem_insert((pitem_t *)proc, kernel.sched.ready)

4.10.3 Typedef Documentation

4.10.3.1 typedef struct _proc_t proc_t

4.10.4 Function Documentation

4.10.4.1 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*)

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.

4.10.4.2 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*)

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.

4.10.4.3 void proc_run_wrapper (proc_t * *proc*)

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.

4.10.4.4 void proc_terminate (void)

4.10.4.5 void _proc_terminate (void)

4.10.4.6 bool_t proc_run (proc_t * *proc*)

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.

4.10.4.7 bool_t proc_run_isr (proc_t * *proc*)

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.

4.10.4.8 bool_t proc_restart (proc_t * *proc*)

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.

4.10.4.9 bool_t proc_restart_isr (proc_t * *proc*)

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.

4.10.4.10 bool_t proc_stop (proc_t * *proc*)

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.

4.10.4.11 bool_t proc_stop_isr (proc_t * *proc*)

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.

4.10.4.12 void proc_self_stop (void)

This function stops caller process.

4.10.4.13 void _proc_self_stop (void)

This function stops caller process.

4.10.4.14 index_t _proc_yeld (void)

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

Returns

Zero if there are no other running processes, none zero if there is at least one.

KERNEL_PREEMPT

KERNEL_PREEMPT

4.10.4.15 index_t proc_yeld (void)

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

Returns

Zero if there are no other running processes, none zero if there is at least one.

4.10.4.16 void proc_reset_watchdog (void)

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.

4.10.4.17 void _proc_reset_watchdog (void)

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.

4.10.4.18 void _proc_run (proc_t * *proc*)

4.10.4.19 void _proc_stop (proc_t * *proc*)

4.10.4.20 void _proc_stop_flags_set (proc_t * *proc*, flag_t *mask*)

4.10.4.21 void _proc_flag_stop (flag_t *mask*)

4.10.4.22 void proc_flag_stop (flag_t *mask*)

4.10.4.23 void _proc_lres_inc (proc_t * *proc*, prio_t *prio*)

Parameters

proc A pointer to a process.

prio Alocked resource priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.10.4.24 void _proc_lres_dec (proc_t * *proc*, prio_t *prio*)

Parameters

proc A pointer to a process.

prio Alocked resource priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.10.4.25 void _proc_prio_control_stoped (proc_t * *proc*)

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

Parameters

proc - A pointer to a process.

4.10.4.26 void _proc_prio_control_running (proc_t * *proc*)

Used with CONFIG_USE_HIGHEST_LOCKER option. A process must be running when the routine is called.

Parameters

proc - A pointer to a process.

4.11 bugurtos/include/sched.h File Reference

Ascheduler header.

Data Structures

- struct _sched_t
  A scheduler.

Defines

- #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.

4.11.1 Detailed Description

Warning

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

4.11.2 Define Documentation

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

Initialization wrapper for sched variable in sched_schedule and sched_reschedule functions.

4.11.3 Typedef Documentation

4.11.3.1 typedef struct _sched_t sched_t

4.11.4 Function Documentation

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

This function prepares a scheduler object for work.

Parameters

sched - A sceduler pointer.

idle - An IDLE process pointer.

4.11.4.2 void sched_schedule (void)

This function switches processes in system timer interrupt handler.

KERNEL_PREEMPT

4.11.4.3 void sched_reschedule (void)

This function switches processes if needed.

4.12 bugurtos/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_unlock (sem_t *sem)
  Semaphore unlock.
- void sem_unlock_isr (sem_t *sem)
  Semaphore unlock 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.

4.12.1 Detailed Description

4.12.2 Typedef Documentation

4.12.2.1 typedef struct _sem_t sem_t

4.12.3 Function Documentation

4.12.3.1 void sem_init_isr (sem_t * *sem*, count_t *count*)

Parameters

sem A sem_t pointer.

count A counter start value.

4.12.3.2 void sem_init (sem_t * *sem*, count_t *count*)

Parameters

sem A sem_t pointer.

count A counter start value.

4.12.3.3 bool_t sem_lock (sem_t * *sem*)

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.

4.12.3.4 bool_t sem_try_lock (sem_t * *sem*)

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.

4.12.3.5 void sem_unlock (sem_t * *sem*)

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.

4.12.3.6 void sem_unlock_isr (sem_t * *sem*)

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.

4.12.3.7 bool_t _sem_lock (sem_t * *sem*)

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.

KERNEL_PREEMPT

4.12.3.8 bool_t _sem_try_lock (sem_t * *sem*)

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.

4.13 bugurtos/include/sig.h File Reference

A signal header.

Typedefs

- typedef xlist_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)

- 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.

4.13.1 Detailed Description

4.13.2 Typedef Documentation

4.13.2.1 typedef xlist_t sig_t

On one core system a signal is just a wait list.

4.13.3 Function Documentation

4.13.3.1 void sig_init_isr (sig_t * *sig*)

Parameters

sig A sig_t pointer.

4.13.3.2 void sig_init (sig_t * *sig*)

Parameters

sig A sig_t pointer.

4.13.3.3 void sig_wait (sig_t * *sig*)

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.

4.13.3.4 void _sig_wait_prologue (sig_t * *sig*)

This function stops cureent running process and insert it to signal wait list. On multicore system it allso does load prebalancing.

Parameters

sig A sig_t pointer.

4.13.3.5 void _sig_wait_epilogue (void)

KERNEL_PREEMPT

4.13.3.6 void sig_signal (sig_t * *sig*)

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.

4.13.3.7 void sig_broadcast (sig_t * *sig*)

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.

4.13.3.8 void sig_signal_isr (sig_t * *sig*)

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.

4.13.3.9 void sig_broadcast_isr (sig_t * *sig*)

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.

4.14 bugurtos/include/syscall.h File Reference

System call header.

Data Structures

- struct proc_init_arg_t

  An argument for SYSCALL_PROC_INIT.

- struct proc_runtime_arg_t

  An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

- struct sem_init_arg_t

  A SYSCALL_SEM_INIT argument structure.

- struct sem_lock_arg_t

  An argument structure for SYSCALL_SEM_LOCK and SYSCALL_SEM_TRY_LOCK.

- struct mutex_init_arg_t

  An argument structure for SYSCALL_MUTEX_INIT.

- struct mutex_lock_arg_t

  An argument structure for SYSCALL_MUTEX_LOCK and SYSCALL_MUTEX_TRY_LOCK.

- struct ipc_send_arg_t

  An argument structure for SYSCALL_IPC_SEND.

- struct ipc_exchange_arg_t

  An argument structure for SYSCALL_IPC_EXCHANGE.

Defines

- #define SYSCALL_PROC_INIT ((syscall_t)(1))
- #define SYSCALL_PROC_RUN (SYSCALL_PROC_INIT + (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_YELD (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))
- #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_YELD + (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_SIG_INIT (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))
- #define SYSCALL_SIG_WAIT (SYSCALL_SIG_INIT + (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_INIT (SYSCALL_SIG_BROADCAST + (syscall_t)(1))

- #define SYSCALL_SEM_LOCK (SYSCALL_SEM_INIT + (syscall_t)(1))
- #define SYSCALL_SEM_TRY_LOCK (SYSCALL_SEM_LOCK + (syscall_t)(1))
- #define SYSCALL_SEM_UNLOCK (SYSCALL_SEM_TRY_LOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_INIT (SYSCALL_SEM_UNLOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_LOCK (SYSCALL_MUTEX_INIT + (syscall_t)(1))
- #define SYSCALL_MUTEX_TRY_LOCK (SYSCALL_MUTEX_LOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_UNLOCK (SYSCALL_MUTEX_TRY_LOCK + (syscall_t)(1))
- #define SYSCALL_IPC_WAIT (SYSCALL_MUTEX_UNLOCK + (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)
- void scall_proc_init (void *arg)

  A SYSCALL_PROC_INIT handler.

- 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_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_sig_init (void *arg)

  A SYSCALL_SIG_INIT handler.

- void scall_sig_wait (void *arg)

  A SYSCALL_SIG_WAIT hadnler.

- void scall_sig_wakeup (void *arg)
- 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_init (void *arg)
  A SYSCALL_SEM_INIT 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_unlock (void *arg)
  A SYSCALL_SEM_UNLOCK handler.
- void scall_mutex_init (void *arg)
  A SYSCALL_MUTEX_INIT 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_unlock (void *arg)
  A SYSCALL_MUTEX_UNLOCK 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

4.14.1 Detailed Description

4.14.2 Define Documentation

4.14.2.1 #define SYSCALL_PROC_INIT ((syscall_t)(1))

A process initialization.

4.14.2.2 #define SYSCALL_PROC_RUN (SYSCALL_PROC_INIT + (syscall_t)(1))

A process launch.

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

A Process restart.

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

A process stop.

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

A process self stop.

4.14.2.6 #define SYSCALL_PROC_YELD (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))

Transfer control to another process.

4.14.2.7 #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_YELD + (syscall_t)(1))

A process termination.

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

PROC_FLG_PRE_STOP flag processing.

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

A real time process watchdog reset.

4.14.2.10 #define SYSCALL_SIG_INIT (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))

A signal initialization.

4.14.2.11 #define SYSCALL_SIG_WAIT (SYSCALL_SIG_INIT + (syscall_t)(1))

Wait for signal.

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

Signal wakeup processing.

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

Signal to one waiting process.

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

Signal to all waiting processes.

4.14.2.15 #define SYSCALL_SEM_INIT (SYSCALL_SIG_BROADCAST + (syscall_t)(1))

A semaphore initialization.

4.14.2.16 #define SYSCALL_SEM_LOCK (SYSCALL_SEM_INIT + (syscall_t)(1))

Lock a semaphore.

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

Try yo lock a semaphore.

4.14.2.18 #define SYSCALL_SEM_UNLOCK (SYSCALL_SEM_TRY_LOCK + (syscall_t)(1))

Unlock a semaphore.

4.14.2.19 #define SYSCALL_MUTEX_INIT (SYSCALL_SEM_UNLOCK + (syscall_t)(1))

A mutex initialization.

4.14.2.20 #define SYSCALL_MUTEX_LOCK (SYSCALL_MUTEX_INIT + (syscall_t)(1))

Lock a mutex.

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

Try to lock a mutex.

4.14.2.22 #define SYSCALL_MUTEX_UNLOCK (SYSCALL_MUTEX_TRY_LOCK + (syscall_t)(1))

Unlock a mutex.

4.14.2.23 #define SYSCALL_IPC_WAIT (SYSCALL_MUTEX_UNLOCK + (syscall_t)(1))

Wait for data (IPC).

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

Send data via IPC.

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

Exchange data via IPC.

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

A user syscall.

4.14.3 Function Documentation

4.14.3.1 void do_syscall (void)

4.14.3.2 void scall_proc_init (void * *arg*)

This function initiates a proces by proc_init_isr call.

Parameters

arg a proc_init_arg_t pointer.

4.14.3.3 void scall_proc_run (void * *arg*)

This function tries to launch a process by proc_run_isr call.

Parameters

arg A proc_runtime_arg_t pointer.

4.14.3.4 void scall_proc_restart (void * *arg*)

This function tries to restart a process by proc_restart_isr call.

Parameters

arg A proc_runtime_arg_t pointer.

4.14.3.5 void scall_proc_stop (void * *arg*)

This function tries to stop a process by proc_stop_isr call.

Parameters

arg A proc_runtime_arg_t pointer.

4.14.3.6 void scall_proc_self_stop (void * *arg*)

This function stops calling process.

Parameters

arg Not used.

4.14.3.7 void scall_proc_yeld (void * *arg*)

Transfers control to another process.

Parameters

arg Not used.

4.14.3.8 void scall_proc_terminate (void * *arg*)

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

Parameters

arg A pointer to a process.

4.14.3.9 void scall_proc_flag_stop (void * *arg*)

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.

4.14.3.10 void scall_proc_reset_watchdog (void * *arg*)

This function calls _proc_reset_watchdog.

Parameters

arg Not used.

4.14.3.11 void scall_sig_init (void * *arg*)

Initiates a signal by sig_init_isr call.

Parameters

arg A pointer to a signal.

4.14.3.12 void scall_sig_wait (void * *arg*)

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

Parameters

arg A pointer to a signal.

4.14.3.13 void scall_sig_wakeup (void * *arg*)

4.14.3.14 void scall_sig_signal (void * *arg*)

Wakes up one waiting process by sig_signal_isr call.

Warning

On a multicore system processes aren't woken up in a FIFO manner!

Parameters

arg A pointer to a signal.

4.14.3.15 void scall_sig_broadcast (void * *arg*)

This function wakes up all waiting processes by sig_broadcast_isr call.

Parameters

arg A pointer to a signal.

4.14.3.16 void scall_sem_init (void * *arg*)

This function initiates semaphore by sem_init_isr call.

Parameters

arg A pointer to a sem_init_arg_t structure.

4.14.3.17 void scall_sem_lock (void * *arg*)

This function calls _sem_lock.

Parameters

arg A pointer to an sem_lock_arg_t object.

4.14.3.18 void scall_sem_try_lock (void * *arg*)

This function calls _sem_try_lock.

Parameters

arg A pointer to an sem_lock_arg_t object.

4.14.3.19 void scall_sem_unlock (void * *arg*)

This function calls sem_unlock_isr.

Parameters

arg A pointer to a semaphore.

4.14.3.20 void scall_mutex_init (void * *arg*)

This function initiater mutex by mutex_init_isr call.

Parameters

arg A poiner to an mutex_init_arg_t object.

4.14.3.21 void scall_mutex_lock (void * *arg*)

This function calls _mutex_lock.

Parameters

arg A pointer to an mutex_lock_arg_t object.

4.14.3.22 void scall_mutex_try_lock (void * *arg*)

This function calls _mutex_try_lock.

Parameters

arg A mutex_lock_arg_t pointer.

4.14.3.23 void scall_mutex_unlock (void * *arg*)

This function calls _mutex_unlock.

Parameters

arg A pointer to a mutex.

4.14.3.24 void scall_ipc_wait (void * *arg*)

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

Parameters

arg A pointer to storage for data to receive.

4.14.3.25 void scall_ipc_send (void * *arg*)

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

Parameters

arg A ipc_send_arg_t pointer.

4.14.3.26 void scall_ipc_exchange (void * *arg*)

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.

4.14.3.27 void scall_user (void * *arg*)

Calls user function.

Parameters

arg A pointer to a callee.

Warning

Be carefull! Callee pointer is not checked before call!

4.14.4 Variable Documentation

4.14.4.1 syscall_t syscall_num

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

System call number.

4.14.4.2 void* syscall_arg

System call argument.

4.15 bugurtos/include/timer.h File Reference

Asoftware timer headers.

Defines

- #define SPIN_LOCK_KERNEL_TIMER()
  Wrapper macro.
- #define SPIN_UNLOCK_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.

4.15.1 Detailed Description

Software timers used for time-process synchronization.

Warning

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

4.15.2 Define Documentation

4.15.2.1 #define SPIN_LOCK_KERNEL_TIMER()

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

4.15.2.2 #define SPIN_UNLOCK_KERNEL_TIMER()

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

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

Parameters

t A timer variable name.

4.15.2.4 #define TIMER(t) (timer_t)_timer((timer_t)t)

Parameters

t Software timer value.

4.15.3 Function Documentation

4.15.3.1 void wait_time (timer_t *time*)

Caller process spins in a loop for a time.

Parameters

time Wait time.

4.15.3.2 void _clear_timer (timer_t * *t*)

Parameters

t A pointer to a timer.

4.15.3.3 timer_t _timer (timer_t *t*)

Parameters

t A timer value.

4.16 bugurtos/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.

4.16.1 Detailed Description

4.16.2 Typedef Documentation

4.16.2.1 typedef struct _xlist_t xlist_t

4.16.3 Function Documentation

4.16.3.1 void xlist_init (xlist_t * *xlist*)

Parameters

xlist An xlist_t pointer.

4.16.3.2 item_t* xlist_head (xlist_t * *xlist*)

Parameters

xlist An xlist_t pointer.

Returns

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

4.16.3.3 void xlist_switch (xlist_t * *xlist*, prio_t *prio*)

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

Parameters

xlist An xlist_t pointer.

prio A priority to switch.

4.17 bugurtos/kernel/crit_sec.c File Reference

```
#include "../include/bugurt.h"
#include "index.h"
#include "item.h"
#include "xlist.h"
#include "pitem.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"
```

Functions

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

4.17.1 Function Documentation

4.17.1.1 void enter_crit_sec (void)

A critical section start.

4.17.1.2 void exit_crit_sec (void)

A critical section end.

4.18 bugurtos/kernel/index.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- prio_t index_search (index_t index)

4.18.1 Function Documentation

4.18.1.1 prio_t index_search (index_t *index*)

4.19 bugurtos/kernel/ipc.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- void _ipc_wait (void *ipc_pointer)
  Wait for IPC kernel part.
- 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.

4.19.1 Function Documentation

4.19.1.1 void _ipc_wait (void * *ipc_pointer*)

Warning

For internal usage only!!!

Parameters

ipc_pointer A pointer to IPCdata storage.

4.19.1.2 bool_t ipc_send_isr (proc_t * *proc*, ipc_data_t *ipc_data*)

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.

4.19.1.3 bool_t _ipc_exchange (proc_t * *proc*, ipc_data_t *send*, ipc_data_t * *receive*)

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.

4.20 bugurtos/kernel/item.c File Reference

```
#include "../include/bugurt.h"
```

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.

4.20.1 Function Documentation

4.20.1.1 void item_init (item_t * *item*)

Parameters

item An item_t pointer.

4.20.1.2 void item_insert (item_t * *item*, item_t * *head*)

Parameters

item A pointer to an item.
head A pointer to a destignation list head.

4.20.1.3 void item_cut (item_t * *item*)

Parameters

item A pointer to an item to cut.

4.21 bugurtos/kernel/kernel.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- WEAK void idle_main (void *arg)
  An IDLE process main function.
- void kernel_init (void)
  The kernel initiation.

Variables

- kernel_t kernel

  The BuguRTOSkernel.

4.21.1 Function Documentation

4.21.1.1 WEAK void idle_main (void * *arg*)

You can use builtin function, or you can write your own. IDLEprocess can work with timers, fire signals and UNLOCK 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.

4.21.1.2 void kernel_init (void)

This function prepares the kernel to work.

4.21.2 Variable Documentation

4.21.2.1 kernel_t kernel

It's the one for the entire system!

4.22 bugurtos/kernel/mutex.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- void mutex_init_isr (mutex_t *mutex, prio_t prio)

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

- 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.

- void _mutex_unlock (mutex_t *mutex)

  Mutex unlock kernel part.

4.22.1 Function Documentation

4.22.1.1 void mutex_init_isr (mutex_t * *mutex*, prio_t *prio*)

Parameters

mutex A mutex pointer.

prio A mutex priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.22.1.2 bool_t _mutex_lock (mutex_t * *mutex*)

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

Parameters

mutex A mutex pointer.

Returns

1 if mutex was locked without wait, else 0.

4.22.1.3 bool_t _mutex_try_lock (mutex_t * *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.

4.22.1.4 void _mutex_unlock (mutex_t * *mutex*)

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

Parameters

mutex A mutex pointer.

KERNEL_PREEMPT

4.23 bugurtos/kernel/pcounter.c File Reference

```
#include "../include/bugurt.h"
```

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;.

4.23.1 Function Documentation

4.23.1.1 void pcounter_init (pcounter_t * *pcounter*)

Parameters

pcounter A pcounter_t pointer.

4.23.1.2 void pcounter_inc (pcounter_t * *pcounter*, prio_t *prio*)

Parameters

pcounter A pcounter_t pointer.

prio A priority.

4.23.1.3 index_t pcounter_dec (pcounter_t * *pcounter*, prio_t *prio*)

Parameters

pcounter A pcounter_t pointer.

prio A priority.

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

Parameters

pcounter A pcounter_t pointer.

prio A priority.

count A number of increment steps.

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

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.

4.24 bugurtos/kernel/pitem.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- void pitem_init (pitem_t *pitem, prio_t prio)
  A pitem_t object initiation.
- void pitem_insert (pitem_t *pitem, xlist_t *xlist)
  Insert pitem_t object to xlist_t container.
- void pitem_fast_cut (pitem_t *pitem)
  Fast cut pitem_t object from xlist_t container.
- void pitem_cut (pitem_t *pitem)
  Cut pitem_t object from xlist_t container.
- pitem_t * pitem_xlist_chain (xlist_t *src)
  "Chain" pitem_t objects from xlist_t container.

4.24.1 Function Documentation

4.24.1.1 void pitem_init (pitem_t * *pitem*, prio_t *prio*)

Parameters

pitem A pitem_t pointer.

prio A priority.

4.24.1.2 void pitem_insert (pitem_t * *pitem*, xlist_t * *xlist*)

Parameters

pitem A pitem_t pointer.

xlist A pointer to destignation list.

4.24.1.3 void pitem_fast_cut (pitem_t * *pitem*)

This function cuts pitem_t object from xlist_t container without pitem->list field.

Parameters

pitem A pitem_t pointer.

4.24.1.4 void pitem_cut (pitem_t * *pitem*)

This function calls pitem_fast_cut and then nulls pitem->list field.

Parameters

pitem A pitem_t pointer.

4.24.1.5 pitem_t* pitem_xlist_chain (xlist_t * *src*)

Cut all pitem_t objects from xlist_t container and form an ordinary list from them.

Parameters

src A xlist_t pointer.

Returns

An ordinary doublelinked list head pointer.

4.25 bugurtos/kernel/proc.c File Reference

```
#include "../include/bugurt.h"
```

Defines

- #define __proc_stop(proc) pitem_cut((pitem_t *)proc)

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_run (proc_t *proc)

  A low level process run routine. For internal usage.

- 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_isr (proc_t *proc)

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

- void _proc_stop (proc_t *proc)

  A low level process stop routine. For internal usage.

- static void _proc_stop_ensure (proc_t *proc)
- void _proc_stop_flags_set (proc_t *proc, flag_t mask)

  A low level process stop with flags set routine. For internal usage.

- bool_t proc_stop_isr (proc_t *proc)

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

- void _proc_flag_stop (flag_t mask)

  APROC_FLG_PRE_STOP flag processing routine. For internal usage.

- void _proc_self_stop (void)

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

- index_t _proc_yeld (void)

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

- void _proc_terminate (void)

  A process termination routine called after proc->pmain return. Internal usage function.

- void _proc_reset_watchdog (void)

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

- void _proc_lres_inc (proc_t *proc, prio_t prio)

A locked resource counter increment routine. For internal usage.

- void _proc_lres_dec (proc_t *proc, prio_t prio)
  A locked resource counter decrement routine. For internal usage.

- void _proc_prio_control_stoped (proc_t *proc)
  A stopedprocess priority control routine.

4.25.1 Define Documentation

4.25.1.1 #define __proc_stop(proc) pitem_cut((pitem_t *)proc)

4.25.2 Function Documentation

4.25.2.1 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*)

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.

4.25.2.2 void _proc_run (proc_t * *proc*)

4.25.2.3 bool_t proc_run_isr (proc_t * *proc*)

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.

4.25.2.4 bool_t proc_restart_isr (proc_t * *proc*)

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.

4.25.2.5 void _proc_stop (proc_t * *proc*)

4.25.2.6 static void _proc_stop_ensure (proc_t * *proc*) [static]

4.25.2.7 void _proc_stop_flags_set (proc_t * *proc*, flag_t *mask*)

4.25.2.8 bool_t proc_stop_isr (proc_t * *proc*)

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.

4.25.2.9 void _proc_flag_stop (flag_t *mask*)

4.25.2.10 void _proc_self_stop (void)

This function stops caller process.

4.25.2.11 index_t _proc_yeld (void)

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

Returns

Zero if there are no other running processes, none zero if there is at least one.

KERNEL_PREEMPT

KERNEL_PREEMPT

4.25.2.12 void _proc_terminate (void)

4.25.2.13 void _proc_reset_watchdog (void)

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.

4.25.2.14 void _proc_lres_inc (proc_t * *proc*, prio_t *prio*)

Parameters

proc A pointer to a process.

prio Alocked resource priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.25.2.15 void _proc_lres_dec (proc_t * *proc*, prio_t *prio*)

Parameters

proc A pointer to a process.

prio Alocked resource priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.25.2.16 void _proc_prio_control_stoped (proc_t * *proc*)

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

Parameters

proc - A pointer to a process.

4.26 bugurtos/kernel/sched.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- void sched_init (sched_t *sched, proc_t *idle)
  A scheduler initiation routine.
- static void _sched_switch_current (sched_t *sched, proc_t *current_proc)
- void sched_schedule (void)
  A scheduler routine.
- void sched_reschedule (void)
  Recheduler routine.

4.26.1 Function Documentation

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

This function prepares a scheduler object for work.

Parameters

sched - A sceduler pointer.

idle - An IDLE process pointer.

4.26.1.2 static void _sched_switch_current (sched_t * *sched*, proc_t * *current_proc*) `[static]`

4.26.1.3 void sched_schedule (void)

This function switches processes in system timer interrupt handler.

KERNEL_PREEMPT

4.26.1.4 void sched_reschedule (void)

This function switches processes if needed.

4.27 bugurtos/kernel/sem.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- void sem_init_isr (sem_t *sem, count_t count)
  Semaphore initiation from ISR.
- 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.

- void sem_unlock_isr (sem_t *sem)

  Semaphore unlock for ISRusage.

4.27.1 Function Documentation

4.27.1.1 void sem_init_isr (sem_t * *sem*, count_t *count*)

Parameters

sem A sem_t pointer.

count A counter start value.

4.27.1.2 bool_t _sem_lock (sem_t * *sem*)

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.

KERNEL_PREEMPT

4.27.1.3 bool_t _sem_try_lock (sem_t * *sem*)

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.

4.27.1.4 void sem_unlock_isr (sem_t * *sem*)

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.

4.28 bugurtos/kernel/sig.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- void sig_init_isr (sig_t *sig)
  A signal initiation from ISR or critical section.
- void _sig_wait_prologue (sig_t *sig)
  A signal wait prologue kernel part.
- static void _sig_set_wakeup_flags (proc_t *proc)
- static void _sig_wakeup_list_proc (proc_t *proc)
- void _sig_wait_epilogue (void)
- 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.

4.28.1 Function Documentation

4.28.1.1 void sig_init_isr (sig_t * *sig*)

Parameters

sig A sig_t pointer.

4.28.1.2 void _sig_wait_prologue (sig_t * *sig*)

This function stops cureent running process and insert it to signal wait list. On multicore system it allso does load prebalancing.

Parameters

sig A sig_t pointer.

4.28.1.3 static void _sig_set_wakeup_flags (proc_t * *proc*) [static]

4.28.1.4 static void _sig_wakeup_list_proc (proc_t * *proc*) [static]

4.28.1.5 void _sig_wait_epilogue (void)

KERNEL_PREEMPT

4.28.1.6 void sig_signal_isr (sig_t * *sig*)

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.

4.28.1.7 void sig_broadcast_isr (sig_t * *sig*)

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.

4.29 bugurtos/kernel/syscall.c File Reference

```
#include "../include/bugurt.h"
```

Functions

- SYSCALL_TABLE (syscall_routine[ ])
- void do_syscall (void)
- void scall_proc_init (void *arg)

  SYSCALL_PROC_INIT.

- 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 scall_proc_run (void *arg)

A SYSCALL_PROC_RUN handler.

- bool_t proc_run (proc_t *proc)
  A process launch routine.
- void scall_proc_restart (void *arg)
  A SYSCALL_PROC_RESTART handler.
- bool_t proc_restart (proc_t *proc)
  Aprocess restart routine.
- void scall_proc_stop (void *arg)
  A SYSCALL_PROC_STOP handler.
- bool_t proc_stop (proc_t *proc)
  A process stop routine.
- void scall_proc_self_stop (void *arg)
  A SYSCALL_PROC_SELF_STOP handler.
- void proc_self_stop (void)
  A process self stop routine.
- void scall_proc_yeld (void *arg)
  A SYSCALL_PROC_YELD handler.
- index_t proc_yeld (void)
  Pass control to next ready process.
- void scall_proc_terminate (void *arg)
  A SYSCALL_PROC_TERMINATE handler.
- void proc_terminate (void)
  A process termination routine called after proc->pmain return. Internal usage function.
- void scall_proc_flag_stop (void *arg)
  A SYSCALL_PROC_FLAG_STOP handler.
- void proc_flag_stop (flag_t mask)
  APROC_FLG_PRE_STOP flag processing routine.
- void scall_proc_reset_watchdog (void *arg)
  A SYSCALL_PROC_RESET_WATCHDOG handler.
- void proc_reset_watchdog (void)
  A watchdog reset routine for real time processes.
- void scall_sig_init (void *arg)
  A SYSCALL_SIG_INIT handler.

- void sig_init (sig_t *sig)
  Signal initiation.
- void scall_sig_wait (void *arg)
  A SYSCALL_SIG_WAIT hadnler.
- void scall_sig_wakeup (void *arg)
- void sig_wait (sig_t *sig)
  Wait for a singnal.
- void scall_sig_signal (void *arg)
  A SYSCALL_SIG_SIGNAL handler.
- void sig_signal (sig_t *sig)
  Fire a signal, launch one waiting process.
- void scall_sig_broadcast (void *arg)
  A SYSCALL_SIG_BROADCAST handler.
- void sig_broadcast (sig_t *sig)
  Fire a signal, launch all waiting processes.
- void scall_sem_init (void *arg)
  A SYSCALL_SEM_INIT handler.
- void sem_init (sem_t *sem, count_t count)
  Semaphore initiation.
- void scall_sem_lock (void *arg)
  A SYSCALL_SEM_LOCK handler.
- bool_t sem_lock (sem_t *sem)
  A semaphore lock.
- void scall_sem_try_lock (void *arg)
  A SYSCALL_SEM_TRY_LOCK handler.
- bool_t sem_try_lock (sem_t *sem)
  Try to lock a semaphore.
- void scall_sem_unlock (void *arg)
  A SYSCALL_SEM_UNLOCK handler.
- void sem_unlock (sem_t *sem)
  Semaphore unlock.
- void scall_mutex_init (void *arg)
  A SYSCALL_MUTEX_INIT handler.
- void mutex_init (mutex_t *mutex, prio_t prio)

A mutex initiation.

- void scall_mutex_lock (void *arg)
  A SYSCALL_MUTEX_LOCK handler.
- bool_t mutex_lock (mutex_t *mutex)
  Lock a mutex.
- void scall_mutex_try_lock (void *arg)
  A SYSCALL_MUTEX_TRY_LOCK handler.
- bool_t mutex_try_lock (mutex_t *mutex)
  Try to lock a mutex.
- void scall_mutex_unlock (void *arg)
  A SYSCALL_MUTEX_UNLOCK handler.
- void mutex_unlock (mutex_t *mutex)
  Mutex unlock.
- void scall_ipc_wait (void *arg)
  A SYSCALL_IPC_WAIT handler.
- ipc_data_t ipc_wait (void)
  Wait for IPC.
- void scall_ipc_send (void *arg)
  A SYSCALL_IPC_SEND handler.
- bool_t ipc_send (proc_t *proc, ipc_data_t ipc_data)
  IPCdata transmition.
- void scall_ipc_exchange (void *arg)
  A SYSCALL_IPC_EXCHANGE handler.
- 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.
- void scall_user (void *arg)
  A SYSCALL_USER handler.

Variables

- syscall_t syscall_num = (syscall_t)0
  System call processing routine.
- void * syscall_arg = (void *)0

4.29.1 Function Documentation

4.29.1.1 SYSCALL_TABLE (syscall_routine[ ])

4.29.1.2 void do_syscall (void)

4.29.1.3 void scall_proc_init (void * *arg*)

A SYSCALL_PROC_INIT handler.

4.29.1.4 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*)

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.

4.29.1.5 void scall_proc_run (void * *arg*)

This function tries to launch a process by proc_run_isr call.

Parameters

arg A proc_runtime_arg_t pointer.

4.29.1.6 bool_t proc_run (proc_t * *proc*)

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.

4.29.1.7 void scall_proc_restart (void * *arg*)

This function tries to restart a process by proc_restart_isr call.

Parameters

arg A proc_runtime_arg_t pointer.

4.29.1.8 bool_t proc_restart (proc_t * *proc*)

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.

4.29.1.9 void scall_proc_stop (void * *arg*)

This function tries to stop a process by proc_stop_isr call.

Parameters

arg A proc_runtime_arg_t pointer.

4.29.1.10 bool_t proc_stop (proc_t * *proc*)

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.

4.29.1.11 void scall_proc_self_stop (void * *arg*)

This function stops calling process.

Parameters

arg Not used.

4.29.1.12 void proc_self_stop (void)

This function stops caller process.

4.29.1.13 void scall_proc_yeld (void * *arg*)

Transfers control to another process.

Parameters

arg Not used.

4.29.1.14 index_t proc_yeld (void)

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

Returns

Zero if there are no other running processes, none zero if there is at least one.

4.29.1.15 void scall_proc_terminate (void * *arg*)

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

Parameters

arg A pointer to a process.

4.29.1.16 void proc_terminate (void)

4.29.1.17 void scall_proc_flag_stop (void * *arg*)

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.

4.29.1.18 void proc_flag_stop (flag_t *mask*)

4.29.1.19 void scall_proc_reset_watchdog (void * *arg*)

This function calls _proc_reset_watchdog.

Parameters

arg Not used.

4.29.1.20 void proc_reset_watchdog (void)

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.

4.29.1.21 void scall_sig_init (void * *arg*)

Initiates a signal by sig_init_isr call.

Parameters

arg A pointer to a signal.

4.29.1.22 void sig_init (sig_t * *sig*)

Parameters

sig A sig_t pointer.

4.29.1.23 void scall_sig_wait (void * *arg*)

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

Parameters

arg A pointer to a signal.

4.29.1.24 void scall_sig_wakeup (void * *arg*)

4.29.1.25 void sig_wait (sig_t * *sig*)

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.

4.29.1.26 void scall_sig_signal (void * *arg*)

Wakes up one waiting process by sig_signal_isr call.

Warning

On a multicore system processes aren't woken up in a FIFO manner!

Parameters

arg A pointer to a signal.

4.29.1.27 void sig_signal (sig_t * *sig*)

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.

4.29.1.28 void scall_sig_broadcast (void * *arg*)

This function wakes up all waiting processes by sig_broadcast_isr call.

Parameters

arg A pointer to a signal.

4.29.1.29 void sig_broadcast (sig_t * *sig*)

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.

4.29.1.30 void scall_sem_init (void * *arg*)

This function initiates semaphore by sem_init_isr call.

Parameters

arg A pointer to a sem_init_arg_t structure.

4.29.1.31 void sem_init (sem_t * *sem*, count_t *count*)

Parameters

sem A sem_t pointer.

count A counter start value.

4.29.1.32 void scall_sem_lock (void * *arg*)

This function calls _sem_lock.

Parameters

arg A pointer to an sem_lock_arg_t object.

4.29.1.33 bool_t sem_lock (sem_t * *sem*)

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.

4.29.1.34 void scall_sem_try_lock (void * *arg*)

This function calls _sem_try_lock.

Parameters

arg A pointer to an sem_lock_arg_t object.

4.29.1.35 bool_t sem_try_lock (sem_t * *sem*)

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.

4.29.1.36 void scall_sem_unlock (void * *arg*)

This function calls sem_unlock_isr.

Parameters

arg A pointer to a semaphore.

4.29.1.37 void sem_unlock (sem_t * *sem*)

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.

4.29.1.38 void scall_mutex_init (void * *arg*)

This function initiater mutex by mutex_init_isr call.

Parameters

arg A poiner to an mutex_init_arg_t object.

4.29.1.39 void mutex_init (mutex_t * *mutex*, prio_t *prio*)

Parameters

mutex A mutex pointer.

prio A mutex priority. Used with CONFIG_USE_HIGHEST_LOCKER option.

4.29.1.40 void scall_mutex_lock (void * *arg*)

This function calls _mutex_lock.

Parameters

arg A pointer to an mutex_lock_arg_t object.

4.29.1.41 bool_t mutex_lock (mutex_t * *mutex*)

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

Parameters

mutex A mutex pointer.

Returns

1 if mutex was locked without wait, else 0.

4.29.1.42 void scall_mutex_try_lock (void * *arg*)

This function calls _mutex_try_lock.

Parameters

arg A mutex_lock_arg_t pointer.

4.29.1.43 bool_t mutex_try_lock (mutex_t * *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.

4.29.1.44 void scall_mutex_unlock (void * *arg*)

This function calls _mutex_unlock.

Parameters

arg A pointer to a mutex.

4.29.1.45 void mutex_unlock (mutex_t * *mutex*)

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

Parameters

mutex .

4.29.1.46 void scall_ipc_wait (void * *arg*)

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

Parameters

arg A pointer to storage for data to receive.

4.29.1.47 ipc_data_t ipc_wait (void)

Returns

IPC data.

4.29.1.48 void scall_ipc_send (void * *arg*)

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

Parameters

arg A ipc_send_arg_t pointer.

4.29.1.49 bool_t ipc_send (proc_t * *proc*, ipc_data_t *ipc_data*)

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.

4.29.1.50 void scall_ipc_exchange (void * *arg*)

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.

4.29.1.51 bool_t ipc_exchange (proc_t * *proc*, ipc_data_t *send*, ipc_data_t * *receive*)

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.

4.29.1.52 void scall_user (void * *arg*)

Calls user function.

Parameters

arg A pointer to a callee.

Warning

Be carefull! Callee pointer is not checked before call!

4.29.2 Variable Documentation

4.29.2.1 syscall_t syscall_num = (syscall_t)0

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

System call number.

4.29.2.2 void* syscall_arg = (void *)0

System call argument.

4.30 bugurtos/kernel/timer.c File Reference

```
#include "../include/bugurt.h"
```

Functions

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

4.30.1 Function Documentation

4.30.1.1 void _clear_timer (timer_t * *t*)

Parameters

t A pointer to a timer.

4.30.1.2 timer_t _timer (timer_t *t*)

Parameters

t A timer value.

4.30.1.3 void wait_time (timer_t *time*)

Caller process spins in a loop for a time.

Parameters

time Wait time.

4.31 bugurtos/kernel/xlist.c File Reference

```
#include "../include/bugurt.h"
```

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.

4.31.1 Function Documentation

4.31.1.1 void xlist_init (xlist_t * *xlist*)

Parameters

xlist An xlist_t pointer.

4.31.1.2 item_t* xlist_head (xlist_t * *xlist*)

Parameters

xlist An xlist_t pointer.

Returns

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

4.31.1.3 void xlist_switch (xlist_t * *xlist*, prio_t *prio*)

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

Parameters

xlist An xlist_t pointer.

prio A priority to switch.