BuguRTOS

0.8.2

Generated by Doxygen 1.8.1.2

Sat Apr 25 2015 15:53:56

Contents

1 Main Page

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

Warning

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

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 _cond_t Struct Reference

A conditional variable.

```
#include "libs/generic/cond.h"
```

Data Fields

- sync_t wait
- count_t blocked

4.1.1 Detailed Description

A conditional variable.

Conditional variables with mutexes are used for process-event synchronization. A process can block on conditional variable. Other process can launch one or all processes blocked on conditional variable.

4.1.2 Field Documentation

4.1.2.1 sync_t wait

A list of waiting processes.

4.1.2.2 count_t blocked

A list of blocked processes.

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

- cond.h

4.2 _ipc_t Struct Reference

An IPC endpoint.

```
#include "libs/generic/ipc.h"
```

Data Fields

- sync_t wait
- void ∗ msg

4.2.1 Detailed Description

An IPC endpoint.

Used for blocking synchronous or asynchronous IPC protocol implementation.

4.2.2 Field Documentation

4.2.2.1 sync_t wait

A list of waiting processes.

4.2.2.2 void∗ msg

A message buffer pointer.

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

- ipc.h

4.3 _item_t Struct Reference

A list item.

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

Data Fields

- item_t ∗ next
- item_t ∗ prev

4.3.1 Detailed Description

A list item.

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

4.3.2 Field Documentation

4.3.2.1 item_t∗ next

Next item in a list.

4.3.2.2 item_t∗ prev

Previous item in a list.

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

- item.h

4.4 _kernel_t Struct Reference

A BuguRTOS kernel structure.

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

Data Fields

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

4.4.1 Detailed Description

A BuguRTOS kernel structure.

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

4.4.2 Field Documentation

4.4.2.1 sched_t sched

The scheduler.

4.4.2.2 proc_t idle

The IDLE process.

4.4.2.3 timer_t timer

The system timer.

4.4.2.4 void(∗ timer_tick)(void)

The system timer tick hook pointer.

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

- kernel.h

4.5 _mutex_t Struct Reference

A mutex.

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

Data Fields

- sync_t wait

4.5.1 Detailed Description

A mutex.

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

Warning

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

4.5.2 Field Documentation

4.5.2.1 sync_t wait

A list of waiting processes.

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

- mutex.h

4.6 _pcounter_t Struct Reference

A locked resource counter.

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

Data Fields

- count_t counter [BITS_IN_INDEX_T]
- index_t index

4.6.1 Detailed Description

A locked resource counter.

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

4.6.2 Field Documentation

4.6.2.1 count_t counter[BITS_IN_INDEX_T]

A counter array.

4.6.2.2 index_t index

An index to speedup search.

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

- pcounter.h

4.7 _pitem_t Struct Reference

A prioritized list item.

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

Data Fields

- item_t parent
- xlist_t * list
- prio_t prio

4.7.1 Detailed Description

A prioritized list item.

4.7.2 Field Documentation

4.7.2.1 item_t parent

A perrent - item_t.

4.7.2.2 xlist_t* list

A pointer to an xlist_t object.

4.7.2.3 prio_t prio

A rpiority.

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

- pitem.h

4.8 _proc_t Struct Reference

A process.

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

Data Fields

- pitem_t parent
- flag_t flags
- prio_t base_prio
- pcounter_t lres
- timer_t time_quant
- timer_t timer
- struct _sync_t ∗ sync
- count_t cnt_lock
- code_t pmain
- code_t sv_hook
- code_t rs_hook
- void ∗ arg
- stack_t ∗ sstart
- stack_t ∗ spointer

4.8.1 Detailed Description

A process.

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

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

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

Warning

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

4.8.2 Field Documentation

4.8.2.1 pitem_t parent

A parent is pitem_t.

4.8.2.2 flag_t flags

Process state flags (to treat process state quickly).

4.8.2.3 prio_t base_prio

A base process priority.

4.8.2.4 pcounter_t lres

A locked resource counter.

4.8.2.5 timer_t time_quant

A process time slice.

4.8.2.6 timer_t timer

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

4.8.2.7 struct _sync_t* sync

4.8.2.8 count_t cnt_lock

A counter of proc_lock nesting.

4.8.2.9 code_t pmain

A pointer to a process "main" routine.

4.8.2.10 code_t sv_hook

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

4.8.2.11 code_t rs_hook

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

4.8.2.12 void* arg

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

4.8.2.13 stack_t* sstart

A process stack bottom pointer.

4.8.2.14 stack_t* spointer

A process stack top pointer.

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

- proc.h

4.9 _sched_t Struct Reference

A scheduler.

`#include "kernel/sched.h"`

Data Fields

- proc_t * current_proc
- xlist_t * ready
- xlist_t * expired

- xlist_t plst [2]
- count_t nested_crit_sec

4.9.1 Detailed Description

A scheduler.

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

4.9.2 Field Documentation

4.9.2.1 proc_t∗ current_proc

A currently running process.

4.9.2.2 xlist_t∗ ready

A pointer to a ready process list.

4.9.2.3 xlist_t∗ expired

Apointer to an expired process list.

4.9.2.4 xlist_t plst[2]

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

4.9.2.5 count_t nested_crit_sec

A critical section nesting count.

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

- sched.h

4.10 _sem_t Struct Reference

A counting semaphore.

```
#include "libs/generic/sem.h"
```

Data Fields

- sync_t wait
- count_t counter
- count_t blocked

4.10.1 Detailed Description

A counting semaphore.

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

4.10.2 Field Documentation

4.10.2.1 sync_t wait

A list of waiting processes.

4.10.2.2 count_t counter

A resource counter.

4.10.2.3 count_t blocked

A blocked process counter.

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

- sem.h

4.11 _sig_t Struct Reference

A signal.

```
#include "libs/generic/sig.h"
```

Data Fields

- cond_t wakeup
- mutex_t wait

4.11.1 Detailed Description

A signal.

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

4.11.2 Field Documentation

4.11.2.1 cond_t wakeup

Wakeup process list.

4.11.2.2 mutex_t wait

A list of waiting processes.

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

- sig.h

4.12 _sync_t Struct Reference

Basic synchronization primitive.

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

Data Fields

- xlist_t sleep
- proc_t ∗ owner
- count_t dirty
- prio_t prio

4.12.1 Detailed Description

Basic synchronization primitive.

A basic type that handles blocking process synchronization. By wrapping this type one can get traditional synchronization primitives (mutexes, semaphores, conditional variables, message-FIFOs, IPC-ednpoints, etc.).

Basic priority inheritanse protocol is supported.

4.12.2 Field Documentation

4.12.2.1 xlist_t sleep

A list of waiting processes.

4.12.2.2 proc_t∗ owner

A pointer to a process, that holds a sync.

4.12.2.3 count_t dirty

Dirty priority inheritanse transaction counter.

4.12.2.4 prio_t prio

Priority.

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

- sync.h

4.13 _xlist_t Struct Reference

A prioritized list.

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

Data Fields

- item_t ∗ item [BITS_IN_INDEX_T]
- index_t index

4.13.1 Detailed Description

A prioritized list.

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

4.13.2 Field Documentation

4.13.2.1 item_t∗ item[BITS_IN_INDEX_T]

An array of list head pointers.

4.13.2.2 index_t index

Index for fast search.

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

- xlist.h

4.14 proc_runtime_arg_t Struct Reference

An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

Data Fields

- proc_t ∗ proc
- bool_t ret

4.14.1 Detailed Description

An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

4.14.2 Field Documentation

4.14.2.1 proc_t∗ proc

A pointer to a process.

4.14.2.2 bool_t ret

A result storage.

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

- proc.c

4.15 proc_set_prio_arg_t Struct Reference

An argument for system call SYSCALL_PROC_SET_PRIO.

Data Fields

- proc_t ∗ proc
- prio_t prio

4.15.1 Detailed Description

An argument for system call SYSCALL_PROC_SET_PRIO.

4.15.2 Field Documentation

4.15.2.1 proc_t∗ proc

A pointer to a process.

4.15.2.2 prio_t prio

Priority.

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

- sync.c

4.16 scall_user_t Struct Reference

Data Fields

- code_t func
- void ∗ arg

4.16.1 Field Documentation

4.16.1.1 code_t func

4.16.1.2 void∗ arg

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

- syscall.c

4.17 sync_proc_timeout_t Struct Reference

Data Fields

- proc_t ∗ proc
- flag_t status

4.17.1 Field Documentation

4.17.1.1 proc_t∗ proc

4.17.1.2 flag_t status

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

- sync.c

4.18 sync_set_owner_t Struct Reference

Data Fields

- sync_t ∗ sync
- proc_t ∗ proc
- flag_t status

4.18.1 Field Documentation

4.18.1.1 sync_t∗ sync

4.18.1.2 proc_t∗ proc

4.18.1.3 flag_t status

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

- sync.c

4.19 sync_sleep_t Struct Reference

For internal usage.

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

Data Fields

- sync_t ∗ sync
- flag_t status

4.19.1 Detailed Description

For internal usage.

4.19.2 Field Documentation

4.19.2.1 sync_t∗ sync

A sync_t object pointer.

4.19.2.2 flag_t status

Execution status.

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

- sync.h

4.20 sync_wait_t Struct Reference

For internal usage.

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

Data Fields

- sync_t ∗ sync
- proc_t ∗∗ proc
- flag_t block
- flag_t status

4.20.1 Detailed Description

For internal usage.

4.20.2 Field Documentation

4.20.2.1 sync_t* sync

A sync_t object pointer.

4.20.2.2 proc_t** proc

A process buffer pointer.

4.20.2.3 flag_t block

A block flag.

4.20.2.4 flag_t status

Execution status.

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

- sync.h

4.21 sync_wake_and_sleep_t Struct Reference

For internal usage.

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

Data Fields

- sync_sleep_t sleep
- sync_t * wake
- proc_t * proc
- flag_t chown
- flag_t stage

4.21.1 Detailed Description

For internal usage.

4.21.2 Field Documentation

4.21.2.1 sync_sleep_t sleep

Parameters for 2nd stage of the call.

4.21.2.2 sync_t* wake

A sync_t object pointer for 1st stage of the call.

4.21.2.3 proc_t* proc

A process pointer for 1st stage of the call.

4.21.2.4 flag_t chown

A change owner flag.

4.21.2.5 flag_t stage

A stage number.

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

- sync.h

4.22 sync_wake_and_wait_t Struct Reference

For internal usage.

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

Data Fields

- sync_wait_t wait
- sync_t ∗ wake
- proc_t ∗ proc
- flag_t chown
- flag_t stage

4.22.1 Detailed Description

For internal usage.

4.22.2 Field Documentation

4.22.2.1 sync_wait_t wait

Parameters for first stage of the call.

4.22.2.2 sync_t∗ wake

A sync_t object pointer for 1st stage of the call.

4.22.2.3 proc_t∗ proc

A process pointer for 1st stage of the call.

4.22.2.4 flag_t chown

A change owner flag.

4.22.2.5 flag_t stage

A stage number.

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

- sync.h

4.23 sync_wake_t Struct Reference

For internal usage.

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

Data Fields

- sync_t * sync
- proc_t * proc
- flag_t chown
- flag_t status

4.23.1 Detailed Description

For internal usage.

4.23.2 Field Documentation

4.23.2.1 sync_t* sync

A sync_t object pointer.

4.23.2.2 proc_t* proc

A process pointer.

4.23.2.3 flag_t chown

A change owner flag.

4.23.2.4 flag_t status

Execution status.

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

- sync.h

5 File Documentation

5.1 bugurt.h File Reference

The top header file.

```
#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 "sync.h"
#include "timer.h"
#include "syscall.h"
```

Macros

- #define SPIN_INIT(arg)

  Wrapper macro.
- #define SPIN_LOCK(arg)

  Wrapper macro.
- #define SPIN_FREE(arg)

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

  Wrapper macro.

Typedefs

- typedef void(∗ code_t)(void ∗)

  Executable code.

Functions

- void resched (void)

  Rescheduling.
- void disable_interrupts (void)

  Interrupt disable.
- void enable_interrupts (void)

  Interrupt enable.
- proc_t ∗ current_proc (void)

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

  A process stack initialization.
- void init_bugurt (void)

  The Kernel initiation.
- void start_bugurt (void)

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

  A system call.

5.1.1 Detailed Description

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

5.1.2 Macro Definition Documentation

5.1.2.1 #define SPIN_INIT(*arg*)

Wrapper macro.

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

5.1.2.2 #define SPIN_LOCK(*arg*)

Wrapper macro.

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

5.1.2.3 #define SPIN_FREE(*arg*)

Wrapper macro.

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

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

Wrapper macro.

A wrapper for resched function.

5.1.3 Typedef Documentation

5.1.3.1 typedef void(∗ code_t)(void ∗)

Executable code.

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

5.1.4 Function Documentation

5.1.4.1 void resched (void)

Rescheduling.

Launces a reschedule sequence.

5.1.4.2 void disable_interrupts (void)

Interrupt disable.

Disables interrupts globally.

5.1.4.3 void enable_interrupts (void)

Interrupt enable.

Enables interrupts globally.

5.1.4.4 proc_t∗ current_proc (void)

Current process.

Current process.

Returns

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

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

A process stack initialization.

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

Parameters

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

Returns

a pointer to a prepared process stack top.

5.1.4.6 void init_bugurt (void)

The Kernel initiation.

Initiates the Kernel before the OSstart.

5.1.4.7 void start_bugurt (void)

The OSstart.

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

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

A system call.

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

Parameters

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

5.2 cond.c File Reference

```
#include "cond.h"
```

Functions

- void cond_init_isr (cond_t ∗cond)

  A conditional variable initiation from ISR or critical section.
- void cond_init (cond_t ∗cond)

  A conditional variable initiation.
- flag_t cond_wait (cond_t ∗cond, mutex_t ∗mutex)

  Wait for a condition.
- flag_t cond_signal (cond_t ∗cond)

  Launch one waiting process.
- flag_t cond_broadcast (cond_t ∗cond)

  Launch all waiting processes.

5.2.1 Function Documentation

5.2.1.1 void cond_init_isr (**cond_t** ∗ *cond*)

A conditional variable initiation from ISR or critical section.

Parameters

cond	A cond_t pointer.

5.2.1.2 void cond_init (**cond_t** ∗ *cond*)

A conditional variable initiation.

Parameters

cond	A cond_t pointer.

5.2.1.3 flag_t cond_wait (**cond_t** ∗ *cond*, **mutex_t** ∗ *mutex*)

Wait for a condition.

This function stops caller process and inserts it to conditional variable wait list.

Parameters

cond	A cond_t pointer.
mutex	A pointer to a mutex which protects a conditional variable.

Returns

SYNC_ST_OK on success, or error number.

5.2.1.4 flag_t cond_signal (**cond_t** ∗ *cond*)

Launch one waiting process.

Launches the head of waiting process list.

Warning

Caller must lock mutex first!

Parameters

cond	A cond_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.2.1.5 flag_t cond_broadcast (**cond_t** ∗ *cond*)

Launch all waiting processes.

Launches all processes from waiting process list.

Warning

Caller must lock mutex first!

Parameters

cond	A cond_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.3 cond.h File Reference

A conditional variable header.

```
#include <bugurt.h>
#include "mutex.h"
```

Data Structures

- struct _cond_t

    A conditional variable.

Typedefs

- typedef struct _cond_t cond_t

Functions

- void cond_init_isr (cond_t ∗cond)

    A conditional variable initiation from ISR or critical section.
- void cond_init (cond_t ∗cond)

    A conditional variable initiation.
- flag_t cond_wait (cond_t ∗cond, mutex_t ∗mutex)

    Wait for a condition.
- flag_t cond_signal (cond_t ∗cond)

    Launch one waiting process.
- flag_t cond_broadcast (cond_t ∗cond)

    Launch all waiting processes.

5.3.1 Detailed Description

A conditional variable header.

5.3.2 Typedef Documentation

5.3.2.1 typedef struct _cond_t cond_t

See _cond_t;

5.3.3 Function Documentation

5.3.3.1 void cond_init_isr (cond_t ∗ *cond*)

A conditional variable initiation from ISR or critical section.

Parameters

cond	A cond_t pointer.

5.3.3.2 void cond_init (cond_t ∗ *cond*)

A conditional variable initiation.

Parameters

cond	A cond_t pointer.

5.3.3.3 flag_t cond_wait (cond_t ∗ *cond,* mutex_t ∗ *mutex*)

Wait for a condition.

This function stops caller process and inserts it to conditional variable wait list.

Parameters

cond	A cond_t pointer.
mutex	A pointer to a mutex which protects a conditional variable.

Returns

SYNC_ST_OK on success, or error number.

5.3.3.4 flag_t cond_signal (cond_t ∗ *cond*)

Launch one waiting process.

Launches the head of waiting process list.

Warning

Caller must lock mutex first!

Parameters

cond	A cond_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.3.3.5 flag_t cond_broadcast (cond_t ∗ *cond*)

Launch all waiting processes.

Launches all processes from waiting process list.

Warning

Caller must lock mutex first!

Parameters

cond	A cond_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.4 crit_sec.c File Reference

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

Functions

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

5.4.1 Function Documentation

5.4.1.1 void enter_crit_sec (void)

A critical section start.

5.4.1.2 void exit_crit_sec (void)

A critical section end.

5.5 crit_sec.h File Reference

A critical section header.

Macros

- #define ENTER_CRIT_SEC() enter_crit_sec()

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

  A wraper macro.

Functions

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

5.5.1 Detailed Description

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

5.5.2 Macro Definition Documentation

5.5.2.1 #define ENTER_CRIT_SEC( ) enter_crit_sec()

A wraper macro.

A critical section start.

Warning

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

5.5.2.2 #define EXIT_CRIT_SEC( ) exit_crit_sec()

A wraper macro.

A critical section end.

Warning

Must be used at the end of a code block.

5.5.3 Function Documentation

5.5.3.1 void enter_crit_sec (void)

A critical section start.

5.5.3.2 void exit_crit_sec (void)

A critical section end.

5.6 index.c File Reference

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

Functions

- prio_t index_search (index_t index)

5.6.1 Function Documentation

5.6.1.1 prio_t index_search (index_t *index*)

An index search.

Parameters

index	An index.

Returns

Highest priority of an index (with minimal value).

5.7 index.h File Reference

An index search header.

Functions

- prio_t index_search (index_t index)

5.7.1 Detailed Description

An index search header.

5.7.2 Function Documentation

5.7.2.1 prio_t index_search (index_t *index*)

An index search.

Parameters

index	An index.

Returns

Highest priority of an index (with minimal value).

5.8 ipc.c File Reference

```
#include "ipc.h"
```

Functions

- void ipc_init_isr (ipc_t ∗endpoint)
  IPC endpoint initiation from ISR or critical section.
- void ipc_init (ipc_t ∗endpoint)
  IPC endpoint initiation.
- flag_t ipc_send (ipc_t ∗out, void ∗msg)
  IPCdata transmition.
- flag_t ipc_wait (ipc_t ∗in, proc_t ∗∗proc, flag_t block)
  Wait for IPC.
- flag_t ipc_reply (ipc_t ∗in, proc_t ∗proc)
  Unblock a sender process, which message has been received.

5.8.1 Function Documentation

5.8.1.1 void ipc_init_isr (ipc_t ∗ *endpoint*)

IPC endpoint initiation from ISR or critical section.

Parameters

endpoint	A pointer to the endpoint.

5.8.1.2 void ipc_init (ipc_t ∗ *endpoint*)

IPC endpoint initiation.

Parameters

endpoint	A pointer to the endpoint.

5.8.1.3 flag_t ipc_send (ipc_t ∗ *out*, void ∗ *msg*)

IPCdata transmition.

This function transfers a pinter to the message buffer throuth IPC. Senders are blocked on IPC endpoint and wait for their turn, receiver inherits senders priorities.

Parameters

out	An IPC endpoint pointer.
msg	A message buffer pointer.

Returns

SYNC_ST_OK on success, or error number.

5.8.1.4 flag_t ipc_wait (ipc_t ∗ *in*, proc_t ∗∗ *proc*, flag_t *block*)

Wait for IPC.

A buffer must be used to set or get sender process. A buffer pointer must be passed as a second parameter.

Parameters

in	An IPC endpoint pointer.
proc	A double pointer to the process which is supposed to send a message (if ∗proc == 0 then every message is received).
block	A caller block flag. If non zero, then caller is blocked untill message is sent.

Returns

SYNC_ST_OK on success, or error number.

5.8.1.5 flag_t ipc_reply (ipc_t ∗ *in*, proc_t ∗ *proc*)

Unblock a sender process, which message has been received.

Parameters

in	An IPC endpoint pointer.
proc	A sender process pointer.

Returns

SYNC_ST_OK on success, or error number.

5.9 ipc.h File Reference

An IPC header.

```
#include <bugurt.h>
```

Data Structures

- struct _ipc_t

    An IPC endpoint.

Typedefs

- typedef struct _ipc_t ipc_t

Functions

- void ipc_init_isr (ipc_t ∗endpoint)

    IPC endpoint initiation from ISR or critical section.
- void ipc_init (ipc_t ∗endpoint)

    IPC endpoint initiation.
- flag_t ipc_send (ipc_t ∗out, void ∗msg)

    IPCdata transmition.
- flag_t ipc_wait (ipc_t ∗in, proc_t ∗∗proc, flag_t block)

    Wait for IPC.
- flag_t ipc_reply (ipc_t ∗in, proc_t ∗proc)

    Unblock a sender process, which message has been received.

5.9.1 Detailed Description

An IPC header.

5.9.2 Typedef Documentation

5.9.2.1 typedef struct _ipc_t ipc_t

See _ipc_t;

5.9.3 Function Documentation

5.9.3.1 void ipc_init_isr (ipc_t ∗ *endpoint*)

IPC endpoint initiation from ISR or critical section.

Parameters

endpoint	A pointer to the endpoint.

5.9.3.2 void ipc_init (ipc_t ∗ *endpoint*)

IPC endpoint initiation.

Parameters

endpoint	A pointer to the endpoint.

5.9.3.3 flag_t ipc_send (ipc_t ∗ *out*, void ∗ *msg*)

IPCdata transmition.

This function transfers a pinter to the message buffer throuth IPC. Senders are blocked on IPC endpoint and wait for their turn, receiver inherits senders priorities.

Parameters

out	An IPC endpoint pointer.
msg	A message buffer pointer.

Returns

SYNC_ST_OK on success, or error number.

5.9.3.4 flag_t ipc_wait (ipc_t ∗ *in*, proc_t ∗∗ *proc*, flag_t *block*)

Wait for IPC.

A buffer must be used to set or get sender process. A buffer pointer must be passed as a second parameter.

Parameters

in	An IPC endpoint pointer.
proc	A double pointer to the process which is supposed to send a message (if ∗proc == 0 then every message is received).
block	A caller block flag. If non zero, then caller is blocked untill message is sent.

Returns

SYNC_ST_OK on success, or error number.

5.9.3.5 flag_t ipc_reply (ipc_t ∗ *in*, proc_t ∗ *proc*)

Unblock a sender process, which message has been received.

Parameters

in	An IPC endpoint pointer.
proc	A sender process pointer.

Returns

SYNC_ST_OK on success, or error number.

5.10 item.c File Reference

```
#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.

5.10.1 Function Documentation

5.10.1.1 void item_init (item_t ∗ *item*)

An item_t object initiation.

Parameters

item	An item_t pointer.

5.10.1.2 void item_insert (item_t ∗ *item,* item_t ∗ *head*)

Insert an item to a list.

Parameters

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

5.10.1.3 void item_cut (item_t ∗ *item*)

Cut an item from a list.

Parameters

item	A pointer to an item to cut.

5.11 item.h File Reference

A list item header.

Data Structures

- struct _item_t

  A list item.

Macros

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

Typedefs

- typedef struct _item_t item_t

Functions

- void item_init (item_t ∗item)

An item_t object initiation.

- void item_insert (item_t ∗item, item_t ∗head)

  Insert an item to a list.

- void item_cut (item_t ∗item)

  Cut an item from a list.

5.11.1 Detailed Description

A list item header.

5.11.2 Macro Definition Documentation

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

Static item initiation.

Parameters

a	An item_t variable name.

5.11.3 Typedef Documentation

5.11.3.1 typedef struct _item_t item_t

See _item_t;

5.11.4 Function Documentation

5.11.4.1 void item_init (item_t ∗ *item*)

An item_t object initiation.

Parameters

item	An item_t pointer.

5.11.4.2 void item_insert (item_t ∗ *item,* item_t ∗ *head*)

Insert an item to a list.

Parameters

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

5.11.4.3 void item_cut (item_t ∗ *item*)

Cut an item from a list.

Parameters

item	A pointer to an item to cut.

5.12 kernel.c File Reference

```
#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.

5.12.1 Function Documentation

5.12.1.1 WEAK void idle_main (void ∗ *arg*)

An IDLE process main function.

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

Warning

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

Parameters

arg	An argument pointer.

5.12.1.2 void kernel_init (void)

The kernel initiation.

This function prepares the kernel to work.

5.12.2 Variable Documentation

5.12.2.1 kernel_t kernel

The BuguRTOSkernel.

It's the one for the entire system!

5.13 kernel.h File Reference

A kernel header.

Data Structures

- struct _kernel_t

A BuguRTOS kernel structure.

Typedefs

- typedef struct _kernel_t kernel_t

Functions

- void kernel_init (void)

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

  An IDLE process main function.

Variables

- kernel_t kernel

  The BuguRTOSkernel.

5.13.1 Detailed Description

A kernel header.

5.13.2 Typedef Documentation

5.13.2.1 typedef struct _kernel_t kernel_t

See _kernel_t;

5.13.3 Function Documentation

5.13.3.1 void kernel_init (void)

The kernel initiation.

This function prepares the kernel to work.

5.13.3.2 void idle_main (void ∗ *arg*)

An IDLE process main function.

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

Warning

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

Parameters

arg	An argument pointer.

5.13.4 Variable Documentation

5.13.4.1 kernel_t kernel

The BuguRTOSkernel.

It's the one for the entire system!

5.14 mutex.c File Reference

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

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.
- flag_t mutex_try_lock (mutex_t ∗mutex)

  Try to lock a mutex.
- flag_t mutex_lock (mutex_t ∗mutex)

  Lock a mutex.
- flag_t mutex_free (mutex_t ∗mutex)

  Mutex free.

5.14.1 Function Documentation

5.14.1.1 void mutex_init_isr (mutex_t ∗ *mutex*, prio_t *prio*)

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

Parameters

mutex	A mutex pointer.
prio	A mutex priority.

5.14.1.2 void mutex_init (mutex_t ∗ *mutex*, prio_t *prio*)

A mutex initiation.

Parameters

mutex	A mutex pointer.
prio	A mutex priority.

5.14.1.3 flag_t mutex_try_lock (mutex_t ∗ *mutex*)

Try to lock a mutex.

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

Parameters

mutex	A mutex pointer.

Returns

SYNC_ST_OK - if mutex was succefully locked else - SYNC_ST_ROLL.

5.14.1.4 flag_t mutex_lock (mutex_t * *mutex*)

Lock a mutex.

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

Parameters

mutex	A mutex pointer.

Returns

SYNC_ST_OK on success, or error number.

5.14.1.5 flag_t mutex_free (mutex_t * *mutex*)

Mutex free.

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

Parameters

mutex	A mutex pointer.

Returns

SYNC_ST_OK on success, or error number.

5.15 mutex.h File Reference

A mutex header.

```
#include <bugurt.h>
```

Data Structures

- struct _mutex_t

  A mutex.

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.
- flag_t mutex_try_lock (mutex_t *mutex)

Try to lock a mutex.
- flag_t mutex_lock (mutex_t ∗mutex)

  Lock a mutex.
- flag_t mutex_free (mutex_t ∗mutex)

  Mutex free.

5.15.1 Detailed Description

A mutex header.

5.15.2 Typedef Documentation

5.15.2.1 typedef struct _mutex_t mutex_t

See _mutex_t;

5.15.3 Function Documentation

5.15.3.1 void mutex_init_isr (mutex_t ∗ *mutex,* prio_t *prio*)

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

Parameters

mutex	A mutex pointer.
prio	A mutex priority.

5.15.3.2 void mutex_init (mutex_t ∗ *mutex,* prio_t *prio*)

A mutex initiation.

Parameters

mutex	A mutex pointer.
prio	A mutex priority.

5.15.3.3 flag_t mutex_try_lock (mutex_t ∗ *mutex*)

Try to lock a mutex.

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

Parameters

mutex	A mutex pointer.

Returns

SYNC_ST_OK - if mutex was succefully locked else - SYNC_ST_ROLL.

5.15.3.4 flag_t mutex_lock (mutex_t ∗ *mutex*)

Lock a mutex.

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

Parameters

mutex	A mutex pointer.

Returns

SYNC_ST_OK on success, or error number.

5.15.3.5 flag_t mutex_free (mutex_t ∗ *mutex*)

Mutex free.

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

Parameters

mutex	A mutex pointer.

Returns

SYNC_ST_OK on success, or error number.

5.16 pcounter.c File Reference

```
#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;.

5.16.1 Function Documentation

5.16.1.1 void pcounter_init (pcounter_t ∗ *pcounter*)

A pcounter_t object initiation.

Parameters

pcounter	A pcounter_t pointer.

5.16.1.2 void pcounter_inc (pcounter_t ∗ *pcounter,* prio_t *prio*)

Increment counter.

Parameters

pcounter	A pcounter_t pointer.
prio	A priority.

5.16.1.3 index_t pcounter_dec (pcounter_t ∗ *pcounter*, prio_t *prio*)

Decrement counter.

Parameters

pcounter	A pcounter_t pointer.
prio	A priority.

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

Increase counter by a number of steps.

Parameters

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

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

Decrease counter by a number of steps;.

Parameters

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

Returns

0 if correspondent counter is nulled, not 0 else.

5.17 pcounter.h File Reference

A locked resource counter header.

Data Structures

- struct _pcounter_t

  A locked resource counter.

Typedefs

- typedef struct _pcounter_t pcounter_t

Functions

- void pcounter_init (pcounter_t ∗pcounter)

  A pcounter_t object initiation.

- void pcounter_inc (pcounter_t ∗pcounter, prio_t prio)

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

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

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

    Decrease counter by a number of steps;.

5.17.1 Detailed Description

A locked resource counter header.

5.17.2 Typedef Documentation

5.17.2.1 typedef struct _pcounter_t pcounter_t

See _pcounter_t;

5.17.3 Function Documentation

5.17.3.1 void pcounter_init (pcounter_t ∗ *pcounter*)

A pcounter_t object initiation.

Parameters

pcounter	A pcounter_t pointer.

5.17.3.2 void pcounter_inc (pcounter_t ∗ *pcounter,* prio_t *prio*)

Increment counter.

Parameters

pcounter	A pcounter_t pointer.
prio	A priority.

5.17.3.3 index_t pcounter_dec (pcounter_t ∗ *pcounter,* prio_t *prio*)

Decrement counter.

Parameters

pcounter	A pcounter_t pointer.
prio	A priority.

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

Increase counter by a number of steps.

Parameters

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

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

Decrease counter by a number of steps;.

Parameters

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

Returns

0 if correspondent counter is nulled, not 0 else.

5.18 pitem.c File Reference

```
#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.

5.18.1 Function Documentation

5.18.1.1 void pitem_init (pitem_t ∗ *pitem,* prio_t *prio*)

A pitem_t object initiation.

Parameters

pitem	A pitem_t pointer.
prio	A priority.

5.18.1.2 void pitem_insert (pitem_t ∗ *pitem,* xlist_t ∗ *xlist*)

Insert pitem_t object to xlist_t container.

Parameters

pitem	A pitem_t pointer.
xlist	A pointer to destignation list.

5.18.1.3 void pitem_fast_cut (pitem_t ∗ *pitem*)

Fast cut pitem_t object from xlist_t container.

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

Parameters

pitem	A pitem_t pointer.

5.18.1.4 void pitem_cut (pitem_t ∗ *pitem*)

Cut pitem_t object from xlist_t container.

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

Parameters

pitem	A pitem_t pointer.

5.18.1.5 pitem_t∗ pitem_xlist_chain (xlist_t ∗ *src*)

"Chain" pitem_t objects from xlist_t container.

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.

5.19 pitem.h File Reference

A prioritixed lis item header.

Data Structures

- struct _pitem_t

  A prioritized list item.

Macros

- #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.

5.19.1 Detailed Description

A prioritixed lis item header.

5.19.2 Macro Definition Documentation

5.19.2.1 #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.

5.19.3 Typedef Documentation

5.19.3.1 typedef struct _pitem_t pitem_t

5.19.4 Function Documentation

5.19.4.1 void pitem_init (pitem_t ∗ *pitem,* prio_t *prio*)

A pitem_t object initiation.

Parameters

pitem	A pitem_t pointer.
prio	A priority.

5.19.4.2 void pitem_insert (pitem_t ∗ *pitem,* xlist_t ∗ *xlist*)

Insert pitem_t object to xlist_t container.

Parameters

pitem	A pitem_t pointer.
xlist	A pointer to designation list.

5.19.4.3 void pitem_fast_cut (pitem_t ∗ *pitem*)

Fast cut pitem_t object from xlist_t container.

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

Parameters

pitem	A pitem_t pointer.

5.19.4.4 void pitem_cut (pitem_t ∗ *pitem*)

Cut pitem_t object from xlist_t container.

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

Parameters

pitem	A pitem_t pointer.

5.19.4.5 pitem_t∗ pitem_xlist_chain (xlist_t ∗ *src*)

"Chain" pitem_t objects from xlist_t container.

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.

5.20 proc.c File Reference

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

Data Structures

- struct proc_runtime_arg_t

  An argument for system calls SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

Functions

- void _proc_stop_ensure (proc_t ∗proc)

  Stops a process. For internel 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_prio_control_stoped (proc_t ∗proc)

  A stopedprocess priority control routine.
- 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_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.
- 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.
- void scall_proc_run (void ∗arg)

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

  A SYSCALL_PROC_RESTART handler.
- 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 scall_proc_stop (void ∗arg)

  A SYSCALL_PROC_STOP handler.
- void proc_lock (void)

  Set PROC_FLG_LOCK for caller process.
- void _proc_lock (void)

  Set PROC_FLG_LOCK for caller process.
- void scall_proc_lock (void ∗arg)

  A SYSCALL_PROC_LOCK handler.
- void proc_free (void)

  A PROC_FLG_PRE_STOP flag processing routine.
- void _proc_free (void)

  A PROC_FLG_PRE_STOP flag processing routine. For internal usage.
- void scall_proc_free (void ∗arg)

  A SYSCALL_PROC_FREE handler.
- void proc_self_stop (void)

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

  A process self stop routine (for internal usage only!).
- void scall_proc_self_stop (void ∗arg)

  A SYSCALL_PROC_SELF_STOP handler.
- 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.
- void scall_proc_terminate (void ∗arg)

  A SYSCALL_PROC_TERMINATE handler.
- 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 scall_proc_reset_watchdog (void ∗arg)

  A SYSCALL_PROC_RESET_WATCHDOG handler.

5.20.1 Function Documentation

5.20.1.1 void _proc_stop_ensure (proc_t ∗ *proc*)

Stops a process. For internel usage.

Stops aprocess for sure.

Parameters

proc	- A pointer to a process.

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

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

5.20.1.3 void _proc_prio_control_stoped (proc_t * *proc*)

A stopedprocess priority control routine.

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

Parameters

proc	- A pointer to a process.

5.20.1.4 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*)

A process initialization.

Parameters

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

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

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

Parameters

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

5.20.1.6 bool_t proc_run (proc_t * *proc*)

A process launch routine.

This function schedules a process if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.20.1.7 bool_t proc_run_isr (proc_t ∗ *proc*)

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

This function schedules a process if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.20.1.8 void scall_proc_run (void ∗ *arg*)

A SYSCALL_PROC_RUN handler.

This function tries to launch a process by proc_run_isr call.

Parameters

arg	A proc_runtime_arg_t pointer.

5.20.1.9 bool_t proc_restart (proc_t ∗ *proc*)

Aprocess restart routine.

This function reinitializes a process and schedules it if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.20.1.10 bool_t proc_restart_isr (proc_t ∗ *proc*)

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

This function reinitializes a process and schedules it if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.20.1.11 void scall_proc_restart (void ∗ *arg*)

A SYSCALL_PROC_RESTART handler.

This function tries to restart a process by proc_restart_isr call.

Parameters

arg	A proc_runtime_arg_t pointer.

5.20.1.12 bool_t proc_stop (proc_t ∗ *proc*)

A process stop routine.

This function stops a process if possible.

Parameters

proc	- A pointer to a process to stop.

Returns

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

5.20.1.13 bool_t proc_stop_isr (proc_t ∗ *proc*)

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

This function stops a process if possible.

Parameters

proc	- A pointer to a process to stop.

Returns

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

5.20.1.14 void scall_proc_stop (void ∗ *arg*)

A SYSCALL_PROC_STOP handler.

This function tries to stop a process by proc_stop_isr call.

Parameters

arg	A proc_runtime_arg_t pointer.

5.20.1.15 void proc_lock (void)

Set PROC_FLG_LOCK for caller process.

5.20.1.16 void _proc_lock (void)

Set PROC_FLG_LOCK for caller process.

5.20.1.17 void scall_proc_lock (void ∗ *arg*)

A SYSCALL_PROC_LOCK handler.

Sets #PROC_FLG_NONSTOP for caller process, increases proc->lres counter.

5.20.1.18 void proc_free (void)

A PROC_FLG_PRE_STOP flag processing routine.

5.20.1.19 void _proc_free (void)

A PROC_FLG_PRE_STOP flag processing routine. For internal usage.

5.20.1.20 void scall_proc_free (void * *arg*)

A SYSCALL_PROC_FREE handler.

This function decreases proc->lres counter, clears PROC_FLG_LOCK if needed and, process PROC_FLG_PRE-_STOP of the calling process and clears masked flags of a calling process. It calls _proc_free.

Parameters

arg	A poointer to a flag mask.

5.20.1.21 void proc_self_stop (void)

A process self stop routine.

This function stops caller process.

5.20.1.22 void _proc_self_stop (void)

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

This function stops caller process.

5.20.1.23 void scall_proc_self_stop (void * *arg*)

A SYSCALL_PROC_SELF_STOP handler.

This function stops calling process.

Parameters

arg	Not used.

5.20.1.24 void proc_terminate (void)

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

5.20.1.25 void _proc_terminate (void)

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

5.20.1.26 void scall_proc_terminate (void * *arg*)

A SYSCALL_PROC_TERMINATE handler.

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

Parameters

arg	A pointer to a process.

5.20.1.27 void proc_reset_watchdog (void)

A watchdog reset routine for real time processes.

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

5.20.1.28 void _proc_reset_watchdog (void)

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

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

5.20.1.29 void scall_proc_reset_watchdog (void ∗ *arg*)

A SYSCALL_PROC_RESET_WATCHDOG handler.

This function calls _proc_reset_watchdog.

Parameters

arg	Not used.

5.21 proc.h File Reference

A process header.

Data Structures

- struct _proc_t

  A process.

Macros

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

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

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

  Wrapper macro.
- #define PROC_FLG_RT ((flag_t)0x80)

  A real time flag.
- #define PROC_FLG_RR ((flag_t)0x40)
- #define PROC_FLG_LOCK ((flag_t)0x20)

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

  A proces stop preparation flag.
- #define PROC_FLG_LOCK_MASK ((flag_t)(PROC_FLG_LOCK))

  A PROC_FLG_LOCK.
- #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

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

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

  An execution state mask.
- #define PROC_STATE_RESTART_MASK ((flag_t)0x8)

  A process execution state check mask.
- #define PROC_STATE_RUN_MASK ((flag_t)0x3)

  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)

  Initial state, stoped.
- #define PROC_STATE_END ((flag_t)0x1)

  Normal process termination.
- #define PROC_STATE_READY ((flag_t)0x2)

  Is ready to run.
- #define PROC_STATE_RUNNING ((flag_t)0x3)

  Is running.
- #define PROC_STATE_WD_STOPED ((flag_t)0x4)

  Watchdog termination.
- #define PROC_STATE_DEAD ((flag_t)0x5)

  Abnormal termination, terminated with waiting ipc transactions.
- #define PROC_STATE_TO_READY ((flag_t)0x6)

  Is ready to run.
- #define PROC_STATE_TO_RUNNING ((flag_t)0x7)

  Is running.
- #define PROC_STATE_SYNC_WAIT ((flag_t)0x8)

  Is waiting for sleaping processes.
- #define PROC_STATE_SYNC_SLEEP ((flag_t)0x9)

  Is waiting for wakeup.
- #define PROC_STATE_SYNC_READY ((flag_t)0xA)

  Is ready to run.
- #define PROC_STATE_SYNC_RUNNING ((flag_t)0xB)

  Is running.
- #define PROC_STATE_PI_PEND ((flag_t)0xC)

  A process is waiting for priority change.
- #define PROC_STATE_PI_DONE ((flag_t)0xD)

  A process has been run during priority change.
- #define PROC_STATE_PI_READY ((flag_t)0xE)

  Is ready to run.
- #define PROC_STATE_PI_RUNNING ((flag_t)0xF)

  Is running.
- #define PROC_PRE_STOP_TEST(a) ((a->flags & PROC_FLG_PRE_STOP) && (!(a->flags & PROC_F-LG_LOCK_MASK)))

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

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

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

  Sets process state.
- #define PROC_PRIO_LOWEST ((prio_t)BITS_IN_INDEX_T - (prio_t)1)

  Lowest priority level.

Typedefs

- typedef struct _proc_t proc_t

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  A watchdog reset routine for real time processes for internal usage.
- void _proc_prio_propagate (proc_t ∗proc)

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

  A low level process stop with flags set routine. For internal usage.
- void _proc_lock (void)

  Set PROC_FLG_LOCK for caller process.
- void proc_lock (void)

  Set PROC_FLG_LOCK for caller process.
- void _proc_free (void)

  A PROC_FLG_PRE_STOP flag processing routine. For internal usage.
- void proc_free (void)

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

  A stopedprocess priority control routine.
- void proc_set_prio (proc_t ∗proc, prio_t prio)

  Set a priotity of a process.
- void _proc_set_prio (proc_t ∗proc, prio_t prio)

Set a priotity of a process. For internel usage.

- void _proc_lres_inc (proc_t ∗proc, prio_t prio)

  Process priority control. For internel usage.
- void _proc_lres_dec (proc_t ∗proc, prio_t prio)

  Process priority control. For internel usage.
- void _proc_stop_ensure (proc_t ∗proc)

  Stops a process. For internel usage.

5.21.1 Detailed Description

A process header.

5.21.2 Macro Definition Documentation

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

Wrapper macro.

Initiates proc->lres field of a process.

Parameters

a	a pointer to a process.

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

Wrapper macro.

An increment of locked mutex counter field of a process.

Parameters

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

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

Wrapper macro.

A decrement of locked mutex counter field of a process.

Parameters

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

5.21.2.4 #define PROC_FLG_RT ((flag_t)0x80)

A real time flag.

This flag enables real time process scheduling policy.

5.21.2.5 #define PROC_FLG_RR ((flag_t)0x40)

5.21.2.6 #define PROC_FLG_LOCK ((flag_t)0x20)

A mutex lock flag.

A process has locked some mutex controled resources.

5.21.2.7 #define PROC_FLG_PRE_STOP ((flag_t)0x10)

A proces stop preparation flag.

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

5.21.2.8 #define PROC_FLG_LOCK_MASK ((flag_t)(PROC_FLG_LOCK))

A PROC_FLG_LOCK.

Used to test if a process has locked some resources.

5.21.2.9 #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

An execution state clear mask.

Used clear execution state bitts in proc->flags.

5.21.2.10 #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xFC)

An execution state clear mask.

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

5.21.2.11 #define PROC_STATE_MASK ((flag_t)0x0F)

An execution state mask.

5.21.2.12 #define PROC_STATE_RESTART_MASK ((flag_t)0x8)

A process execution state check mask.

Used by proc_restart and proc_restart_isr to check for restart posibility.

5.21.2.13 #define PROC_STATE_RUN_MASK ((flag_t)0x3)

A process execution state check mask.

Used to check if the process has been run.

5.21.2.14 #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

A process execution state check mask.

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

5.21.2.15 #define PROC_STATE_STOPED ((flag_t)0x0)

Initial state, stoped.

5.21.2.16 #define PROC_STATE_END ((flag_t)0x1)

Normal process termination.

5.21.2.17 #define PROC_STATE_READY ((flag_t)0x2)

Is ready to run.

5.21.2.18 #define PROC_STATE_RUNNING ((flag_t)0x3)

Is running.

5.21.2.19 #define PROC_STATE_WD_STOPED ((flag_t)0x4)

Watchdog termination.

5.21.2.20 #define PROC_STATE_DEAD ((flag_t)0x5)

Abnormal termination, terminated with waiting ipc transactions.

5.21.2.21 #define PROC_STATE_TO_READY ((flag_t)0x6)

Is ready to run.

5.21.2.22 #define PROC_STATE_TO_RUNNING ((flag_t)0x7)

Is running.

5.21.2.23 #define PROC_STATE_SYNC_WAIT ((flag_t)0x8)

Is waiting for sleaping processes.

5.21.2.24 #define PROC_STATE_SYNC_SLEEP ((flag_t)0x9)

Is waiting for wakeup.

5.21.2.25 #define PROC_STATE_SYNC_READY ((flag_t)0xA)

Is ready to run.

5.21.2.26 #define PROC_STATE_SYNC_RUNNING ((flag_t)0xB)

Is running.

5.21.2.27 #define PROC_STATE_PI_PEND ((flag_t)0xC)

A process is waiting for priority change.

5.21.2.28 #define PROC_STATE_PI_DONE ((flag_t)0xD)

A process has been run during priority change.

5.21.2.29 #define PROC_STATE_PI_READY ((flag_t)0xE)

Is ready to run.

5.21.2.30 #define PROC_STATE_PI_RUNNING ((flag_t)0xF)

Is running.

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

A PROC_FLG_PRE_STOP condition test macro.

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

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

Check if process is ready or running.

5.21.2.33 #define PROC_GET_STATE(*a*) (a->flags & **PROC_STATE_MASK**)

Reads a process state.

5.21.2.34 #define PROC_SET_STATE(*a,* *b*) (a->flags &= **PROC_STATE_CLEAR_MASK**, a->flags |= b)

Sets process state.

5.21.2.35 #define PROC_PRIO_LOWEST ((prio_t)BITS_IN_INDEX_T - (prio_t)1)

Lowest priority level.

5.21.3 Typedef Documentation

5.21.3.1 typedef struct _proc_t proc_t

See _proc_t;

5.21.4 Function Documentation

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

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

Parameters

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

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

A process initialization.

Parameters

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

5.21.4.3 void proc_run_wrapper (proc_t ∗ *proc*)

A wrapper for process "main" routines.

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

Parameters

proc	- A pointer to a process to launch.

5.21.4.4 void proc_terminate (void)

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

5.21.4.5 void _proc_terminate (void)

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

5.21.4.6 bool_t proc_run (proc_t * *proc*)

A process launch routine.

This function schedules a process if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.21.4.7 bool_t proc_run_isr (proc_t * *proc*)

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

This function schedules a process if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.21.4.8 bool_t proc_restart (proc_t * *proc*)

Aprocess restart routine.

This function reinitializes a process and schedules it if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.21.4.9 bool_t proc_restart_isr (proc_t * *proc*)

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

This function reinitializes a process and schedules it if possible.

Parameters

proc	- A pointer to a process to launch.

Returns

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

5.21.4.10 bool_t proc_stop (proc_t * *proc*)

A process stop routine.

This function stops a process if possible.

Parameters

proc	- A pointer to a process to stop.

Returns

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

5.21.4.11 bool_t proc_stop_isr (proc_t * *proc*)

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

This function stops a process if possible.

Parameters

proc	- A pointer to a process to stop.

Returns

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

5.21.4.12 void proc_self_stop (void)

A process self stop routine.

This function stops caller process.

5.21.4.13 void _proc_self_stop (void)

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

This function stops caller process.

5.21.4.14 void proc_reset_watchdog (void)

A watchdog reset routine for real time processes.

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

5.21.4.15 void _proc_reset_watchdog (void)

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

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

5.21.4.16 void _proc_prio_propagate (proc_t * *proc*)

Propagation of priority through a blovked process chain. For internal usage.

5.21.4.17 void _proc_stop_flags_set (proc_t ∗ *proc,* flag_t *mask*)

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

5.21.4.18 void _proc_lock (void)

Set PROC_FLG_LOCK for caller process.

5.21.4.19 void proc_lock (void)

Set PROC_FLG_LOCK for caller process.

5.21.4.20 void _proc_free (void)

A PROC_FLG_PRE_STOP flag processing routine. For internal usage.

5.21.4.21 void proc_free (void)

A PROC_FLG_PRE_STOP flag processing routine.

5.21.4.22 void _proc_prio_control_stoped (proc_t ∗ *proc*)

A stopedprocess priority control routine.

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

Parameters

proc	- A pointer to a process.

5.21.4.23 void proc_set_prio (proc_t ∗ *proc,* prio_t *prio*)

Set a priotity of a process.

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

Parameters

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

5.21.4.24 void _proc_set_prio (proc_t ∗ *proc,* prio_t *prio*)

Set a priotity of a process. For internel usage.

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

Parameters

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

5.21.4.25 void _proc_lres_inc (proc_t ∗ *proc,* prio_t *prio*)

Process priority control. For internel usage.

Increments proc->lres counter, sets PROC_FLG_LOCK flag.

Parameters

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

5.21.4.26 void _proc_lres_dec (proc_t ∗ *proc,* prio_t *prio*)

Process priority control. For internel usage.

Decrements proc->lres counter, clears PROC_FLG_LOCK flag if needed.

Parameters

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

5.21.4.27 void _proc_stop_ensure (proc_t ∗ *proc*)

Stops a process. For internel usage.

Stops aprocess for sure.

Parameters

proc	- A pointer to a process.

5.22 sched.c File Reference

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

Macros

- #define SCHED_STAT_UPDATE_RUN(a) (&kernel.sched)

Functions

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

  A scheduler initiation routine.
- void sched_proc_run (proc_t ∗proc, flag_t state)

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

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

  Pass control to next ready process.
- bool_t _sched_proc_yeld (void)

  Pass control to next ready process (for internal usage only!).
- void scall_sched_proc_yeld (void ∗arg)

  A SYSCALL_SCHED_PROC_YELD handler.

5.22.1 Macro Definition Documentation

5.22.1.1 #define SCHED_STAT_UPDATE_RUN(*a*) (&kernel.sched)

5.22.2 Function Documentation

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

A scheduler initiation routine.

This function prepares a scheduler object for work.

Parameters

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

5.22.2.2 void sched_proc_run (proc_t * *proc*, flag_t *state*)

A low level process run routine. For internal usage.

5.22.2.3 void sched_proc_stop (proc_t * *proc*)

A low level process stop routine. For internal usage.

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

5.22.2.5 void sched_schedule (void)

A scheduler routine.

This function switches processes in system timer interrupt handler.

5.22.2.6 void sched_reschedule (void)

Recheduler routine.

This function switches processes if needed.

5.22.2.7 bool_t sched_proc_yeld (void)

Pass control to next ready process.

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

Returns

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

5.22.2.8 bool_t _sched_proc_yeld (void)

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

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

Returns

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

5.22.2.9 void scall_sched_proc_yeld (void * *arg*)

A SYSCALL_SCHED_PROC_YELD handler.

Transfers control to another process.

Parameters

arg	Not used.

5.23 sched.h File Reference

Ascheduler header.

Data Structures

- struct _sched_t

  A scheduler.

Macros

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

  Wrapper macro.

Typedefs

- typedef struct _sched_t sched_t

Functions

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

  A scheduler initiation routine.
- void sched_schedule (void)

  A scheduler routine.
- void sched_reschedule (void)

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

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

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

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

  Pass control to next ready process.

5.23.1 Detailed Description

Ascheduler header.

Warning

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

5.23.2 Macro Definition Documentation

5.23.2.1 #define _SCHED_INIT( ) ((sched_t ∗)&kernel.sched)

Wrapper macro.

Initialization wrapper for sched variable in sched_schedule and sched_reschedule functions.

5.23.3 Typedef Documentation

5.23.3.1 typedef struct _sched_t sched_t

See _sched_t;

5.23.4 Function Documentation

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

A scheduler initiation routine.

This function prepares a scheduler object for work.

Parameters

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

5.23.4.2 void sched_schedule (void)

A scheduler routine.

This function switches processes in system timer interrupt handler.

5.23.4.3 void sched_reschedule (void)

Recheduler routine.

This function switches processes if needed.

5.23.4.4 void sched_proc_run (proc_t * *proc*, flag_t *state*)

A low level process run routine. For internal usage.

5.23.4.5 void sched_proc_stop (proc_t * *proc*)

A low level process stop routine. For internal usage.

5.23.4.6 bool_t _sched_proc_yeld (void)

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

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

Returns

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

5.23.4.7 bool_t sched_proc_yeld (void)

Pass control to next ready process.

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

Returns

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

5.24 sem.c File Reference

```
#include "sem.h"
```

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.
- flag_t sem_try_lock (sem_t ∗sem)

  Try to lock a semaphore.
- flag_t sem_lock (sem_t ∗sem)

  A semaphore lock.
- flag_t sem_free (sem_t ∗sem)

  Semaphore free.

5.24.1 Function Documentation

5.24.1.1 void sem_init_isr (sem_t ∗ *sem*, count_t *count*)

Semaphore initiation from ISR.

Parameters

sem	A sem_t pointer.
count	A counter start value.

5.24.1.2 void sem_init (sem_t ∗ *sem*, count_t *count*)

Semaphore initiation.

Parameters

sem	A sem_t pointer.
count	A counter start value.

5.24.1.3 flag_t sem_try_lock (sem_t ∗ *sem*)

Try to lock a semaphore.

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

Parameters

sem	A sem_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.24.1.4 flag_t sem_lock (sem_t * *sem*)

A semaphore lock.

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

Parameters

sem	A sem_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.24.1.5 flag_t sem_free (sem_t * *sem*)

Semaphore free.

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

Parameters

sem	A sem_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.25 sem.h File Reference

A counting semaphores header.

```
#include <bugurt.h>
```

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.
- flag_t sem_lock (sem_t *sem)

A semaphore lock.

- flag_t sem_try_lock (sem_t ∗sem)

  Try to lock a semaphore.

- flag_t sem_free (sem_t ∗sem)

  Semaphore free.

5.25.1 Detailed Description

A counting semaphores header.

5.25.2 Typedef Documentation

5.25.2.1 typedef struct _sem_t sem_t

See _sem_t;

5.25.3 Function Documentation

5.25.3.1 void sem_init_isr (sem_t ∗ *sem,* count_t *count*)

Semaphore initiation from ISR.

Parameters

sem	A sem_t pointer.
count	A counter start value.

5.25.3.2 void sem_init (sem_t ∗ *sem,* count_t *count*)

Semaphore initiation.

Parameters

sem	A sem_t pointer.
count	A counter start value.

5.25.3.3 flag_t sem_lock (sem_t ∗ *sem*)

A semaphore lock.

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

Parameters

sem	A sem_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.25.3.4 flag_t sem_try_lock (sem_t ∗ *sem*)

Try to lock a semaphore.

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

Parameters

sem	A sem_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.25.3.5 flag_t sem_free (sem_t ∗ *sem*)

Semaphore free.

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

Parameters

sem	A sem_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.26 sig.c File Reference

```
#include "sig.h"
```

Functions

- void sig_init (sig_t ∗sig)

  Signal initiation.
- void sig_init_isr (sig_t ∗sig)

  A signal initiation from ISR or critical section.
- flag_t sig_wait (sig_t ∗sig)

  Wait for a singnal.
- flag_t sig_signal (sig_t ∗sig)

  Fire a signal, launch one waiting process.
- count_t sig_broadcast (sig_t ∗sig)

  Fire a signal, launch all waiting processes.

5.26.1 Function Documentation

5.26.1.1 void sig_init (sig_t ∗ *sig*)

Signal initiation.

Parameters

sig	A sig_t pointer.

5.26.1.2 void sig_init_isr (sig_t ∗ *sig*)

A signal initiation from ISR or critical section.

Parameters

sig	A sig_t pointer.

5.26.1.3 flag_t sig_wait (sig_t ∗ *sig*)

Wait for a singnal.

This function stops caller process and inserts it to signal wait list.

Parameters

sig	A sig_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.26.1.4 flag_t sig_signal (sig_t ∗ *sig*)

Fire a signal, launch one waiting process.

Parameters

sig	A sig_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.26.1.5 count_t sig_broadcast (sig_t ∗ *sig*)

Fire a signal, launch all waiting processes.

This function launches all processes waiting for certain signal.

Parameters

sig	A sig_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.27 sig.h File Reference

A signal header.

```
#include <bugurt.h>
#include "cond.h"
```

Data Structures

- struct _sig_t

  A signal.

Typedefs

- typedef struct _sig_t sig_t

Functions

- void sig_init_isr (sig_t ∗sig)

  A signal initiation from ISR or critical section.
- void sig_init (sig_t ∗sig)

  Signal initiation.
- flag_t sig_wait (sig_t ∗sig)

  Wait for a singnal.
- flag_t sig_signal (sig_t ∗sig)

  Fire a signal, launch one waiting process.
- count_t sig_broadcast (sig_t ∗sig)

  Fire a signal, launch all waiting processes.

5.27.1 Detailed Description

A signal header.

5.27.2 Typedef Documentation

5.27.2.1 typedef struct _sig_t sig_t

See _sig_t;

5.27.3 Function Documentation

5.27.3.1 void sig_init_isr (sig_t ∗ *sig*)

A signal initiation from ISR or critical section.

Parameters

sig	A sig_t pointer.

5.27.3.2 void sig_init (sig_t ∗ *sig*)

Signal initiation.

Parameters

sig	A sig_t pointer.

5.27.3.3 flag_t sig_wait (sig_t ∗ *sig*)

Wait for a singnal.

This function stops caller process and inserts it to signal wait list.

Parameters

sig	A sig_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.27.3.4 flag_t sig_signal (sig_t ∗ *sig*)

Fire a signal, launch one waiting process.

Parameters

sig	A sig_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.27.3.5 count_t sig_broadcast (sig_t ∗ *sig*)

Fire a signal, launch all waiting processes.

This function launches all processes waiting for certain signal.

Parameters

sig	A sig_t pointer.

Returns

SYNC_ST_OK on success, or error number.

5.28 sync.c File Reference

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

Data Structures

- struct proc_set_prio_arg_t

    An argument for system call SYSCALL_PROC_SET_PRIO.
- struct sync_set_owner_t
- struct sync_proc_timeout_t

Macros

- #define PROC_PRIO_PROP_HOOK()
- #define PROC_PROC_PRIO_PROPAGATE(p) _proc_prio_propagate(p)
- #define SYNC_PROC_PRIO_PROPAGATE(p, m) _proc_prio_propagate(p)

Functions

- prio_t _sync_prio (sync_t ∗sync)

    Returns current sync_t object priority. For internal usage.
- void _proc_prio_propagate (proc_t ∗proc)

    Propagation of priority through a blovked process chain. For internal usage.
- void proc_set_prio (proc_t ∗proc, prio_t prio)

Set a priotity of a process.
- void _proc_set_prio (proc_t ∗proc, prio_t prio)

  Set a priotity of a process. For internel usage.
- void scall_proc_set_prio (void ∗arg)

  A SYSCALL_PROC_SET_PRIO handler.
- void sync_init (sync_t ∗sync, prio_t prio)

  A sync initiation for usage in ISRs or in critical sections.
- void sync_init_isr (sync_t ∗sync, prio_t prio)

  A basic synchronization promitive initiation.
- proc_t ∗ sync_get_owner (sync_t ∗sync)

  Get current sync_t object owner.
- flag_t sync_set_owner (sync_t ∗sync, proc_t ∗proc)

  Set sync_t object owner.
- flag_t _sync_set_owner (sync_t ∗sync, proc_t ∗proc)

  For internal usage. Watch sync_set_owner.
- void scall_sync_set_owner (void ∗arg)

  A SYSCALL_SYNC_SET_OWNER handler.
- void sync_clear_owner (sync_t ∗sync)

  Clear sync_t object owner.
- void _sync_clear_owner (sync_t ∗sync)

  For internal usage. Watch sync_clear_owner.
- void scall_sync_clear_owner (void ∗arg)

  A SYSCALL_SYNC_CLEAR_OWNER handler.
- flag_t sync_sleep (sync_t ∗sync)

  Sleep to wait for synchronization.
- flag_t _sync_sleep (sync_t ∗sync)

  For internal usage. Watch sync_sleep.
- void scall_sync_sleep (void ∗arg)

  A SYSCALL_SYNC_SLEEP handler.
- static void _sync_owner_block (proc_t ∗owner)
- flag_t sync_wait (sync_t ∗sync, proc_t ∗∗proc, flag_t block)

  Sleep to wait for synchronization.
- flag_t _sync_wait (sync_t ∗sync, proc_t ∗∗proc, flag_t block)

  For internal usage. Watch sync_wait.
- void scall_sync_wait (void ∗arg)

  A SYSCALL_SYNC_WAIT handler.
- flag_t sync_wake (sync_t ∗sync, proc_t ∗proc, flag_t chown)

  Sleep to wait for synchronization.
- flag_t _sync_wake (sync_t ∗sync, proc_t ∗proc, flag_t chown)

  For internal usage. Watch sync_wake.
- void scall_sync_wake (void ∗arg)

  A SYSCALL_SYNC_WAKE handler.
- flag_t sync_wake_and_sleep (sync_t ∗wake, proc_t ∗proc, flag_t chown, sync_t ∗sleep)

  Watch sync_wake and sync_sleep.
- void scall_sync_wake_and_sleep (void ∗arg)

  A SYSCALL_SYNC_WAKE_AND_SLEEP handler.
- flag_t sync_wake_and_wait (sync_t ∗wake, proc_t ∗proc_wake, flag_t chown, sync_t ∗wait, proc_t ∗∗proc_-wait, flag_t block)

  Watch sync_wake and sync_wait.
- void scall_sync_wake_and_wait (void ∗arg)

  A SYSCALL_SYNC_WAKE_AND_WAIT handler.

- flag_t sync_proc_timeout (proc_t ∗proc)

  Wake a process on timeout.
- void scall_sync_proc_timeout (void ∗arg)

  A SYSCALL_SYNC_PROC_TIMEOUT handler.
- flag_t _sync_proc_timeout (proc_t ∗proc)

  For internal usage. Watch sync_proc_timeout.

5.28.1 Macro Definition Documentation

5.28.1.1 #define PROC_PRIO_PROP_HOOK( )

5.28.1.2 #define PROC_PROC_PRIO_PROPAGATE(*p*) _proc_prio_propagate(p)

5.28.1.3 #define SYNC_PROC_PRIO_PROPAGATE(*p, m*) _proc_prio_propagate(p)

5.28.2 Function Documentation

5.28.2.1 prio_t _sync_prio (sync_t ∗ *sync*)

Returns current sync_t object priority. For internal usage.

5.28.2.2 void _proc_prio_propagate (proc_t ∗ *proc*)

Propagation of priority through a blovked process chain. For internal usage.

5.28.2.3 void proc_set_prio (proc_t ∗ *proc,* prio_t *prio*)

Set a priotity of a process.

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

Parameters

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

5.28.2.4 void _proc_set_prio (proc_t ∗ *proc,* prio_t *prio*)

Set a priotity of a process. For internel usage.

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

Parameters

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

5.28.2.5 void scall_proc_set_prio (void ∗ *arg*)

A SYSCALL_PROC_SET_PRIO handler.

This function calls _proc_set_prio.

Parameters

arg	A pointer to proc_set_prio_arg_t object.

5.28.2.6 void sync_init (sync_t ∗ *sync*, prio_t *prio*)

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

Parameters

sync	A sync pointer.
prio	A priority.

5.28.2.7 void sync_init_isr (sync_t ∗ *sync*, prio_t *prio*)

A basic synchronization promitive initiation.

Parameters

sync	A sync pointer.
prio	A priority.

5.28.2.8 proc_t∗ sync_get_owner (sync_t ∗ *sync*)

Get current sync_t object owner.

Parameters

sync	A pointer to the object of interest.

Returns

A pointer to sync_t opbject owner.

5.28.2.9 flag_t sync_set_owner (sync_t ∗ *sync*, proc_t ∗ *proc*)

Set sync_t object owner.

Parameters

sync	A pointer to the object of interest.
proc	A pointer to new sync_t opbject owner.

Returns

SYNC_ST_OK if owner was set, SYNC_ST_ROLL if sync_t object already had an owner.

5.28.2.10 flag_t _sync_set_owner (sync_t ∗ *sync*, proc_t ∗ *proc*)

For internal usage. Watch sync_set_owner.

5.28.2.11 void scall_sync_set_owner (void ∗ *arg*)

A SYSCALL_SYNC_SET_OWNER handler.

This function calls _sync_set_owner.

5.28.2.12 void sync_clear_owner (sync_t ∗ *sync*)

Clear sync_t object owner.

Parameters

sync	A pointer to the object of interest.

5.28.2.13 void _sync_clear_owner (**sync_t** ∗ *sync*)

For internal usage. Watch sync_clear_owner.

5.28.2.14 void scall_sync_clear_owner (void ∗ *arg*)

A SYSCALL_SYNC_CLEAR_OWNER handler.

This function calls _sync_clear_owner.

5.28.2.15 flag_t sync_sleep (**sync_t** ∗ *sync*)

Sleep to wait for synchronization.

Blocks caller process.

Parameters

sync	A pointer to the object of interest.

Returns

SYNC_ST_OK on success, or error number.

5.28.2.16 flag_t _sync_sleep (**sync_t** ∗ *sync*)

For internal usage. Watch sync_sleep.

5.28.2.17 void scall_sync_sleep (void ∗ *arg*)

A SYSCALL_SYNC_SLEEP handler.

This function calls _sync_sleep.

5.28.2.18 static void _sync_owner_block (**proc_t** ∗ *owner*) `[static]`

5.28.2.19 flag_t sync_wait (**sync_t** ∗ *sync*, **proc_t** ∗∗ *proc*, flag_t *block*)

Sleep to wait for synchronization.

Wait until target process is blocked on target sync_t object.

Parameters

sync	A sync_t object pointer.
proc	A double pointer to a process, that is supposed to block. If ∗proc is zero, then caller may wait for first process to block on sync_t object.
block	Block flag. If non 0 and caller process must wait, then caller is blocked until terget process is blocked on sync_t object.

Returns

SYNC_ST_OK if target process has blocked on target sync_t object, SYNC_ST_ROLL if caller must wait for target procerr to block, or error code.

5.28.2.20 flag_t _sync_wait (**sync_t** ∗ *sync*, **proc_t** ∗∗ *proc*, flag_t *block*)

For internal usage. Watch sync_wait.

5.28.2.21 void scall_sync_wait (void ∗ *arg*)

A SYSCALL_SYNC_WAIT handler.

This function calls _sync_wait.

5.28.2.22 flag_t sync_wake (sync_t ∗ *sync,* proc_t ∗ *proc,* flag_t *chown*)

Sleep to wait for synchronization.

Unblock some waiting process. A process should be blocked on target sync_t object.

Parameters

sync	A sync_t object pointer.
proc	A pointer to a process, that is supposed to wake up. If 0, then try to wake up wait list head.
chown	A change owner flag. If non 0, then ownership is given to wake up process.

Returns

SYNC_ST_OK on process wakeup, or error code.

5.28.2.23 flag_t _sync_wake (sync_t ∗ *sync,* proc_t ∗ *proc,* flag_t *chown*)

For internal usage. Watch sync_wake.

5.28.2.24 void scall_sync_wake (void ∗ *arg*)

A SYSCALL_SYNC_WAKE handler.

This function calls _sync_wake.

5.28.2.25 flag_t sync_wake_and_sleep (sync_t ∗ *wake,* proc_t ∗ *proc,* flag_t *chown,* sync_t ∗ *sleep*)

Watch sync_wake and sync_sleep.

5.28.2.26 void scall_sync_wake_and_sleep (void ∗ *arg*)

A SYSCALL_SYNC_WAKE_AND_SLEEP handler.

5.28.2.27 flag_t sync_wake_and_wait (sync_t ∗ *wake,* proc_t ∗ *proc_wake,* flag_t *chown,* sync_t ∗ *wait,* proc_t ∗∗ *proc_wait,* flag_t *block*)

Watch sync_wake and sync_wait.

5.28.2.28 void scall_sync_wake_and_wait (void ∗ *arg*)

A SYSCALL_SYNC_WAKE_AND_WAIT handler.

5.28.2.29 flag_t sync_proc_timeout (proc_t ∗ *proc*)

Wake a process on timeout.

Parameters

proc	A pointer to a process, that is supposed to wake up.

Returns

SYNC_ST_OK if target process has been woken up, SYNC_ST_ROLL if caller must do next itteration, or error code.

5.28.2.30 void scall_sync_proc_timeout (void ∗ *arg*)

A SYSCALL_SYNC_PROC_TIMEOUT handler.

5.28.2.31 flag_t _sync_proc_timeout (proc_t ∗ *proc*)

For internal usage. Watch sync_proc_timeout.

5.29 sync.h File Reference

A sync header.

Data Structures

- struct _sync_t

  Basic synchronization primitive.
- struct sync_sleep_t

  For internal usage.
- struct sync_wait_t

  For internal usage.
- struct sync_wake_t

  For internal usage.
- struct sync_wake_and_sleep_t

  For internal usage.
- struct sync_wake_and_wait_t

  For internal usage.

Macros

- #define SYNC_ST_OK 0

  Success.
- #define SYNC_ST_ENULL 1

  Null pointer argument.
- #define SYNC_ST_EOWN 2

  Ownership error.
- #define SYNC_ST_EEMPTY 3

  Wait process list is empty.
- #define SYNC_ST_ESYNC 4

  Wrong sync_t object.
- #define SYNC_ST_ETIMEOUT 5

  Timeout expired.
- #define SYNC_ST_ROLL 6

  Next itteration needed.
- #define SYNC_PRIO(s) _sync_prio(s)

  Calculates a sync_t object priority.
- #define SYNC_INIT(s, p) sync_init((sync_t ∗)s, (prio_t)p)

  Watch sync_init.
- #define SYNC_INIT_ISR(s, p) sync_init_isr((sync_t ∗)s, (prio_t)p)

  Watch sync_init_isr.
- #define SYNC_GET_OWNER(s) sync_get_owner((sync_t ∗)s)

  Watch sync_get_owner.

- #define SYNC_SET_OWNER(s, p) sync_set_owner((sync_t ∗)s, (proc_t ∗)p)

  Watch sync_set_owner.
- #define SYNC_CLEAR_OWNER(s) sync_clear_owner((sync_t ∗)s)

  Watch sync_clear_owner.
- #define SYNC_SLEEP(s) sync_sleep((sync_t ∗)s)

  Watch sync_sleep.
- #define SYNC_WAIT(s, p, b, st)

  Watch sync_wait.
- #define SYNC_WAKE(s, p, c, st)

  Watch sync_wake.
- #define SYNC_WAKE_AND_SLEEP(w, p, c, s, st)

  Watch sync_wake_and_sleep.
- #define SYNC_WAKE_AND_WAIT(wk, pwk, c, wt, pwt, b, st)

  Watch sync_wake_and_wait.

Typedefs

- typedef struct _sync_t sync_t

Functions

- prio_t _sync_prio (sync_t ∗sync)

  Returns current sync_t object priority. For internal usage.
- void sync_init (sync_t ∗sync, prio_t prio)

  A sync initiation for usage in ISRs or in critical sections.
- void sync_init_isr (sync_t ∗sync, prio_t prio)

  A basic synchronization promitive initiation.
- proc_t ∗ sync_get_owner (sync_t ∗sync)

  Get current sync_t object owner.
- flag_t sync_set_owner (sync_t ∗sync, proc_t ∗proc)

  Set sync_t object owner.
- void sync_clear_owner (sync_t ∗sync)

  Clear sync_t object owner.
- flag_t sync_sleep (sync_t ∗sync)

  Sleep to wait for synchronization.
- flag_t sync_wait (sync_t ∗sync, proc_t ∗∗proc, flag_t block)

  Sleep to wait for synchronization.
- flag_t sync_wake (sync_t ∗sync, proc_t ∗proc, flag_t chown)

  Sleep to wait for synchronization.
- flag_t sync_wake_and_sleep (sync_t ∗wake, proc_t ∗proc, flag_t chown, sync_t ∗sleep)

  Watch sync_wake and sync_sleep.
- flag_t sync_wake_and_wait (sync_t ∗wake, proc_t ∗proc_wake, flag_t chown, sync_t ∗wait, proc_t ∗∗proc_-wait, flag_t block)

  Watch sync_wake and sync_wait.
- flag_t sync_proc_timeout (proc_t ∗proc)

  Wake a process on timeout.
- flag_t _sync_set_owner (sync_t ∗sync, proc_t ∗proc)

  For internal usage. Watch sync_set_owner.
- void _sync_clear_owner (sync_t ∗sync)

  For internal usage. Watch sync_clear_owner.
- flag_t _sync_wake (sync_t ∗sync, proc_t ∗proc, flag_t chown)

For internal usage. Watch sync_wake.
- flag_t _sync_sleep (sync_t ∗sync)

  For internal usage. Watch sync_sleep.
- flag_t _sync_wait (sync_t ∗sync, proc_t ∗∗proc, flag_t block)

  For internal usage. Watch sync_wait.
- flag_t _sync_proc_timeout (proc_t ∗proc)

  For internal usage. Watch sync_proc_timeout.

5.29.1 Detailed Description

A sync header.

5.29.2 Macro Definition Documentation

5.29.2.1 #define SYNC_ST_OK 0

Success.

5.29.2.2 #define SYNC_ST_ENULL 1

Null pointer argument.

5.29.2.3 #define SYNC_ST_EOWN 2

Ownership error.

5.29.2.4 #define SYNC_ST_EEMPTY 3

Wait process list is empty.

5.29.2.5 #define SYNC_ST_ESYNC 4

Wrong sync_t object.

5.29.2.6 #define SYNC_ST_ETIMEOUT 5

Timeout expired.

5.29.2.7 #define SYNC_ST_ROLL 6

Next itteration needed.

5.29.2.8 #define SYNC_PRIO(*s*) _sync_prio(s)

Calculates a sync_t object priority.

5.29.2.9 #define SYNC_INIT(*s, p*) sync_init((sync_t ∗)s, (prio_t)p)

Watch sync_init.

5.29.2.10 #define SYNC_INIT_ISR(*s, p*) sync_init_isr((sync_t ∗)s, (prio_t)p)

Watch sync_init_isr.

5.29.2.11 #define SYNC_GET_OWNER(*s*) sync_get_owner((sync_t ∗)s)

Watch sync_get_owner.

5.29.2.12 #define SYNC_SET_OWNER(*s*, *p*) sync_set_owner((sync_t ∗)s, (proc_t ∗)p)

Watch sync_set_owner.

5.29.2.13 #define SYNC_CLEAR_OWNER(*s*) sync_clear_owner((sync_t ∗)s)

Watch sync_clear_owner.

5.29.2.14 #define SYNC_SLEEP(*s*) sync_sleep((sync_t ∗)s)

Watch sync_sleep.

5.29.2.15 #define SYNC_WAIT(*s*, *p*, *b*, *st*)

Value:

```
do                                                              \
{                                                               \
    volatile sync_wait_t scarg;                                 \
    scarg.status = SYNC_ST_ROLL;
         \
    scarg.sync = (sync_t *)(s);                                 \
    scarg.proc = (proc_t **)(p);                                \
    scarg.block = (flag_t)(b);                                  \
    do                                                          \
    {                                                           \
        syscall_bugurt( SYSCALL_SYNC_WAIT, (void *)&scarg );
        \
    }                                                           \
    while( scarg.status >= SYNC_ST_ROLL );                      \
    (st) = scarg.status;                                        \
}                                                               \
while(0)
```

Watch sync_wait.

5.29.2.16 #define SYNC_WAKE(*s*, *p*, *c*, *st*)

Value:

```
do                                                              \
{                                                               \
    volatile sync_wake_t scarg;                                 \
    scarg.status = SYNC_ST_ROLL;
         \
    scarg.sync = (sync_t *)(s);                                 \
    scarg.proc = (proc_t *)(p);                                 \
    scarg.chown = (flag_t)(c);                                  \
    do                                                          \
    {                                                           \
        syscall_bugurt( SYSCALL_SYNC_WAKE, (void *)&scarg );
        \
    }                                                           \
    while( scarg.status >= SYNC_ST_ROLL );                      \
    (st) = scarg.status;                                        \
}                                                               \
while(0)
```

Watch sync_wake.

5.29.2.17 #define SYNC_WAKE_AND_SLEEP(*w*, *p*, *c*, *s*, *st*)

Value:

```
do                                                                      \
{                                                                       \
    volatile sync_wake_and_sleep_t scarg;
                    \
    scarg.sleep.sync = (sync_t *)(s);
          \
    scarg.sleep.status = SYNC_ST_ROLL;
            \
    scarg.chown = (flag_t)(c);                                          \
    scarg.wake = (sync_t *)(w);                                         \
    scarg.proc = (proc_t *)(p);                                         \
    scarg.stage = (flag_t)0;                                            \
    do                                                                  \
    {                                                                   \
        syscall_bugurt( SYSCALL_SYNC_WAKE_AND_SLEEP,
       (void *)&scarg );  \
    }                                                                   \
    while( scarg.sleep.status >= SYNC_ST_ROLL );
           \
    (st) = scarg.sleep.status;                                          \
}                                                                       \
while(0)
```

Watch sync_wake_and_sleep.

5.29.2.18 #define SYNC_WAKE_AND_WAIT(*wk,* *pwk,* *c,* *wt,* *pwt,* *b,* *st*)

Value:

```
do                                                                      \
{                                                                       \
    volatile sync_wake_and_wait_t scarg;
                  \
    scarg.wait.sync   = (sync_t *)(wt);
            \
    scarg.wait.proc   = (proc_t **)(pwt);                               \
    scarg.wait.block  = (flag_t)(b);                                    \
    scarg.wait.status = SYNC_ST_ROLL;
          \
    scarg.wake      = (sync_t *)(wk);                                   \
    scarg.proc      = (proc_t *)(pwk);                                  \
    scarg.chown     = (flag_t)(c);                                      \
    scarg.stage     = (flag_t)0;                                        \
    do                                                                  \
    {                                                                   \
        syscall_bugurt( SYSCALL_SYNC_WAKE_AND_WAIT, (
      void *)&scarg );   \
    }                                                                   \
    while( scarg.wait.status >= SYSCALL_SYNC_WAKE_AND_WAIT
       );            \
    (st) = scarg.wait.status;                                           \
}                                                                       \
while(0)
```

Watch sync_wake_and_wait.

5.29.3 Typedef Documentation

5.29.3.1 typedef struct _sync_t sync_t

See _sync_t;

5.29.4 Function Documentation

5.29.4.1 prio_t _sync_prio (sync_t * *sync*)

Returns current sync_t object priority. For internal usage.

5.29.4.2 void sync_init (sync_t * *sync,* prio_t *prio*)

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

Parameters

sync	A sync pointer.
prio	A priority.

5.29.4.3 void sync_init_isr (sync_t ∗ *sync*, prio_t *prio*)

A basic synchronization promitive initiation.

Parameters

sync	A sync pointer.
prio	A priority.

5.29.4.4 proc_t∗ sync_get_owner (sync_t ∗ *sync*)

Get current sync_t object owner.

Parameters

sync	A pointer to the object of interest.

Returns

A pointer to sync_t opbject owner.

5.29.4.5 flag_t sync_set_owner (sync_t ∗ *sync*, proc_t ∗ *proc*)

Set sync_t object owner.

Parameters

sync	A pointer to the object of interest.
proc	A pointer to new sync_t opbject owner.

Returns

SYNC_ST_OK if owner was set, SYNC_ST_ROLL if sync_t object already had an owner.

5.29.4.6 void sync_clear_owner (sync_t ∗ *sync*)

Clear sync_t object owner.

Parameters

sync	A pointer to the object of interest.

5.29.4.7 flag_t sync_sleep (sync_t ∗ *sync*)

Sleep to wait for synchronization.

Blocks caller process.

Parameters

sync	A pointer to the object of interest.

Returns

SYNC_ST_OK on success, or error number.

5.29.4.8 flag_t sync_wait (sync_t ∗ *sync*, proc_t ∗∗ *proc*, flag_t *block*)

Sleep to wait for synchronization.

Wait until target process is blocked on target sync_t object.

Parameters

sync	A sync_t object pointer.
proc	A double pointer to a process, that is supposed to block. If ∗proc is zero, then caller may wait for first process to block on sync_t object.
block	Block flag. If non 0 and caller process must wait, then caller is blocked until terget process is blocked on sync_t object.

Returns

SYNC_ST_OK if target process has blocked on target sync_t object, SYNC_ST_ROLL if caller must wait for target procerr to block, or error code.

5.29.4.9 flag_t sync_wake (sync_t ∗ *sync*, proc_t ∗ *proc*, flag_t *chown*)

Sleep to wait for synchronization.

Unblock some waiting process. A process should be blocked on target sync_t object.

Parameters

sync	A sync_t object pointer.
proc	A pointer to a process, that is supposed to wake up. If 0, then try to wake up wait list head.
chown	A change owner flag. If non 0, then ownership is given to wake up process.

Returns

SYNC_ST_OK on process wakeup, or error code.

5.29.4.10 flag_t sync_wake_and_sleep (sync_t ∗ *wake*, proc_t ∗ *proc*, flag_t *chown*, sync_t ∗ *sleep*)

Watch sync_wake and sync_sleep.

5.29.4.11 flag_t sync_wake_and_wait (sync_t ∗ *wake*, proc_t ∗ *proc_wake*, flag_t *chown*, sync_t ∗ *wait*, proc_t ∗∗ *proc_wait*, flag_t *block*)

Watch sync_wake and sync_wait.

5.29.4.12 flag_t sync_proc_timeout (proc_t ∗ *proc*)

Wake a process on timeout.

Parameters

proc	A pointer to a process, that is supposed to wake up.

Returns

SYNC_ST_OK if target process has been woken up, SYNC_ST_ROLL if caller must do next itteration, or error code.

5.29.4.13 flag_t _sync_set_owner (sync_t ∗ *sync*, proc_t ∗ *proc*)

For internal usage. Watch sync_set_owner.

5.29.4.14 void _sync_clear_owner (**sync_t** ∗ *sync*)

For internal usage. Watch sync_clear_owner.

5.29.4.15 flag_t _sync_wake (**sync_t** ∗ *sync,* **proc_t** ∗ *proc,* flag_t *chown*)

For internal usage. Watch sync_wake.

5.29.4.16 flag_t _sync_sleep (**sync_t** ∗ *sync*)

For internal usage. Watch sync_sleep.

5.29.4.17 flag_t _sync_wait (**sync_t** ∗ *sync,* **proc_t** ∗∗ *proc,* flag_t *block*)

For internal usage. Watch sync_wait.

5.29.4.18 flag_t _sync_proc_timeout (**proc_t** ∗ *proc*)

For internal usage. Watch sync_proc_timeout.

5.30 syscall.c File Reference

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

Data Structures

- struct scall_user_t

Functions

- SYSCALL_TABLE (syscall_routine[ ])
- void do_syscall (void)

  System call processing routine.
- void scall_user (void ∗arg)

  A SYSCALL_USER handler.

Variables

- syscall_t syscall_num = (syscall_t)0

  System call number.
- void ∗ syscall_arg = (void ∗)0

  System call argument.

5.30.1 Function Documentation

5.30.1.1 SYSCALL_TABLE (syscall_routine *[ ]*)

5.30.1.2 void do_syscall (void)

System call processing routine.

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

5.30.1.3 void scall_user (void * *arg*)

A SYSCALL_USER handler.

5.30.2 Variable Documentation

5.30.2.1 syscall_t syscall_num = (syscall_t)0

System call number.

5.30.2.2 void* syscall_arg = (void *)0

System call argument.

5.31 syscall.h File Reference

System call header.

Macros

- #define SYSCALL_PROC_RUN ((syscall_t)(1))

  A process launch.
- #define SYSCALL_PROC_RESTART (SYSCALL_PROC_RUN + (syscall_t)(1))

  A Process restart.
- #define SYSCALL_PROC_STOP (SYSCALL_PROC_RESTART + (syscall_t)(1))

  A process stop.
- #define SYSCALL_PROC_SELF_STOP (SYSCALL_PROC_STOP + (syscall_t)(1))

  A process self stop.
- #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))

  A process termination.
- #define SYSCALL_PROC_LOCK (SYSCALL_PROC_TERMINATE + (syscall_t)(1))

  PROC_FLG_LOCK for caller process.
- #define SYSCALL_PROC_FREE (SYSCALL_PROC_LOCK + (syscall_t)(1))

  PROC_FLG_PRE_STOP flag processing.
- #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FREE + (syscall_t)(1))

  A real time process watchdog reset.
- #define SYSCALL_PROC_SET_PRIO (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))

  Set a process priority.
- #define SYSCALL_SCHED_PROC_YELD (SYSCALL_PROC_SET_PRIO + (syscall_t)(1))

  Transfer control to another process.
- #define SYSCALL_SYNC_SET_OWNER (SYSCALL_SCHED_PROC_YELD + (syscall_t)(1))

  Set new sync_t object owner.
- #define SYSCALL_SYNC_CLEAR_OWNER (SYSCALL_SYNC_SET_OWNER + (syscall_t)(1))

  Clear sync_t object owner.
- #define SYSCALL_SYNC_SLEEP (SYSCALL_SYNC_CLEAR_OWNER + (syscall_t)(1))

  Block process for synchronization.
- #define SYSCALL_SYNC_WAKE (SYSCALL_SYNC_SLEEP + (syscall_t)(1))

  Run a process waiting for synchronization.
- #define SYSCALL_SYNC_WAIT (SYSCALL_SYNC_WAKE + (syscall_t)(1))

  Wait for process to block on sync_t object.
- #define SYSCALL_SYNC_WAKE_AND_SLEEP (SYSCALL_SYNC_WAIT + (syscall_t)(1))

  Watch SYSCALL_SYNC_WAKE and SYSCALL_SYNC_SLEEP.

- #define SYSCALL_SYNC_WAKE_AND_WAIT (SYSCALL_SYNC_WAKE_AND_SLEEP + (syscall_t)(1))

  Watch SYSCALL_SYNC_WAKE and SYSCALL_SYNC_WAIT.
- #define SYSCALL_SYNC_PROC_TIMEOUT (SYSCALL_SYNC_WAKE_AND_WAIT + (syscall_t)(1))

  Wake a process on timeout.
- #define SYSCALL_USER (SYSCALL_SYNC_PROC_TIMEOUT + (syscall_t)(1))

  User system call.

Functions

- void do_syscall (void)

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

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

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

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

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

  A SYSCALL_SCHED_PROC_YELD handler.
- void scall_proc_terminate (void ∗arg)

  A SYSCALL_PROC_TERMINATE handler.
- void scall_proc_lock (void ∗arg)

  A SYSCALL_PROC_LOCK handler.
- void scall_proc_free (void ∗arg)

  A SYSCALL_PROC_FREE handler.
- void scall_proc_reset_watchdog (void ∗arg)

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

  A SYSCALL_PROC_SET_PRIO handler.
- void scall_sync_set_owner (void ∗arg)

  A SYSCALL_SYNC_SET_OWNER handler.
- void scall_sync_clear_owner (void ∗arg)

  A SYSCALL_SYNC_CLEAR_OWNER handler.
- void scall_sync_sleep (void ∗arg)

  A SYSCALL_SYNC_SLEEP handler.
- void scall_sync_wake (void ∗arg)

  A SYSCALL_SYNC_WAKE handler.
- void scall_sync_wait (void ∗arg)

  A SYSCALL_SYNC_WAIT handler.
- void scall_sync_wake_and_sleep (void ∗arg)

  A SYSCALL_SYNC_WAKE_AND_SLEEP handler.
- void scall_sync_wake_and_wait (void ∗arg)

  A SYSCALL_SYNC_WAKE_AND_WAIT handler.
- void scall_sync_proc_timeout (void ∗arg)

  A SYSCALL_SYNC_PROC_TIMEOUT handler.
- void scall_user (void ∗arg)

  A SYSCALL_USER handler.

Variables

- syscall_t syscall_num
  System call number.
- void ∗ syscall_arg
  System call argument.

5.31.1 Detailed Description

System call header.

5.31.2 Macro Definition Documentation

5.31.2.1 #define SYSCALL_PROC_RUN ((syscall_t)(1))

A process launch.

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

A Process restart.

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

A process stop.

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

A process self stop.

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

A process termination.

5.31.2.6 #define SYSCALL_PROC_LOCK (SYSCALL_PROC_TERMINATE + (syscall_t)(1))

PROC_FLG_LOCK for caller process.

5.31.2.7 #define SYSCALL_PROC_FREE (SYSCALL_PROC_LOCK + (syscall_t)(1))

PROC_FLG_PRE_STOP flag processing.

5.31.2.8 #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FREE + (syscall_t)(1))

A real time process watchdog reset.

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

Set a process priority.

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

Transfer control to another process.

5.31.2.11 #define SYSCALL_SYNC_SET_OWNER (SYSCALL_SCHED_PROC_YELD + (syscall_t)(1))

Set new sync_t object owner.

5.31.2.12 #define SYSCALL_SYNC_CLEAR_OWNER (SYSCALL_SYNC_SET_OWNER + (syscall_t)(1))

Clear sync_t object owner.

5.31.2.13 #define SYSCALL_SYNC_SLEEP (SYSCALL_SYNC_CLEAR_OWNER + (syscall_t)(1))

Block process for synchronization.

5.31.2.14 #define SYSCALL_SYNC_WAKE (SYSCALL_SYNC_SLEEP + (syscall_t)(1))

Run a process waiting for synchronization.

5.31.2.15 #define SYSCALL_SYNC_WAIT (SYSCALL_SYNC_WAKE + (syscall_t)(1))

Wait for process to block on sync_t object.

5.31.2.16 #define SYSCALL_SYNC_WAKE_AND_SLEEP (SYSCALL_SYNC_WAIT + (syscall_t)(1))

Watch SYSCALL_SYNC_WAKE and SYSCALL_SYNC_SLEEP.

5.31.2.17 #define SYSCALL_SYNC_WAKE_AND_WAIT (SYSCALL_SYNC_WAKE_AND_SLEEP + (syscall_t)(1))

Watch SYSCALL_SYNC_WAKE and SYSCALL_SYNC_WAIT.

5.31.2.18 #define SYSCALL_SYNC_PROC_TIMEOUT (SYSCALL_SYNC_WAKE_AND_WAIT + (syscall_t)(1))

Wake a process on timeout.

5.31.2.19 #define SYSCALL_USER (SYSCALL_SYNC_PROC_TIMEOUT + (syscall_t)(1))

User system call.

5.31.3 Function Documentation

5.31.3.1 void do_syscall (void)

System call processing routine.

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

5.31.3.2 void scall_proc_run (void ∗ *arg*)

A SYSCALL_PROC_RUN handler.

This function tries to launch a process by proc_run_isr call.

Parameters

arg	A proc_runtime_arg_t pointer.

5.31.3.3 void scall_proc_restart (void ∗ *arg*)

A SYSCALL_PROC_RESTART handler.

This function tries to restart a process by proc_restart_isr call.

Parameters

arg	A proc_runtime_arg_t pointer.

5.31.3.4 void scall_proc_stop (void ∗ *arg*)

A SYSCALL_PROC_STOP handler.

This function tries to stop a process by proc_stop_isr call.

Parameters

arg	A proc_runtime_arg_t pointer.

5.31.3.5 void scall_proc_self_stop (void ∗ *arg*)

A SYSCALL_PROC_SELF_STOP handler.

This function stops calling process.

Parameters

arg	Not used.

5.31.3.6 void scall_sched_proc_yeld (void ∗ *arg*)

A SYSCALL_SCHED_PROC_YELD handler.

Transfers control to another process.

Parameters

arg	Not used.

5.31.3.7 void scall_proc_terminate (void ∗ *arg*)

A SYSCALL_PROC_TERMINATE handler.

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

Parameters

arg	A pointer to a process.

5.31.3.8 void scall_proc_lock (void ∗ *arg*)

A SYSCALL_PROC_LOCK handler.

Sets #PROC_FLG_NONSTOP for caller process, increases proc->lres counter.

5.31.3.9 void scall_proc_free (void ∗ *arg*)

A SYSCALL_PROC_FREE handler.

This function decreases proc->lres counter, clears PROC_FLG_LOCK if needed and, process PROC_FLG_PRE-_STOP of the calling process and clears masked flags of a calling process. It calls _proc_free.

Parameters

arg	A poointer to a flag mask.

5.31.3.10 void scall_proc_reset_watchdog (void ∗ *arg*)

A SYSCALL_PROC_RESET_WATCHDOG handler.

This function calls _proc_reset_watchdog.

Parameters

arg	Not used.

5.31.3.11 void scall_proc_set_prio (void ∗ *arg*)

A SYSCALL_PROC_SET_PRIO handler.

This function calls _proc_set_prio.

Parameters

arg	A pointer to proc_set_prio_arg_t object.

5.31.3.12 void scall_sync_set_owner (void ∗ *arg*)

A SYSCALL_SYNC_SET_OWNER handler.

This function calls _sync_set_owner.

5.31.3.13 void scall_sync_clear_owner (void ∗ *arg*)

A SYSCALL_SYNC_CLEAR_OWNER handler.

This function calls _sync_clear_owner.

5.31.3.14 void scall_sync_sleep (void ∗ *arg*)

A SYSCALL_SYNC_SLEEP handler.

This function calls _sync_sleep.

5.31.3.15 void scall_sync_wake (void ∗ *arg*)

A SYSCALL_SYNC_WAKE handler.

This function calls _sync_wake.

5.31.3.16 void scall_sync_wait (void ∗ *arg*)

A SYSCALL_SYNC_WAIT handler.

This function calls _sync_wait.

5.31.3.17 void scall_sync_wake_and_sleep (void ∗ *arg*)

A SYSCALL_SYNC_WAKE_AND_SLEEP handler.

5.31.3.18 void scall_sync_wake_and_wait (void ∗ *arg*)

A SYSCALL_SYNC_WAKE_AND_WAIT handler.

5.31.3.19 void scall_sync_proc_timeout (void ∗ *arg*)

A SYSCALL_SYNC_PROC_TIMEOUT handler.

5.31.3.20 void scall_user (void ∗ *arg*)

A SYSCALL_USER handler.

5.31.4 Variable Documentation

5.31.4.1 syscall_t syscall_num

System call number.

5.31.4.2 void∗ syscall_arg

System call argument.

5.32 timer.c File Reference

```
#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.

5.32.1 Function Documentation

5.32.1.1 void _clear_timer (timer_t ∗ *t*)

Clear software timer. For unternal usage.

Parameters

t	A pointer to a timer.

5.32.1.2 timer_t _timer (timer_t *t*)

Get software timer. For internal usage.

Parameters

t	A timer value.

5.32.1.3 void wait_time (timer_t *time*)

Wait for certain time.

Caller process spins in a loop for a time.

Parameters

time	Wait time.

5.33 timer.h File Reference

Asoftware timer headers.

Macros

- #define SPIN_LOCK_KERNEL_TIMER()

  Wrapper macro.

- #define SPIN_FREE_KERNEL_TIMER()

  Wrapper macro.
- #define CLEAR_TIMER(t) _clear_timer((timer_t ∗)&t)

  Reset software timer.
- #define TIMER(t) (timer_t)_timer((timer_t)t)

  Get software timer value.

Functions

- void wait_time (timer_t time)

  Wait for certain time.
- void _clear_timer (timer_t ∗t)

  Clear software timer. For unternal usage.
- timer_t _timer (timer_t t)

  Get software timer. For internal usage.

5.33.1 Detailed Description

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

Warning

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

5.33.2 Macro Definition Documentation

5.33.2.1 #define SPIN_LOCK_KERNEL_TIMER( )

Wrapper macro.

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

5.33.2.2 #define SPIN_FREE_KERNEL_TIMER( )

Wrapper macro.

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

5.33.2.3 #define CLEAR_TIMER(*t*) _clear_timer((timer_t ∗)&t)

Reset software timer.

Parameters

t	A timer variable name.

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

Get software timer value.

Parameters

t	Software timer value.

5.33.3 Function Documentation

5.33.3.1 void wait_time (timer_t *time*)

Wait for certain time.

Caller process spins in a loop for a time.

Parameters

time	Wait time.

5.33.3.2 void _clear_timer (timer_t ∗ *t*)

Clear software timer. For unternal usage.

Parameters

t	A pointer to a timer.

5.33.3.3 timer_t _timer (timer_t *t*)

Get software timer. For internal usage.

Parameters

t	A timer value.

5.34 xlist.c File Reference

```
#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.

5.34.1 Function Documentation

5.34.1.1 void xlist_init (xlist_t ∗ *xlist*)

An xlist_t object initiation.

Parameters

xlist	An xlist_t pointer.

5.34.1.2 item_t∗ xlist_head (xlist_t ∗ *xlist*)

List head search.

Parameters

xlist	An xlist_t pointer.

Returns

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

5.34.1.3 void xlist_switch (xlist_t ∗ *xlist,* prio_t *prio*)

Switch a head pointer.

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

Parameters

xlist	An xlist_t pointer.
prio	A priority to switch.

5.35 xlist.h File Reference

A prioritized list header.

Data Structures

- struct _xlist_t

  A prioritized list.

Typedefs

- typedef struct _xlist_t xlist_t

Functions

- void xlist_init (xlist_t ∗xlist)

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

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

  Switch a head pointer.

5.35.1 Detailed Description

A prioritized list header.

5.35.2 Typedef Documentation

5.35.2.1 typedef struct _xlist_t xlist_t

See _xlist_t;

5.35.3 Function Documentation

5.35.3.1 void xlist_init (**xlist_t** ∗ *xlist*)

An xlist_t object initiation.

Parameters

xlist	An xlist_t pointer.

5.35.3.2 **item_t**∗ xlist_head (**xlist_t** ∗ *xlist*)

List head search.

Parameters

xlist	An xlist_t pointer.

Returns

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

5.35.3.3 void xlist_switch (**xlist_t** ∗ *xlist,* prio_t *prio*)

Switch a head pointer.

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

Parameters

xlist	An xlist_t pointer.
prio	A priority to switch.