BuguRTOS

0.7.0

Создано системой Doxygen 1.8.1.2

Сб 17 Май 2014 21:16:04

Содержание

1

BuguRTOS - ядро операционной системы реального времени. Написано анонимусом ДЛЯ УДОВОЛЬСТВИЯ.

Распространяется под измененной лицензией GPLv3, смотрите exception.txt.

2

2.1

Структуры данных с их кратким описанием.

3

3.1

Полный список документированных файлов.

4

4.1 _gitem_t

Элемент группированного списка.

#include "include/gitem.h"

- item_t parent
- group_t * group
- group_t grp

4.1.1

Элемент группированного списка.

Такой элемент постоянно состоит в одной из групп (смотри group_t). В каждом таком элементе gitem_t есть поле типа group_t, это группа, в которую изначально включен элемент.

4.1.2

4.1.2.1 item_t parent

Родитель - элемент 2-связного списка.

4.1.2.2 group_t* group

Указатель на группу, в которую сейчас включен элемент.

4.1.2.3 group_t grp

Выделение памяти под группу, в нее изначально будет включен элемент.

Объявления и описания членов структуры находятся в файле:

- include/gitem.h

4.2 _group_t

Группа элементов типа gitem_t.

#include "include/gitem.h"

- void * link
- prio_t prio
- count_t el_num

4.2.1

Группа элементов типа gitem_t.

Группа элементов типа gitem_t, хранит информацию о приоритете входящих в нее элементов, а так же указатель на список xlist_t, которому принадлежат эти элементы.

Каждый элемент gitem_t имеет поле типа group_t, изначально принадлежит этой самой группе, однако при включении элемента в группированный список, где уже есть элементы с таким же приоритетом, собственная группа будет передана в Пул, и элемент будет включен в группу, которая уже есть в списке.

4.2.2

4.2.2.1 void* link

Поле используется для хранения указателя на список, либо для хранения указателя на следующую группу в gуле.

4.2.2.2 prio_t prio

Приоритет группы.

4.2.2.3 count_t el_num

Количество элементов в группе, подсчет ссылок же!

Объявления и описания членов структуры находятся в файле:

- include/gitem.h

4.3 _gxlist_t

- xlist_t parent
- pool_t pool

4.3.1

4.3.1.1 xlist_t parent

Родительский тип - xlist_t.

4.3.1.2 pool_t pool

Пул для хранения неиспользуемых групп.

Объявления и описания членов структуры находятся в файле:

- include/gitem.h

4.4 _item_t

Элемент 2-связного списка.

#include "include/item.h"

- item_t * next
- item_t * prev

4.4.1

Элемент 2-связного списка.

Все структуры, где будут применяться 2-связные списки, унаследуют свойства и методы item_t.

4.4.2

4.4.2.1 item_t* next

Следующий элемент.

4.4.2.2 item_t* prev

Предыдущий элемент.

Объявления и описания членов структуры находятся в файле:

- include/item.h

4.5 _kernel_t

Ядро BuguRTOS.

#include "include/kernel.h"

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

4.5.1

Ядро BuguRTOS.

В ядре хранится информация о запущенных процессах, процессе(ах) холостого хода.

4.5.2

4.5.2.1 sched_t sched

Планировщик.

4.5.2.2 proc_t idle

Процесс холостого хода.

4.5.2.3 timer_t timer

Системный таймер.

4.5.2.4 void(* timer_tick)(void)

Хук обработчика системного таймера.

Объявления и описания членов структуры находятся в файле:

- include/kernel.h

4.6 _mutex_t

Мьютекс.

#include "include/mutex.h"

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

4.6.1

Мьютекс.

Используется для управления доступом к общим ресурсам, в тех случаях, когда общий ресурс нужен в течение долгого времени. Поддерживается произвольная вложенность мьютексов.

> Мьютексы захватываются и освобожаются только процессами. Нельзя делать это из обработчиков прерываний.
> Мьютекс должен освободить ИМЕННО ТОТ процесс, который его захватил.

4.6.2

4.6.2.1 xlist_t wait

Список ожидающих процессов.

4.6.2.2 bool_t free

Флаг "свободен", 1 - если мьютекс свободен, 0 - если занят.

4.6.2.3 proc_t* owner

Указатель на процесс, удерживающий мьютекс.

4.6.2.4 count_t dirty

Счетчик незавершенных транзакций наследования приоритетов.

Объявления и описания членов структуры находятся в файле:

- include/mutex.h

4.7 _pcounter_t

Счетчик захваченных ресурсов.

#include "include/pcounter.h"

- count_t counter [BITS_IN_INDEX_T]
- index_t index

4.7.1

Счетчик захваченных ресурсов.

При использовании опции CONFIG_USE_HIGHEST_LOCKER используется для пересчета захваченных процессом ресурсов.

4.7.2

4.7.2.1 count_t counter[BITS_IN_INDEX_T]

Массив счетчиков.

4.7.2.2 index_t index

Индекс для ускорения поиска.

Объявления и описания членов структуры находятся в файле:

- include/pcounter.h

4.8 _pool_t

Пул;.

#include "include/gitem.h"

- group_t * top
- group_t * bot

4.8.1

Пул;.

Стек для хранения неиспользуемых групп.

4.8.2

4.8.2.1 group_t* top

Вершина пула.

4.8.2.2 group_t* bot

Дно пула.

Объявления и описания членов структуры находятся в файле:

- include/gitem.h

4.9 _proc_t

Процесс.

#include "include/proc.h"

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

4.9.1

Процесс.

В разных ОС это называется по разному: процесс, поток, задача и пр., суть такова: это независимый поток исполнения инструкций процессора.

То есть это исполняющийся кусок твоей программы, у которого есть своя собственная «main» (смотри поле pmain), и эта «main» может быть написана так, как будто других процессов нет!

Можно использовать 1 функцию pmain для нескольких процессов, каждый запущенный экземпляр pmain не зависит от других, но есть одно но.

> Осторожно со статическими переменными, они будут общими для всех запущенных экземпляров, доступ к ним необходимо организовывать только с помощью средств синхронизации процессов.

4.9.2

4.9.2.1 gitem_t parent

Родитель - gitem_t.

4.9.2.2 flag_t flags

Флаги (для ускорения анализа состояния процесса).

4.9.2.3 prio_t base_prio

Базовый приоритет.

4.9.2.4 pcounter_t lres

Счетчик захваченных ресурсов.

4.9.2.5 timer_t time_quant

Квант времени процесса.

4.9.2.6 timer_t timer

Таймер процесса, для процессов жесткого реального времени используется как watchdog.

4.9.2.7 void* buf

Указатель на хранилище для передачи данных через IPC.

4.9.2.8 code_t pmain

Главная функция процесса.

4.9.2.9 code_t sv_hook

Хук, исполняется планировщиком после сохранения контекста процесса.

4.9.2.10 code_t rs_hook

Хук, исполняется планировщиком перед восстановлением контекста процесса.

4.9.2.11 void* arg

Аргумент для pmain, sv_hook, rs_hook, может хранить ссылку на локальные данные конкретного экземпляра процесса.

4.9.2.12 stack_t* sstart

Указатель на дно стека экземпляра процесса.

4.9.2.13 stack_t* spointer

Указатель на вершину стека экземпляра процесса.

Объявления и описания членов структуры находятся в файле:

- include/proc.h

4.10 _sched_t

Планировщик.

#include "include/sched.h"

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

4.10.1

Планировщик.

Планировщик содержит информацию о процессах, запущенных на процессоре (процессорном ядре).

4.10.2

4.10.2.1 proc_t* current_proc

Текущий процесс.

4.10.2.2 xlist_t* ready

Указатель на список готовых к выполнению процессов.

4.10.2.3 xlist_t* expired

Указатель на список процессов, исчерпавших свой квант времени.

4.10.2.4 xlist_t plist[2]

Сами списки процесов.

4.10.2.5 count_t nested_crit_sec

Счетчик вложенности критических секций.

Объявления и описания членов структуры находятся в файле:

- include/sched.h

4.11 _sem_t

Счетный семафор.

#include "include/sem.h"

- xlist_t wait
- count_t counter

4.11.1

Счетный семафор.

Счетные семафоры используются для синхронизации процессов. Не рекомендуется их использовать для организации доступа к общим ресурсам, т.к. здесь нет управления приоритетами. Счетный семафор может быть захвачен 1 процессом, а освобожден другим.

4.11.2

4.11.2.1 xlist_t wait

Потомок списка, да.

4.11.2.2 count_t counter

Счетчик семафора.

Объявления и описания членов структуры находятся в файле:

- include/sem.h

4.12 _sig_t

Сигнал.

#include "include/sig.h"

- gxlist_t wait
- gxlist_t wakeup

4.12.1

Сигнал.

Сигналы используются для синхронизации процессов по событиям. Процесс может встать в список ожидания сигнала. Другой процесс, или обработчик прерывания может подать сигнал и возобновить выполнение 1 или всех процессов, ожидающих этот сигнал.

4.12.2

4.12.2.1 gxlist_t wait

Список ожидающих процессов.

4.12.2.2 gxlist_t wakeup

Список "пробуждаемых" процессов.

Объявления и описания членов структуры находятся в файле:

- include/sig.h

4.13 _xlist_t

Список с приоритетами.

#include "include/xlist.h"

- item_t * item [BITS_IN_INDEX_T]
- index_t index

4.13.1

Список с приоритетами.

Такой список хранит ссылки на структуры типа item_t. Фактически в нем будут храниться ссылки на элементы типа #pitem_t.

4.13.2

4.13.2.1 item_t* item[BITS_IN_INDEX_T]

Массив указателей на элементы.

4.13.2.2 index_t index

Индекс, показывает, где в массиве ненулевые указатели.

Объявления и описания членов структуры находятся в файле:

- include/xlist.h

4.14 ipc_exchange_arg_t

Параметр системного вызова SYSCALL_IPC_EXCHANGE.

- ipc_send_arg_t send
- ipc_data_t * receive

4.14.1

Параметр системного вызова SYSCALL_IPC_EXCHANGE.

4.14.2

4.14.2.1 ipc_send_arg_t send

Родитель.

4.14.2.2 ipc_data_t* receive

Указатель на хранилище принимаемых данных.

Объявления и описания членов структуры находятся в файле:

- kernel/ipc.c

4.15 ipc_send_arg_t

Параметр системного вызова SYSCALL_IPC_SEND.

- proc_t * proc
- bool_t ret
- ipc_data_t ipc_data

4.15.1

Параметр системного вызова SYSCALL_IPC_SEND.

4.15.2

4.15.2.1 proc_t* proc

указатель на процесс-адресат.

4.15.2.2 bool_t ret

хранилище результата выполнения операции.

4.15.2.3 ipc_data_t ipc_data

данные для передачи.

Объявления и описания членов структуры находятся в файле:

- kernel/ipc.c

4.16 mutex_lock_arg_t

Параметр системных вызовов SYSCALL_MUTEX_LOCK и SYSCALL_MUTEX_TRY_LOCK.

- mutex_t * mutex
- bool_t ret

4.16.1

Параметр системных вызовов SYSCALL_MUTEX_LOCK и SYSCALL_MUTEX_TRY_LOCK.

4.16.2

4.16.2.1 mutex_t* mutex

указатель на мьютекс.

4.16.2.2 bool_t ret

хранилище результата выполнения операции.

Объявления и описания членов структуры находятся в файле:

- kernel/mutex.c

4.17 proc_runtime_arg_t

Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

- proc_t * proc
- bool_t ret

4.17.1

Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

4.17.2

4.17.2.1 proc_t* proc

Указатель на процесс.

4.17.2.2 bool_t ret

Результат выполнения системного вызова.

Объявления и описания членов структуры находятся в файле:

- kernel/proc.c

4.18 proc_set_prio_arg_t

Параметр системного вызова SYSCALL_PROC_SET_PRIO.

- proc_t * proc
- prio_t prio

4.18.1

Параметр системного вызова SYSCALL_PROC_SET_PRIO.

4.18.2

4.18.2.1 proc_t* proc

Указатель на процесс.

4.18.2.2 prio_t prio

Приоритет.

Объявления и описания членов структуры находятся в файле:

- kernel/proc.c

4.19 sem_lock_arg_t

Параметр системных вызовов SYSCALL_SEM_LOCK и SYSCALL_SEM_TRY_LOCK.

- sem_t * sem
- bool_t ret

4.19.1

Параметр системных вызовов SYSCALL_SEM_LOCK и SYSCALL_SEM_TRY_LOCK.

4.19.2

4.19.2.1 sem_t* sem

указатель на семафор.

4.19.2.2 bool_t ret

хранилище результата выполнения операции.

Объявления и описания членов структуры находятся в файле:

- kernel/sem.c

5

5.1 include/bugurt.h

Главный заголовочный файл.

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

- #define SPIN_INIT(arg)
  Макрос-обертка.
- #define SPIN_LOCK(arg)
  Макрос-обертка.
- #define SPIN_FREE(arg)
  Макрос-обертка.
- #define RESCHED_PROC(proc) resched()
  Макрос-обертка.

- typedef void(* code_t)(void *)
  Исполняемый код.

- void resched (void)

  Перепланировка.
- void disable_interrupts (void)

  Запрет прерываний.
- void enable_interrupts (void)

  Разрешение прерываний.
- proc_t * current_proc (void)

  Текущий процесс.
- stack_t * proc_stack_init (stack_t *sstart, code_t pmain, void *arg, void(*return_address)(void))

  Инициализация стека процесса.
- void init_bugurt (void)

  Инициализация Ядра.
- void start_bugurt (void)

  Запуск Ядра.
- void syscall_bugurt (syscall_t num, void *arg)

  Системный вызов.

5.1.1

Главный заголовочный файл. В этот файл включены все заголовочные файлы BuguRTOS. В свою очередь все исходные тексты включают этот файл.

5.1.2

5.1.2.1 #define SPIN_INIT(*arg*)

Макрос-обертка.

Обертка инициализации спин-блокировки arg->lock, на однопроцессорной системе - пустой макрос.

5.1.2.2 #define SPIN_LOCK(*arg*)

Макрос-обертка.

Обертка захвата спин-блокировки arg->lock, на однопроцессорной системе - пустой макрос.

5.1.2.3 #define SPIN_FREE(*arg*)

Макрос-обертка.

Обертка освобождения спин-блокировки arg->lock, на однопроцессорной системе - пустой макрос.

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

Макрос-обертка.

Обертка функции resched.

5.1.3

5.1.3.1 typedef void(* code_t)(void *)

Исполняемый код.

Указатель на функцию типа void, принимающую в качестве аргумента указатель типа void.

5.1.4

5.1.4.1 void resched (void)

Перепланировка.

Запускает перепланировку.

5.1.4.2 void disable_interrupts (void)

Запрет прерываний.

Глобальный запрет прерываний на системе.

5.1.4.3 void enable_interrupts (void)

Разрешение прерываний.

Глобальное разрешение прерываний на системе.

5.1.4.4 proc_t* current_proc (void)

Текущий процесс.

Текущий процесс.

Указатель на текущий процесс, исполняемый на локальном процессре.

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

Инициализация стека процесса.

Подготовка стека к запуску процесса. Делает так, что после восстановления контекста процесса происходит вызов функции code(arg).

sstart	Дно стека.
code	Функция, которая будет вызвана после восстановления контекста.
arg	Аргумент вызываемой функции.
return_address	адрес возврата из pmain.

Указатель на вершину подготовленного стека.

5.1.4.6 void init_bugurt (void)

Инициализация Ядра.

Подготовка Ядра к запуску.

5.1.4.7 void start_bugurt (void)

Запуск Ядра.

Запуск Ядра. После вызова этой функции можно ничего не писать - всеравно исполняться не будет.

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

Системный вызов.

Код Ядра всегда выполняется в контексте Ядра. Это нужно для экономии памяти в стеках процессов. Соответственно, если мы хотим выполнить какие либо операции над процессами, мьютексами,

семафорами, сигналами, то нам нужно "попросить" Ядро стелать эту работу.

Именно для этого существует функция syscall_bugurt, которая передает управление Ядру для выполнения треуемой работы.

num	номер системного вызова (что именно надо выполнить).
arg	аргумент системного вызова (над чем это надо выполнить).

5.2 include/crit_sec.h

Заголовок критических секций.

- #define ENTER_CRIT_SEC() enter_crit_sec()

  Макрос-обертка.
- #define EXIT_CRIT_SEC() exit_crit_sec()

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

  Выход из критической секции.

5.2.1

Заголовок критических секций. Критическая секция - область кода, в которой запрещены все прерывания. Критические секции используются, когда надо использовать общий ресурс в течение короткого ввремени.

Критические секции могут быть вложенные, в этом случае прерывания разрешаются, когда произошел выход из всех критических секций.

5.2.2

5.2.2.1 #define ENTER_CRIT_SEC( ) enter_crit_sec()

Макрос-обертка.

Вход в критическую секцию.

> Использовать в начале блока!
> Все локальные переменные должны быть объявлены до ENTER_CRIT_SEC

Макрос-обертка.

Выход из критической секции.

> Использовать в конце блока!

5.2.3

5.2.3.1 void enter_crit_sec (void)

Вход в критическую секцию.

5.2.3.2 void exit_crit_sec (void)

Выход из критической секции.

Вход в критическую секцию.

5.3 include/gitem.h

Заголовок элементов группированного списка. Типы данных: group_t, pool_t, gitem_t, gxlist_t.

- struct _group_t
  Группа элементов типа gitem_t.
- struct _pool_t
  Пул;.
- struct _gitem_t
  Элемент группированного списка.
- struct _gxlist_t

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

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

- void pool_init (pool_t *pool)
- void pool_merge (pool_t *src, pool_t *dst)
- void group_init (group_t *group, prio_t prio)
  Инициализация группы.
- void group_push (group_t *group, pool_t *pool)
  Положить группу в пул.
- group_t * group_pop (pool_t *pool)
  Взять группу из Пула.
- void gitem_init (gitem_t *gitem, prio_t prio)
  Инициализация объекта типа gitem_t.
- void gitem_insert (gitem_t *gitem, xlist_t *xlist)
  Вставка элемента типа gitem_t в список типа xlist_t без группировки.
- void gitem_insert_group (gitem_t *gitem, gxlist_t *gxlist)
  Вставка элемента типа gitem_t в список типа gxlist_t.
- void gitem_fast_cut (gitem_t *gitem)
  Быстро вырезать из списка.
- void gitem_cut (gitem_t *gitem)

Вырезать из списка.

- void gxlist_init (gxlist_t *gxlist)

  Инициация списка типа gxlist_t.

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

  Переносит все элементы из одного группированного списка в другой

5.3.1

Заголовок элементов группированного списка. Типы данных: group_t, pool_t, gitem_t, gxlist_t.

5.3.2

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

Статическая инициализация объекта типа pool_t

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

Статическая инициализация объекта типа group_t

p	Приоритет.

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

Статическая инициализация объекта типа gitem_t.

a	- Имя переменной типа gitem_t.
p	- Приоритет.

5.3.3

5.3.3.1 typedef struct _group_t group_t

Смотри _group_t;

5.3.3.2 typedef struct _pool_t pool_t

Смотри _pool_t;

5.3.3.3 typedef struct _gitem_t gitem_t

Элемент группированного списка с приоритетами.

5.3.3.4 typedef struct _gxlist_t gxlist_t

Группированный список с приоритетами (Расширение типа #xlixt_t)

5.3.4

5.3.4.1 void pool_init (pool_t * *pool*)

Инициализация объекта типа pool_t

pool	- указаткль на пул.

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

Слияние двух пулов.

src	Указатель на пул-источник.
dst	Указатель на пул-приемник.

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

Инициализация группы.

group	Указатель на объект group_t.
prio	Приоритет.

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

Положить группу в пул.

group	Указатель на объект group_t.
pool	Указатель на контейнер типа pool_t.

5.3.4.5 group_t* group_pop (pool_t * *pool*)

Взять группу из Пула.

pool	Указатель на контейнер типа pool_t.

Указатель на объект group_t, который был взят из пула.

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

Инициализация объекта типа gitem_t.

gitem	Указатель на объект gitem_t.
prio	Приоритет элемента.

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

Вставка элемента типа gitem_t в список типа xlist_t без группировки.

gitem	Указатель на объект gitem_t.
xlist	Указатель на список.

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

Вставка элемента типа gitem_t в список типа gxlist_t.

Вставляет в часть списка с приоритетом prio = gitem->group->prio, и переносит элемент в группу gxlist->parent.item[prio]->group, при этом gitem->group переходит в пул.

gitem	Указатель на объект gitem_t.
gxlist	Указатель на список.

5.3.4.9 void gitem_fast_cut (gitem_t * *gitem*)

Быстро вырезать из списка.

Вырезает объект типа gitem_t, из соответствующего списка, если объект был сгруппирован, то он вырезается из группы, и для него выделяется группа из Пула. При этом не обнуляется указатель gitem->group->link.

gitem	Указатель на объект gitem_t.

5.3.4.10 void gitem_cut (gitem_t * *gitem*)

Вырезать из списка.

Вызывает gitem_fast_cut, а потом таки обнуляет gitem->group->link

gitem	Указатель на объект gitem_t.

5.3.4.11 void gxlist_init (gxlist_t * *gxlist*)

Инициация списка типа gxlist_t.

gxlist	Указатель типа gxlist_t *.

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

Переносит все элементы из одного группированного списка в другой

Перенести все элементы из группированного списка типа gxlist_t в другой список типа gxlist_t. Выполняется за O(1) шагов.

source	Указатель на список из которого переносим элементы.
destignation	Указатель на список в который преносим элементы.

5.4 include/index.h

Заголовок функции поиска в бинарном индексе.

- prio_t index_search (index_t index)

  Поиск в бинарном индексе.

5.4.1

Заголовок функции поиска в бинарном индексе.

5.4.2

5.4.2.1 prio_t index_search (index_t *index*)

Поиск в бинарном индексе.

index	Бинарный индекс.

Наивысший (с минимальным значением) приоритет в индексе.

5.5 include/ipc.h

Заголовок IPC.

- void _ipc_wait (void *ipc_pointer)
  Переход процесса к ожиданию получения данных через IPC.
- ipc_data_t ipc_wait (void)
  Переход процесса к ожиданию получения данных через IPC.
- bool_t ipc_send (proc_t *proc, ipc_data_t ipc_data)
  Посылка данных процессу через IPC.
- bool_t ipc_send_isr (proc_t *proc, ipc_data_t ipc_data)
  Посылка данных процессу через IPC. Для вызова из обработчиков прерываний.
- bool_t _ipc_exchange (proc_t *proc, ipc_data_t send, ipc_data_t *receive)
  Посылка данных процессу через IPC, прием ответа через IPC.
- bool_t ipc_exchange (proc_t *proc, ipc_data_t send, ipc_data_t *receive)
  Посылка данных процессу через IPC, прием ответа через IPC.

5.5.1

Заголовок IPC.

5.5.2

5.5.2.1 void _ipc_wait (void * *ipc_pointer*)

Переход процесса к ожиданию получения данных через IPC.

Для внутреннего использования.

ipc_pointer	Указатель на хранилище для передаваемых данных.

5.5.2.2 ipc_data_t ipc_wait (void)

Переход процесса к ожиданию получения данных через IPC.

Данные.

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

Посылка данных процессу через IPC.

Проверяет, действительно ли процесс-адресат ждет получения данных через IPC. Если процесс-адресат действительно ждет, Ядро передает данные и запускает процесс-адресат.

proc	Указатель на процесс-адресат.
data	Данные для передачи.

1 - Если удалось передать данные, 0 - если нет.

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

Посылка данных процессу через IPC. Для вызова из обработчиков прерываний.

Для вызова из обработчиков прерываний!

Проверяет, действительно ли процесс-адресат ждет получения данных через IPC. Если процесс-адресат действительно ждет, Ядро передает данные и запускает процесс-адресат.

proc	Указатель на процесс-адресат.
data	Данные для передачи.

1 - Если удалось передать данные, 0 - если нет.

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

Посылка данных процессу через IPC, прием ответа через IPC.

Для внутреннего использования!

Проверяет, действительно ли процесс-адресат ждет получения данных через IPC. Если процесс-адресат действительно ждет, Ядро передает данные и запускает процесс-адресат, при этом процесс отправитель переходит к ожиданию данных через IPC.

proc	Указатель на процесс-адресат.
send	Данные для передачи.
receive	Указатель на хранилище данных для приема.

1 - если удалось передать данные, 0 - если нет.

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

Посылка данных процессу через IPC, прием ответа через IPC.

Проверяет, действительно ли процесс-адресат ждет получения данных через IPC. Если процесс-адресат действительно ждет, Ядро передает данные и запускает процесс-адресат, при этом процесс отправитель переходит к ожиданию данных через IPC.

proc	Указатель на процесс-адресат.
send	Данные для передачи.
receive	указатель на хранилище данных для приема.

1 - если удалось передать данные, 0 - если нет.

5.6 include/item.h

Заголовок элементов 2-связного списка.

- struct _item_t

  Элемент 2-связного списка.

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

- typedef struct _item_t item_t

- void item_init (item_t *item)

  Инициализация объекта типа item_t.
- void item_insert (item_t *item, item_t *head)

  Вставка элемента типа item_t в список.
- void item_cut (item_t *item)

  Вырезать элемент типа item_t из списка.

5.6.1

Заголовок элементов 2-связного списка.

5.6.2

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

Статическая инициализация объекта типа item_t.

a	Имя переменной типа item_t.

5.6.3

5.6.3.1 typedef struct _item_t item_t

Смотри _item_t;

5.6.4

5.6.4.1 void item_init (item_t * *item*)

Инициализация объекта типа item_t.

item	Указатель на объект item_t.

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

Вставка элемента типа item_t в список.

item	Указатель на объект типа item_t, который будем вставлять.
head	Указатель на голову списка типа item_t.

5.6.4.3 void item_cut (item_t * *item*)

Вырезать элемент типа item_t из списка.

item	Указатель на объект типа item_t, который будем вырезать.

5.7 include/kernel.h

Заголовок Ядра.

- struct _kernel_t
  Ядро BuguRTOS.

- typedef struct _kernel_t kernel_t

- void kernel_init (void)

  Инициализация Ядра.
- void idle_main (void *arg)

  Главная функция процесса холостого хода.

- kernel_t kernel

  Ядро BuguRTOS.

5.7.1

Заголовок Ядра.

5.7.2

5.7.2.1 typedef struct _kernel_t kernel_t

Смотри _kernel_t;

5.7.3

5.7.3.1 void kernel_init (void)

Инициализация Ядра.

Готовит ядро к запуску.

5.7.3.2 void idle_main (void * *arg*)

Главная функция процесса холостого хода.

Можно использовать встроенную функцию, а можно определить ее самому. Из idle_main можно работать с программными таймерами, подавать сигналы, ОСВОБОЖДАТЬ семафоры.

> Ни в коем случае нельзя делать return, останавливать процесс idle, захватывать семафомы и мьютексы из idle!!! Кто будет это все делать, того ждут Страшный суд, АдЪ и ПогибельЪ. Я предупредил!

arg	Указатель на аргумент.

5.7.4

5.7.4.1 kernel_t kernel

Ядро BuguRTOS.

Оно одно на всю систему!

5.8 include/mutex.h

Заголовок мьютекса.

- struct _mutex_t
  Мьютекс.

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

- typedef struct _mutex_t mutex_t

- void mutex_init_isr (mutex_t *mutex)
  Инициализация мьютекса из критической секции, или обработчика прерываний.
- void mutex_init (mutex_t *mutex)
  Инициализация мьютекса.
- bool_t mutex_lock (mutex_t *mutex)
  Захват мьютекса.
- bool_t mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса.
- void mutex_free (mutex_t *mutex)
  Освобождение мьютекса.
- bool_t _mutex_lock (mutex_t *mutex)
  Захват мьютекса, для внутреннего использования.
- bool_t _mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса, для внутреннего использования.
- bool_t _mutex_free (mutex_t *mutex)
  Освобождение мьютекса, для внутреннего использования.

5.8.1

Заголовок мьютекса.

5.8.2

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

Считает приоритет мьютекса.

5.8.3

5.8.3.1 typedef struct _mutex_t mutex_t

Смотри _mutex_t;

5.8.4

5.8.4.1 void mutex_init_isr (mutex_t * *mutex*)

Инициализация мьютекса из критической секции, или обработчика прерываний.

Да, инициировать из обработчика прерывания можно!

mutex	Указатель на мьютекс.

5.8.4.2 void mutex_init (mutex_t * *mutex*)

Инициализация мьютекса.

mutex	Указатель на мьютекс.

5.8.4.3 bool_t mutex_lock (mutex_t * *mutex*)

Захват мьютекса.

Если мьютекс свободен - процесс захватывает его и продолжает выполняться, если уже занят - процесс останавливается и записывается в список ожидающих.

mutex	Указатель на мьютекс.

1 - если удалось захватить без ожидания, 0 - если пришлось ждать.

5.8.4.4 bool_t mutex_try_lock (mutex_t * *mutex*)

Попытка захвата мьютекса.

Если мьютекс свободен - процесс захватывает его и продолжает выполняться, если уже занят - процесс продолжает выполнение.

mutex	Указатель на мьютекс.

1 - если уддалось захватить, 0 - если не удалось.

5.8.4.5 void mutex_free (mutex_t * *mutex*)

Освобождение мьютекса.

Если список ожидающих процессов пуст - вызывающий процесс освобождает мьютекс, если список не пуст - ставит на выполнение голову списка. Также происходит обработка флагов, при необходимости вызывающий процесс останавливается.

mutex	Указатель на мьютекс.

5.8.4.6 bool_t _mutex_lock (mutex_t * *mutex*)

Захват мьютекса, для внутреннего использования.

Если мьютекс свободен - процесс захватывает его и продолжает выполняться, если уже занят - процесс останавливается и записывается в список ожидающих.

mutex	Указатель на мьютекс.

1 - если удалось захватить без ожидания, 0 - если пришлось ждать.

5.8.4.7 bool_t _mutex_try_lock (mutex_t * *mutex*)

Попытка захвата мьютекса, для внутреннего использования.

Если мьютекс свободен - процесс захватывает его и продолжает выполняться, если уже занят - процесс продолжает выполнение.

mutex	Указатель на мьютекс.

1 - если уддалось захватить, 0 - если не удалось.

5.8.4.8 bool_t _mutex_free (mutex_t * *mutex*)

Освобождение мьютекса, для внутреннего использования.

Если список ожидающих процессов пуст - вызывающий процесс освобождает мьютекс, если список не пуст - ставит на выполнение голову списка. Также происходит обработка флагов, при необходимости вызывающий процесс останавливается.

mutex	Указатель на мьютекс.

5.9 include/pcounter.h

Заголовок счетчиков захваченных ресурсов.

- struct _pcounter_t
  Счетчик захваченных ресурсов.

- typedef struct _pcounter_t pcounter_t

- void pcounter_init (pcounter_t *pcounter)
  Инициализация счетчика.
- void pcounter_inc (pcounter_t *pcounter, prio_t prio)

Инкремент счетчика.

- index_t pcounter_dec (pcounter_t *pcounter, prio_t prio)

  Декремент счетчика.
- void pcounter_plus (pcounter_t *pcounter, prio_t prio, count_t count)

  Увеличение счетчика на произвольное количество единиц.
- index_t pcounter_minus (pcounter_t *pcounter, prio_t prio, count_t count)

  Уменьшение счетчика на произвольное количество единиц.

5.9.1

Заголовок счетчиков захваченных ресурсов.

5.9.2

5.9.2.1 typedef struct _pcounter_t pcounter_t

Смотри _pcounter_t;

5.9.3

5.9.3.1 void pcounter_init (pcounter_t * *pcounter*)

Инициализация счетчика.

pcounter	Указатель на счетчик.

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

Инкремент счетчика.

pcounter	Указатель на счетчик.
prio	Приоритет.

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

Декремент счетчика.

pcounter	Указатель на счетчик.
prio	Приоритет.

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

Увеличение счетчика на произвольное количество единиц.

pcounter	Указатель на счетчик.
prio	Приоритет.
count	Количество единиц.

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

Уменьшение счетчика на произвольное количество единиц.

pcounter	Указатель на счетчик.
prio	Приоритет.
count	Количество единиц.

0 - если соответствующая часть счетчика обнулилась, не 0 - в других случаях.

5.10 include/proc.h

Заголовок процессов.

- struct _proc_t
  Процесс.

- #define PROC_LRES_INIT(a) pcounter_init(&a->lres)
  Макрос-обертка.
- #define PROC_LRES_INC(a, b) pcounter_inc(&a->lres, b)
  Макрос-обертка.
- #define PROC_LRES_DEC(a, b) pcounter_dec(&a->lres, b)
  Макрос-обертка.
- #define PROC_FLG_RT ((flag_t)0x80)
  Флаг реального времени.
- #define PROC_FLG_MUTEX ((flag_t)0x40)
  Флаг захвата мьютексов.
- #define PROC_FLG_SEM ((flag_t)0x20)
  Флаг захвата семафора.
- #define PROC_FLG_PRE_STOP ((flag_t)0x10)
  Флаг запроса останова.
- #define PROC_FLG_LOCK_MASK ((flag_t)(PROC_FLG_MUTEX|PROC_FLG_SEM))
  Маска PROC_FLG_MUTEX или PROC_FLG_SEM.
- #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)
  Маска очистки состояния исполнения процесса.
- #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)
  Маска очистки состояния исполнения процесса.
- #define PROC_STATE_MASK ((flag_t)0x0F)
  Маска состояния исполнения процесса.
- #define PROC_STATE_RESTART_MASK ((flag_t)0xC)
  Маска проверки состояния процесса.
- #define PROC_STATE_RUN_MASK ((flag_t)0x7)
  Маска проверки состояния процесса.
- #define PROC_STATE_WAIT_MASK ((flag_t)0x8)
  Маска проверки состояния процесса.

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

  Макрос проверки условий останова по флагу PROC_FLG_PRE_STOP.
- #define PROC_RUN_TEST(a) ((a->flags & PROC_STATE_RUN_MASK) >= PROC_STATE_READY)

  Проверяет, запущен ли процесс.
- #define PROC_GET_STATE(a) (a->flags & PROC_STATE_MASK)

  Читает состояние процесса.
- #define PROC_SET_STATE(a, b) (a->flags &= PROC_STATE_CLEAR_MASK, proc->flags |= b)

  Устанавливает состояние процесса.
- #define PROC_IPC_TEST(a) (PROC_GET_STATE(a) == PROC_STATE_W_IPC)

  Проверяет ждет ли процесс IPC.
- #define PROC_PRIO_LOWEST ((prio_t)BITS_IN_INDEX_T - (prio_t)1)

  Низший приоритет.

- typedef struct _proc_t proc_t

- 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)

  Инициализация процесса из обработчика прерывания, либо из критической секции.
- 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)

  Инициализация процесса.
- void proc_run_wrapper (proc_t *proc)

  Обертка для запуска процессов.
- void proc_terminate (void)

  Завершение работы процесса после возврата из proc->pmain. Для внутреннего использования.
- void _proc_terminate (void)

  Завершение работы процесса после возврата из proc->pmain. Для внутреннего использования.
- bool_t proc_run (proc_t *proc)

  Запуск процесса.
- bool_t proc_run_isr (proc_t *proc)

Запуск процесса из критической секции, либо обработчика прерывания.

- bool_t proc_restart (proc_t *proc)

  Перезапуск процесса.
- bool_t proc_restart_isr (proc_t *proc)

  Перезапуск процесса из критической секции или обработчика прерывания.
- bool_t proc_stop (proc_t *proc)

  Останов процесса.
- bool_t proc_stop_isr (proc_t *proc)

  Останов процесса из критической секции или обработчика прерывания.
- void proc_self_stop (void)

  Самоостанов процесса.
- void _proc_self_stop (void)

  Самоостанов процесса (для внутреннего использования).
- void proc_reset_watchdog (void)

  Сброс watchdog для процесса реального времени.
- void _proc_reset_watchdog (void)

  Сброс watchdog для процесса реального времени из обработчика прерывания (для внутреннего использования).
- void _proc_dont_stop (proc_t *proc, flag_t flags)

  Запуск остановленного процесса с флагом PROC_FLG_PRE_STOP. Для внутреннего использования.
- void _proc_cut_and_run (proc_t *proc, flag_t state)

  Вырезать проесс из списка ожидающих и его запуск. Для внутреннего использования.
- void _proc_prio_propagate (proc_t *proc)

  Передача приоритетов по цепи заблокированных процессов. Для внутреннего использования.
- void _proc_stop_flags_set (proc_t *proc, flag_t mask)

  "Низкоуровневый" останов процесса с установкой флагов, для внутреннего использования.
- void _proc_flag_stop (flag_t mask)

  Останов процесса по флагу PROC_FLG_PRE_STOP из критической секции или обработчика прерывания, для внутреннего использования.
- void proc_flag_stop (flag_t mask)

  Останов процесса по флагу PROC_FLG_PRE_STOP.
- void _proc_prio_control_stoped (proc_t *proc)

  Управление приоритетом процесса, для внутреннего использования.
- void proc_set_prio (proc_t *proc, prio_t prio)

  Управление приоритетом процесса.
- void _proc_set_prio (proc_t *proc, prio_t prio)

  Управление приоритетом процесса. Для внктреннего использования.

5.10.1

Заголовок процессов.

5.10.2

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

Макрос-обертка.

Инициирует поле proc->lres процесса.

a	указатель на процесс.

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

Макрос-обертка.

Инкремент счетчика захваченных мьютексов.

a	указатель на процесс.
b	приоритет захваченного мьютекса, если используется протокол highest locker.

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

Макрос-обертка.

Декремент счетчика захваченных мьютексов.

a	указатель на процесс.
b	приоритет захваченного мьютекса, если используется протокол highest locker.

5.10.2.4 #define PROC_FLG_RT ((flag_t)0x80)

Флаг реального времени.

Для этого процесса используется политика планирования жесткого реального времени.

5.10.2.5 #define PROC_FLG_MUTEX ((flag_t)0x40)

Флаг захвата мьютексов.

Процесс удерживает мьютекс.

5.10.2.6 #define PROC_FLG_SEM ((flag_t)0x20)

Флаг захвата семафора.

Выставляется при вызове sem_lock и при удачном вызове sem_try_lock. Обнулять необходимо вручную, при освобождении общего ресурса, охраняемого семафором. Обнуляется вызовом proc_flag_stop.

5.10.2.7 #define PROC_FLG_PRE_STOP ((flag_t)0x10)

Флаг запроса останова.

Произошел запрос на останов процесса. Процесс будет остановлен при первой же возможности.

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

Маска PROC_FLG_MUTEX или PROC_FLG_SEM.

Нужна, чтобы определить, удерживает ли процесс общие ресурсы.

5.10.2.9 #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

Маска очистки состояния исполнения процесса.

Нужна, чтобы очистить биты стотояния выполнения процесса в поле proc->flags.

5.10.2.10 #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)

Маска очистки состояния исполнения процесса.

Нужна, чтобы очистить младшие биты стотояния выполнения процесса в поле proc->flags.

5.10.2.11 #define PROC_STATE_RESTART_MASK ((flag_t)0xC)

Маска проверки состояния процесса.

Используется функциями proc_restart и proc_restart_isr, для проверки возможности перезапуска.

5.10.2.12 #define PROC_STATE_RUN_MASK ((flag_t)0x7)

Маска проверки состояния процесса.

Используется для того, чтобы проверить, запущен ли процесс.

5.10.2.13 #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

Маска проверки состояния процесса.

Используется для того, чтобы проверить, ожидате ли процесс получения семафора, мьютекса, сообщения через IPC или сигнала.

5.10.2.14 #define PROC_STATE_STOPED ((flag_t)0x0)

Начальное состояние, остановлен.

5.10.2.15 #define PROC_STATE_END ((flag_t)0x1)

Завершен.

5.10.2.16 #define PROC_STATE_W_WD_STOPED ((flag_t)0x2)

Остановлен по вачдог в состоянии W_RUNNING.

5.10.2.17 #define PROC_STATE_WD_STOPED ((flag_t)0x3)

Остановлен по вачдог.

5.10.2.18 #define PROC_STATE_DEAD ((flag_t)0x4)

Завершен до освобождения общих ресурсов.

5.10.2.19 #define PROC_STATE_PCHANGE ((flag_t)0x5)

Запущен при смене приоритета

5.10.2.20 #define PROC_STATE_READY ((flag_t)0x6)

Готов к выполнению.

5.10.2.21 #define PROC_STATE_RUNNING ((flag_t)0x7)

Выполняется.

5.10.2.22 #define PROC_STATE_W_MUT ((flag_t)0x8)

Ожидает мьютекса.

5.10.2.23 #define PROC_STATE_W_SEM ((flag_t)0x9)

Ожидает семафора.

5.10.2.24 #define PROC_STATE_W_SIG ((flag_t)0xA)

Ожидает сигнала.

5.10.2.25 #define PROC_STATE_W_IPC ((flag_t)0xB)

Ожидает IPC.

5.10.2.26 #define PROC_STATE_W_DEAD ((flag_t)0xC)

Остановлен по вачдог в состоянии W_RUNNING до освобождения общих ресурсов.

5.10.2.27 #define PROC_STATE_W_PCHANGE ((flag_t)0xD)

Остановлен для смены приоритета.

5.10.2.28 #define PROC_STATE_W_READY ((flag_t)0xE)

Готов к выполнению (специальное).

5.10.2.29 #define PROC_STATE_W_RUNNING ((flag_t)0xF)

Выполняется (специальное).

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

Макрос проверки условий останова по флагу PROC_FLG_PRE_STOP.

Используется для проверки процессов на возможность останова по флагу PROC_FLG_PRE_STOP. Процесс не должен удерживать общие ресурсы в момент останова по флагу.

5.10.3

5.10.3.1 typedef struct _proc_t proc_t

Смотри _proc_t;

5.10.4

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

Инициализация процесса из обработчика прерывания, либо из критической секции.

proc	Указатель на инициируемый процесс.
pmain	Указатель на главную функцию процесса.
sv_hook	Указатель на хук proc->sv_hook.
rs_hook	Указатель на хук proc->rs_hook.
arg	Указатель на аргумент.
sstart	Указатель на дно стека процесса.
prio	Приоритет.
time_quant	Квант времени.
is_rt	Флаг реального времени, если true, значит процесс будет иметь поведение RT.

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

Инициализация процесса.

proc	Указатель на инициируемый процесс.
pmain	Указатель на главную функцию процесса.
sv_hook	Указатель на хук proc->sv_hook.
rs_hook	Указатель на хук proc->rs_hook.
arg	Указатель на аргумент.
sstart	Указатель на дно стека процесса.
prio	Приоритет.
time_quant	Квант времени.
is_rt	Флаг реального времени, если true, значит процесс будет иметь поведение RT.

5.10.4.3 void proc_run_wrapper (proc_t * *proc*)

Обертка для запуска процессов.

Эта функция вызывает proc->pmain(proc->arg), и если происходит возврат из pmain, то proc_run_wrapper корректно завершает процесс.

proc	- Указатель на запускаемый процесс.

5.10.4.4 bool_t proc_run (proc_t * *proc*)

Запуск процесса.

Ставит процесс в список готовых к выполнению, если можно (процесс не запущен, еще не завершил работу, не был "убит"), и производит перепланировку.

proc	- Указатель на запускаемый процесс.

1 - если процесс был вставлен в список готовых к выполнению, 0 во всех остальных случаях.

5.10.4.5 bool_t proc_run_isr (proc_t * *proc*)

Запуск процесса из критической секции, либо обработчика прерывания.

Ставит процесс в список готовых к выполнению, если можно (процесс не запущен, еще не завершил работу, не был "убит"), и производит перепланировку.

proc	- Указатель на запускаемый процесс.

1 - если процесс был вставлен в список готовых к выполнению, 0 во всех остальных случаях.

5.10.4.6 bool_t proc_restart (proc_t * *proc*)

Перезапуск процесса.

Если можно (процесс не запущен, завершил работу, не был "убит"), приводит структуру proc в состояние, которое было после вызова proc_init, и ставит процесс в список готовых к выполнению, и производит перепланировку.

proc	- Указатель на запускаемый процесс.

1 - если процесс был вставлен в список готовых к выполнению, 0 во всех остальных случаях.

5.10.4.7 bool_t proc_restart_isr (proc_t * *proc*)

Перезапуск процесса из критической секции или обработчика прерывания.

Если можно (процесс не запущен, завершил работу, не был "убит"), приводит структуру proc в состояние, которое было после вызова proc_init, и ставит процесс в список готовых к выполнению, производит перепланировку.

proc	- Указатель на запускаемый процесс.

1 - если процесс был вставлен в список готовых к выполнению, 0 во всех остальных случаях.

5.10.4.8 bool_t proc_stop (proc_t * *proc*)

Останов процесса.

Вырезает процесс из списка готовых к выполнению и производит перепланировку.

proc	- Указатель на останавливаемый процесс.

1 - если процесс был вырезан из списка готовых к выполнению, 0 во всех остальных случаях.

5.10.4.9 bool_t proc_stop_isr (proc_t * *proc*)

Останов процесса из критической секции или обработчика прерывания.

Вырезает процесс из списка готовых к выполнению и производит перепланировку.

proc	- Указатель на останавливаемый процесс.

1 - если процесс был вырезан из списка готовых к выполнению, 0 во всех остальных случаях.

5.10.4.10 void proc_self_stop (void)

Самоостанов процесса.

Вырезает вызывающий процесс из списка готовых к выполнению и производит перепланировку.

5.10.4.11 void _proc_self_stop (void)

Самоостанов процесса (для внутреннего использования).

Вырезает вызывающий процесс из списка готовых к выполнению и производит перепланировку.

5.10.4.12 void proc_reset_watchdog (void)

Сброс watchdog для процесса реального времени.

Если функцию вызывает процесс реального времени, то функция сбрасывает его таймер. Если процесс завис, и таймер не был вовремя сброшен, то планировщик остановит такой процесс и передаст

управление другому.

5.10.4.13 void _proc_reset_watchdog (void)

Сброс watchdog для процесса реального времени из обработчика прерывания (для внутреннего использования).

Если функцию вызывает процесс реального времени, то функция сбрасывает его таймер. Если процесс завис, и таймер не был вовремя сброшен, то планировщик остановит такой процесс и передаст управление другому.

5.10.4.14 void _proc_prio_control_stoped (proc_t * *proc*)

Управление приоритетом процесса, для внутреннего использования.

Используется совместно с опцией CONFIG_USE_HIGHEST_LOCKER. Процесс должен быть остановлен на момент вызова.

proc	- Указатель на процесс.

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

Управление приоритетом процесса.

Устанавливает приоритет процесса, находящегося в любом состоянии.

proc	- Указатель на процесс.
prio	- Новое значение приоритета.

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

Управление приоритетом процесса. Для внктреннего использования.

Устанавливает приоритет процесса, находящегося в любом состоянии.

proc	- Указатель на процесс.
prio	- Новое значение приоритета.

5.11 include/sched.h

Заголовок планировщика

- struct _sched_t
  Планировщик.

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

- typedef struct _sched_t sched_t

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

  Инициализация планировщика.
- void sched_schedule (void)

  Функция планирования.
- void sched_reschedule (void)

  Функция перепланирования.
- void sched_proc_run (proc_t *proc, flag_t state)

  "Низкоуровневый" запуск процесса, для внутреннего использования.
- void sched_proc_stop (proc_t *proc)

  "Низкоуровневый" останов процесса, для внутреннего использования.
- bool_t _sched_proc_yeld (void)

  Передача управления следующему процессу (для внутреннего использования).
- bool_t sched_proc_yeld (void)

  Передача управления следующему процессу.

5.11.1

Заголовок планировщика

Все функции в этом файле для внутреннего использования!!!

5.11.2

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

Макрос-обертка.

Обертка инициализации переменной sched в функциях sched_schedule и sched_reschedule.

5.11.3

5.11.3.1 typedef struct _sched_t sched_t

Смотри _sched_t;

5.11.4

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

Инициализация планировщика.

Готовит планировщик к запуску.

sched	- Указатель на планировщик.
idle	- Указатель на процесс холостого хода.

5.11.4.2 void sched_schedule (void)

Функция планирования.

Переключает процессы в обработчике прерывания системного таймера.

5.11.4.3 void sched_reschedule (void)

Функция перепланирования.

Переключает процессы в случае необходимости.

5.11.4.4 bool_t _sched_proc_yeld (void)

Передача управления следующему процессу (для внутреннего использования).

Передает управление следующему процессу, если такой процесс есть.

0 если нет других выполняющихся процессов, не 0 - если есть.

5.11.4.5 bool_t sched_proc_yeld (void)

Передача управления следующему процессу.

Передает управление следующему процессу, если такой процесс есть.

0 если нет других выполняющихся процессов, не 0 - если есть.

5.12 include/sem.h

Заголовок счетных семафоров.

- struct _sem_t
  Счетный семафор.

- typedef struct _sem_t sem_t

- void sem_init_isr (sem_t *sem, count_t count)
  Инициализация семафора из обработчика прерывания или критической секции.
- void sem_init (sem_t *sem, count_t count)
  Инициализация семафора.
- bool_t sem_lock (sem_t *sem)
  Захват семафора.
- bool_t sem_try_lock (sem_t *sem)
  Попытка захвата семафора.
- void sem_free (sem_t *sem)
  Освобождение семафора.
- void sem_free_isr (sem_t *sem)
  Освобождение для использования в обработчиках прерываний
- bool_t _sem_lock (sem_t *sem)
  Захват семафора для внутреннего использования.
- bool_t _sem_try_lock (sem_t *sem)
  Попытка захвата семафора для внутреннего использования.

5.12.1

Заголовок счетных семафоров.

5.12.2

5.12.2.1 typedef struct _sem_t sem_t

Смотри _sem_t;

5.12.3

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

Инициализация семафора из обработчика прерывания или критической секции.

sem	Указатель на семафор.
count	Начальное значение счетчика.

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

Инициализация семафора.

sem	Указатель на семафор.
count	Начальное значение счетчика.

5.12.3.3 bool_t sem_lock (sem_t * *sem*)

Захват семафора.

Если значение счетчика семафора больше 0, то процесс уменьшает счетчик семафора на 1 и продолжает выполняться. Если значение счетчика семафора равно 0, процесс останавливается и встает в список ожидающих освобождения семафора.

sem	Указатель на семафор.

1 если удалось захватить семафор без ожидания, 0 если не удалось.

5.12.3.4 bool_t sem_try_lock (sem_t * *sem*)

Попытка захвата семафора.

Если значение счетчика семафора больше 0, то процесс уменьшает счетчик семафора на 1 и продолжает выполняться. Если значение счетчика семафора равно 0, процесс просто продолжает выполняться.

sem	Указатель на семафор.

1 если удалось захватить семафор, 0 если не удалось.

5.12.3.5 void sem_free (sem_t * *sem*)

Освобождение семафора.

Если список ожидающих захвата семафора пуст, то счетчик семафора увеличиваем на 1. Если не пуст - возобновляем работу головы списка.

sem	Указатель на семафор.

5.12.3.6 void sem_free_isr (sem_t * *sem*)

Освобождение для использования в обработчиках прерываний

Если список ожидающих захвата семафора пуст, то счетчик семафора увеличиваем на 1. Если не пуст - возобновляем работу головы списка.

sem	Указатель на семафор.

5.12.3.7 bool_t _sem_lock (sem_t * *sem*)

Захват семафора для внутреннего использования.

Если значение счетчика семафора больше 0, то процесс уменьшает счетчик семафора на 1 и продолжает выполняться. Если значение счетчика семафора равно 0, процесс останавливается и встает в список ожидающих освобождения семафора.

sem	Указатель на семафор.

1 если удалось захватить семафор без ожидания, 0 если не удалось.

5.12.3.8 bool_t _sem_try_lock (sem_t * *sem*)

Попытка захвата семафора для внутреннего использования.

Если значение счетчика семафора больше 0, то процесс уменьшает счетчик семафора на 1 и продолжает выполняться. Если значение счетчика семафора равно 0, процесс просто продолжает выполняться.

sem	Указатель на семафор.

1 если удалось захватить семафор, 0 если не удалось.

5.13 include/sig.h

Заголовок сигналов.

- struct _sig_t
  Сигнал.

- typedef struct _sig_t sig_t

- void sig_init_isr (sig_t *sig)
  Инициализация сигнала из обработчика прерывания или критической секции.
- void sig_init (sig_t *sig)
  Инициализация сигнала.
- void sig_wait (sig_t *sig)
  Встать в список ожидания сигнала.
- void _sig_wait_prologue (sig_t *sig)
  Встать в список ожидания сигнала, для внутреннего использования.
- void _sig_wait_epilogue (void)
  Эпилог ожидания сигнала. Для внутреннего использования.
- void sig_signal (sig_t *sig)
  Возобновить работу 1 процесса ожидающего сигнал.
- void sig_broadcast (sig_t *sig)
  Возобновить работу всех ожидающих процессов.
- void sig_signal_isr (sig_t *sig)
  Возобновить работу 1 процесса ожидающего сигнал из обработчика прерывания.
- void sig_broadcast_isr (sig_t *sig)
  Возобновить работу всех ожидающих процессов из обработчика прерывания.

5.13.1

Заголовок сигналов.

5.13.2

5.13.2.1 typedef struct _sig_t sig_t

Смотри _sig_t;

5.13.3

5.13.3.1 void sig_init_isr (sig_t * *sig*)

Инициализация сигнала из обработчика прерывания или критической секции.

sig	Указатель на сигнал.

5.13.3.2 void sig_init (sig_t * *sig*)

Инициализация сигнала.

sig	Указатель на сигнал.

5.13.3.3 void sig_wait (sig_t * *sig*)

Встать в список ожидания сигнала.

Останавливает вызвавший процесс и ставит его в список ожидания. На многопроцессорной системе при этом происходит предварительная балансировка нагрузки. После возобновления работы процесса делается попытка остановить его по флагу PROC_FLG_PRE_STOP.

sig	Указатель на сигнал.

5.13.3.4 void _sig_wait_prologue (sig_t * *sig*)

Встать в список ожидания сигнала, для внутреннего использования.

Останавливает вызвавший процесс и ставит его в список ожидания.

sig	Указатель на сигнал.

5.13.3.5 void _sig_wait_epilogue (void)

Эпилог ожидания сигнала. Для внутреннего использования.

При необходимости пробуждает следующий процесс в sig->wakeup;

5.13.3.6 void sig_signal (sig_t * *sig*)

Возобновить работу 1 процесса ожидающего сигнал.

На мнгогопроцессорной системе: Ищет в массиве статистики сигнала самое "нагруженное" ядро. Далее возобновляет работу головы списка ожидающих сигнал для этого ядра, при этом происходит балансировка нагрузци - запускаемый процесс будет выполняться на самом ненагруженном процессорном ядре из возможных.

На 1 процессорной системе: просто возобновляет работу голоы списка ожидающих.

sig	Указатель на сигнал.

5.13.3.7 void sig_broadcast (sig_t * *sig*)

Возобновить работу всех ожидающих процессов.

Возобновляет работу всех ожидающих процессов. Процессы в списках ожидания сигналов сгруппированы, что дает возможность за ограниченное время перенести весь список ожидающих в соотетствующий список готовых к выполнению.

sig	Указатель на сигнал.

5.13.3.8 void sig_signal_isr (sig_t * *sig*)

Возобновить работу 1 процесса ожидающего сигнал из обработчика прерывания.

На мнгогопроцессорной системе: Ищет в массиве статистики сигнала самое "нагруженное" ядро. Далее возобновляет работу головы списка ожидающих сигнал для этого ядра, при этом происходит балансировка нагрузци - запускаемый процесс будет выполняться на самом ненагруженном процессорном ядре из возможных.

На 1 процессорной системе: просто возобновляет работу голоы списка ожидающих.

sig	Указатель на сигнал.

5.13.3.9 void sig_broadcast_isr (sig_t * *sig*)

Возобновить работу всех ожидающих процессов из обработчика прерывания.

Возобновляет работу всех ожидающих процессов. Процессы в списках ожидания сигналов сгруппированы, что дает возможность за ограниченное время перенести весь список ожидающих в соответствующий список готовых к выполнению.

sig	Указатель на сигнал.

5.14 include/syscall.h

Заголовок системных вызовов.

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

- void do_syscall (void)

  Обработка системного вызова.
- void scall_proc_run (void *arg)

  Обработчик вызова SYSCALL_PROC_RUN.
- void scall_proc_restart (void *arg)

  Обработчик вызова SYSCALL_PROC_RESTART.

- void scall_proc_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_STOP.
- void scall_proc_self_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_SELF_STOP.
- void scall_sched_proc_yeld (void *arg)
  Обработчик вызова #SYSCALL_PROC_YELD.
- void scall_proc_terminate (void *arg)
  Обработчик вызова SYSCALL_PROC_TERMINATE.
- void scall_proc_flag_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_FLAG_STOP.
- void scall_proc_reset_watchdog (void *arg)
  Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.
- void scall_proc_set_prio (void *arg)
  Обработчик вызова SYSCALL_PROC_SET_PRIO.
- void scall_sig_wait (void *arg)
  Обработчик вызова SYSCALL_SIG_WAIT.
- void scall_sig_wakeup (void *arg)
  Обработчик вызова SYSCALL_SIG_WAKEUP.
- void scall_sig_signal (void *arg)
  Обработчик вызова SYSCALL_SIG_SIGNAL.
- void scall_sig_broadcast (void *arg)
  Обработчик вызова SYSCALL_SIG_BROADCAST.
- void scall_sem_lock (void *arg)
  Обработчик вызова SYSCALL_SEM_LOCK.
- void scall_sem_try_lock (void *arg)
  Обработчик вызова SYSCALL_SEM_TRY_LOCK.
- void scall_sem_free (void *arg)
  Обработчик вызова SYSCALL_SEM_FREE.
- void scall_mutex_lock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_LOCK.
- void scall_mutex_try_lock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_TRY_LOCK.
- void scall_mutex_free (void *arg)
  Обработчик вызова SYSCALL_MUTEX_FREE.
- void scall_ipc_wait (void *arg)
  Обработчик вызова SYSCALL_IPC_WAIT.
- void scall_ipc_send (void *arg)
  Обработчик вызова SYSCALL_IPC_SEND.
- void scall_ipc_exchange (void *arg)
  Обработчик вызова SYSCALL_IPC_EXCHANGE.
- void scall_user (void *arg)
  Обработчик вызова SYSCALL_USER.

- syscall_t syscall_num
  Обработка системного вызова.
- void * syscall_arg

5.14.1

Заголовок системных вызовов.

5.14.2

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

Запуск процесса.

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

Перезапуск процесса.

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

Останов процесса.

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

Самоостанов процесса.

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

Завершение работы процесса.

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

Останов процесса по флагу PROC_FLG_PRE_STOP.

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

Сброс watchdog процесса реального времени.

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

Установить приоритет процесса

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

Передача управления другому процессу.

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

Ожидание сигнала.

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

Обработка запуска по сигналу.

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

Подача сигнала одному процессу.

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

Подача сигнала всем ожидающим процессам.

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

Захват семафора.

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

Попытка захвата семафора.

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

Освобождение семафора.

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

Захват мьютекса.

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

Попытка захвата мьютекса.

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

Освобождение мьютекса.

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

Ожидание передачи данных.

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

Передача данных.

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

Обмен данными.

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

Пользовательский системный вызов.

5.14.3

5.14.3.1 void do_syscall (void)

Обработка системного вызова.

Зпускает обработчик системного вызова и передает ему аргумент.

5.14.3.2 void scall_proc_run (void * *arg*)

Обработчик вызова SYSCALL_PROC_RUN.

Пытается запустить процесс, вызывая proc_run_isr.

arg	указатель на структуру proc_runtime_arg_t.

5.14.3.3 void scall_proc_restart (void * *arg*)

Обработчик вызова SYSCALL_PROC_RESTART.

Пытается перезапустить процесс, вызывая proc_restart_isr.

arg	указатель на структуру proc_runtime_arg_t.

5.14.3.4 void scall_proc_stop (void * *arg*)

Обработчик вызова SYSCALL_PROC_STOP.

Пытается остановить процесс, вызывая proc_stop_isr.

arg	указатель на структуру proc_runtime_arg_t.

5.14.3.5 void scall_proc_self_stop (void * *arg*)

Обработчик вызова SYSCALL_PROC_SELF_STOP.

Останавливает вызывающий процесс.

arg	не используется.

5.14.3.6 void scall_sched_proc_yeld (void * *arg*)

Обработчик вызова #SYSCALL_PROC_YELD.

Передает управление следующему процессу.

arg	не используется.

5.14.3.7 void scall_proc_terminate (void * *arg*)

Обработчик вызова SYSCALL_PROC_TERMINATE.

Завершает выполенение процесса после выхода из pmain. Вызывает _proc_terminate.

arg	указатель на процесс.

5.14.3.8 void scall_proc_flag_stop (void * *arg*)

Обработчик вызова SYSCALL_PROC_FLAG_STOP.

Пытается остановить вызывающий процесс по флагу PROC_FLG_PRE_STOP, обнуляет флаги, заданные маской. Вызывает _proc_flag_stop.

arg	указатель на маску обнуления флагов процесса.

5.14.3.9 void scall_proc_reset_watchdog (void * *arg*)

Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.

Вызывает _proc_reset_watchdog.

arg	не используется.

5.14.3.10 void scall_proc_set_prio (void * *arg*)

Обработчик вызова SYSCALL_PROC_SET_PRIO.

Вызывает _proc_set_prio.

arg	Указатель на переменную типа proc_set_prio_arg_t.

5.14.3.11 void scall_sig_wait (void * *arg*)

Обработчик вызова SYSCALL_SIG_WAIT.

Переводит вызваший процесс в состояние ожидания сигнала, вызывает _sig_wait_prologue.

arg	указатель на сигнал.

5.14.3.12 void scall_sig_wakeup (void * *arg*)

Обработчик вызова SYSCALL_SIG_WAKEUP.

Вызывает _sig_wait_epilogue, обрабатывает флаг PROC_FLG_PRE_STOP

arg	не используется.

5.14.3.13 void scall_sig_signal (void * *arg*)

Обработчик вызова SYSCALL_SIG_SIGNAL.

"Будит" один из процессов, ожидающих сигнала, вызывает sig_signal_isr.

arg	указатель на сигнал.

5.14.3.14 void scall_sig_broadcast (void * *arg*)

Обработчик вызова SYSCALL_SIG_BROADCAST.

"Будит" все процессы , ожидающие сигнала, вызывает sig_broadcast_isr.

arg	указатель на сигнал.

5.14.3.15 void scall_sem_lock (void * *arg*)

Обработчик вызова SYSCALL_SEM_LOCK.

Вызывает _sem_lock.

arg	указатель на аргумент типа sem_lock_arg_t.

5.14.3.16 void scall_sem_try_lock (void * *arg*)

Обработчик вызова SYSCALL_SEM_TRY_LOCK.

Вызывает _sem_try_lock.

arg	указатель на аргумент типа sem_lock_arg_t.

5.14.3.17 void scall_sem_free (void * *arg*)

Обработчик вызова SYSCALL_SEM_FREE.

Вызывает sem_free_isr.

arg	указатель на семафор.

5.14.3.18 void scall_mutex_lock (void * *arg*)

Обработчик вызова SYSCALL_MUTEX_LOCK.

Вызывает _mutex_lock.

arg	указатель на аргумент типа mutex_lock_arg_t.

5.14.3.19 void scall_mutex_try_lock (void * *arg*)

Обработчик вызова SYSCALL_MUTEX_TRY_LOCK.

Вызывает _mutex_try_lock.

arg	указатель на аргумент типа mutex_lock_arg_t.

5.14.3.20 void scall_mutex_free (void * *arg*)

Обработчик вызова SYSCALL_MUTEX_FREE.

Вызывает _mutex_free.

arg	указатель на мьютекс.

5.14.3.21 void scall_ipc_wait (void * *arg*)

Обработчик вызова SYSCALL_IPC_WAIT.

Переводит вызывающий процесс в состояние ожидания получения данных через IPC. Вызывает _ipc_wait.

arg	указатель на хранилище для передачи данных.

5.14.3.22 void scall_ipc_send (void * *arg*)

Обработчик вызова SYSCALL_IPC_SEND.

Вызывает ipc_send_isr.

arg	указатель на аргумент типа ipc_send_arg_t.

5.14.3.23 void scall_ipc_exchange (void * *arg*)

Обработчик вызова SYSCALL_IPC_EXCHANGE.

Вызывает _ipc_exchange.

arg	указатель на аргумент типа ipc_send_arg_t.

5.14.3.24 void scall_user (void * *arg*)

Обработчик вызова SYSCALL_USER.

Вызывает пользовательскую функцию.

arg	указатель на функцию пользователя.

Осторожно! Параметр не проверяется!

5.14.4

5.14.4.1 syscall_t syscall_num

Обработка системного вызова.

Зпускает обработчик системного вызова и передает ему аргумент.

Номер системного вызова.

5.14.4.2 void* syscall_arg

Аргумент системного вызова.

5.15 include/timer.h

Заголовок программных таймеров.

- #define SPIN_LOCK_KERNEL_TIMER()
  Макрос-обертка.
- #define SPIN_FREE_KERNEL_TIMER()
- #define CLEAR_TIMER(t) _clear_timer((timer_t *)&t)
  Сброс программного таймера.
- #define TIMER(t) (timer_t)_timer((timer_t)t)
  Получить значение программного таймера, для внутреннего использования.

- void wait_time (timer_t time)
  Подождать заданный интервал времени.
- void _clear_timer (timer_t *t)
  Сброс программного таймера, для внутреннего использования.
- timer_t _timer (timer_t t)
  Получить значение программного таймера, для внутреннего использования.

5.15.1

Заголовок программных таймеров. Программные таймеры используются для синхронизации процессов по времени.

Программные таймеры нельзя использовать для точного измерения интервалов времени!

5.15.2

5.15.2.1 #define SPIN_LOCK_KERNEL_TIMER( )

Макрос-обертка.

Обертка захвата спин-блокировки таймера ядра, на однопроцессорной системе - пустой макрос.

Макрос-обертка.

Обертка освобождения спин-блокировки таймера ядра, на однопроцессорной системе - пустой макрос.

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

Сброс программного таймера.

t	Имя переменной таймера.

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

Получить значение программного таймера, для внутреннего использования.

t	Значение таймера.

5.15.3

5.15.3.1 void wait_time (timer_t *time*)

Подождать заданный интервал времени.

Просто ждет в цикле пока пройдет время time.

time	Время ожидания.

5.15.3.2 void _clear_timer (timer_t * *t*)

Сброс программного таймера, для внутреннего использования.

t	Указатель на таймер.

5.15.3.3 timer_t _timer (timer_t *t*)

Получить значение программного таймера, для внутреннего использования.

t	Значение таймера.

5.16 include/xlist.h

Заголовок списков с приоритетами.

- struct _xlist_t
  Список с приоритетами.

- typedef struct _xlist_t xlist_t

- void xlist_init (xlist_t *xlist)
  Инициализация списка.
- item_t * xlist_head (xlist_t *xlist)
  Поиск головы списка.
- void xlist_switch (xlist_t *xlist, prio_t prio)
  Переключение списка.

5.16.1

Заголовок списков с приоритетами.

5.16.2

5.16.2.1 typedef struct _xlist_t xlist_t

Смотри _xlist_t;

5.16.3

5.16.3.1 void xlist_init (xlist_t * *xlist*)

Инициализация списка.

xlist	Указатель на список.

5.16.3.2 item_t* xlist_head (xlist_t * *xlist*)

Поиск головы списка.

xlist	Указатель на список.

Указатель на голову - самый приоритетный элемент в массиве указателей.

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

Переключение списка.

Изменяет указатель xlist->item[prio] на xlist->item[prio]->next.

xlist	Указатель на список.
prio	Приоритет переключаемой части списка.