BuguRTOS

0.8.2

Создано системой Doxygen 1.8.1.2

Сб 25 Апр 2015 15:57:32

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 файлов.

4

4.1 _cond_t

Условная переменная.

#include "libs/generic/cond.h"

- sync_t wait
- count_t blocked

4.1.1

Условная переменная.

Условные переменные используются в сочитании с мьютексами для синхронизации процессов по событиям. Процесс может заблокировать себя на условной переменной. Другой процесс может возобновить выполнение 1 или всех процессов, заблокированных на условной переменной.

4.1.2

4.1.2.1 sync_t wait

Список ожидающих процессов.

4.1.2.2 count_t blocked

Счетчик ожидающих процессов.

Объявления и описания членов структуры находятся в файле:

- cond.h

4.2 _ipc_t

Конечная точка IPC.

#include "libs/generic/ipc.h"

- sync_t wait
- void * msg

4.2.1

Конечная точка IPC.

Используется для реализации блокирующего синхронного или асинхронного протокола IPC.

4.2.2

4.2.2.1 sync_t wait

Список ожидающих процессов.

4.2.2.2 void* msg

Указатель на буфер с сообщением.

Объявления и описания членов структуры находятся в файле:

- ipc.h

4.3 _item_t

Элемент 2-связного списка.

#include "kernel/item.h"

- item_t * next
- item_t * prev

4.3.1

Элемент 2-связного списка.

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

4.3.2

4.3.2.1 item_t* next

Следующий элемент.

4.3.2.2 item_t* prev

Предыдущий элемент.

Объявления и описания членов структуры находятся в файле:

- item.h

4.4 _kernel_t

Ядро BuguRTOS.

#include "kernel/kernel.h"

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

4.4.1

Ядро BuguRTOS.

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

4.4.2

4.4.2.1 sched_t sched

Планировщик.

4.4.2.2 proc_t idle

Процесс холостого хода.

4.4.2.3 timer_t timer

Системный таймер.

4.4.2.4 void(* timer_tick)(void)

Хук обработчика системного таймера.

Объявления и описания членов структуры находятся в файле:

- kernel.h

4.5 _mutex_t

Мьютекс.

#include "libs/generic/mutex.h"

- sync_t wait

4.5.1

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.

4.5.2

4.5.2.1 sync_t wait

Список ожидающих процессов.

Объявления и описания членов структуры находятся в файле:

- mutex.h

4.6 _pcounter_t

Счетчик захваченных ресурсов.

#include "kernel/pcounter.h"

- count_t counter [BITS_IN_INDEX_T]
- index_t index

4.6.1

Счетчик захваченных ресурсов.

При использовании опции CONFIG_USE_HIGHEST_LOCKER используется для пересчета захваченных процессом ресурсов.

4.6.2

4.6.2.1 count_t counter[BITS_IN_INDEX_T]

Массив счетчиков.

4.6.2.2 index_t index

Индекс для ускорения поиска.

Объявления и описания членов структуры находятся в файле:

- pcounter.h

4.7 _pitem_t

Элемент списка с приоритетами

#include "kernel/pitem.h"

- item_t parent
- xlist_t * list
- prio_t prio

4.7.1

Элемент списка с приоритетами

4.7.2

4.7.2.1 item_t parent

Родитель - item_t.

4.7.2.2 xlist_t* list

Указательна сниок в который будем вставлять.

4.7.2.3 prio_t prio

Приоритет.

Объявления и описания членов структуры находятся в файле:

- pitem.h

4.8 _proc_t

Процесс.

#include "kernel/proc.h"

- 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

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.

4.8.2

4.8.2.1 pitem_t parent

Родитель - pitem_t.

4.8.2.2 flag_t flags

Флаги (для ускорения анализа состояния процесса).

4.8.2.3 prio_t base_prio

Базовый приоритет.

4.8.2.4 pcounter_t lres

Счетчик захваченных ресурсов.

4.8.2.5 timer_t time_quant

Квант времени процесса.

4.8.2.6 timer_t timer

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

4.8.2.7 struct _sync_t* sync

4.8.2.8 count_t cnt_lock

Счетчик уровней вложенности proc_lock.

4.8.2.9 code_t pmain

Главная функция процесса.

4.8.2.10 code_t sv_hook

Хук, исполняется планировщиком после сохранения контекста процесса.

4.8.2.11 code_t rs_hook

Хук, исполняется планировщиком перед восстановлением контекста процесса.

4.8.2.12 void* arg

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

4.8.2.13 stack_t* sstart

Указатель на дно стека экземпляра процесса.

4.8.2.14 stack_t* spointer

Указатель на вершину стека экземпляра процесса.

Объявления и описания членов структуры находятся в файле:

- proc.h

4.9 _sched_t

Планировщик.

#include "kernel/sched.h"

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

4.9.1

Планировщик.

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

4.9.2

4.9.2.1 proc_t* current_proc

Текущий процесс.

4.9.2.2 xlist_t* ready

Указатель на список готовых к выполнению процессов.

4.9.2.3 xlist_t* expired

Указатель на список процессов, исчерпавших свой квант времени.

4.9.2.4 xlist_t plst[2]

Сами списки процесов.

4.9.2.5 count_t nested_crit_sec

Счетчик вложенности критических секций.

Объявления и описания членов структуры находятся в файле:

- sched.h

4.10 _sem_t

Счетный семафор.

#include "libs/generic/sem.h"

- sync_t wait
- count_t counter
- count_t blocked

4.10.1

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.

4.10.2

4.10.2.1 sync_t wait

Список ожидающих процессов.

4.10.2.2 count_t counter

Счетчик реусрсов.

4.10.2.3 count_t blocked

Счетчик блокированных процессов.

Объявления и описания членов структуры находятся в файле:

- sem.h

4.11 _sig_t

Сигнал.

#include "libs/generic/sig.h"

- cond_t wakeup
- mutex_t wait

4.11.1

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.

4.11.2

4.11.2.1 cond_t wakeup

Список процессов для пробуждения.

4.11.2.2 mutex_t wait

Список ожидающих процессов.

Объявления и описания членов структуры находятся в файле:

- sig.h

4.12 _sync_t

Базовый примитив синхронизации.

#include "kernel/sync.h"

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

4.12.1

Базовый примитив синхронизации.

Базовый тип, отвечающий за влокирующую синхронизацию процессов. Путем "обертывания" данного типа можно получить привычные примитивы синхронизации (мьютексы, семафоры, условные переменные, FIFO-буферы, блокироующий IPC, и т.д.).

Поддерживает протокол наследования приоритетов (Basic Priority Inheritance).

4.12.2

4.12.2.1 xlist_t sleep

Список ожидающих процессов.

4.12.2.2 proc_t* owner

Указатель на процесс, удерживающий мьютекс.

4.12.2.3 count_t dirty

Счетчик незавершенных транзакций наследования приоритетов.

4.12.2.4 prio_t prio

Приоритет.

Объявления и описания членов структуры находятся в файле:

- sync.h

4.13 _xlist_t

Список с приоритетами.

#include "kernel/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:

- xlist.h

4.14 proc_runtime_arg_t

Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

- proc_t * proc
- bool_t ret

4.14.1

Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

4.14.2

4.14.2.1 proc_t* proc

Указатель на процесс.

4.14.2.2 bool_t ret

Результат выполнения системного вызова.

Объявления и описания членов структуры находятся в файле:

- proc.c

4.15 proc_set_prio_arg_t

Параметр системного вызова SYSCALL_PROC_SET_PRIO.

- proc_t * proc
- prio_t prio

4.15.1

Параметр системного вызова SYSCALL_PROC_SET_PRIO.

4.15.2

4.15.2.1 proc_t* proc

Указатель на процесс.

4.15.2.2 prio_t prio

Приоритет.

Объявления и описания членов структуры находятся в файле:

- sync.c

4.16 scall_user_t

- code_t func
- void * arg

4.16.1

4.16.1.1 code_t func

4.16.1.2 void* arg

Объявления и описания членов структуры находятся в файле:

- syscall.c

4.17 sync_proc_timeout_t

- proc_t * proc
- flag_t status

4.17.1

4.17.1.1 proc_t* proc

4.17.1.2 flag_t status

Объявления и описания членов структуры находятся в файле:

- sync.c

4.18 sync_set_owner_t

- sync_t * sync
- proc_t * proc
- flag_t status

4.18.1

4.18.1.1 sync_t* sync

4.18.1.2 proc_t* proc

4.18.1.3 flag_t status

Объявления и описания членов структуры находятся в файле:

- sync.c

4.19 sync_sleep_t

Для внутреннего пользования.

#include "kernel/sync.h"

- sync_t * sync
- flag_t status

4.19.1

Для внутреннего пользования.

4.19.2

4.19.2.1 sync_t* sync

Указатель на объект типа sync_t.

4.19.2.2 flag_t status

Результат выполнения.

Объявления и описания членов структуры находятся в файле:

- sync.h

4.20 sync_wait_t

Для внутреннего пользования.

#include "kernel/sync.h"

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

4.20.1

Для внутреннего пользования.

4.20.2

4.20.2.1 sync_t* sync

Указатель на объект типа sync_t.

4.20.2.2 proc_t** proc

Указатель на буфер процесса.

4.20.2.3 flag_t block

Флаг блокирования.

4.20.2.4 flag_t status

Результат выполнения.

Объявления и описания членов структуры находятся в файле:

- sync.h

4.21 sync_wake_and_sleep_t

Для внутреннего пользования.

#include "kernel/sync.h"

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

4.21.1

Для внутреннего пользования.

4.21.2

4.21.2.1 sync_sleep_t sleep

Аргументы для второй части вызова.

4.21.2.2 sync_t* wake

Указатель на объект типа sync_t для 1й части вызова.

4.21.2.3 proc_t* proc

Указатель на процесс для 1й части вызова.

4.21.2.4 flag_t chown

Флаг смены хозяина.

4.21.2.5 flag_t stage

Номер части вызова.

Объявления и описания членов структуры находятся в файле:

- sync.h

4.22 sync_wake_and_wait_t

Для внутреннего пользования.

#include "kernel/sync.h"

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

4.22.1

Для внутреннего пользования.

4.22.2

4.22.2.1 sync_wait_t wait

Аргументы для первой части вызова.

4.22.2.2 sync_t* wake

Указатель на объект типа sync_t для 1й части вызова.

4.22.2.3 proc_t* proc

Указатель на процесс для 1й части вызова.

4.22.2.4 flag_t chown

Флаг смены хозяина.

4.22.2.5 flag_t stage

Номер части вызова.

Объявления и описания членов структуры находятся в файле:

- sync.h

4.23 sync_wake_t

Для внутреннего пользования.

#include "kernel/sync.h"

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

4.23.1

Для внутреннего пользования.

4.23.2

4.23.2.1 sync_t* sync

Указатель на объект типа sync_t.

4.23.2.2 proc_t* proc

Указатель на процесс.

4.23.2.3 flag_t chown

Флаг смены хозяина.

4.23.2.4 flag_t status

Результат выполнения.

Объявления и описания членов структуры находятся в файле:

- sync.h

5

5.1 bugurt.h

Главный заголовочный файл.

```
#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"
```

- #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 pmain(arg).

sstart	Дно стека.
pmain	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 cond.c

#include "cond.h"

- void cond_init_isr (cond_t *cond)

  Инициализация условной переменной из обработчика прерывания или критической секции.
- void cond_init (cond_t *cond)

  Инициализация условной переменной.
- flag_t cond_wait (cond_t *cond, mutex_t *mutex)

  Встать в список ожидания условной переменной.
- flag_t cond_signal (cond_t *cond)

  Возобновить работу 1 процесса ожидающего условной переменной.
- flag_t cond_broadcast (cond_t *cond)

  Возобновить работу вех процессов ожидающих условной переменной.

5.2.1

5.2.1.1 void cond_init_isr (cond_t * *cond*)

Инициализация условной переменной из обработчика прерывания или критической секции.

cond	Указатель на условную переменную.

5.2.1.2 void cond_init (cond_t * *cond*)

Инициализация условной переменной.

cond	Указатель на условную переменную.

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

Встать в список ожидания условной переменной.

Останавливает вызвавший процесс и ставит его в список ожидания.

cond	Указатель на условную переменную.
mutex	Указатель на мьютекс, защищающий условную переменную.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.2.1.4 flag_t cond_signal (cond_t * *cond*)

Возобновить работу 1 процесса ожидающего условной переменной.

Возобновляет работу головы списка ожидающих процессов.

Вызывать только при захваченном мьютексе!

cond	Указатель на условную переменную.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.2.1.5 flag_t cond_broadcast (cond_t * *cond*)

Возобновить работу вех процессов ожидающих условной переменной.

Возобновляет работу всех процессов из списка ожидающих.

Вызывать только при захваченном мьютексе!

cond	Указатель на условную переменную.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.3 cond.h

Заговок условных переменных.

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

- struct _cond_t

  Условная переменная.

- typedef struct _cond_t cond_t

- void cond_init_isr (cond_t *cond)

  Инициализация условной переменной из обработчика прерывания или критической секции.
- void cond_init (cond_t *cond)

  Инициализация условной переменной.
- flag_t cond_wait (cond_t *cond, mutex_t *mutex)

  Встать в список ожидания условной переменной.
- flag_t cond_signal (cond_t *cond)

  Возобновить работу 1 процесса ожидающего условной переменной.
- flag_t cond_broadcast (cond_t *cond)

  Возобновить работу вех процессов ожидающих условной переменной.

5.3.1

Заговок условных переменных.

5.3.2

5.3.2.1 typedef struct _cond_t cond_t

Смотри _cond_t;

5.3.3

5.3.3.1 void cond_init_isr (cond_t * *cond*)

Инициализация условной переменной из обработчика прерывания или критической секции.

cond	Указатель на условную переменную.

5.3.3.2 void cond_init (cond_t * *cond*)

Инициализация условной переменной.

cond	Указатель на условную переменную.

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

Встать в список ожидания условной переменной.

Останавливает вызвавший процесс и ставит его в список ожидания.

cond	Указатель на условную переменную.
mutex	Указатель на мьютекс, защищающий условную переменную.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.3.3.4 flag_t cond_signal (cond_t * *cond*)

Возобновить работу 1 процесса ожидающего условной переменной.

Возобновляет работу головы списка ожидающих процессов.

Вызывать только при захваченном мьютексе!

cond	Указатель на условную переменную.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.3.3.5 flag_t cond_broadcast (cond_t * *cond*)

Возобновить работу вех процессов ожидающих условной переменной.

Возобновляет работу всех процессов из списка ожидающих.

Вызывать только при захваченном мьютексе!

cond	Указатель на условную переменную.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.4 crit_sec.c

#include "bugurt.h"

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

  Выход из критической секции.

5.4.1

5.4.1.1 void enter_crit_sec (void)

Вход в критическую секцию.

5.4.1.2 void exit_crit_sec (void)

Выход из критической секции.

Вход в критическую секцию.

5.5 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.

5.5.1

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.

5.5.2

5.5.2.1 #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!

5.5.2.2 #define EXIT_CRIT_SEC( ) exit_crit_sec()

5.5.3

5.5.3.1 void enter_crit_sec (void)

Вход в критическую секцию.

5.5.3.2 void exit_crit_sec (void)

Выход из критической секции.

Вход в критическую секцию.

5.6 index.c

#include "bugurt.h"

- prio_t index_search (index_t index)
  Поиск в бинарном индексе.

5.6.1

5.6.1.1 prio_t index_search (index_t *index*)

Поиск в бинарном индексе.

index	Бинарный индекс.

Наивысший (с минимальным значением) приоритет в индексе.

5.7 index.h

Заголовок функции поиска в бинарном индексе.

- prio_t index_search (index_t index)
  Поиск в бинарном индексе.

5.7.1

Заголовок функции поиска в бинарном индексе.

5.7.2

5.7.2.1 prio_t index_search (index_t *index*)

Поиск в бинарном индексе.

index	Бинарный индекс.

Наивысший (с минимальным значением) приоритет в индексе.

5.8 ipc.c

#include "ipc.h"

- void ipc_init_isr (ipc_t *endpoint)

  Инициализация конечной точки IPC из критической секции или обработчика прерывания.
- void ipc_init (ipc_t *endpoint)

  Инициализация конечной точки IPC.
- flag_t ipc_send (ipc_t *out, void *msg)

  Посылка данных процессу через IPC.
- flag_t ipc_wait (ipc_t *in, proc_t **proc, flag_t block)

  Переход процесса к ожиданию получения данных через IPC.
- flag_t ipc_reply (ipc_t *in, proc_t *proc)

  Разблокировать процесс, от которого получено сообщение.

5.8.1

5.8.1.1 void ipc_init_isr (ipc_t * *endpoint*)

Инициализация конечной точки IPC из критической секции или обработчика прерывания.

endpoint	Указатель на конечную точку.

5.8.1.2 void ipc_init (ipc_t * *endpoint*)

Инициализация конечной точки IPC.

endpoint	Указатель на конечную точку.

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

Посылка данных процессу через IPC.

Пересылает указатель на буфер с сообщением через IPC, отправители блокируются и ждут своей очереди, получатель наследует приоритеты отправителей.

out	Указатель на конечную точку IPC.
msg	Указатель на буфер с сообщением.

SYNC_ST_OK в случае успеха, или номер ошибки.

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

Переход процесса к ожиданию получения данных через IPC.

Для для указания отправителя или получения указателя на отправитель используется буфер, адрес котороко передается вторым аргументом.

in	Указатель на конечную точку IPC.
proc	Двойной указатель на процесс, от которого ожидается сообщение (если *proc == 0, то принимаются сообщения от любых процессов).
block	Флаг блокировки вызывающего процесса, если не 0, то вызывающий процесс блокируется до готовности сообщения.

SYNC_ST_OK в случае успеха, или номер ошибки.

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

Разблокировать процесс, от которого получено сообщение.

in	Указатель на конечную точку IPC.
proc	Указатель на процесс-отправитель.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.9 ipc.h

Заголовок IPC.

#include <bugurt.h>

- struct _ipc_t
  Конечная точка IPC.

- typedef struct _ipc_t ipc_t

- void ipc_init_isr (ipc_t *endpoint)
  Инициализация конечной точки IPC из критической секции или обработчика прерывания.
- void ipc_init (ipc_t *endpoint)
  Инициализация конечной точки IPC.
- flag_t ipc_send (ipc_t *out, void *msg)
  Посылка данных процессу через IPC.
- flag_t ipc_wait (ipc_t *in, proc_t **proc, flag_t block)
  Переход процесса к ожиданию получения данных через IPC.
- flag_t ipc_reply (ipc_t *in, proc_t *proc)
  Разблокировать процесс, от которого получено сообщение.

5.9.1

Заголовок IPC.

5.9.2

5.9.2.1 typedef struct _ipc_t ipc_t

Смотри _ipc_t;

5.9.3

5.9.3.1 void ipc_init_isr (ipc_t * *endpoint*)

Инициализация конечной точки IPC из критической секции или обработчика прерывания.

endpoint	Указатель на конечную точку.

5.9.3.2 void ipc_init (ipc_t * *endpoint*)

Инициализация конечной точки IPC.

endpoint	Указатель на конечную точку.

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

Посылка данных процессу через IPC.

Пересылает указатель на буфер с сообщением через IPC, отправители блокируются и ждут своей очереди, получатель наследует приоритеты отправителей.

out	Указатель на конечную точку IPC.
msg	Указатель на буфер с сообщением.

SYNC_ST_OK в случае успеха, или номер ошибки.

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

Переход процесса к ожиданию получения данных через IPC.

Для для указания отправителя или получения указателя на отправитель используется буфер, адрес котороко передается вторым аргументом.

in	Указатель на конечную точку IPC.
proc	Двойной указатель на процесс, от которого ожидается сообщение (если *proc == 0, то принимаются сообщения от любых процессов).
block	Флаг блокировки вызывающего процесса, если не 0, то вызывающий процесс блокируется до готовности сообщения.

SYNC_ST_OK в случае успеха, или номер ошибки.

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

Разблокировать процесс, от которого получено сообщение.

in	Указатель на конечную точку IPC.
proc	Указатель на процесс-отправитель.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.10 item.c

#include "bugurt.h"

- 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.

5.10.1

5.10.1.1 void item_init (item_t * *item*)

Инициализация объекта типа item_t.

item	Указатель на объект item_t.

5.10.1.2 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.

5.10.1.3 void item_cut (item_t * *item*)

Вырезать элемент типа item_t из списка.

item	Указатель на объект типа item_t, который будем вырезать.

5.11 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.

5.11.1

Заголовок элементов 2-связного списка.

5.11.2

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

Статическая инициализация объекта типа item_t.

a	Имя переменной типа item_t.

5.11.3

5.11.3.1 typedef struct _item_t item_t

Смотри _item_t;

5.11.4

5.11.4.1 void item_init (item_t * *item*)

Инициализация объекта типа item_t.

item	Указатель на объект item_t.

5.11.4.2 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.

5.11.4.3 void item_cut (item_t * *item*)

Вырезать элемент типа item_t из списка.

item	Указатель на объект типа item_t, который будем вырезать.

5.12 kernel.c

#include "bugurt.h"

- WEAK void idle_main (void *arg)

  Главная функция процесса холостого хода.
- void kernel_init (void)

  Инициализация Ядра.

- kernel_t kernel

  Ядро BuguRTOS.

5.12.1

5.12.1.1 WEAK 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 семафоры и мьютексы из idle!!! Кто будет это все делать, того ждут Страшный суд, АдЪ и ПогибельЪ. Я предупредил!

arg	Указатель на аргумент.

5.12.1.2 void kernel_init (void)

Инициализация Ядра.

Готовит ядро к запуску.

5.12.2

5.12.2.1 kernel_t kernel

Ядро BuguRTOS.

Оно одно на всю систему!

5.13 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.

5.13.1

Заголовок Ядра.

5.13.2

5.13.2.1 typedef struct _kernel_t kernel_t

Смотри _kernel_t;

5.13.3

5.13.3.1 void kernel_init (void)

Инициализация Ядра.

Готовит ядро к запуску.

5.13.3.2 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 семафоры и мьютексы из idle!!! Кто будет это все делать, того ждут Страшный суд, АдЪ и ПогибельЪ. Я предупредил!

arg	Указатель на аргумент.

5.13.4

5.13.4.1 kernel_t kernel

Ядро BuguRTOS.

Оно одно на всю систему!

5.14 mutex.c

#include "mutex.h"

- void mutex_init_isr (mutex_t *mutex, prio_t prio)
  Инициализация мьютекса из критической секции, или обработчика прерываний.
- void mutex_init (mutex_t *mutex, prio_t prio)
  Инициализация мьютекса.
- flag_t mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса.
- flag_t mutex_lock (mutex_t *mutex)
  Захват мьютекса.
- flag_t mutex_free (mutex_t *mutex)
  Освобождение мьютекса.

5.14.1

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

Инициализация мьютекса из критической секции, или обработчика прерываний.

Да, инициировать из обработчика прерывания можно!

mutex	Указатель на мьютекс.
prio	Приоритет мьютекса.

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

Инициализация мьютекса.

mutex	Указатель на мьютекс.
prio	Приоритет мьютекса.

5.14.1.3 flag_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.

SYNC_ST_OK - если уддалось захватить, SYNC_ST_ROLL - если не удалось.

5.14.1.4 flag_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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.14.1.5 flag_t 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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.15 mutex.h

Заголовок мьютексов.

#include <bugurt.h>

- struct _mutex_t
  Мьютекс.

- typedef struct _mutex_t mutex_t

- void mutex_init_isr (mutex_t *mutex, prio_t prio)
  Инициализация мьютекса из критической секции, или обработчика прерываний.
- void mutex_init (mutex_t *mutex, prio_t prio)
  Инициализация мьютекса.
- flag_t mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса.

- flag_t mutex_lock (mutex_t *mutex)

  Захват мьютекса.
- flag_t mutex_free (mutex_t *mutex)

  Освобождение мьютекса.

5.15.1

Заголовок мьютексов.

5.15.2

5.15.2.1 typedef struct _mutex_t mutex_t

Смотри _mutex_t;

5.15.3

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

Инициализация мьютекса из критической секции, или обработчика прерываний.

Да, инициировать из обработчика прерывания можно!

mutex	Указатель на мьютекс.
prio	Приоритет мьютекса.

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

Инициализация мьютекса.

mutex	Указатель на мьютекс.
prio	Приоритет мьютекса.

5.15.3.3 flag_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.

SYNC_ST_OK - если уддалось захватить, SYNC_ST_ROLL - если не удалось.

5.15.3.4 flag_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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.15.3.5 flag_t 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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.16 pcounter.c

#include "bugurt.h"

- 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.

5.16.1

5.16.1.1 void pcounter_init (pcounter_t * *pcounter*)

Инициализация счетчика.

pcounter	Указатель на счетчик.

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

Инкремент счетчика.

pcounter	Указатель на счетчик.
prio	Приоритет.

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

Декремент счетчика.

pcounter	Указатель на счетчик.
prio	Приоритет.

5.16.1.4 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.

5.16.1.5 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.

5.17 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.

5.17.1

Заголовок счетчиков захваченных ресурсов.

5.17.2

5.17.2.1 typedef struct _pcounter_t pcounter_t

Смотри _pcounter_t;

5.17.3

5.17.3.1 void pcounter_init (pcounter_t * *pcounter*)

Инициализация счетчика.

pcounter	Указатель на счетчик.

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

Инкремент счетчика.

pcounter	Указатель на счетчик.
prio	Приоритет.

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

Декремент счетчика.

pcounter	Указатель на счетчик.
prio	Приоритет.

5.17.3.4 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.

5.17.3.5 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.

5.18 pitem.c

#include "bugurt.h"

- void pitem_init (pitem_t *pitem, prio_t prio)
  Инициализация объект а типа pitem_t.
- void pitem_insert (pitem_t *pitem, xlist_t *xlist)
  Вставка элемента типа pitem_t в список типа xlist_t.
- void pitem_fast_cut (pitem_t *pitem)
  Быстро вырезать из списка.
- void pitem_cut (pitem_t *pitem)
  Вырезать из списка.
- pitem_t * pitem_xlist_chain (xlist_t *src)
  "Сцепить" список типа xlist_t.

5.18.1

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

Инициализация объект а типа pitem_t.

pitem	Указатель на объект pitem_t.
prio	Приоритет элемента.

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

Вставка элемента типа pitem_t в список типа xlist_t.

pitem	Указатель на объект pitem_t.
xlist	Указатель на список.

5.18.1.3 void pitem_fast_cut (pitem_t * *pitem*)

Быстро вырезать из списка.

Вырезает объект типа pitem_t, из спика типа xlist_t, не обнуляет указатель pitem->list.

pitem	Указатель на объект pitem_t.

5.18.1.4 void pitem_cut (pitem_t * *pitem*)

Вырезать из списка.

Вызывает pitem_fast_cut и обнуляет указатель pitem->list.

pitem	Указатель на объект pitem_t.

5.18.1.5 pitem_t* pitem_xlist_chain (xlist_t * *src*)

"Сцепить" список типа xlist_t.

Вырезать из списка типа xlist_t все элементы типа pitem_t и сделать из них простой 2-связный список.

src	Указатель на объект xlist_t.

Указаьель на голову 2-связного списка.

5.19 pitem.h

Заголовок элементов списка с приоритетами.

- struct _pitem_t
  Элемент списка с приоритетами

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

- typedef struct _pitem_t pitem_t

- void pitem_init (pitem_t *pitem, prio_t prio)
  Инициализация объект а типа pitem_t.
- void pitem_insert (pitem_t *pitem, xlist_t *xlist)
  Вставка элемента типа pitem_t в список типа xlist_t.
- void pitem_fast_cut (pitem_t *pitem)
  Быстро вырезать из списка.
- void pitem_cut (pitem_t *pitem)
  Вырезать из списка.
- pitem_t * pitem_xlist_chain (xlist_t *src)
  "Сцепить" список типа xlist_t.

5.19.1

Заголовок элементов списка с приоритетами.

5.19.2

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

Статическая инициализация объекта типа pitem_t

a	Имя переменной.
p	Приоритет.

5.19.3

5.19.3.1 typedef struct _pitem_t pitem_t

5.19.4

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

Инициализация объект а типа pitem_t.

pitem	Указатель на объект pitem_t.
prio	Приоритет элемента.

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

Вставка элемента типа pitem_t в список типа xlist_t.

pitem	Указатель на объект pitem_t.
xlist	Указатель на список.

5.19.4.3 void pitem_fast_cut (pitem_t * *pitem*)

Быстро вырезать из списка.

Вырезает объект типа pitem_t, из спика типа xlist_t, не обнуляет указатель pitem->list.

pitem	Указатель на объект pitem_t.

5.19.4.4 void pitem_cut (pitem_t * *pitem*)

Вырезать из списка.

Вызывает pitem_fast_cut и обнуляет указатель pitem->list.

pitem	Указатель на объект pitem_t.

5.19.4.5 pitem_t* pitem_xlist_chain (xlist_t * *src*)

"Сцепить" список типа xlist_t.

Вырезать из списка типа xlist_t все элементы типа pitem_t и сделать из них простой 2-связный список.

src	Указатель на объект xlist_t.

Указаьель на голову 2-связного списка.

5.20 proc.c

#include "bugurt.h"

- struct proc_runtime_arg_t

  Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

- void _proc_stop_ensure (proc_t *proc)

  Останов процесса. Для внктреннего использования.
- void _proc_stop_flags_set (proc_t *proc, flag_t mask)

  "Низкоуровневый" останов процесса с установкой флагов, для внутреннего использования.
- void _proc_prio_control_stoped (proc_t *proc)

  Управление приоритетом процесса, для внутреннего использования.
- 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_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.
- 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.
- void scall_proc_run (void *arg)

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

  Обработчик вызова SYSCALL_PROC_RESTART.
- 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 scall_proc_stop (void *arg)

  Обработчик вызова SYSCALL_PROC_STOP.
- void proc_lock (void)

  Установка флага PROC_FLG_LOCK для вызывающего процесса.
- void _proc_lock (void)

  Установка флага PROC_FLG_LOCK для вызывающего процесса.
- void scall_proc_lock (void *arg)

  Обработчик вызова SYSCALL_PROC_LOCK.
- void proc_free (void)

  Останов процесса по флагу PROC_FLG_PRE_STOP.
- void _proc_free (void)

  Останов процесса по флагу PROC_FLG_PRE_STOP из критической секции или обработчика прерывания, для внутреннего использования.
- void scall_proc_free (void *arg)

  Обработчик вызова SYSCALL_PROC_FREE.
- void proc_self_stop (void)

  Самоостанов процесса.
- void _proc_self_stop (void)

  Самоостанов процесса (для внутреннего использования).
- void scall_proc_self_stop (void *arg)

  Обработчик вызова SYSCALL_PROC_SELF_STOP.
- 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.
- void scall_proc_terminate (void *arg)

  Обработчик вызова SYSCALL_PROC_TERMINATE.
- 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 scall_proc_reset_watchdog (void *arg)

  Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.

5.20.1

5.20.1.1 void _proc_stop_ensure (proc_t * *proc*)

Останов процесса. Для внктреннего использования.

Гарантированно останавливает процесс.

proc	- Указатель на процесс.

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

"Низкоуровневый" останов процесса с установкой флагов, для внутреннего использования.

5.20.1.3 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.

5.20.1.4 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*)

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.

5.20.1.5 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*)

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.

5.20.1.6 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.

5.20.1.7 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.

5.20.1.8 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.

5.20.1.9 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.

5.20.1.10 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.

5.20.1.11 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.

5.20.1.12 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.

5.20.1.13 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.

5.20.1.14 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.

5.20.1.15 void proc_lock (void)

Установка флага PROC_FLG_LOCK для вызывающего процесса.

5.20.1.16 void _proc_lock (void)

Установка флага PROC_FLG_LOCK для вызывающего процесса.

5.20.1.17 void scall_proc_lock (void * *arg*)

Обработчик вызова SYSCALL_PROC_LOCK.

Устанавливает флаг PROC_FLG_LOCK для вызывающего процесса, увеличивает счетчик proc->lres.

5.20.1.18 void proc_free (void)

Останов процесса по флагу PROC_FLG_PRE_STOP.

5.20.1.19 void _proc_free (void)

Останов процесса по флагу PROC_FLG_PRE_STOP из критической секции или обработчика прерывания, для внутреннего использования.

5.20.1.20 void scall_proc_free (void * arg)

Обработчик вызова SYSCALL_PROC_FREE.

Уменьшает счетчик proc->lres, при необходимости обнуляет флаг PROC_FLG_LOCK, пытается остановить вызывающий процесс по флагу PROC_FLG_PRE_STOP. Вызывает _proc_free.

arg	указатель на маску обнуления флагов процесса.

5.20.1.21 void proc_self_stop (void)

Самоостанов процесса.

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

5.20.1.22 void _proc_self_stop (void)

Самоостанов процесса (для внутреннего использования).

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

5.20.1.23 void scall_proc_self_stop (void * arg)

Обработчик вызова SYSCALL_PROC_SELF_STOP.

Останавливает вызывающий процесс.

arg	не используется.

5.20.1.24 void proc_terminate (void)

Завершение работы процесса после возврата из proc->pmain. Для внутреннего использования.

5.20.1.25 void _proc_terminate (void)

Завершение работы процесса после возврата из proc->pmain. Для внутреннего использования.

5.20.1.26 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.

5.20.1.27 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.

5.20.1.28 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.

5.20.1.29 void scall_proc_reset_watchdog (void * *arg*)

Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.

Вызывает _proc_reset_watchdog.

arg	не используется.

5.21 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_RR ((flag_t)0x40)
- #define PROC_FLG_LOCK ((flag_t)0x20)
  Флаг захвата мьютексов.
- #define PROC_FLG_PRE_STOP ((flag_t)0x10)
  Флаг запроса останова.
- #define PROC_FLG_LOCK_MASK ((flag_t)(PROC_FLG_LOCK))
  Маска PROC_FLG_LOCK.
- #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)
  Маска очистки состояния исполнения процесса.
- #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xFC)
  Маска очистки состояния исполнения процесса.
- #define PROC_STATE_MASK ((flag_t)0x0F)
  Маска состояния исполнения процесса.
- #define PROC_STATE_RESTART_MASK ((flag_t)0x8)
  Маска проверки состояния процесса.
- #define PROC_STATE_RUN_MASK ((flag_t)0x3)
  Маска проверки состояния процесса.

- #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

  Маска проверки состояния процесса.
- #define PROC_STATE_STOPED ((flag_t)0x0)

  Начальное состояние, остановлен.
- #define PROC_STATE_END ((flag_t)0x1)

  Завершен.
- #define PROC_STATE_READY ((flag_t)0x2)

  Готов к выполнению.
- #define PROC_STATE_RUNNING ((flag_t)0x3)

  Выполняется.
- #define PROC_STATE_WD_STOPED ((flag_t)0x4)

  Остановлен по вачдог.
- #define PROC_STATE_DEAD ((flag_t)0x5)

  Завершен до завершения ipc-транзакций.
- #define PROC_STATE_TO_READY ((flag_t)0x6)

  Готов к выполнению.
- #define PROC_STATE_TO_RUNNING ((flag_t)0x7)

  Выполняется.
- #define PROC_STATE_SYNC_WAIT ((flag_t)0x8)

  Ожидает приема спящих процессов.
- #define PROC_STATE_SYNC_SLEEP ((flag_t)0x9)

  Ожидает пробуждения.
- #define PROC_STATE_SYNC_READY ((flag_t)0xA)

  Готов к выполнению.
- #define PROC_STATE_SYNC_RUNNING ((flag_t)0xB)

  Выполняется.
- #define PROC_STATE_PI_PEND ((flag_t)0xC)

  Ожидает смены приоритета
- #define PROC_STATE_PI_DONE ((flag_t)0xD)

  Запущен при смене приоритета
- #define PROC_STATE_PI_READY ((flag_t)0xE)

  Готов к выполнению.
- #define PROC_STATE_PI_RUNNING ((flag_t)0xF)

  Выполняется.
- #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, a->flags |= b)

  Устанавливает состояние процесса.
- #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_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_lock (void)

  Установка флага PROC_FLG_LOCK для вызывающего процесса.
- void proc_lock (void)

  Установка флага PROC_FLG_LOCK для вызывающего процесса.
- void _proc_free (void)

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

  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.
- void _proc_lres_inc (proc_t *proc, prio_t prio)

  Управление приоритетом процесса. Для внктреннего использования.
- void _proc_lres_dec (proc_t *proc, prio_t prio)

  Управление приоритетом процесса. Для внктреннего использования.
- void _proc_stop_ensure (proc_t *proc)

  Останов процесса. Для внктреннего использования.

5.21.1

Заголовок процессов.

5.21.2

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

Макрос-обертка.

Инициирует поле proc->lres процесса.

a	указатель на процесс.

5.21.2.2 #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.

5.21.2.3 #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.

5.21.2.4 #define PROC_FLG_RT ((flag_t)0x80)

Флаг реального времени.

Для этого процесса используется политика планирования жесткого реального времени.

5.21.2.5 #define PROC_FLG_RR ((flag_t)0x40)

5.21.2.6 #define PROC_FLG_LOCK ((flag_t)0x20)

Флаг захвата мьютексов.

Процесс удерживает мьютекс.

5.21.2.7 #define PROC_FLG_PRE_STOP ((flag_t)0x10)

Флаг запроса останова.

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

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

Маска PROC_FLG_LOCK.

Нужна, чтобы определить, удерживает ли процесс общие ресурсы.

5.21.2.9 #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

Маска очистки состояния исполнения процесса.

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

5.21.2.10 #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xFC)

Маска очистки состояния исполнения процесса.

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

5.21.2.11 #define PROC_STATE_MASK ((flag_t)0x0F)

Маска состояния исполнения процесса.

5.21.2.12 #define PROC_STATE_RESTART_MASK ((flag_t)0x8)

Маска проверки состояния процесса.

Используется функциями proc_restart и proc_restart_isr, для проверки возможности перезапуска.

5.21.2.13 #define PROC_STATE_RUN_MASK ((flag_t)0x3)

Маска проверки состояния процесса.

Используется для того, чтобы проверить, запущен ли процесс.

5.21.2.14 #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.

5.21.2.15 #define PROC_STATE_STOPED ((flag_t)0x0)

Начальное состояние, остановлен.

5.21.2.16 #define PROC_STATE_END ((flag_t)0x1)

Завершен.

5.21.2.17 #define PROC_STATE_READY ((flag_t)0x2)

Готов к выполнению.

5.21.2.18 #define PROC_STATE_RUNNING ((flag_t)0x3)

Выполняется.

5.21.2.19 #define PROC_STATE_WD_STOPED ((flag_t)0x4)

Остановлен по вачдог.

5.21.2.20 #define PROC_STATE_DEAD ((flag_t)0x5)

Завершен до завершения ipc-транзакций.

5.21.2.21 #define PROC_STATE_TO_READY ((flag_t)0x6)

Готов к выполнению.

5.21.2.22 #define PROC_STATE_TO_RUNNING ((flag_t)0x7)

Выполняется.

5.21.2.23 #define PROC_STATE_SYNC_WAIT ((flag_t)0x8)

Ожидает приема спящих процессов.

5.21.2.24 #define PROC_STATE_SYNC_SLEEP ((flag_t)0x9)

Ожидает пробуждения.

5.21.2.25 #define PROC_STATE_SYNC_READY ((flag_t)0xA)

Готов к выполнению.

5.21.2.26 #define PROC_STATE_SYNC_RUNNING ((flag_t)0xB)

Выполняется.

5.21.2.27 #define PROC_STATE_PI_PEND ((flag_t)0xC)

Ожидает смены приоритета

5.21.2.28 #define PROC_STATE_PI_DONE ((flag_t)0xD)

Запущен при смене приоритета

5.21.2.29 #define PROC_STATE_PI_READY ((flag_t)0xE)

Готов к выполнению.

5.21.2.30 #define PROC_STATE_PI_RUNNING ((flag_t)0xF)

Выполняется.

5.21.2.31 #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.

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

Проверяет, запущен ли процесс.

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

Читает состояние процесса.

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

Устанавливает состояние процесса.

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

Низший приоритет.

5.21.3

5.21.3.1 typedef struct _proc_t proc_t

Смотри _proc_t;

5.21.4

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*)

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.

5.21.4.2 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*)

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.

5.21.4.3 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.

5.21.4.4 void proc_terminate (void)

Завершение работы процесса после возврата из proc->pmain. Для внутреннего использования.

5.21.4.5 void _proc_terminate (void)

Завершение работы процесса после возврата из proc->pmain. Для внутреннего использования.

5.21.4.6 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.

5.21.4.7 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.

5.21.4.8 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.

5.21.4.9 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.

5.21.4.10 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.

5.21.4.11 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.

5.21.4.12 void proc_self_stop (void)

Самоостанов процесса.

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

5.21.4.13 void _proc_self_stop (void)

Самоостанов процесса (для внутреннего использования).

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

5.21.4.14 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.

5.21.4.15 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.

5.21.4.16 void _proc_prio_propagate (proc_t * proc)

Передача приоритетов по цепи заблокированных процессов. Для внутреннего использования.

5.21.4.17 void _proc_stop_flags_set (proc_t * proc, flag_t mask)

"Низкоуровневый" останов процесса с установкой флагов, для внутреннего использования.

5.21.4.18 void _proc_lock (void)

Установка флага PROC_FLG_LOCK для вызывающего процесса.

5.21.4.19 void proc_lock (void)

Установка флага PROC_FLG_LOCK для вызывающего процесса.

5.21.4.20 void _proc_free (void)

Останов процесса по флагу PROC_FLG_PRE_STOP из критической секции или обработчика прерывания, для внутреннего использования.

5.21.4.21 void proc_free (void)

Останов процесса по флагу PROC_FLG_PRE_STOP.

5.21.4.22 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.

5.21.4.23 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.

5.21.4.24 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.

5.21.4.25 void _proc_lres_inc (proc_t * proc, prio_t prio)

Управление приоритетом процесса. Для внктреннего использования.

Инкрементирует счетчик proc->lres, устанавливает флаг PROC_FLG_LOCK.

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

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

Управление приоритетом процесса. Для внктреннего использования.

Декрементирует счетчик proc->lres, сбрасывает флаг PROC_FLG_LOCK при необходимости.

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

5.21.4.27 void _proc_stop_ensure (proc_t * *proc*)

Останов процесса. Для внктреннего использования.

Гарантированно останавливает процесс.

proc	- Указатель на процесс.

5.22 sched.c

#include "bugurt.h"

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

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

  Инициализация планировщика.
- 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.
- static void _sched_switch_current (sched_t *sched, proc_t *current_proc)
- void sched_schedule (void)

  Функция планирования.
- void sched_reschedule (void)

  Функция перепланирования.
- bool_t sched_proc_yeld (void)

  Передача управления следующему процессу.
- bool_t _sched_proc_yeld (void)

  Передача управления следующему процессу (для внутреннего использования).
- void scall_sched_proc_yeld (void *arg)

  Обработчик вызова SYSCALL_SCHED_PROC_YELD.

5.22.1

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

5.22.2

5.22.2.1 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.

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

"Низкоуровневый" запуск процесса, для внутреннего использования.

5.22.2.3 void sched_proc_stop (proc_t * ***proc***)

"Низкоуровневый" останов процесса, для внутреннего использования.

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)

Функция планирования.

Переключает процессы в обработчике прерывания системного таймера.

5.22.2.6 void sched_reschedule (void)

Функция перепланирования.

Переключает процессы в случае необходимости.

5.22.2.7 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.

5.22.2.8 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.

5.22.2.9 void scall_sched_proc_yeld (void * *arg*)

Обработчик вызова SYSCALL_SCHED_PROC_YELD.

Передает управление следующему процессу.

arg	не используется.

5.23 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.

5.23.1

Заголовок планировщика

Все функции в этом файле для внутреннего использования!!!

5.23.2

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

Макрос-обертка.

Обертка инициализации переменной sched в функциях sched_schedule и sched_reschedule.

5.23.3

5.23.3.1 typedef struct _sched_t sched_t

Смотри _sched_t;

5.23.4

5.23.4.1 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.

5.23.4.2 void sched_schedule (void)

Функция планирования.

Переключает процессы в обработчике прерывания системного таймера.

5.23.4.3 void sched_reschedule (void)

Функция перепланирования.

Переключает процессы в случае необходимости.

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

"Низкоуровневый" запуск процесса, для внутреннего использования.

5.23.4.5 void sched_proc_stop (proc_t * *proc*)

"Низкоуровневый" останов процесса, для внутреннего использования.

5.23.4.6 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.

5.23.4.7 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.

5.24 sem.c

#include "sem.h"

- 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.
- flag_t sem_try_lock (sem_t *sem)

  Попытка захвата семафора.
- flag_t sem_lock (sem_t *sem)

  Захват семафора.
- flag_t sem_free (sem_t *sem)

  Освобождение семафора.

5.24.1

5.24.1.1 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.

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

Инициализация семафора.

sem	Указатель на семафор.
count	Начальное значение счетчика.

5.24.1.3 flag_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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.24.1.4 flag_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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.24.1.5 flag_t 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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.25 sem.h

Заголовок счетных семафоров.

#include <bugurt.h>

- 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.
- flag_t sem_lock (sem_t *sem)
  Захват семафора.
- flag_t sem_try_lock (sem_t *sem)
  Попытка захвата семафора.
- flag_t sem_free (sem_t *sem)
  Освобождение семафора.

5.25.1

Заголовок счетных семафоров.

5.25.2

5.25.2.1 typedef struct _sem_t sem_t

Смотри _sem_t;

5.25.3

5.25.3.1 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.

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

Инициализация семафора.

sem	Указатель на семафор.
count	Начальное значение счетчика.

5.25.3.3 flag_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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.25.3.4 flag_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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.25.3.5 flag_t 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.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.26 sig.c

#include "sig.h"

- void sig_init (sig_t *sig)
  Инициализация сигнала.
- void sig_init_isr (sig_t *sig)
  Инициализация сигнала из обработчика прерывания или критической секции.
- flag_t sig_wait (sig_t *sig)
  Встать в список ожидания сигнала.
- flag_t sig_signal (sig_t *sig)
  Возобновить работу 1 процесса ожидающего сигнал.
- count_t sig_broadcast (sig_t *sig)
  Возобновить работу всех ожидающих процессов.

5.26.1

5.26.1.1 void sig_init (sig_t * *sig*)

Инициализация сигнала.

sig	Указатель на сигнал.

5.26.1.2 void sig_init_isr (sig_t * *sig*)

Инициализация сигнала из обработчика прерывания или критической секции.

sig	Указатель на сигнал.

5.26.1.3 flag_t sig_wait (sig_t * *sig*)

Встать в список ожидания сигнала.

Останавливает вызвавший процесс и ставит его в список ожидания.

sig	Указатель на сигнал.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.26.1.4 flag_t sig_signal (sig_t * *sig*)

Возобновить работу 1 процесса ожидающего сигнал.

На многопроцессорной системе:

sig	Указатель на сигнал.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.26.1.5 count_t sig_broadcast (sig_t * *sig*)

Возобновить работу всех ожидающих процессов.

Возобновляет работу всех ожидающих процессов.

sig	Указатель на сигнал.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.27 sig.h

Заголовок сигналов.

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

- 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.
- flag_t sig_wait (sig_t *sig)
  Встать в список ожидания сигнала.
- flag_t sig_signal (sig_t *sig)
  Возобновить работу 1 процесса ожидающего сигнал.
- count_t sig_broadcast (sig_t *sig)
  Возобновить работу всех ожидающих процессов.

5.27.1

Заголовок сигналов.

5.27.2

5.27.2.1 typedef struct _sig_t sig_t

Смотри _sig_t;

5.27.3

5.27.3.1 void sig_init_isr (sig_t * *sig*)

Инициализация сигнала из обработчика прерывания или критической секции.

sig	Указатель на сигнал.

5.27.3.2 void sig_init (sig_t * *sig*)

Инициализация сигнала.

sig	Указатель на сигнал.

5.27.3.3 flag_t sig_wait (sig_t * *sig*)

Встать в список ожидания сигнала.

Останавливает вызвавший процесс и ставит его в список ожидания.

sig	Указатель на сигнал.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.27.3.4 flag_t sig_signal (sig_t * *sig*)

Возобновить работу 1 процесса ожидающего сигнал.

На многопроцессорной системе:

sig	Указатель на сигнал.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.27.3.5 count_t sig_broadcast (sig_t * *sig*)

Возобновить работу всех ожидающих процессов.

Возобновляет работу всех ожидающих процессов.

sig	Указатель на сигнал.

SYNC_ST_OK в случае успеха, или номер ошибки.

5.28 sync.c

#include "bugurt.h"

- struct proc_set_prio_arg_t
  Параметр системного вызова SYSCALL_PROC_SET_PRIO.
- struct sync_set_owner_t
- struct sync_proc_timeout_t

- #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)

- prio_t _sync_prio (sync_t *sync)
  Возвращает текущий приоритет объекта типа sync_t. Для внктренннего использования.
- void _proc_prio_propagate (proc_t *proc)
  Передача приоритетов по цепи заблокированных процессов. 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.
- void scall_proc_set_prio (void *arg)
  Обработчик вызова SYSCALL_PROC_SET_PRIO.
- void sync_init (sync_t *sync, prio_t prio)
  Инициализация из критической секции, или обработчика прерываний.
- void sync_init_isr (sync_t *sync, prio_t prio)
  Инициализация базового примитива синхронизации.
- proc_t * sync_get_owner (sync_t *sync)
  Получить хозяина примитива.
- flag_t sync_set_owner (sync_t *sync, proc_t *proc)
  Назначить хозяина объекта типа sync_t.
- flag_t _sync_set_owner (sync_t *sync, proc_t *proc)
  Для внутреннего использования. Смотри sync_set_owner.
- void scall_sync_set_owner (void *arg)
  Обработчик вызова SYSCALL_SYNC_SET_OWNER.
- void sync_clear_owner (sync_t *sync)
  Сбросить хозяина объекта типа sync_t.
- void _sync_clear_owner (sync_t *sync)
  Для внутреннего использования. Смотри sync_clear_owner.
- void scall_sync_clear_owner (void *arg)
  Обработчик вызова SYSCALL_SYNC_CLEAR_OWNER.
- flag_t sync_sleep (sync_t *sync)

"Уснуть" в ожидании синхронизации sync_t.

- flag_t _sync_sleep (sync_t *sync)

  Для внутреннего использования. Смотри sync_sleep.
- void scall_sync_sleep (void *arg)

  Обработчик вызова SYSCALL_SYNC_SLEEP.
- static void _sync_owner_block (proc_t *owner)
- flag_t sync_wait (sync_t *sync, proc_t **proc, flag_t block)

  "Ожидать", блокировки процесса.
- flag_t _sync_wait (sync_t *sync, proc_t **proc, flag_t block)

  Для внутреннего использования. Смотри sync_wait.
- void scall_sync_wait (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAIT.
- flag_t sync_wake (sync_t *sync, proc_t *proc, flag_t chown)

  "Разбудить" ожидающий процесс.
- flag_t _sync_wake (sync_t *sync, proc_t *proc, flag_t chown)

  Для внутреннего использования. Смотри sync_wake.
- void scall_sync_wake (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAKE.
- flag_t sync_wake_and_sleep (sync_t *wake, proc_t *proc, flag_t chown, sync_t *sleep)

  Смотри sync_wake и sync_sleep.
- void scall_sync_wake_and_sleep (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAKE_AND_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)

  Смотри sync_wake и sync_wait.
- void scall_sync_wake_and_wait (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAKE_AND_WAIT.
- flag_t sync_proc_timeout (proc_t *proc)

  "Разбудить", процесс по таймауту.
- void scall_sync_proc_timeout (void *arg)

  Обработчик вызова SYSCALL_SYNC_PROC_TIMEOUT.
- flag_t _sync_proc_timeout (proc_t *proc)

  Для внутреннего использования. Смотри sync_proc_timeout.

5.28.1

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

5.28.2.1 prio_t _sync_prio (sync_t * *sync*)

Возвращает текущий приоритет объекта типа sync_t. Для внктренннего использования.

5.28.2.2 void _proc_prio_propagate (proc_t * *proc*)

Передача приоритетов по цепи заблокированных процессов. Для внутреннего использования.

5.28.2.3 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.

5.28.2.4 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.

5.28.2.5 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.

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

Инициализация из критической секции, или обработчика прерываний.

Да, инициировать из обработчика прерывания можно!

sync	Указатель на объект типа sync_t.
prio	Приоритет.

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

Инициализация базового примитива синхронизации.

sync	Указатель на базвоый примитив синхронизации.
prio	Приоритет.

5.28.2.8 proc_t* sync_get_owner (sync_t * *sync*)

Получить хозяина примитива.

sync	Указатель на интересующий объект типа sync_t.

Указатель на процесс-хозяин объекта типа sync_t.

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

Назначить хозяина объекта типа sync_t.

sync	Указатель на объект типа sync_t.
proc	Указатель на новый процесс-хозяин объекта типа sync_t.

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

Для внутреннего использования. Смотри sync_set_owner.

5.28.2.11 void scall_sync_set_owner (void * *arg*)

Обработчик вызова SYSCALL_SYNC_SET_OWNER.

Вызывает _sync_set_owner.

5.28.2.12 void sync_clear_owner (sync_t * *sync*)

Сбросить хозяина объекта типа sync_t.

sync	Указатель на объект типа sync_t.

5.28.2.13 void _sync_clear_owner (sync_t * *sync*)

Для внутреннего использования. Смотри sync_clear_owner.

5.28.2.14 void scall_sync_clear_owner (void * *arg*)

Обработчик вызова SYSCALL_SYNC_CLEAR_OWNER.

Вызывает _sync_clear_owner.

5.28.2.15 flag_t sync_sleep (sync_t * *sync*)

"Уснуть" в ожидании синхронизации sync_t.

Блокирует вызывающий процесс.

sync	Указатель на объект типа sync_t.

SYNC_ST_OK в случае успеха, иначе - код ошибки.

5.28.2.16 flag_t _sync_sleep (sync_t * *sync*)

Для внутреннего использования. Смотри sync_sleep.

5.28.2.17 void scall_sync_sleep (void * *arg*)

Обработчик вызова SYSCALL_SYNC_SLEEP.

Вызывает _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*)

"Ожидать", блокировки процесса.

Подождать того момента, как целевой процесс будет заблокирован на целевом примимтиве синхронизации.

sync	Указатель на объект типа sync_t.
proc	Двойной указатель на процес, который надо подождать, если *proc==0, то вызывающий процесс будет ждать первой блокировки процесса на объекте типа sync_t.
block	Флаг блокировки вызывающего процесса, если не 0 и нужно ждать, вызывающий процесс будет заблокирован.

SYNC_ST_OK в случае если дождался блокировки целевого процесса, SYNC_ST_ROLL, если нужна следующая иттерация, иначе - код ошибки.

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

Для внутреннего использования. Смотри sync_wait.

5.28.2.21 void scall_sync_wait (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAIT.

Вызывает _sync_wait.

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

"Разбудить" ожидающий процесс.

Запускает ожидающий процесс. Может запустить "голову" списка ожидающих процессов, или какой-то конкретный прооцесс, в случае, если он заблокирован на целевом примитиве синхронизации.

sync	Указатель на объект типа sync_t.
proc	Указатель на процес, который надо запустить, если 0, то пытается запустить "голову" списка ожидающих.
chown	Флаг смены хозяина, если не 0, то запускаемый процесс станет новым хозяином примитива синхронизации.

SYNC_ST_OK в случае если удалось запустить процесс, иначе - код ошибки.

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

Для внутреннего использования. Смотри sync_wake.

5.28.2.24 void scall_sync_wake (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAKE.

Вызывает _sync_wake.

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

Смотри sync_wake и sync_sleep.

5.28.2.26 void scall_sync_wake_and_sleep (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAKE_AND_SLEEP.

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*)

Смотри sync_wake и sync_wait.

5.28.2.28 void scall_sync_wake_and_wait (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAKE_AND_WAIT.

5.28.2.29 flag_t sync_proc_timeout (proc_t * *proc*)

"Разбудить", процесс по таймауту.

proc	Указатель на процес, который надо подождать, если *proc==0, то вызывающий процесс будет ждать первой блокировки процесса на объекте типа sync_t.

SYNC_ST_OK в случае, если дождался разбудил целевой процесс, SYNC_ST_ROLL, если нужна следующая иттерация, иначе - код ошибки.

5.28.2.30 void scall_sync_proc_timeout (void * *arg*)

Обработчик вызова SYSCALL_SYNC_PROC_TIMEOUT.

5.28.2.31 flag_t _sync_proc_timeout (proc_t * *proc*)

Для внутреннего использования. Смотри sync_proc_timeout.

5.29 sync.h

Заголовок базового примитива синхронизации.

- struct _sync_t
  Базовый примитив синхронизации.
- struct sync_sleep_t
  Для внутреннего пользования.
- struct sync_wait_t
  Для внутреннего пользования.
- struct sync_wake_t
  Для внутреннего пользования.
- struct sync_wake_and_sleep_t
  Для внутреннего пользования.
- struct sync_wake_and_wait_t
  Для внутреннего пользования.

- #define SYNC_ST_OK 0
  Удачное завершение.
- #define SYNC_ST_ENULL 1
  Передан нулевой указатель.
- #define SYNC_ST_EOWN 2
  Ошибка владения.
- #define SYNC_ST_EEMPTY 3
  Список спящих процессов пуст.
- #define SYNC_ST_ESYNC 4
  Не тот объект типа sync_t.
- #define SYNC_ST_ETIMEOUT 5
  ИСтек таймаут sync_t.
- #define SYNC_ST_ROLL 6
  Нужна следующая иттерация.
- #define SYNC_PRIO(s) _sync_prio(s)
  Считает приоритет щбъекта типа sync_t.
- #define SYNC_INIT(s, p) sync_init((sync_t *)s, (prio_t)p)
  Смотри sync_init.
- #define SYNC_INIT_ISR(s, p) sync_init_isr((sync_t *)s, (prio_t)p)
  Смотри sync_init_isr.
- #define SYNC_GET_OWNER(s) sync_get_owner((sync_t *)s)
  Смотри sync_get_owner.
- #define SYNC_SET_OWNER(s, p) sync_set_owner((sync_t *)s, (proc_t *)p)
  Смотри sync_set_owner.
- #define SYNC_CLEAR_OWNER(s) sync_clear_owner((sync_t *)s)
  Смотри sync_clear_owner.
- #define SYNC_SLEEP(s) sync_sleep((sync_t *)s)
  Смотри sync_sleep.
- #define SYNC_WAIT(s, p, b, st)
  Смотри sync_wait.
- #define SYNC_WAKE(s, p, c, st)
  Смотри sync_wake.
- #define SYNC_WAKE_AND_SLEEP(w, p, c, s, st)
  Смотри sync_wake_and_sleep.
- #define SYNC_WAKE_AND_WAIT(wk, pwk, c, wt, pwt, b, st)
  Смотри sync_wake_and_wait.

- typedef struct _sync_t sync_t

- prio_t _sync_prio (sync_t *sync)
  Возвращает текущий приоритет объекта типа sync_t. Для внктренннего использования.
- void sync_init (sync_t *sync, prio_t prio)
  Инициализация из критической секции, или обработчика прерываний.
- void sync_init_isr (sync_t *sync, prio_t prio)
  Инициализация базового примитива синхронизации.

- proc_t * sync_get_owner (sync_t *sync)

  Получить хозяина примитива.
- flag_t sync_set_owner (sync_t *sync, proc_t *proc)

  Назначить хозяина объекта типа sync_t.
- void sync_clear_owner (sync_t *sync)

  Сбросить хозяина объекта типа sync_t.
- flag_t sync_sleep (sync_t *sync)

  "Уснуть" в ожидании синхронизации sync_t.
- flag_t sync_wait (sync_t *sync, proc_t **proc, flag_t block)

  "Ожидать", блокировки процесса.
- flag_t sync_wake (sync_t *sync, proc_t *proc, flag_t chown)

  "Разбудить" ожидающий процесс.
- flag_t sync_wake_and_sleep (sync_t *wake, proc_t *proc, flag_t chown, sync_t *sleep)

  Смотри sync_wake и 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)

  Смотри sync_wake и sync_wait.
- flag_t sync_proc_timeout (proc_t *proc)

  "Разбудить", процесс по таймауту.
- flag_t _sync_set_owner (sync_t *sync, proc_t *proc)

  Для внутреннего использования. Смотри sync_set_owner.
- void _sync_clear_owner (sync_t *sync)

  Для внутреннего использования. Смотри sync_clear_owner.
- flag_t _sync_wake (sync_t *sync, proc_t *proc, flag_t chown)

  Для внутреннего использования. Смотри sync_wake.
- flag_t _sync_sleep (sync_t *sync)

  Для внутреннего использования. Смотри sync_sleep.
- flag_t _sync_wait (sync_t *sync, proc_t **proc, flag_t block)

  Для внутреннего использования. Смотри sync_wait.
- flag_t _sync_proc_timeout (proc_t *proc)

  Для внутреннего использования. Смотри sync_proc_timeout.

5.29.1

Заголовок базового примитива синхронизации.

5.29.2

5.29.2.1 #define SYNC_ST_OK 0

Удачное завершение.

5.29.2.2 #define SYNC_ST_ENULL 1

Передан нулевой указатель.

5.29.2.3 #define SYNC_ST_EOWN 2

Ошибка владения.

5.29.2.4 #define SYNC_ST_EEMPTY 3

Список спящих процессов пуст.

5.29.2.5 #define SYNC_ST_ESYNC 4

Не тот объект типа sync_t.

5.29.2.6 #define SYNC_ST_ETIMEOUT 5

ИСтек таймаут sync_t.

5.29.2.7 #define SYNC_ST_ROLL 6

Нужна следующая иттерация.

5.29.2.8 #define SYNC_PRIO(s) _sync_prio(s)

Считает приоритет щбъекта типа sync_t.

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

Смотри sync_init.

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

Смотри sync_init_isr.

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

Смотри sync_get_owner.

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

Смотри sync_set_owner.

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

Смотри sync_clear_owner.

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

Смотри sync_sleep.

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

Макроопределение:

```
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)
```

Смотри sync_wait.

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

Макроопределение:

```
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)
```

Смотри sync_wake.

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

Макроопределение:

```
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)
```

Смотри sync_wake_and_sleep.

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

Макроопределение:

```
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)
```

Смотри sync_wake_and_wait.

5.29.3

5.29.3.1 typedef struct _sync_t sync_t

Смотри _sync_t;

5.29.4

5.29.4.1 prio_t _sync_prio (sync_t * *sync*)

Возвращает текущий приоритет объекта типа sync_t. Для внктренннего использования.

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

Инициализация из критической секции, или обработчика прерываний.

Да, инициировать из обработчика прерывания можно!

sync	Указатель на объект типа sync_t.
prio	Приоритет.

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

Инициализация базового примитива синхронизации.

sync	Указатель на базвоый примитив синхронизации.
prio	Приоритет.

5.29.4.4 proc_t* sync_get_owner (sync_t * *sync*)

Получить хозяина примитива.

sync	Указатель на интересующий объект типа sync_t.

Указатель на процесс-хозяин объекта типа sync_t.

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

Назначить хозяина объекта типа sync_t.

sync	Указатель на объект типа sync_t.
proc	Указатель на новый процесс-хозяин объекта типа sync_t.

5.29.4.6 void sync_clear_owner (sync_t * *sync*)

Сбросить хозяина объекта типа sync_t.

sync	Указатель на объект типа sync_t.

5.29.4.7 flag_t sync_sleep (sync_t * *sync*)

"Уснуть" в ожидании синхронизации sync_t.

Блокирует вызывающий процесс.

sync	Указатель на объект типа sync_t.

SYNC_ST_OK в случае успеха, иначе - код ошибки.

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

"Ожидать", блокировки процесса.

Подождать того момента, как целевой процесс будет заблокирован на целевом примимтиве синхронизации.

sync	Указатель на объект типа sync_t.
proc	Двойной указатель на процес, который надо подождать, если *proc==0, то вызывающий процесс будет ждать первой блокировки процесса на объекте типа sync_t.
block	Флаг блокировки вызывающего процесса, если не 0 и нужно ждать, вызывающий процесс будет заблокирован.

SYNC_ST_OK в случае если дождался блокировки целевого процесса, SYNC_ST_ROLL, если нужна следующая иттерация, иначе - код ошибки.

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

"Разбудить" ожидающий процесс.

Запускает ожидающий процесс. Может запустить "голову" списка ожидающих процессов, или какой-то конкретный прооцесс, в случае, если он заблокирован на целевом примитиве синхронизации.

sync	Указатель на объект типа sync_t.
proc	Указатель на процес, который надо запустить, если 0, то пытается запустить "голову" списка ожидающих.
chown	Флаг смены хозяина, если не 0, то запускаемый процесс станет новым хозяином примитива синхронизации.

SYNC_ST_OK в случае если удалось запустить процесс, иначе - код ошибки.

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

Смотри sync_wake и 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*)

Смотри sync_wake и sync_wait.

5.29.4.12 flag_t sync_proc_timeout (proc_t * *proc*)

"Разбудить", процесс по таймауту.

proc	Указатель на процес, который надо подождать, если *proc==0, то вызывающий процесс будет ждать первой блокировки процесса на объекте типа sync_t.

SYNC_ST_OK в случае, если дождался разбудил целевой процесс, SYNC_ST_ROLL, если нужна следующая иттерация, иначе - код ошибки.

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

Для внутреннего использования. Смотри sync_set_owner.

5.29.4.14 void _sync_clear_owner (sync_t * *sync*)

Для внутреннего использования. Смотри sync_clear_owner.

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

Для внутреннего использования. Смотри sync_wake.

5.29.4.16 flag_t _sync_sleep (sync_t * *sync*)

Для внутреннего использования. Смотри sync_sleep.

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

Для внутреннего использования. Смотри sync_wait.

5.29.4.18 flag_t _sync_proc_timeout (proc_t * *proc*)

Для внутреннего использования. Смотри sync_proc_timeout.

5.30 syscall.c

#include "bugurt.h"

- struct scall_user_t

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

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

  Обработчик вызова SYSCALL_USER.

- syscall_t syscall_num = (syscall_t)0

Номер системного вызова.

- void * syscall_arg = (void *)0

  Аргумент системного вызова.

5.30.1

5.30.1.1 SYSCALL_TABLE (syscall_routine [])

5.30.1.2 void do_syscall (void)

Обработка системного вызова.

Запускает обработчик системного вызова и передает ему аргумент.

5.30.1.3 void scall_user (void * arg)

Обработчик вызова SYSCALL_USER.

5.30.2

5.30.2.1 syscall_t syscall_num = (syscall_t)0

Номер системного вызова.

5.30.2.2 void* syscall_arg = (void *)0

Аргумент системного вызова.

5.31 syscall.h

Заголовок системных вызовов.

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

  Запуск процесса.
- #define SYSCALL_PROC_RESTART (SYSCALL_PROC_RUN + (syscall_t)(1))

  Перезапуск процесса.
- #define SYSCALL_PROC_STOP (SYSCALL_PROC_RESTART + (syscall_t)(1))

  Останов процесса.
- #define SYSCALL_PROC_SELF_STOP (SYSCALL_PROC_STOP + (syscall_t)(1))

  Самоостанов процесса.
- #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))

  Завершение работы процесса.
- #define SYSCALL_PROC_LOCK (SYSCALL_PROC_TERMINATE + (syscall_t)(1))

  Установить флаг PROC_FLG_LOCK для вызывающего процесса.
- #define SYSCALL_PROC_FREE (SYSCALL_PROC_LOCK + (syscall_t)(1))

  Останов процесса по флагу PROC_FLG_PRE_STOP.
- #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FREE + (syscall_t)(1))

  Сброс watchdog процесса реального времени.
- #define SYSCALL_PROC_SET_PRIO (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))

Установить приоритет процесса

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

  Передача управления другому процессу.
- #define SYSCALL_SYNC_SET_OWNER (SYSCALL_SCHED_PROC_YELD + (syscall_t)(1))

  Установить нового хозяина объекта типа sync_t.
- #define SYSCALL_SYNC_CLEAR_OWNER (SYSCALL_SYNC_SET_OWNER + (syscall_t)(1))

  Сбросить хозяина объекта типа sync_t.
- #define SYSCALL_SYNC_SLEEP (SYSCALL_SYNC_CLEAR_OWNER + (syscall_t)(1))

  Заблокировать процесс в ожидании синхронизации.
- #define SYSCALL_SYNC_WAKE (SYSCALL_SYNC_SLEEP + (syscall_t)(1))

  Запустить процесс, ожидающий синхронизации.
- #define SYSCALL_SYNC_WAIT (SYSCALL_SYNC_WAKE + (syscall_t)(1))

  Подождать блокировки процесса на объекте типа sync_t.
- #define SYSCALL_SYNC_WAKE_AND_SLEEP (SYSCALL_SYNC_WAIT + (syscall_t)(1))

  Смотри SYSCALL_SYNC_WAKE и SYSCALL_SYNC_SLEEP.
- #define SYSCALL_SYNC_WAKE_AND_WAIT (SYSCALL_SYNC_WAKE_AND_SLEEP + (syscall_t)(1))

  Смотри SYSCALL_SYNC_WAKE и SYSCALL_SYNC_WAIT.
- #define SYSCALL_SYNC_PROC_TIMEOUT (SYSCALL_SYNC_WAKE_AND_WAIT + (syscall_t)(1))

  Разбудить процесс по таймауту.
- #define SYSCALL_USER (SYSCALL_SYNC_PROC_TIMEOUT + (syscall_t)(1))

  Пользовательский системный вызов.

- 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_SCHED_PROC_YELD.
- void scall_proc_terminate (void *arg)

  Обработчик вызова SYSCALL_PROC_TERMINATE.
- void scall_proc_lock (void *arg)

  Обработчик вызова SYSCALL_PROC_LOCK.
- void scall_proc_free (void *arg)

  Обработчик вызова SYSCALL_PROC_FREE.
- 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_sync_set_owner (void *arg)

  Обработчик вызова SYSCALL_SYNC_SET_OWNER.
- void scall_sync_clear_owner (void *arg)

  Обработчик вызова SYSCALL_SYNC_CLEAR_OWNER.
- void scall_sync_sleep (void *arg)

  Обработчик вызова SYSCALL_SYNC_SLEEP.
- void scall_sync_wake (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAKE.
- void scall_sync_wait (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAIT.
- void scall_sync_wake_and_sleep (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAKE_AND_SLEEP.
- void scall_sync_wake_and_wait (void *arg)

  Обработчик вызова SYSCALL_SYNC_WAKE_AND_WAIT.
- void scall_sync_proc_timeout (void *arg)

  Обработчик вызова SYSCALL_SYNC_PROC_TIMEOUT.
- void scall_user (void *arg)

  Обработчик вызова SYSCALL_USER.

- syscall_t syscall_num

  Номер системного вызова.
- void * syscall_arg

  Аргумент системного вызова.

5.31.1

Заголовок системных вызовов.

5.31.2

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

Запуск процесса.

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

Перезапуск процесса.

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

Останов процесса.

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

Самоостанов процесса.

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

Завершение работы процесса.

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

Установить флаг PROC_FLG_LOCK для вызывающего процесса.

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

Останов процесса по флагу PROC_FLG_PRE_STOP.

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

Сброс watchdog процесса реального времени.

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

Установить приоритет процесса

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

Передача управления другому процессу.

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

Установить нового хозяина объекта типа sync_t.

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

Сбросить хозяина объекта типа sync_t.

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

Заблокировать процесс в ожидании синхронизации.

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

Запустить процесс, ожидающий синхронизации.

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

Подождать блокировки процесса на объекте типа sync_t.

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

Смотри SYSCALL_SYNC_WAKE и SYSCALL_SYNC_SLEEP.

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

Смотри SYSCALL_SYNC_WAKE и SYSCALL_SYNC_WAIT.

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

Разбудить процесс по таймауту.

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

Пользовательский системный вызов.

5.31.3

5.31.3.1 void do_syscall (void)

Обработка системного вызова.

Зпускает обработчик системного вызова и передает ему аргумент.

5.31.3.2 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.

5.31.3.3 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.

5.31.3.4 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.

5.31.3.5 void scall_proc_self_stop (void * *arg*)

Обработчик вызова SYSCALL_PROC_SELF_STOP.

Останавливает вызывающий процесс.

arg	не используется.

5.31.3.6 void scall_sched_proc_yeld (void * *arg*)

Обработчик вызова SYSCALL_SCHED_PROC_YELD.

Передает управление следующему процессу.

arg	не используется.

5.31.3.7 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.

5.31.3.8 void scall_proc_lock (void * *arg*)

Обработчик вызова SYSCALL_PROC_LOCK.

Устанавливает флаг PROC_FLG_LOCK для вызывающего процесса, увеличивает счетчик proc->lres.

5.31.3.9 void scall_proc_free (void * *arg*)

Обработчик вызова SYSCALL_PROC_FREE.

Уменьшает счетчик proc->lres, при необходимости обнуляет флаг PROC_FLG_LOCK, пытается остановить вызывающий процесс по флагу PROC_FLG_PRE_STOP. Вызывает _proc_free.

arg	указатель на маску обнуления флагов процесса.

5.31.3.10 void scall_proc_reset_watchdog (void * *arg*)

Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.

Вызывает _proc_reset_watchdog.

arg	не используется.

5.31.3.11 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.

5.31.3.12 void scall_sync_set_owner (void * *arg*)

Обработчик вызова SYSCALL_SYNC_SET_OWNER.

Вызывает _sync_set_owner.

5.31.3.13 void scall_sync_clear_owner (void * *arg*)

Обработчик вызова SYSCALL_SYNC_CLEAR_OWNER.

Вызывает _sync_clear_owner.

5.31.3.14 void scall_sync_sleep (void * *arg*)

Обработчик вызова SYSCALL_SYNC_SLEEP.

Вызывает _sync_sleep.

5.31.3.15 void scall_sync_wake (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAKE.

Вызывает _sync_wake.

5.31.3.16 void scall_sync_wait (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAIT.

Вызывает _sync_wait.

5.31.3.17 void scall_sync_wake_and_sleep (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAKE_AND_SLEEP.

5.31.3.18 void scall_sync_wake_and_wait (void * *arg*)

Обработчик вызова SYSCALL_SYNC_WAKE_AND_WAIT.

5.31.3.19 void scall_sync_proc_timeout (void * *arg*)

Обработчик вызова SYSCALL_SYNC_PROC_TIMEOUT.

5.31.3.20 void scall_user (void * *arg*)

Обработчик вызова SYSCALL_USER.

5.31.4

5.31.4.1 syscall_t syscall_num

Номер системного вызова.

5.31.4.2 void* syscall_arg

Аргумент системного вызова.

5.32 timer.c

#include "bugurt.h"

- 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.
- void wait_time (timer_t time)
  Подождать заданный интервал времени.

5.32.1

5.32.1.1 void _clear_timer (timer_t * *t*)

Сброс программного таймера, для внутреннего использования.

t	Указатель на таймер.

5.32.1.2 timer_t _timer (timer_t *t*)

Получить значение программного таймера, для внутреннего использования.

t	Значение таймера.

5.32.1.3 void wait_time (timer_t *time*)

Подождать заданный интервал времени.

Просто ждет в цикле пока пройдет время time.

time	Время ожидания.

5.33 timer.h

Заголовок программных таймеров.

- #define SPIN_LOCK_KERNEL_TIMER()
  Макрос-обертка.
- #define SPIN_FREE_KERNEL_TIMER()
  Макрос-обертка.
- #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.

5.33.1

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

Программные таймеры нельзя использовать для точного измерения интервалов времени!

5.33.2

5.33.2.1 #define SPIN_LOCK_KERNEL_TIMER( )

Макрос-обертка.

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

5.33.2.2 #define SPIN_FREE_KERNEL_TIMER( )

Макрос-обертка.

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

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

Сброс программного таймера.

t	Имя переменной таймера.

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

Получить значение программного таймера, для внутреннего использования.

t	Значение таймера.

5.33.3

5.33.3.1 void wait_time (timer_t *time*)

Подождать заданный интервал времени.

Просто ждет в цикле пока пройдет время time.

time	Время ожидания.

5.33.3.2 void _clear_timer (timer_t * *t*)

Сброс программного таймера, для внутреннего использования.

t	Указатель на таймер.

5.33.3.3 timer_t _timer (timer_t *t*)

Получить значение программного таймера, для внутреннего использования.

t	Значение таймера.

5.34 xlist.c

#include "bugurt.h"

- 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.

5.34.1

5.34.1.1 void xlist_init (xlist_t * *xlist*)

Инициализация списка.

xlist	Указатель на список.

5.34.1.2 item_t* xlist_head (xlist_t * *xlist*)

Поиск головы списка.

xlist	Указатель на список.

Указатель на голову - самый приоритетный элемент в массиве указателей.

5.34.1.3 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.

5.35 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.

5.35.1

Заголовок списков с приоритетами.

5.35.2

5.35.2.1 typedef struct _xlist_t xlist_t

Смотри _xlist_t;

5.35.3

5.35.3.1 void xlist_init (xlist_t * *xlist*)

Инициализация списка.

xlist	Указатель на список.

5.35.3.2 item_t* xlist_head (xlist_t * *xlist*)

Поиск головы списка.

xlist	Указатель на список.

Указатель на голову - самый приоритетный элемент в массиве указателей.

5.35.3.3 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.