BuguRTOS

0.6.1

Создано системой Doxygen 1.6.3

Sun Jun 30 11:25:09 2013

Содержание

1 Титульная страница

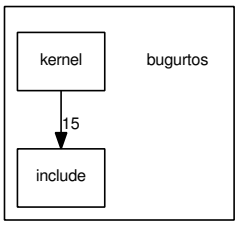
BuguRTOS - ядро операционной системы реального времени. Написано анонимусом ДЛЯ УДОВОЛЬСТВИЯ.

Предупреждения

Распространяется под измененной лицензией GPLv3, смотрите exception.txt.

2 Директории

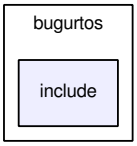
2.1 Содержание директории bugurtos/



Директории

- директория include
- директория kernel

2.2 Содержание директории bugurtos/include/



Файлы

- файл bugurt.h
  Главный заголовочный файл.
- файл crit_sec.h
  Заголовок критических секций.
- файл index.h
  Заголовок функции поиска в бинарном индексе.
- файл ipc.h
  Заголовок IPC.
- файл item.h
  Заголовок элементов 2-связного списка.
- файл kernel.h
  Заголовок Ядра.
- файл mutex.h
  Заголовок мьютекса.
- файл pcounter.h
  Заголовок счетчиков захваченных ресурсов.
- файл pitem.h
  Заголовок элементов списка с приоритетами.
- файл proc.h
  Заголовок процессов.
- файл sched.h
  Заголовок планировщика.
- файл sem.h
  Заголовок счетных семафоров.
- файл sig.h
  Заголовок сигналов.

- файл syscall.h

  Заголовок системных вызовов.

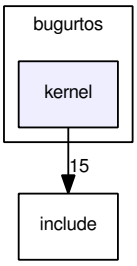
- файл timer.h

  Заголовок программных таймеров.

- файл xlist.h

  Заголовок списков с приоритетами.

2.3 Содержание директории bugurtos/kernel/



Файлы

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

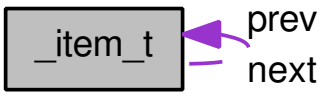
3 Структуры данных

3.1 Структура _item_t

Элемент 2-связного списка.

#include "item.h"

Граф связей класса _item_t:



Поля данных

- item_t * next
- item_t * prev

3.1.1 Подробное описание

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

3.1.2 Поля

3.1.2.1 item_t* next

Следующий элемент.

3.1.2.2 item_t* prev

Предыдущий элемент.

Объявления и описания членов структуры находятся в файле:

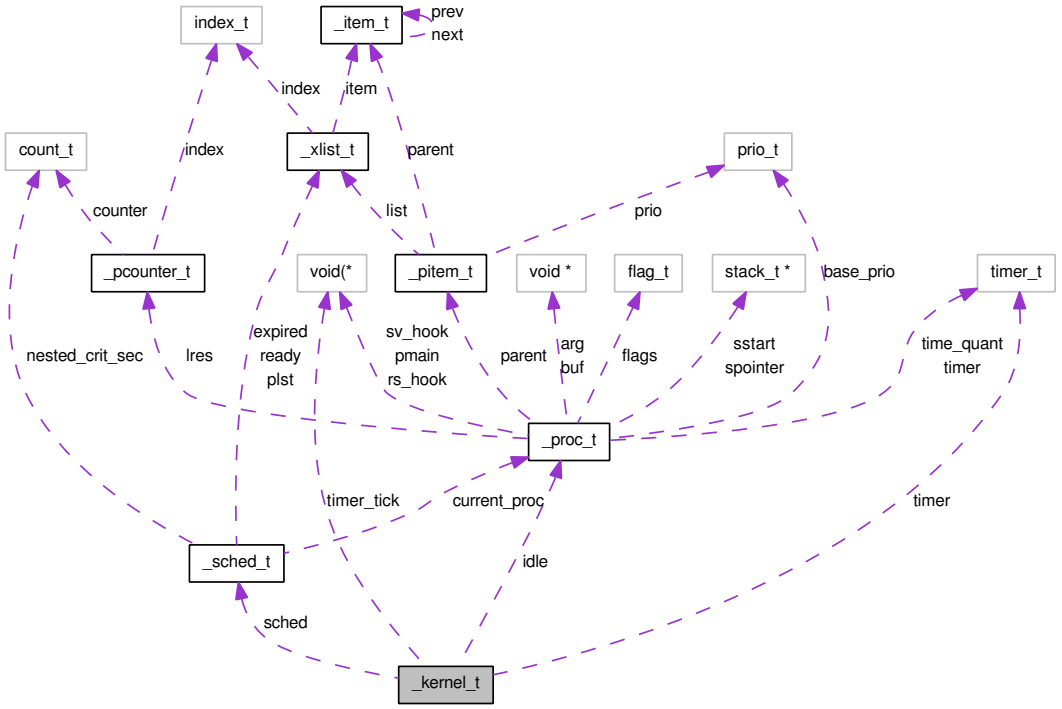
- bugurtos/include/item.h

3.2 Структура _kernel_t

Ядро BuguRTOS.

#include "kernel.h"

Граф связей класса _kernel_t:



Поля данных

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

3.2.1 Подробное описание

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

3.2.2 Поля

3.2.2.1 sched_t sched

Планировшик.

3.2.2.2 proc_t idle

Процесс холостого хода.

3.2.2.3 timer_t timer

Системный таймер.

3.2.2.4 void(* timer_tick)(void)

Хук обработчика системного таймера.

Объявления и описания членов структуры находятся в файле:

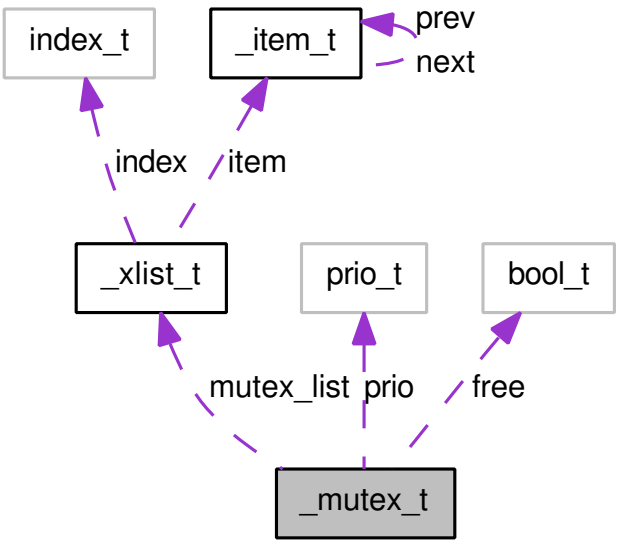
- bugurtos/include/kernel.h

3.3 Структура _mutex_t

Мьютекс.

#include "mutex.h"

Граф связей класса _mutex_t:



Поля данных

- xlist_t mutex_list
- prio_t prio
- bool_t free

3.3.1 Подробное описание

Используется для управления доступом к общим ресурсам, в тех случаях, когда общий ресурс нужен в течение долгого времени. Поддерживается произвольная вложенность мьютексов. При использовании опции CONFIG_USE_HIGHEST_LOCKER, работа с мьютексами производится по протоколу highest locker, более подробно написано в Википедии.

Предупреждения

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

3.3.2 Поля

3.3.2.1 xlist_t mutex_list

Список ожидающих процессов.

3.3.2.2 prio_t prio

Приоритет.

3.3.2.3 bool_t free

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

Объявления и описания членов структуры находятся в файле:

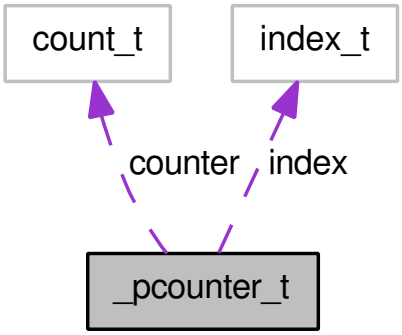
- bugurtos/include/mutex.h

3.4 Структура _pcounter_t

Счетчик захваченных ресурсов.

#include "pcounter.h"

Граф связей класса _pcounter_t:



Поля данных

- count_t counter [BITS_IN_INDEX_T]
- index_t index

3.4.1 Подробное описание

При использовании опции CONFIG_USE_HIGHEST_LOCKER используется для пересчета захваченных процессом ресурсов.

3.4.2 Поля

3.4.2.1 count_t counter[BITS_IN_INDEX_T]

Массив счетчиков.

3.4.2.2 index_t index

Индекс для ускорения поиска.

Объявления и описания членов структуры находятся в файле:

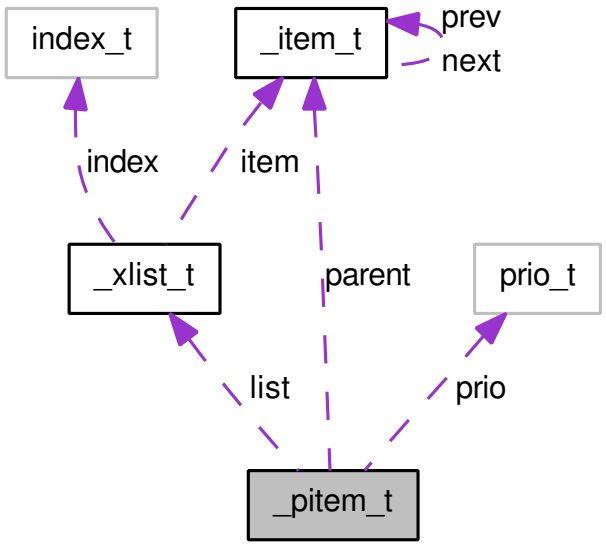
- bugurtos/include/pcounter.h

3.5 Структура _pitem_t

Элемент списка с приоритетами.

#include "pitem.h"

Граф связей класса _pitem_t:



Поля данных

- item_t parent
- xlist_t * list
- prio_t prio

3.5.1 Поля

3.5.1.1 item_t parent

Родитель - item_t.

3.5.1.2 xlist_t* list

Указательна спиок в который будем вставлять.

3.5.1.3 prio_t prio

Приоритет.

Объявления и описания членов структуры находятся в файле:

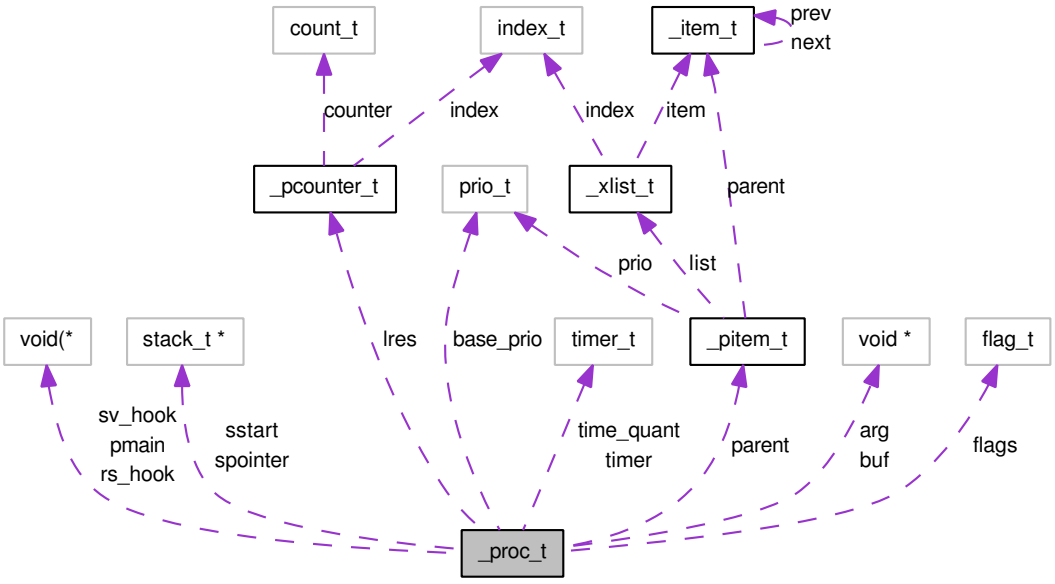
- bugurtos/include/pitem.h

3.6 Структура _proc_t

Процесс.

#include "proc.h"

Граф связей класса _proc_t:



Поля данных

- pitem_t parent
- flag_t flags
- prio_t base_prio
- pcounter_t lres
- timer_t time_quant
- timer_t timer
- void * buf
- code_t pmain
- code_t sv_hook
- code_t rs_hook

- void * arg
- stack_t * sstart
- stack_t * spointer

3.6.1 Подробное описание

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.

Предупреждения

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

3.6.2 Поля

3.6.2.1 pitem_t parent

Родитель - pitem_t.

3.6.2.2 flag_t flags

Флаги (для ускорения анализа состояния процесса).

3.6.2.3 prio_t base_prio

Базовый приоритет.

3.6.2.4 pcounter_t lres

Счетчик захваченных ресурсов.

3.6.2.5 timer_t time_quant

Квант времени процесса.

3.6.2.6 timer_t timer

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

3.6.2.7 void* buf

Указатель на хранилище для передачи данных через IPC.

3.6.2.8 code_t pmain

Главная функция процесса.

3.6.2.9 code_t sv_hook

Хук, исполняется планировщиком после сохранения контекста процесса.

3.6.2.10 code_t rs_hook

Хук, исполняется планировщиком перед восстановлением контекста процесса.

3.6.2.11 void* arg

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

3.6.2.12 stack_t* sstart

Указатель на дно стека экземпляра процесса.

3.6.2.13 stack_t* spointer

Указатель на вершину стека экземпляра процесса.

Объявления и описания членов структуры находятся в файле:

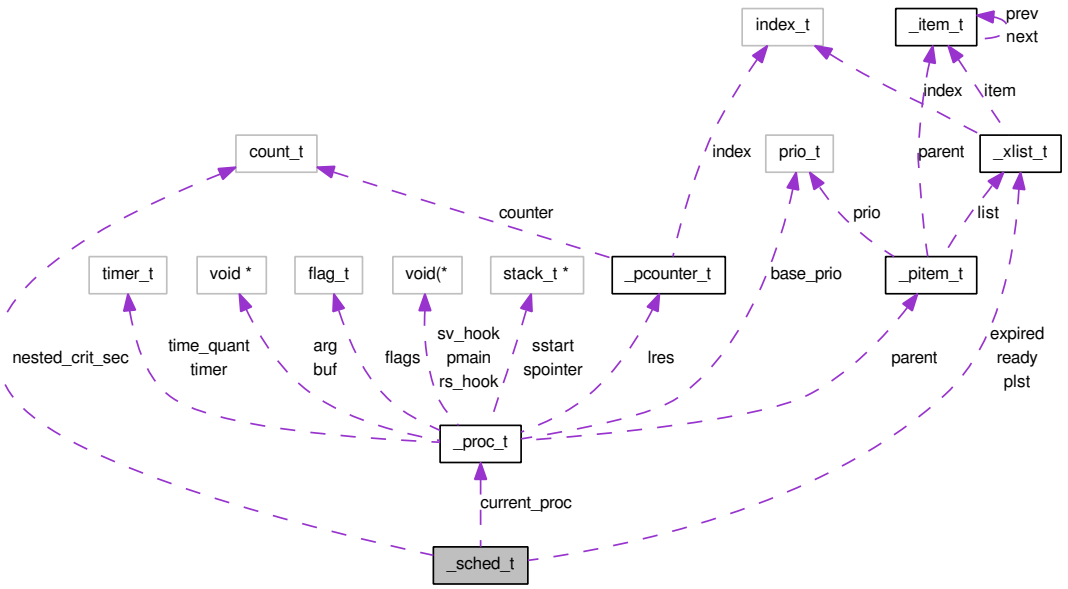
- bugurtos/include/proc.h

3.7 Структура _sched_t

Планировщик.

#include "sched.h"

Граф связей класса _sched_t:



Поля данных

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

3.7.1 Подробное описание

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

3.7.2 Поля

3.7.2.1 proc_t* current_proc

Текущий процесс.

3.7.2.2 xlist_t* ready

Указатель на список готовых к выполнению процессов.

3.7.2.3 xlist_t* expired

Указатель на список процессов, исчерпавших свой квант времени.

3.7.2.4 xlist_t plst[2]

Сами списки процесов.

3.7.2.5 count_t nested_crit_sec

Счетчик вложенности критических секций.

Объявления и описания членов структуры находятся в файле:

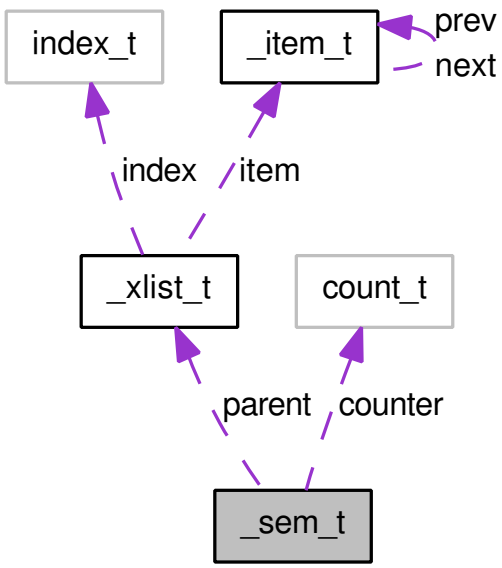
- bugurtos/include/sched.h

3.8 Структура _sem_t

Счетный семафор.

#include "sem.h"

Граф связей класса _sem_t:



Поля данных

- xlist_t parent
- count_t counter

3.8.1 Подробное описание

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.

3.8.2 Поля

3.8.2.1 xlist_t parent

Потомок списка, да.

3.8.2.2 count_t counter

Счетчик семафора.

Объявления и описания членов структуры находятся в файле:

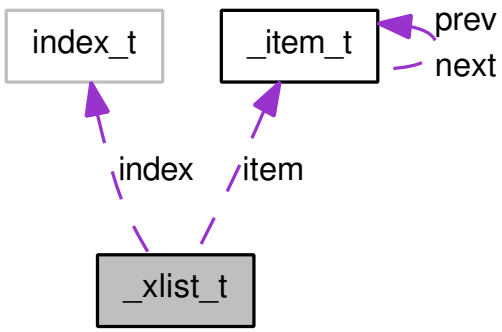
- bugurtos/include/sem.h

3.9 Структура _xlist_t

Список с приоритетами.

#include "xlist.h"

Граф связей класса _xlist_t:



Поля данных

- item_t * item [BITS_IN_INDEX_T]
- index_t index

3.9.1 Подробное описание

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

3.9.2 Поля

3.9.2.1 item_t* item[BITS_IN_INDEX_T]

Массив указателей на элементы.

3.9.2.2 index_t index

Индекс, показывает, где в массиве ненулевые указатели.

Объявления и описания членов структуры находятся в файле:

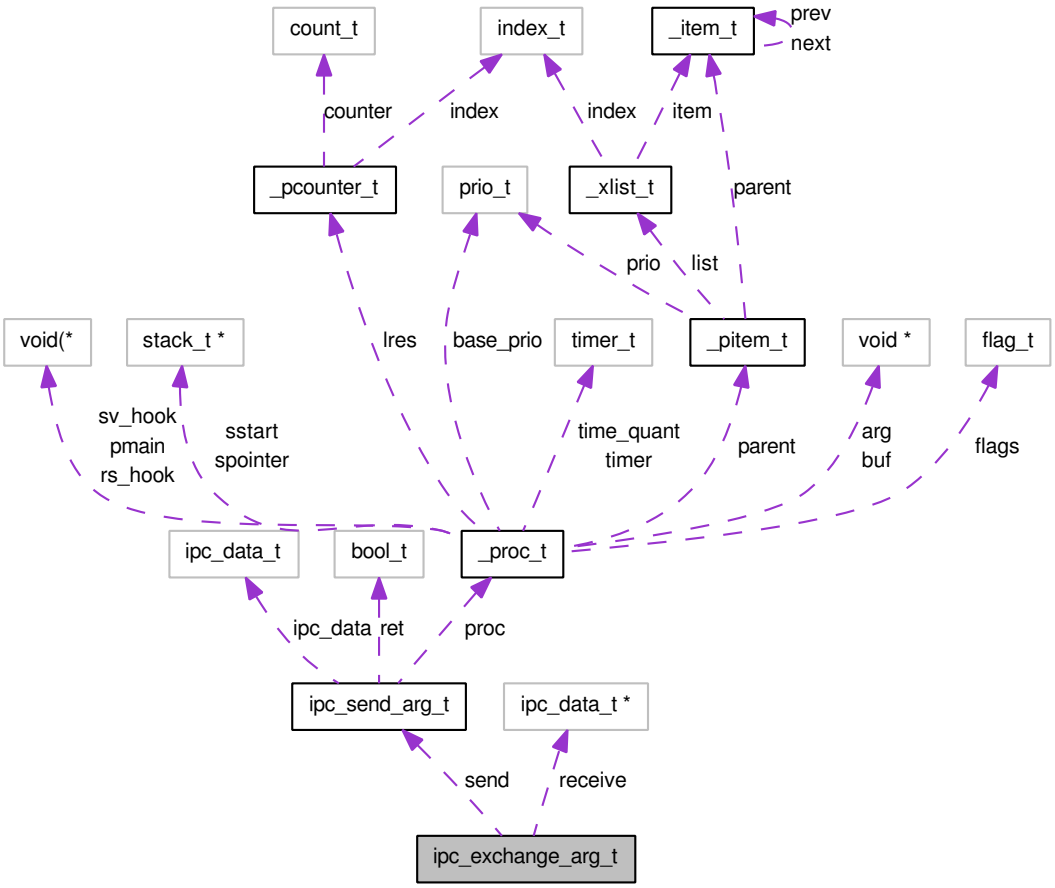
- bugurtos/include/xlist.h

3.10 Структура ipc_exchange_arg_t

Параметр системного вызова SYSCALL_IPC_EXCHANGE.

#include "syscall.h"

Граф связей класса ipc_exchange_arg_t:



Поля данных

- ipc_send_arg_t send
- ipc_data_t * receive

3.10.1 Поля

3.10.1.1 ipc_send_arg_t send

Родитель.

3.10.1.2 ipc_data_t* receive

Указатель на хранилище принимаемых данных.

Объявления и описания членов структуры находятся в файле:

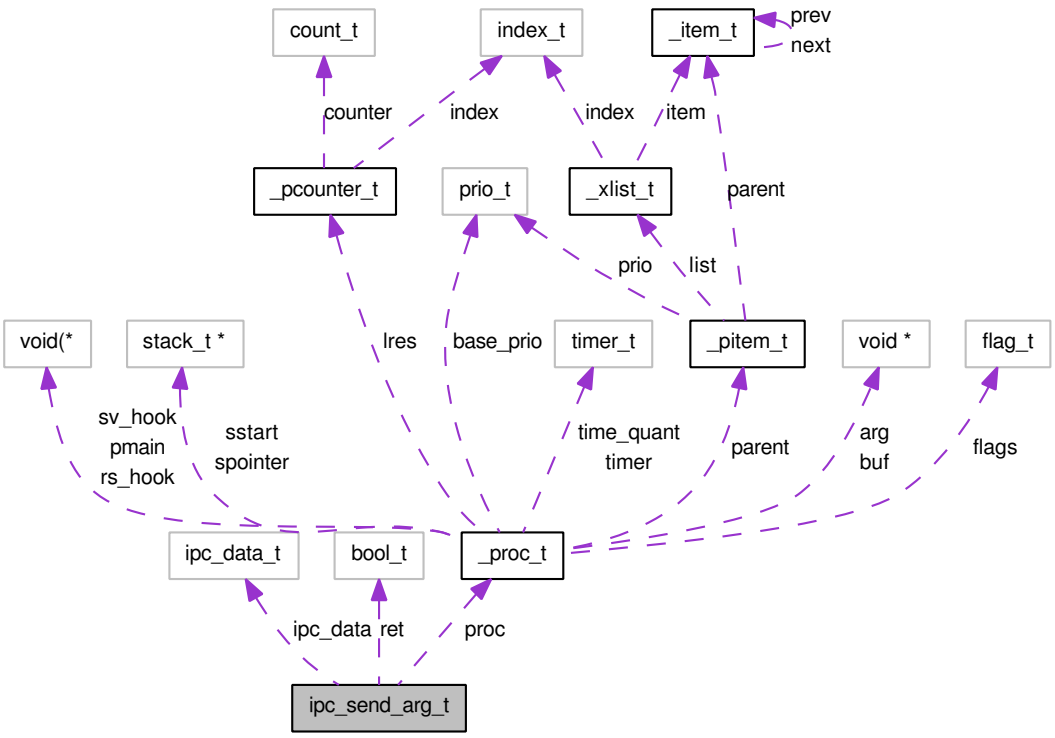
- bugurtos/include/syscall.h

3.11 Структура ipc_send_arg_t

Параметр системного вызова SYSCALL_IPC_SEND.

#include "syscall.h"

Граф связей класса ipc_send_arg_t:



Поля данных

- proc_t * proc
- bool_t ret
- ipc_data_t ipc_data

3.11.1 Поля

3.11.1.1 proc_t* proc

указатель на процесс-адресат.

3.11.1.2 bool_t ret

хранилище результата выполнения операции.

3.11.1.3 ipc_data_t ipc_data

данные для передачи.

Объявления и описания членов структуры находятся в файле:

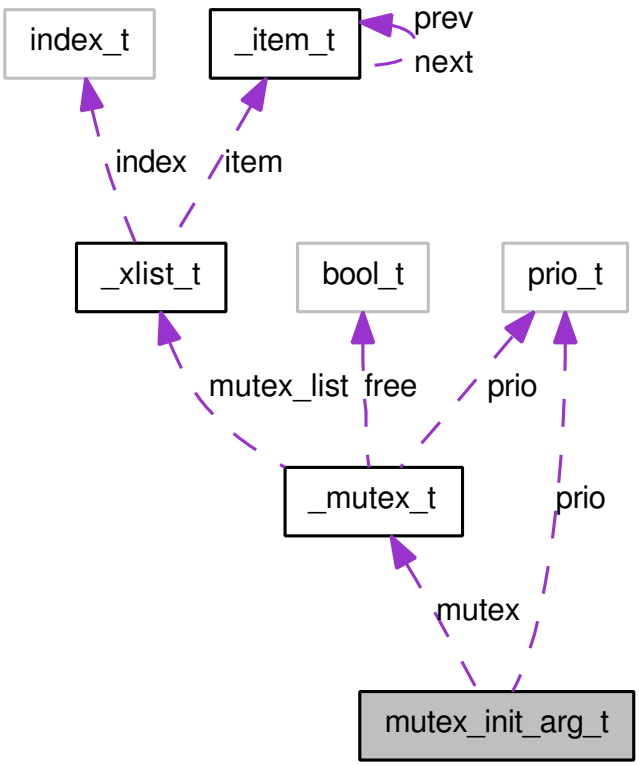
- bugurtos/include/syscall.h

3.12 Структура mutex_init_arg_t

Параметр системного вызова SYSCALL_MUTEX_INIT.

#include "syscall.h"

Граф связей класса mutex_init_arg_t:



Поля данных

- mutex_t * mutex
- prio_t prio

3.12.1 Поля

3.12.1.1 mutex_t* mutex

указатель на мьютекс.

3.12.1.2 prio_t prio

приоритет мьютекса

Объявления и описания членов структуры находятся в файле:

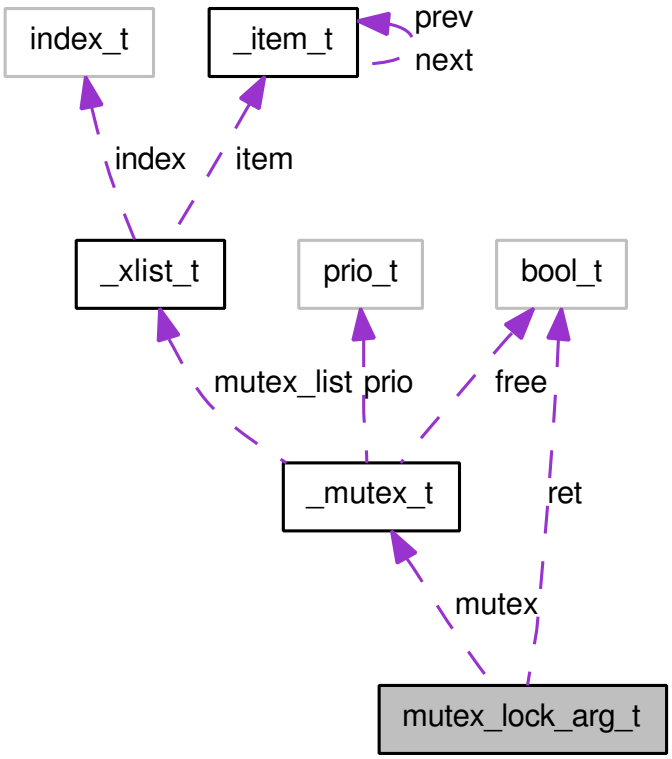
- bugurtos/include/syscall.h

3.13 Структура mutex_lock_arg_t

Параметр системных вызовов SYSCALL_MUTEX_LOCK и SYSCALL_MUTEX_TRY_LOCK.

#include "syscall.h"

Граф связей класса mutex_lock_arg_t:



Поля данных

- mutex_t * mutex
- bool_t ret

3.13.1 Поля

3.13.1.1 mutex_t* mutex

указатель на мьютекс.

3.13.1.2 bool_t ret

хранилище результата выполнения операции.

Объявления и описания членов структуры находятся в файле:

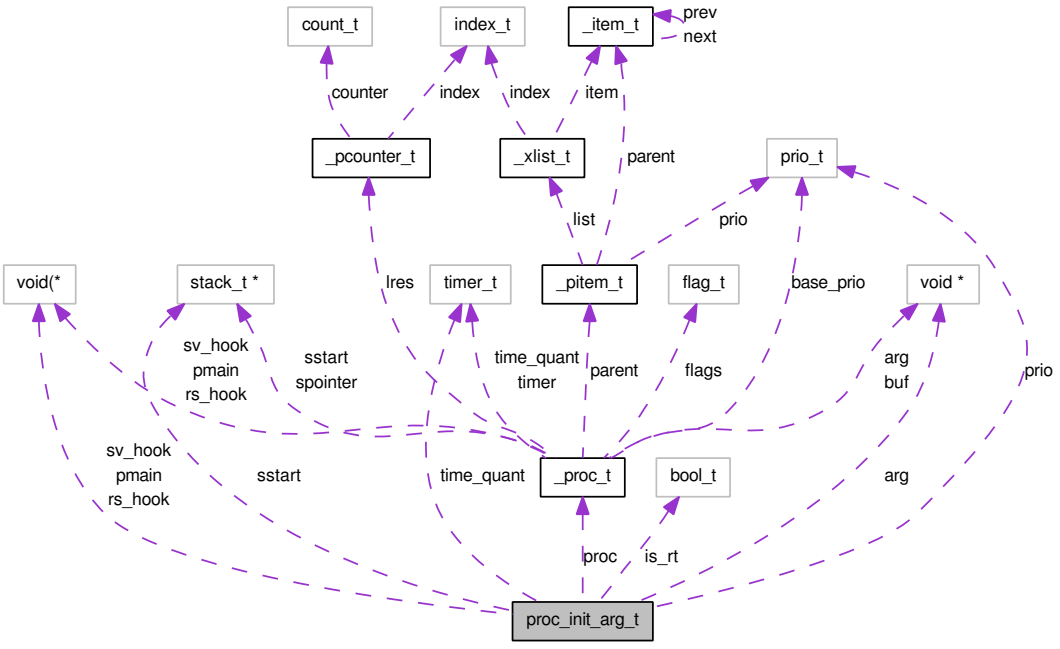
- bugurtos/include/syscall.h

3.14 Структура proc_init_arg_t

Параметр системного вызова SYSCALL_PROC_INIT.

#include "syscall.h"

Граф связей класса proc_init_arg_t:



Поля данных

- proc_t * proc

- code_t pmain
- code_t sv_hook
- code_t rs_hook
- void * arg
- stack_t * sstart
- prio_t prio
- timer_t time_quant
- bool_t is_rt

3.14.1 Подробное описание

Содержит информацию о процессе, и его свойствах.

3.14.2 Поля

3.14.2.1 proc_t* proc

Указатель на инициируемый процесс.

3.14.2.2 code_t pmain

Указатель на главную функцию процесса.

3.14.2.3 code_t sv_hook

Указатель на хук proc->sv_hook.

3.14.2.4 code_t rs_hook

Указатель на хук proc->rs_hook. Хук, исполняется планировщиком перед восстановлением контекста процесса.

3.14.2.5 void* arg

Указатель на аргумент.

3.14.2.6 stack_t* sstart

Указатель на дно стека процесса.

3.14.2.7 prio_t prio

Приоритет.

3.14.2.8 timer_t time_quant

Квант времени.

3.14.2.9 bool_t is_rt

Объявления и описания членов структуры находятся в файле:

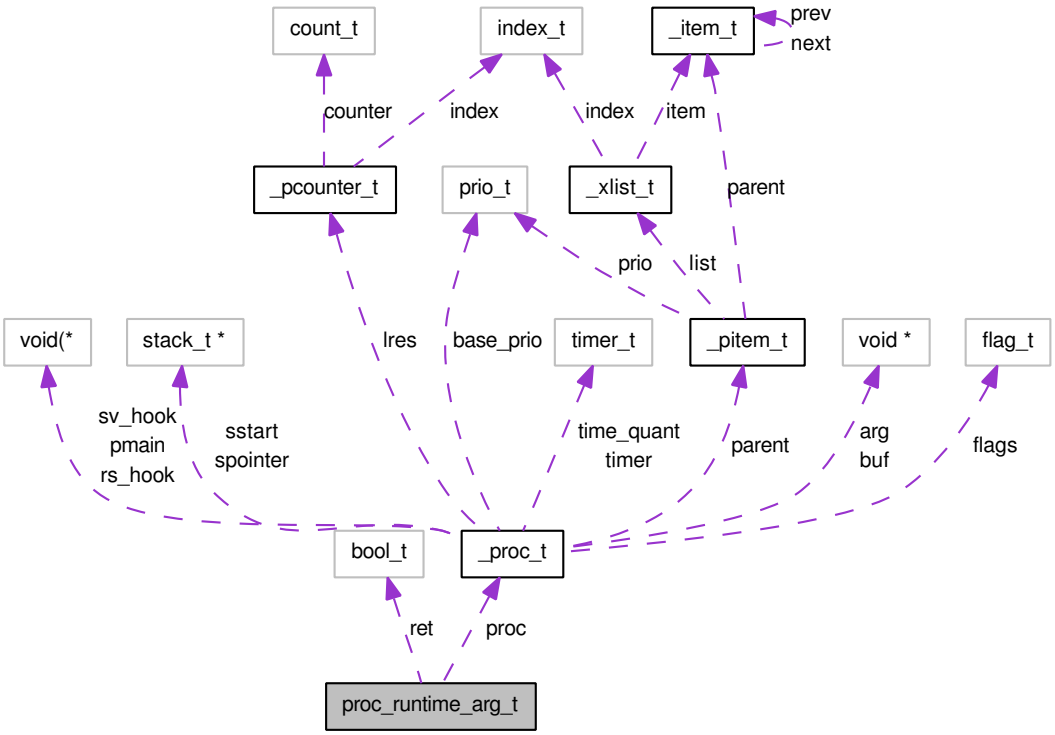
- bugurtos/include/syscall.h

3.15 Структура proc_runtime_arg_t

Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

#include "syscall.h"

Граф связей класса proc_runtime_arg_t:



Поля данных

- proc_t * proc
- bool_t ret

3.15.1 Поля

3.15.1.1 proc_t* proc

Указатель на процесс.

3.15.1.2 bool_t ret

Результат выполнения системного вызова.

Объявления и описания членов структуры находятся в файле:

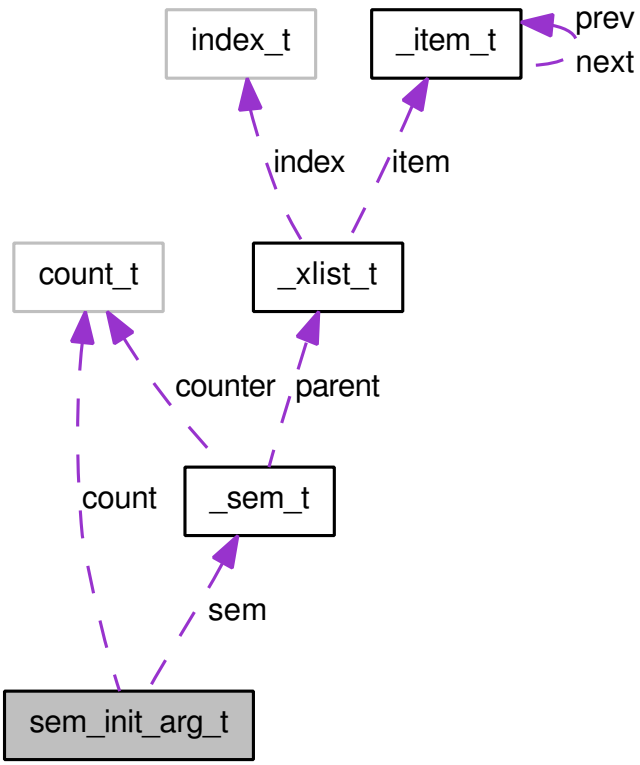
- bugurtos/include/syscall.h

3.16 Структура sem_init_arg_t

Параметр системного вызова SYSCALL_SEM_INIT.

#include "syscall.h"

Граф связей класса sem_init_arg_t:



Поля данных

- sem_t * sem
- count_t count

3.16.1 Поля

3.16.1.1 sem_t* sem

указатель на семафор.

3.16.1.2 count_t count

начальное значение счетчика семафора.

Объявления и описания членов структуры находятся в файле:

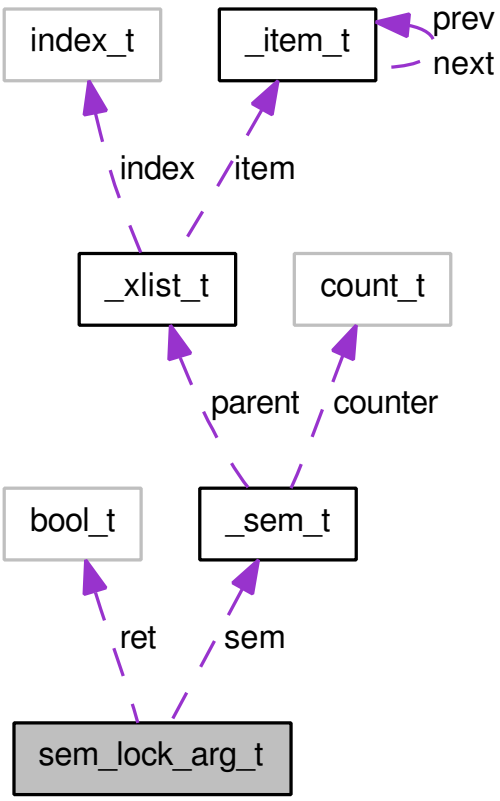
- bugurtos/include/syscall.h

3.17 Структура sem_lock_arg_t

Параметр системных вызовов SYSCALL_SEM_LOCK и SYSCALL_SEM_TRY_LOCK.

#include "syscall.h"

Граф связей класса sem_lock_arg_t:



Поля данных

- sem_t * sem
- bool_t ret

3.17.1 Поля

3.17.1.1 sem_t* sem

указатель на семафор.

3.17.1.2 bool_t ret

хранилище результата выполнения операции.

Объявления и описания членов структуры находятся в файле:

- bugurtos/include/syscall.h

4 Файлы

4.1 Файл bugurtos/include/bugurt.h

Главный заголовочный файл.

#include "index.h"

Макросы

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

Определения типов

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

Функции

- 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.

4.1.1 Подробное описание

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

4.1.2 Макросы

4.1.2.1 #define SPIN_INIT(arg)

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

4.1.2.2 #define SPIN_LOCK(arg)

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

4.1.2.3 #define SPIN_UNLOCK(arg)

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

4.1.2.4 #define RESCHED_PROC(proc) resched()

Обертка функции resched.

4.1.3 Типы

4.1.3.1 typedef void(* code_t)(void *)

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

4.1.4 Функции

4.1.4.1 void resched (void)

Запускает перепланировку.

4.1.4.2 void disable_interrupts (void)

Глобальный запрет прерываний на системе.

4.1.4.3 void enable_interrupts (void)

Глобальное разрешение прерываний на системе.

4.1.4.4 proc_t* current_proc (void)

Текущий процесс.

Возвращает

Указатель на текущий процесс, исполняемый на локальном процессре.

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

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

Аргументы

sstart Дно стека.

code Функция, которая будет вызвана после восстановления контекста.

arg Аргумент вызываемой функции.

Возвращает

Указатель на вершину подготовленного стека.

4.1.4.6 void init_bugurt (void)

Подготовка Ядра к запуску.

4.1.4.7 void start_bugurt (void)

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

4.1.4.8 void syscall_bugurt (syscall_t num, void * arg)

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.

Аргументы

num номер системного вызова (что именно надо выполнить).

arg аргумент системного вызова (над чем это надо выполнить).

4.2 Файл bugurtos/include/crit_sec.h

Заголовок критических секций.

Макросы

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

Функции

- void enter_crit_sec (void)
- void exit_crit_sec (void)
  Выход из критической секции.

4.2.1 Подробное описание

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.

4.2.2 Макросы

4.2.2.1 #define ENTER_CRIT_SEC() enter_crit_sec()

Вход в критическую секцию.

Предупреждения

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

Макрос-обертка.Выход из критической секции.

Предупреждения

Использовать в конце блока!

4.2.2.2 #define EXIT_CRIT_SEC() exit_crit_sec()

4.2.3 Функции

4.2.3.1 void enter_crit_sec (void)

Вход в критическую секцию.

4.2.3.2 void exit_crit_sec (void)

Вход в критическую секцию.

4.3 Файл bugurtos/include/index.h

Заголовок функции поиска в бинарном индексе.

Функции

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

4.3.1 Подробное описание

4.3.2 Функции

4.3.2.1 prio_t index_search (index_t index)

Аргументы

index Бинарный индекс.

Возвращает

Наивысший (с минимальным значением) приоритет в индексе.

An index search.

Аргументы

index An index.

Возвращает

Highest priority of an index (with minimal value).

4.4 Файл bugurtos/include/ipc.h

Заголовок IPC.

Функции

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

4.4.1 Подробное описание

4.4.2 Функции

4.4.2.1 void _ipc_wait (void * ipc_pointer)

Предупреждения

Для внутреннего использования.

Аргументы

ipc_pointer Указатель на хранилище для передаваемых данных.

4.4.2.2 ipc_data_t ipc_wait (void)

Возвращает

Данные.

4.4.2.3 bool_t ipc_send (proc_t * proc, ipc_data_t ipc_data)

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

Аргументы

proc Указатель на процесс-адресат.

data Данные для передачи.

Возвращает

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

4.4.2.4 bool_t ipc_send_isr (proc_t * proc, ipc_data_t ipc_data)

Предупреждения

Для вызова из обработчиков прерываний!

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

Аргументы

proc Указатель на процесс-адресат.

data Данные для передачи.

Возвращает

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

Обработка флага останова целевого процесса

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

Предупреждения

Для внутреннего использования!

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

Аргументы

proc Указатель на процесс-адресат.

send Данные для передачи.

receive Указатель на хранилище данных для приема.

Возвращает

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

Обработка флага останова целевого процесса

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

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

Аргументы

proc Указатель на процесс-адресат.

send Данные для передачи.

receive указатель на хранилище данных для приема.

Возвращает

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

4.5 Файл bugurtos/include/item.h

Заголовок элементов 2-связного списка.

Структуры данных

- struct _item_t
  Элемент 2-связного списка.

Макросы

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

Определения типов

- typedef struct _item_t item_t

Функции

- 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.

4.5.1 Подробное описание

4.5.2 Макросы

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

Статическая инициализация объекта типа item_t.

Аргументы

a Имя переменной типа item_t.

4.5.3 Типы

4.5.3.1 typedef struct _item_t item_t

4.5.4 Функции

4.5.4.1 void item_init (item_t * item)

Аргументы

item Указатель на объект item_t.

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

Аргументы

item Указатель на объект типа item_t, который будем вставлять.

head Указатель на голову списка типа item_t.

4.5.4.3 void item_cut (item_t * item)

Аргументы

item Указатель на объект типа item_t, который будем вырезать.

4.6 Файл bugurtos/include/kernel.h

Заголовок Ядра.

Структуры данных

- struct _kernel_t
  Ядро BuguRTOS.

Определения типов

- typedef struct _kernel_t kernel_t

Функции

- void kernel_init (void)
  Инициализация Ядра.
- void idle_main (void *arg)
  Главная функция процесса холостого хода.

Переменные

- kernel_t kernel
  Ядро BuguRTOS.

4.6.1 Подробное описание

4.6.2 Типы

4.6.2.1 typedef struct _kernel_t kernel_t

4.6.3 Функции

4.6.3.1 void kernel_init (void)

Готовит ядро к запуску.

4.6.3.2 void idle_main (void * arg)

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

Предупреждения

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!

Аргументы

arg Указатель на аргумент.

4.6.4 Переменные

4.6.4.1 kernel_t kernel

Оно одно на всю систему!

4.7 Файл bugurtos/include/mutex.h

Заголовок мьютекса.

Структуры данных

- struct _mutex_t
  Мьютекс.

Макросы

- #define GET_PRIO(mutex) mutex->prio

Определения типов

- typedef struct _mutex_t mutex_t

Функции

- 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.
- bool_t mutex_lock (mutex_t *mutex)
  Захват мьютекса.
- bool_t mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса.
- void mutex_unlock (mutex_t *mutex)
  Освобождение мьютекса.

- bool_t _mutex_lock (mutex_t *mutex)
  Захват мьютекса, для внутреннего использования.
- bool_t _mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса, для внутреннего использования.
- void _mutex_unlock (mutex_t *mutex)
  Освобождение мьютекса, для внутреннего использования.

4.7.1 Подробное описание

4.7.2 Макросы

4.7.2.1 #define GET_PRIO(mutex) mutex->prio

4.7.3 Типы

4.7.3.1 typedef struct _mutex_t mutex_t

4.7.4 Функции

4.7.4.1 void mutex_init_isr (mutex_t * mutex, prio_t prio)

Да, инициировать из обработчика прерывания можно!

Аргументы

mutex Указатель на мьютекс.

prio В случае использования CONFIG_USE_HIGHEST_LOCKER, - приоритет мьютекса.

4.7.4.2 void mutex_init (mutex_t * mutex, prio_t prio)

Аргументы

mutex Указатель на мьютекс.

prio В случае использования CONFIG_USE_HIGHEST_LOCKER, - приоритет мьютекса.

4.7.4.3 bool_t mutex_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.7.4.4 bool_t mutex_try_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.7.4.5 void mutex_unlock (mutex_t * mutex)

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.

Аргументы

mutex Указатель на мьютекс.

4.7.4.6 bool_t _mutex_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.7.4.7 bool_t _mutex_try_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.7.4.8 void _mutex_unlock (mutex_t * mutex)

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.

Аргументы

mutex Указатель на мьютекс.

KERNEL_PREEMPT

4.8 Файл bugurtos/include/pcounter.h

Заголовок счетчиков захваченных ресурсов.

Структуры данных

- struct _pcounter_t
  Счетчик захваченных ресурсов.

Определения типов

- typedef struct _pcounter_t pcounter_t

Функции

- 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.

4.8.1 Подробное описание

4.8.2 Типы

4.8.2.1 typedef struct _pcounter_t pcounter_t

4.8.3 Функции

4.8.3.1 void pcounter_init (pcounter_t * pcounter)

Аргументы

pcounter Указатель на счетчик.

4.8.3.2 void pcounter_inc (pcounter_t * pcounter, prio_t prio)

Аргументы

pcounter Указатель на счетчик.
prio Приоритет.

4.8.3.3 index_t pcounter_dec (pcounter_t * pcounter, prio_t prio)

Аргументы

pcounter Указатель на счетчик.
prio Приоритет.

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

Аргументы

pcounter Указатель на счетчик.

prio Приоритет.

count Количество единиц.

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

Аргументы

pcounter Указатель на счетчик.

prio Приоритет.

count Количество единиц.

Возвращает

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

4.9 Файл bugurtos/include/pitem.h

Заголовок элементов списка с приоритетами.

Структуры данных

- struct _pitem_t
  Элемент списка с приоритетами.

Макросы

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

Определения типов

- typedef struct _pitem_t pitem_t

Функции

- 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.

4.9.1 Подробное описание

4.9.2 Макросы

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

Статическая инициализация объекта типа pitem_t

Аргументы

a Имя переменной.
p Приоритет.

4.9.3 Типы

4.9.3.1 typedef struct _pitem_t pitem_t

4.9.4 Функции

4.9.4.1 void pitem_init (pitem_t * pitem, prio_t prio)

Аргументы

pitem Указатель на объект pitem_t.
prio Приоритет элемента.

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

Аргументы

pitem Указатель на объект pitem_t.
xlist Указатель на список.

4.9.4.3 void pitem_fast_cut (pitem_t * pitem)

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

Аргументы

pitem Указатель на объект pitem_t.

4.9.4.4 void pitem_cut (pitem_t * pitem)

Вызывает pitem_fast_cut и обнуляет указатель pitem->list.

Аргументы

pitem Указатель на объект pitem_t.

4.9.4.5 pitem_t* pitem_xlist_chain (xlist_t * src)

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

Аргументы

src Указатель на объект xlist_t.

Возвращает

Указаьель на голову 2-связного списка.

4.10 Файл bugurtos/include/proc.h

Заголовок процессов.

Структуры данных

- struct _proc_t
  Процесс.

Макросы

- #define PROC_LRES_INIT(a) pcounter_init(&a->lres)
  Макрос-обертка.
- #define PROC_LRES_INC(a, b) _proc_lres_inc(a,b)
  Макрос-обертка.
- #define PROC_LRES_DEC(a, b) _proc_lres_dec(a,b)
  Макрос-обертка.
- #define PROC_PRIO_CONTROL_STOPED(a) _proc_prio_control_stoped(a)
  Макрос-обертка.
- #define PROC_FLG_RT ((flag_t)0x80)
  Флаг реального времени.
- #define PROC_FLG_MUTEX ((flag_t)0x40)
  Флаг захвата мьютексов.

- #define PROC_FLG_SEM ((flag_t)0x20)

  Флаг захвата семафора.

- #define PROC_FLG_PRE_STOP ((flag_t)0x10)

  Флаг запроса останова.

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

  Маска PROC_FLG_MUTEX или PROC_FLG_SEM.

- #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

  Маска очистки состояния исполнения процесса.

- #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)

  Маска очистки состояния исполнения процесса.

- #define PROC_STATE_MASK ((flag_t)0x0F)

  Маска состояния исполнения процесса.

- #define PROC_STATE_RESTART_MASK ((flag_t)0xC)

  Маска проверки состояния процесса.

- #define PROC_STATE_RUN_MASK ((flag_t)0x7)

  Маска проверки состояния процесса.

- #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

  Маска проверки состояния процесса.

- #define PROC_STATE_STOPED ((flag_t)0x0)
- #define PROC_STATE_END ((flag_t)0x1)
- #define PROC_STATE_W_WD_STOPED ((flag_t)0x2)
- #define PROC_STATE_WD_STOPED ((flag_t)0x3)
- #define PROC_STATE_DEAD ((flag_t)0x4)
- #define PROC_STATE_READY ((flag_t)0x5)
- #define PROC_STATE_RESERVED_0x6 ((flag_t)0x6)
- #define PROC_STATE_RUNNING ((flag_t)0x7)
- #define PROC_STATE_W_MUT ((flag_t)0x8)
- #define PROC_STATE_W_SEM ((flag_t)0x9)
- #define PROC_STATE_W_SIG ((flag_t)0xA)
- #define PROC_STATE_W_IPC ((flag_t)0xB)
- #define PROC_STATE_W_DEAD ((flag_t)0xC)
- #define PROC_STATE_W_READY ((flag_t)0xD)
- #define PROC_STATE_RESERVED_0xE ((flag_t)0xE)
- #define PROC_STATE_W_RUNNING ((flag_t)0xF)
- #define PROC_PRE_STOP_TEST(a) ((a->flags & PROC_FLG_PRE_STOP) && (!(a->flags & PROC_FLG_LOCK_MASK)))

  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_IPC_TEST(a) ((a->flags & PROC_STATE_MASK) == PROC_STATE_W_IPC)

  Проверяет ждет ли процесс IPC.

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

  Вставка процесса в список готовых к выполнению, для внутреннего использования.

Определения типов

- typedef struct _proc_t proc_t

Функции

- 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).
- index_t _proc_yeld (void)
  Передача управления следующему процессу (для внутреннего использования).
- index_t proc_yeld (void)
  Передача управления следующему процессу.
- 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_run (proc_t *proc)
  "Низкоуровневый" запуск процесса, для внутреннего использования.
- void _proc_stop (proc_t *proc)
  "Низкоуровневый" останов процесса, для внутреннего использования.
- void _proc_stop_flags_set (proc_t *proc, flag_t mask)
  "Низкоуровневый" останов процесса с установкой флагов, для внутреннего использования.
- void _proc_flag_stop (flag_t mask)
  Останов процесса по флагу PROC_FLG_PRE_STOP из критической секции или обработчика прерывания, для внутреннего использования.
- void proc_flag_stop (flag_t mask)
  Останов процесса по флагу PROC_FLG_PRE_STOP.
- void _proc_lres_inc (proc_t *proc, prio_t prio)
  Инкремент счетчика захваченных ресурсов, для внутреннего использования.
- void _proc_lres_dec (proc_t *proc, prio_t prio)
  Декремент счетчика захваченных ресурсов, для внутреннего использования.
- void _proc_prio_control_stoped (proc_t *proc)
  Управление приоритетом процесса, для внутреннего использования.
- void _proc_prio_control_running (proc_t *proc)
  Управление приоритетом процесса, для внутреннего использования.

4.10.1 Подробное описание

4.10.2 Макросы

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

Инициирует поле proc->lres процесса.

Аргументы

a указатель на процесс.

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

Инкремент счетчика захваченных мьютексов.

Аргументы

a указатель на процесс.

b приоритет захваченного мьютекса, если используется протокол highest locker.

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

Декремент счетчика захваченных мьютексов.

Аргументы

a указатель на процесс.

b приоритет захваченного мьютекса, если используется протокол highest locker.

4.10.2.4 #define PROC_PRIO_CONTROL_STOPED(a) _proc_prio_control_stoped(a)

Управление приоритетом остановленного процесса. В случае использования протокола highest locker приводит поле proc->group->prio в соответствие с полем proc->lres, иначе - пустой макрос.

Аргументы

a указатель на процесс.

4.10.2.5 #define PROC_FLG_RT ((flag_t)0x80)

Для этого процесса используется политика планирования жесткого реального времени.

4.10.2.6 #define PROC_FLG_MUTEX ((flag_t)0x40)

Процесс удерживает мьютекс.

4.10.2.7 #define PROC_FLG_SEM ((flag_t)0x20)

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

4.10.2.8 #define PROC_FLG_PRE_STOP ((flag_t)0x10)

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

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

Нужна, чтобы определить, удерживает ли процесс общие ресурсы.

4.10.2.10 #define PROC_STATE_CLEAR_MASK ((flag_t)0xF0)

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

4.10.2.11 #define PROC_STATE_CLEAR_RUN_MASK ((flag_t)0xF8)

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

4.10.2.12 #define PROC_STATE_MASK ((flag_t)0x0F)

4.10.2.13 #define PROC_STATE_RESTART_MASK ((flag_t)0xC)

Используется функциями proc_restart и proc_restart_isr, для проверки возможности перезапуска.

4.10.2.14 #define PROC_STATE_RUN_MASK ((flag_t)0x7)

Используется для того, чтобы проверить, запущен ли процесс.

4.10.2.15 #define PROC_STATE_WAIT_MASK ((flag_t)0x8)

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

4.10.2.16 #define PROC_STATE_STOPED ((flag_t)0x0)

Начальное состояние, остановлен.

4.10.2.17 #define PROC_STATE_END ((flag_t)0x1)

Завершен.

4.10.2.18 #define PROC_STATE_W_WD_STOPED ((flag_t)0x2)

Остановлен по вачдог в состоянии W_RUNNING.

4.10.2.19 #define PROC_STATE_WD_STOPED ((flag_t)0x3)

Остановлен по вачдог.

4.10.2.20 #define PROC_STATE_DEAD ((flag_t)0x4)

Завершен до освобождения общих ресурсов.

4.10.2.21 #define PROC_STATE_READY ((flag_t)0x5)

Готов к выполнению.

4.10.2.22 #define PROC_STATE_RESERVED_0x6 ((flag_t)0x6)

Зарезервировано.

4.10.2.23 #define PROC_STATE_RUNNING ((flag_t)0x7)

Выполняется.

4.10.2.24 #define PROC_STATE_W_MUT ((flag_t)0x8)

Ожидает мьютекса.

4.10.2.25 #define PROC_STATE_W_SEM ((flag_t)0x9)

Ожидает семафора.

4.10.2.26 #define PROC_STATE_W_SIG ((flag_t)0xA)

Ожидает сигнала.

4.10.2.27 #define PROC_STATE_W_IPC ((flag_t)0xB)

Ожидает IPC.

4.10.2.28 #define PROC_STATE_W_DEAD ((flag_t)0xC)

Остановлен по вачдог в состоянии W_RUNNING до освобождения общих ресурсов.

4.10.2.29 #define PROC_STATE_W_READY ((flag_t)0xD)

Готов к выполнению (специальное).

4.10.2.30 #define PROC_STATE_RESERVED_0xE ((flag_t)0xE)

Зарезервировано.

4.10.2.31 #define PROC_STATE_W_RUNNING ((flag_t)0xF)

Выполняется (специальное).

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

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

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

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

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

4.10.3 Типы

4.10.3.1 typedef struct _proc_t proc_t

4.10.4 Функции

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

Аргументы

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.

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

Аргументы

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.

4.10.4.3 void proc_run_wrapper (proc_t * proc)

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

Аргументы

proc - Указатель на запускаемый процесс.

4.10.4.4 void proc_terminate (void)

4.10.4.5 void _proc_terminate (void)

4.10.4.6 bool_t proc_run (proc_t * proc)

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

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.10.4.7 bool_t proc_run_isr (proc_t * proc)

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

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.10.4.8 bool_t proc_restart (proc_t * proc)

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.

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.10.4.9 bool_t proc_restart_isr (proc_t * proc)

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.

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.10.4.10 bool_t proc_stop (proc_t * proc)

Вырезает процесс из списка готовых к выполнению и производит перепланировку.

Аргументы

proc - Указатель на останавливаемый процесс.

Возвращает

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

4.10.4.11 bool_t proc_stop_isr (proc_t * proc)

Вырезает процесс из списка готовых к выполнению и производит перепланировку.

Аргументы

proc - Указатель на останавливаемый процесс.

Возвращает

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

4.10.4.12 void proc_self_stop (void)

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

4.10.4.13 void _proc_self_stop (void)

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

4.10.4.14 index_t _proc_yeld (void)

Передает управление следующему процессу, если такой процесс есть.

Возвращает

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

KERNEL_PREEMPT

KERNEL_PREEMPT

4.10.4.15 index_t proc_yeld (void)

Передает управление следующему процессу, если такой процесс есть.

Возвращает

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

4.10.4.16 void proc_reset_watchdog (void)

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.

4.10.4.17 void _proc_reset_watchdog (void)

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.

4.10.4.18 void _proc_run (proc_t * proc)

4.10.4.19 void _proc_stop (proc_t * proc)

4.10.4.20 void _proc_stop_flags_set (proc_t * proc, flag_t mask)

4.10.4.21 void _proc_flag_stop (flag_t mask)

4.10.4.22 void proc_flag_stop (flag_t mask)

4.10.4.23 void _proc_lres_inc (proc_t * proc, prio_t prio)

Аргументы

proc Указатель на процесс, захвативший ресурс.

prio Приоритет захваченного ресурса, используется совместно с опцией CONFIG_USE_HIGHEST_LOCKER.

4.10.4.24 void _proc_lres_dec (proc_t * proc, prio_t prio)

Аргументы

proc Указатель на процесс, захвативший ресурс.

prio Приоритет захваченного ресурса, используется совместно с опцией CONFIG_USE_HIGHEST_LOCKER.

4.10.4.25 void _proc_prio_control_stoped (proc_t * proc)

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

Аргументы

proc - Указатель на процесс.

4.10.4.26 void _proc_prio_control_running (proc_t * proc)

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

Аргументы

proc - Указатель на процесс.

4.11 Файл bugurtos/include/sched.h

Заголовок планировщика.

Структуры данных

- struct _sched_t
  Планировщик.

Макросы

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

Определения типов

- typedef struct _sched_t sched_t

Функции

- void sched_init (sched_t *sched, proc_t *idle)
  Инициализация планировщика.
- void sched_schedule (void)
  Функция планирования.
- void sched_reschedule (void)
  Функция перепланирования.

4.11.1 Подробное описание

Предупреждения

Все функции в этом файле для внутреннего использования!!!

4.11.2 Макросы

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

Обертка инициализации переменной sched в функциях sched_schedule и sched_reschedule.

4.11.3 Типы

4.11.3.1 typedef struct _sched_t sched_t

4.11.4 Функции

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

Готовит планировщик к запуску.

Аргументы

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

4.11.4.2 void sched_schedule (void)

Переключает процессы в обработчике прерывания системного таймера.

KERNEL_PREEMPT

4.11.4.3 void sched_reschedule (void)

Переключает процессы в случае необходимости.

4.12 Файл bugurtos/include/sem.h

Заголовок счетных семафоров.

Структуры данных

- struct _sem_t
  Счетный семафор.

Определения типов

- typedef struct _sem_t sem_t

Функции

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

4.12.1 Подробное описание

4.12.2 Типы

4.12.2.1 typedef struct _sem_t sem_t

4.12.3 Функции

4.12.3.1 void sem_init_isr (sem_t * sem, count_t count)

Аргументы

sem Указатель на семафор.

count Начальное значение счетчика.

4.12.3.2 void sem_init (sem_t * sem, count_t count)

Аргументы

sem Указатель на семафор.

count Начальное значение счетчика.

4.12.3.3 bool_t sem_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

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

4.12.3.4 bool_t sem_try_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

1 если удалось захватить семафор, 0 если не удалось.

4.12.3.5 void sem_unlock (sem_t * sem)

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

Аргументы

sem Указатель на семафор.

4.12.3.6 void sem_unlock_isr (sem_t * sem)

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

Аргументы

sem Указатель на семафор.

4.12.3.7 bool_t _sem_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

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

KERNEL_PREEMPT

4.12.3.8 bool_t _sem_try_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

1 если удалось захватить семафор, 0 если не удалось.

4.13 Файл bugurtos/include/sig.h

Заголовок сигналов.

Определения типов

- typedef xlist_t sig_t

Функции

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

4.13.1 Подробное описание

4.13.2 Типы

4.13.2.1 typedef xlist_t sig_t

На 1 процессорной системе сигнал - просто список ожидания.

4.13.3 Функции

4.13.3.1 void sig_init_isr (sig_t * sig)

Аргументы

sig Указатель на сигнал.

4.13.3.2 void sig_init (sig_t * sig)

Аргументы

sig Указатель на сигнал.

4.13.3.3 void sig_wait (sig_t * sig)

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

Аргументы

sig Указатель на сигнал.

4.13.3.4 void _sig_wait_prologue (sig_t * sig)

Останавливает вызвавший процесс и ставит его в список ожидания. На многопроцессорной системе при этом происходит предварительная балансировка нагрузки.

Аргументы

sig Указатель на сигнал.

4.13.3.5 void _sig_wait_epilogue (void)

KERNEL_PREEMPT

4.13.3.6 void sig_signal (sig_t * sig)

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

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

Аргументы

sig Указатель на сигнал.

4.13.3.7 void sig_broadcast (sig_t * sig)

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

Аргументы

sig Указатель на сигнал.

4.13.3.8 void sig_signal_isr (sig_t * sig)

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

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

Аргументы

sig Указатель на сигнал.

4.13.3.9 void sig_broadcast_isr (sig_t * sig)

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

Аргументы

sig Указатель на сигнал.

4.14 Файл bugurtos/include/syscall.h

Заголовок системных вызовов.

Структуры данных

- struct proc_init_arg_t

  Параметр системного вызова SYSCALL_PROC_INIT.

- struct proc_runtime_arg_t

  Параметр системных вызовов SYSCALL_PROC_RUN, SYSCALL_PROC_RESTART, SYSCALL_PROC_STOP.

- struct sem_init_arg_t

  Параметр системного вызова SYSCALL_SEM_INIT.

- struct sem_lock_arg_t

Параметр системных вызовов SYSCALL_SEM_LOCK и SYSCALL_SEM_TRY_LOCK.

- struct mutex_init_arg_t

  Параметр системного вызова SYSCALL_MUTEX_INIT.

- struct mutex_lock_arg_t

  Параметр системных вызовов SYSCALL_MUTEX_LOCK и SYSCALL_MUTEX_TRY_LOCK.

- struct ipc_send_arg_t

  Параметр системного вызова SYSCALL_IPC_SEND.

- struct ipc_exchange_arg_t

  Параметр системного вызова SYSCALL_IPC_EXCHANGE.

Макросы

- #define SYSCALL_PROC_INIT ((syscall_t)(1))
- #define SYSCALL_PROC_RUN (SYSCALL_PROC_INIT + (syscall_t)(1))
- #define SYSCALL_PROC_RESTART (SYSCALL_PROC_RUN + (syscall_t)(1))
- #define SYSCALL_PROC_STOP (SYSCALL_PROC_RESTART + (syscall_t)(1))
- #define SYSCALL_PROC_SELF_STOP (SYSCALL_PROC_STOP + (syscall_t)(1))
- #define SYSCALL_PROC_YELD (SYSCALL_PROC_SELF_STOP + (syscall_t)(1))
- #define SYSCALL_PROC_TERMINATE (SYSCALL_PROC_YELD + (syscall_t)(1))
- #define SYSCALL_PROC_FLAG_STOP (SYSCALL_PROC_TERMINATE + (syscall_t)(1))
- #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FLAG_STOP + (syscall_t)(1))
- #define SYSCALL_SIG_INIT (SYSCALL_PROC_RESET_WATCHDOG + (syscall_t)(1))
- #define SYSCALL_SIG_WAIT (SYSCALL_SIG_INIT + (syscall_t)(1))
- #define SYSCALL_SIG_WAKEUP (SYSCALL_SIG_WAIT + (syscall_t)(1))
- #define SYSCALL_SIG_SIGNAL (SYSCALL_SIG_WAKEUP + (syscall_t)(1))
- #define SYSCALL_SIG_BROADCAST (SYSCALL_SIG_SIGNAL + (syscall_t)(1))
- #define SYSCALL_SEM_INIT (SYSCALL_SIG_BROADCAST + (syscall_t)(1))
- #define SYSCALL_SEM_LOCK (SYSCALL_SEM_INIT + (syscall_t)(1))
- #define SYSCALL_SEM_TRY_LOCK (SYSCALL_SEM_LOCK + (syscall_t)(1))
- #define SYSCALL_SEM_UNLOCK (SYSCALL_SEM_TRY_LOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_INIT (SYSCALL_SEM_UNLOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_LOCK (SYSCALL_MUTEX_INIT + (syscall_t)(1))
- #define SYSCALL_MUTEX_TRY_LOCK (SYSCALL_MUTEX_LOCK + (syscall_t)(1))
- #define SYSCALL_MUTEX_UNLOCK (SYSCALL_MUTEX_TRY_LOCK + (syscall_t)(1))
- #define SYSCALL_IPC_WAIT (SYSCALL_MUTEX_UNLOCK + (syscall_t)(1))
- #define SYSCALL_IPC_SEND (SYSCALL_IPC_WAIT + (syscall_t)(1))
- #define SYSCALL_IPC_EXCHANGE (SYSCALL_IPC_SEND + (syscall_t)(1))
- #define SYSCALL_USER (SYSCALL_IPC_EXCHANGE + (syscall_t)(1))

Функции

- void do_syscall (void)
- void scall_proc_init (void *arg)
  Обработчик вызова SYSCALL_PROC_INIT.
- 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_proc_yeld (void *arg)
  Обработчик вызова SYSCALL_PROC_YELD.
- void scall_proc_terminate (void *arg)
  Обработчик вызова SYSCALL_PROC_TERMINATE.
- void scall_proc_flag_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_FLAG_STOP.
- void scall_proc_reset_watchdog (void *arg)
  Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.
- void scall_sig_init (void *arg)
  Обработчик вызова SYSCALL_SIG_INIT.
- void scall_sig_wait (void *arg)
  Обработчик вызова SYSCALL_SIG_WAIT.
- void scall_sig_wakeup (void *arg)
- void scall_sig_signal (void *arg)
  Обработчик вызова SYSCALL_SIG_SIGNAL.
- void scall_sig_broadcast (void *arg)
  Обработчик вызова SYSCALL_SIG_BROADCAST.
- void scall_sem_init (void *arg)
  Обработчик вызова SYSCALL_SEM_INIT.
- void scall_sem_lock (void *arg)
  Обработчик вызова SYSCALL_SEM_LOCK.
- void scall_sem_try_lock (void *arg)

Обработчик вызова SYSCALL_SEM_TRY_LOCK.

- void scall_sem_unlock (void *arg)
  Обработчик вызова SYSCALL_SEM_UNLOCK.
- void scall_mutex_init (void *arg)
  Обработчик вызова SYSCALL_MUTEX_INIT.
- void scall_mutex_lock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_LOCK.
- void scall_mutex_try_lock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_TRY_LOCK.
- void scall_mutex_unlock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_UNLOCK.
- void scall_ipc_wait (void *arg)
  Обработчик вызова SYSCALL_IPC_WAIT.
- void scall_ipc_send (void *arg)
  Обработчик вызова SYSCALL_IPC_SEND.
- void scall_ipc_exchange (void *arg)
  Обработчик вызова SYSCALL_IPC_EXCHANGE.
- void scall_user (void *arg)
  Обработчик вызова SYSCALL_USER.

Переменные

- syscall_t syscall_num
  Обработка системного вызова.
- void * syscall_arg

4.14.1 Подробное описание

4.14.2 Макросы

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

Инициация процесса.

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

Запуск процесса.

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

Перезапуск процесса.

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

Останов процесса.

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

Самоостанов процесса.

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

Передача управления другому процессу.

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

Завершение работы процесса.

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

Останов процесса по флагу PROC_FLG_PRE_STOP.

4.14.2.9 #define SYSCALL_PROC_RESET_WATCHDOG (SYSCALL_PROC_FLAG_-STOP + (syscall_t)(1))

Сброс watchdog процесса реального времени.

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

Инициация сигнала.

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

Ожидание сигнала.

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

Обработка запуска по сигналу.

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

Подача сигнала одному процессу.

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

Подача сигнала всем ожидающим процессам.

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

Инициация семафора.

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

Захват семафора.

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

Попытка захвата семафора.

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

Освобождение семафора.

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

Инициация мьютекса.

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

Захват мьютекса.

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

Попытка захвата мьютекса.

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

Освобождение мьютекса.

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

Ожидание передачи данных.

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

Передача данных.

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

Обмен данными.

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

Пользовательский системный вызов.

4.14.3 Функции

4.14.3.1 void do_syscall (void)

4.14.3.2 void scall_proc_init (void * arg)

Инициализирует процесс, вызывая proc_init_isr.

Аргументы

arg указатель на структуру proc_init_arg_t.

4.14.3.3 void scall_proc_run (void * arg)

Пытается запустить процесс, вызывая proc_run_isr.

Аргументы

arg указатель на структуру proc_runtime_arg_t.

4.14.3.4 void scall_proc_restart (void * arg)

Пытается перезапустить процесс, вызывая proc_restart_isr.

Аргументы

arg указатель на структуру proc_runtime_arg_t.

4.14.3.5 void scall_proc_stop (void * arg)

Пытается остановить процесс, вызывая proc_stop_isr.

Аргументы

arg указатель на структуру proc_runtime_arg_t.

4.14.3.6 void scall_proc_self_stop (void * arg)

Останавливает вызывающий процесс.

Аргументы

arg не используется.

4.14.3.7 void scall_proc_yeld (void * arg)

Передает управление следующему процессу.

Аргументы

arg не используется.

4.14.3.8 void scall_proc_terminate (void * arg)

Завершает выполенение процесса после выхода из pmain. Вызывает _proc_terminate.

Аргументы

arg указатель на процесс.

4.14.3.9 void scall_proc_flag_stop (void * arg)

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

Аргументы

arg указатель на маску обнуления флагов процесса.

4.14.3.10 void scall_proc_reset_watchdog (void * arg)

Вызывает _proc_reset_watchdog.

Аргументы

arg не используется.

4.14.3.11 void scall_sig_init (void * arg)

Инициализирует сигнал, вызывает sig_init_isr.

Аргументы

arg указатель на сигнал.

4.14.3.12 void scall_sig_wait (void * arg)

Переводит вызваший процесс в состояние ожидания сигнала, вызывает _sig_wait_prologue.

Аргументы

arg указатель на сигнал.

4.14.3.13 void scall_sig_wakeup (void * arg)

4.14.3.14 void scall_sig_signal (void * arg)

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

Предупреждения

На многопроцессорной не действует принцип FIFO при "пробуждении" процессов, вставленных в списки ожидания для разных процессоров!

Аргументы

arg указатель на сигнал.

4.14.3.15 void scall_sig_broadcast (void * arg)

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

Аргументы

arg указатель на сигнал.

4.14.3.16 void scall_sem_init (void * arg)

Инициирует семафор, вызывает sem_init_isr.

Аргументы

arg указатель на аргумент типа sem_init_arg_t.

4.14.3.17 void scall_sem_lock (void * arg)

Вызывает _sem_lock.

Аргументы

arg указатель на аргумент типа sem_lock_arg_t.

4.14.3.18 void scall_sem_try_lock (void * arg)

Вызывает _sem_try_lock.

Аргументы

arg указатель на аргумент типа sem_lock_arg_t.

4.14.3.19 void scall_sem_unlock (void * arg)

Вызывает sem_unlock_isr.

Аргументы

arg указатель на семафор.

4.14.3.20 void scall_mutex_init (void * arg)

Инициирует мьютекс, вызывает mutex_init_isr.

Аргументы

arg указатель на аргумент типа mutex_init_arg_t.

4.14.3.21 void scall_mutex_lock (void * arg)

Вызывает _mutex_lock.

Аргументы

arg указатель на аргумент типа mutex_lock_arg_t.

4.14.3.22 void scall_mutex_try_lock (void * arg)

Вызывает _mutex_try_lock.

Аргументы

arg указатель на аргумент типа mutex_lock_arg_t.

4.14.3.23 void scall_mutex_unlock (void * arg)

Вызывает _mutex_unlock.

Аргументы

arg указатель на мьютекс.

4.14.3.24 void scall_ipc_wait (void * arg)

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

Аргументы

arg указатель на хранилище для передачи данных.

4.14.3.25 void scall_ipc_send (void * arg)

Вызывает ipc_send_isr.

Аргументы

arg указатель на аргумент типа ipc_send_arg_t.

4.14.3.26 void scall_ipc_exchange (void * arg)

Вызывает _ipc_exchange.

Аргументы

arg указатель на аргумент типа ipc_send_arg_t.

4.14.3.27 void scall_user (void * arg)

Вызывает пользовательскую функцию.

Аргументы

arg указатель на функцию пользователя.

Предупреждения

Осторожно! Параметр не проверяется!

4.14.4 Переменные

4.14.4.1 syscall_t syscall_num

Зпускает обработчик системного вызова и передает ему аргумент.

Номер системного вызова.

4.14.4.2 void* syscall_arg

Аргумент системного вызова.

4.15 Файл bugurtos/include/timer.h

Заголовок программных таймеров.

Макросы

- #define SPIN_LOCK_KERNEL_TIMER()

  Макрос-обертка.

- #define SPIN_UNLOCK_KERNEL_TIMER()
- #define CLEAR_TIMER(t) _clear_timer((timer_t *)&t)

  Сброс программного таймера.

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

  Получить значение программного таймера, для внутреннего использования.

Функции

- 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.

4.15.1 Подробное описание

Программные таймеры используются для синхронизации процессов по времени.

Предупреждения

Программные таймеры нельзя использовать для точного измерения интервалов времени!

4.15.2 Макросы

4.15.2.1 #define SPIN_LOCK_KERNEL_TIMER()

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

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

4.15.2.2 #define SPIN_UNLOCK_KERNEL_TIMER()

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

Аргументы

t Имя переменной таймера.

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

Аргументы

t Значение таймера.

4.15.3 Функции

4.15.3.1 void wait_time (timer_t time)

Просто ждет в цикле пока пройдет время time.

Аргументы

time Время ожидания.

4.15.3.2 void _clear_timer (timer_t * t)

Аргументы

t Указатель на таймер.

4.15.3.3 timer_t _timer (timer_t t)

Аргументы

t Значение таймера.

4.16 Файл bugurtos/include/xlist.h

Заголовок списков с приоритетами.

Структуры данных

- struct _xlist_t
  Список с приоритетами.

Определения типов

- typedef struct _xlist_t xlist_t

Функции

- 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.

4.16.1 Подробное описание

4.16.2 Типы

4.16.2.1 typedef struct _xlist_t xlist_t

4.16.3 Функции

4.16.3.1 void xlist_init (xlist_t * xlist)

Аргументы

xlist Указатель на список.

4.16.3.2 item_t* xlist_head (xlist_t * xlist)

Аргументы

xlist Указатель на список.

Возвращает

Указатель на голову - самый приоритетный элемент в массиве указателей.

4.16.3.3 void xlist_switch (xlist_t * xlist, prio_t prio)

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

Аргументы

xlist Указатель на список.

prio Приоритет переключаемой части списка.

4.17 Файл bugurtos/kernel/crit_sec.c

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

Функции

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

  Выход из критической секции.

4.17.1 Функции

4.17.1.1 void enter_crit_sec (void)

Вход в критическую секцию.

4.17.1.2 void exit_crit_sec (void)

Вход в критическую секцию.

4.18 Файл bugurtos/kernel/index.c

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

Функции

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

4.18.1 Функции

4.18.1.1 prio_t index_search (index_t index)

Аргументы

index Бинарный индекс.

Возвращает

Наивысший (с минимальным значением) приоритет в индексе.

An index search.

Аргументы

index An index.

Возвращает

Highest priority of an index (with minimal value).

4.19 Файл bugurtos/kernel/ipc.c

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

Функции

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

4.19.1 Функции

4.19.1.1 void _ipc_wait (void * ipc_pointer)

Предупреждения

Для внутреннего использования.

Аргументы

ipc_pointer Указатель на хранилище для передаваемых данных.

4.19.1.2 bool_t ipc_send_isr (proc_t * proc, ipc_data_t ipc_data)

Предупреждения

Для вызова из обработчиков прерываний!

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

Аргументы

proc Указатель на процесс-адресат.

data Данные для передачи.

Возвращает

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

Обработка флага останова целевого процесса

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

Предупреждения

Для внутреннего использования!

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

Аргументы

proc Указатель на процесс-адресат.

send Данные для передачи.

receive Указатель на хранилище данных для приема.

Возвращает

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

Обработка флага останова целевого процесса

4.20 Файл bugurtos/kernel/item.c

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

Функции

- 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.

4.20.1 Функции

4.20.1.1 void item_init (item_t * item)

Аргументы

item Указатель на объект item_t.

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

Аргументы

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

4.20.1.3 void item_cut (item_t * item)

Аргументы

item Указатель на объект типа item_t, который будем вырезать.

4.21 Файл bugurtos/kernel/kernel.c

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

Функции

- WEAK void idle_main (void *arg)

  Главная функция процесса холостого хода.

- void kernel_init (void)

  Инициализация Ядра.

Переменные

- kernel_t kernel

  Ядро BuguRTOS.

4.21.1 Функции

4.21.1.1 WEAK void idle_main (void * arg)

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

Предупреждения

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!

Аргументы

arg Указатель на аргумент.

4.21.1.2 void kernel_init (void)

Готовит ядро к запуску.

4.21.2 Переменные

4.21.2.1 kernel_t kernel

Оно одно на всю систему!

4.22 Файл bugurtos/kernel/mutex.c

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

Функции

- void mutex_init_isr (mutex_t *mutex, prio_t prio)
  Инициализация мьютекса из критической секции, или обработчика прерываний.
- bool_t _mutex_lock (mutex_t *mutex)
  Захват мьютекса, для внутреннего использования.
- bool_t _mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса, для внутреннего использования.
- void _mutex_unlock (mutex_t *mutex)
  Освобождение мьютекса, для внутреннего использования.

4.22.1 Функции

4.22.1.1 void mutex_init_isr (mutex_t * mutex, prio_t prio)

Да, инициировать из обработчика прерывания можно!

Аргументы

mutex Указатель на мьютекс.

prio В случае использования CONFIG_USE_HIGHEST_LOCKER, - приоритет мьютекса.

4.22.1.2 bool_t _mutex_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.22.1.3 bool_t _mutex_try_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.22.1.4 void _mutex_unlock (mutex_t * mutex)

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.

Аргументы

mutex Указатель на мьютекс.

KERNEL_PREEMPT

4.23 Файл bugurtos/kernel/pcounter.c

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

Функции

- 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.

4.23.1 Функции

4.23.1.1 void pcounter_init (pcounter_t * pcounter)

Аргументы

pcounter Указатель на счетчик.

4.23.1.2 void pcounter_inc (pcounter_t * pcounter, prio_t prio)

Аргументы

pcounter Указатель на счетчик.
prio Приоритет.

4.23.1.3 index_t pcounter_dec (pcounter_t * pcounter, prio_t prio)

Аргументы

pcounter Указатель на счетчик.

prio Приоритет.

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

Аргументы

pcounter Указатель на счетчик.

prio Приоритет.

count Количество единиц.

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

Аргументы

pcounter Указатель на счетчик.

prio Приоритет.

count Количество единиц.

Возвращает

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

4.24 Файл bugurtos/kernel/pitem.c

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

Функции

- 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.

4.24.1 Функции

4.24.1.1 void pitem_init (pitem_t * pitem, prio_t prio)

Аргументы

pitem Указатель на объект pitem_t.

prio Приоритет элемента.

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

Аргументы

pitem Указатель на объект pitem_t.

xlist Указатель на список.

4.24.1.3 void pitem_fast_cut (pitem_t * pitem)

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

Аргументы

pitem Указатель на объект pitem_t.

4.24.1.4 void pitem_cut (pitem_t * pitem)

Вызывает pitem_fast_cut и обнуляет указатель pitem->list.

Аргументы

pitem Указатель на объект pitem_t.

4.24.1.5 pitem_t* pitem_xlist_chain (xlist_t * src)

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

Аргументы

src Указатель на объект xlist_t.

Возвращает

Указаьель на голову 2-связного списка.

4.25 Файл bugurtos/kernel/proc.c

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

Макросы

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

Функции

- 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_run (proc_t *proc)

  "Низкоуровневый" запуск процесса, для внутреннего использования.

- bool_t proc_run_isr (proc_t *proc)

  Запуск процесса из критической секции, либо обработчика прерывания.

- bool_t proc_restart_isr (proc_t *proc)

  Перезапуск процесса из критической секции или обработчика прерывания.

- void _proc_stop (proc_t *proc)

  "Низкоуровневый" останов процесса, для внутреннего использования.

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

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

- bool_t proc_stop_isr (proc_t *proc)

  Останов процесса из критической секции или обработчика прерывания.

- void _proc_flag_stop (flag_t mask)

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

- void _proc_self_stop (void)

  Самоостанов процесса (для внутреннего использования).

- index_t _proc_yeld (void)

  Передача управления следующему процессу (для внутреннего использования).

- void _proc_terminate (void)

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

- void _proc_reset_watchdog (void)

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

- void _proc_lres_inc (proc_t *proc, prio_t prio)

  Инкремент счетчика захваченных ресурсов, для внутреннего использования.

- void _proc_lres_dec (proc_t *proc, prio_t prio)

  Декремент счетчика захваченных ресурсов, для внутреннего использования.

- void _proc_prio_control_stoped (proc_t *proc)

  Управление приоритетом процесса, для внутреннего использования.

4.25.1 Макросы

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

4.25.2 Функции

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

Аргументы

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.

4.25.2.2 void _proc_run (proc_t * proc)

4.25.2.3 bool_t proc_run_isr (proc_t * proc)

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

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.25.2.4 bool_t proc_restart_isr (proc_t * proc)

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.

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.25.2.5 void _proc_stop (proc_t * proc)

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

4.25.2.7 void _proc_stop_flags_set (proc_t * proc, flag_t mask)

4.25.2.8 bool_t proc_stop_isr (proc_t * proc)

Вырезает процесс из списка готовых к выполнению и производит перепланировку.

Аргументы

proc - Указатель на останавливаемый процесс.

Возвращает

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

4.25.2.9 void _proc_flag_stop (flag_t mask)

4.25.2.10 void _proc_self_stop (void)

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

4.25.2.11 index_t _proc_yeld (void)

Передает управление следующему процессу, если такой процесс есть.

Возвращает

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

KERNEL_PREEMPT

KERNEL_PREEMPT

4.25.2.12 void _proc_terminate (void)

4.25.2.13 void _proc_reset_watchdog (void)

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.

4.25.2.14 void _proc_lres_inc (proc_t * proc, prio_t prio)

Аргументы

proc Указатель на процесс, захвативший ресурс.

prio Приоритет захваченного ресурса, используется совместно с опцией CONFIG_USE_HIGHEST_LOCKER.

4.25.2.15 void _proc_lres_dec (proc_t * proc, prio_t prio)

Аргументы

proc Указатель на процесс, захвативший ресурс.

prio Приоритет захваченного ресурса, используется совместно с опцией CONFIG_USE_HIGHEST_LOCKER.

4.25.2.16 void _proc_prio_control_stoped (proc_t * proc)

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

Аргументы

proc - Указатель на процесс.

4.26 Файл bugurtos/kernel/sched.c

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

Функции

- void sched_init (sched_t *sched, proc_t *idle)
  Инициализация планировщика.
- static void _sched_switch_current (sched_t *sched, proc_t *current_proc)
- void sched_schedule (void)
  Функция планирования.
- void sched_reschedule (void)
  Функция перепланирования.

4.26.1 Функции

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

Готовит планировщик к запуску.

Аргументы

sched - Указатель на планировщик.

idle - Указатель на процесс холостого хода.

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

4.26.1.3 void sched_schedule (void)

Переключает процессы в обработчике прерывания системного таймера.

KERNEL_PREEMPT

4.26.1.4 void sched_reschedule (void)

Переключает процессы в случае необходимости.

4.27 Файл bugurtos/kernel/sem.c

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

Функции

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

4.27.1 Функции

4.27.1.1 void sem_init_isr (sem_t * sem, count_t count)

Аргументы

sem Указатель на семафор.

count Начальное значение счетчика.

4.27.1.2 bool_t _sem_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

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

KERNEL_PREEMPT

4.27.1.3 bool_t _sem_try_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

1 если удалось захватить семафор, 0 если не удалось.

4.27.1.4 void sem_unlock_isr (sem_t * sem)

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

Аргументы

sem Указатель на семафор.

4.28 Файл bugurtos/kernel/sig.c

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

Функции

- void sig_init_isr (sig_t *sig)

  Инициализация сигнала из обработчика прерывания или критической секции.

- void _sig_wait_prologue (sig_t *sig)

  Встать в список ожидания сигнала, для внутреннего использования.

- static void _sig_set_wakeup_flags (proc_t *proc)
- static void _sig_wakeup_list_proc (proc_t *proc)
- void _sig_wait_epilogue (void)
- void sig_signal_isr (sig_t *sig)

  Возобновить работу 1 процесса ожидающего сигнал из обработчика прерывания.

- void sig_broadcast_isr (sig_t *sig)

  Возобновить работу всех ожидающих процессов из обработчика прерывания.

4.28.1 Функции

4.28.1.1 void sig_init_isr (sig_t * sig)

Аргументы

sig Указатель на сигнал.

4.28.1.2 void _sig_wait_prologue (sig_t * sig)

Останавливает вызвавший процесс и ставит его в список ожидания. На многопроцессорной системе при этом происходит предварительная балансировка нагрузки.

Аргументы

sig Указатель на сигнал.

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

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

4.28.1.5 void _sig_wait_epilogue (void)

KERNEL_PREEMPT

4.28.1.6 void sig_signal_isr (sig_t * sig)

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

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

Аргументы

sig Указатель на сигнал.

4.28.1.7 void sig_broadcast_isr (sig_t * sig)

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

Аргументы

sig Указатель на сигнал.

4.29 Файл bugurtos/kernel/syscall.c

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

Функции

- SYSCALL_TABLE (syscall_routine[])
- void do_syscall (void)
- void scall_proc_init (void *arg)
  SYSCALL_PROC_INIT.
- void proc_init (proc_t *proc, code_t pmain, code_t sv_hook, code_t rs_hook, void *arg, stack_t *sstart, prio_t prio, timer_t time_quant, bool_t is_rt)
  Инициализация процесса.
- void scall_proc_run (void *arg)
  Обработчик вызова SYSCALL_PROC_RUN.
- bool_t proc_run (proc_t *proc)
  Запуск процесса.
- void scall_proc_restart (void *arg)
  Обработчик вызова SYSCALL_PROC_RESTART.
- bool_t proc_restart (proc_t *proc)
  Перезапуск процесса.
- void scall_proc_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_STOP.
- bool_t proc_stop (proc_t *proc)
  Останов процесса.
- void scall_proc_self_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_SELF_STOP.
- void proc_self_stop (void)
  Самоостанов процесса.
- void scall_proc_yeld (void *arg)
  Обработчик вызова SYSCALL_PROC_YELD.
- index_t proc_yeld (void)
  Передача управления следующему процессу.
- void scall_proc_terminate (void *arg)
  Обработчик вызова SYSCALL_PROC_TERMINATE.
- void proc_terminate (void)

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

- void scall_proc_flag_stop (void *arg)
  Обработчик вызова SYSCALL_PROC_FLAG_STOP.
- void proc_flag_stop (flag_t mask)
  Останов процесса по флагу PROC_FLG_PRE_STOP.
- void scall_proc_reset_watchdog (void *arg)
  Обработчик вызова SYSCALL_PROC_RESET_WATCHDOG.
- void proc_reset_watchdog (void)
  Сброс watchdog для процесса реального времени.
- void scall_sig_init (void *arg)
  Обработчик вызова SYSCALL_SIG_INIT.
- void sig_init (sig_t *sig)
  Инициализация сигнала.
- void scall_sig_wait (void *arg)
  Обработчик вызова SYSCALL_SIG_WAIT.
- void scall_sig_wakeup (void *arg)
- void sig_wait (sig_t *sig)
  Встать в список ожидания сигнала.
- void scall_sig_signal (void *arg)
  Обработчик вызова SYSCALL_SIG_SIGNAL.
- void sig_signal (sig_t *sig)
  Возобновить работу 1 процесса ожидающего сигнал.
- void scall_sig_broadcast (void *arg)
  Обработчик вызова SYSCALL_SIG_BROADCAST.
- void sig_broadcast (sig_t *sig)
  Возобновить работу всех ожидающих процессов.
- void scall_sem_init (void *arg)
  Обработчик вызова SYSCALL_SEM_INIT.
- void sem_init (sem_t *sem, count_t count)
  Инициализация семафора.
- void scall_sem_lock (void *arg)
  Обработчик вызова SYSCALL_SEM_LOCK.
- bool_t sem_lock (sem_t *sem)

Захват семафора.

- void scall_sem_try_lock (void *arg)
  Обработчик вызова SYSCALL_SEM_TRY_LOCK.
- bool_t sem_try_lock (sem_t *sem)
  Попытка захвата семафора.
- void scall_sem_unlock (void *arg)
  Обработчик вызова SYSCALL_SEM_UNLOCK.
- void sem_unlock (sem_t *sem)
  Освобождение семафора.
- void scall_mutex_init (void *arg)
  Обработчик вызова SYSCALL_MUTEX_INIT.
- void mutex_init (mutex_t *mutex, prio_t prio)
  Инициализация мьютекса.
- void scall_mutex_lock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_LOCK.
- bool_t mutex_lock (mutex_t *mutex)
  Захват мьютекса.
- void scall_mutex_try_lock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_TRY_LOCK.
- bool_t mutex_try_lock (mutex_t *mutex)
  Попытка захвата мьютекса.
- void scall_mutex_unlock (void *arg)
  Обработчик вызова SYSCALL_MUTEX_UNLOCK.
- void mutex_unlock (mutex_t *mutex)
  Освобождение мьютекса.
- void scall_ipc_wait (void *arg)
  Обработчик вызова SYSCALL_IPC_WAIT.
- ipc_data_t ipc_wait (void)
  Переход процесса к ожиданию получения данных через IPC.
- void scall_ipc_send (void *arg)
  Обработчик вызова SYSCALL_IPC_SEND.
- bool_t ipc_send (proc_t *proc, ipc_data_t ipc_data)
  Посылка данных процессу через IPC.

- void scall_ipc_exchange (void *arg)
  Обработчик вызова SYSCALL_IPC_EXCHANGE.
- bool_t ipc_exchange (proc_t *proc, ipc_data_t send, ipc_data_t *receive)
  Посылка данных процессу через IPC, прием ответа через IPC.
- void scall_user (void *arg)
  Обработчик вызова SYSCALL_USER.

Переменные

- syscall_t syscall_num = (syscall_t)0
  Обработка системного вызова.
- void * syscall_arg = (void *)0

4.29.1 Функции

4.29.1.1 SYSCALL_TABLE (syscall_routine[])

4.29.1.2 void do_syscall (void)

4.29.1.3 void scall_proc_init (void * arg)

Обработчик вызова SYSCALL_PROC_INIT.

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

Аргументы

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.

4.29.1.5 void scall_proc_run (void * arg)

Пытается запустить процесс, вызывая proc_run_isr.

Аргументы

arg указатель на структуру proc_runtime_arg_t.

4.29.1.6 bool_t proc_run (proc_t * proc)

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

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.29.1.7 void scall_proc_restart (void * arg)

Пытается перезапустить процесс, вызывая proc_restart_isr.

Аргументы

arg указатель на структуру proc_runtime_arg_t.

4.29.1.8 bool_t proc_restart (proc_t * proc)

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.

Аргументы

proc - Указатель на запускаемый процесс.

Возвращает

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

4.29.1.9 void scall_proc_stop (void * arg)

Пытается остановить процесс, вызывая proc_stop_isr.

Аргументы

arg указатель на структуру proc_runtime_arg_t.

4.29.1.10 bool_t proc_stop (proc_t * proc)

Вырезает процесс из списка готовых к выполнению и производит перепланировку.

Аргументы

proc - Указатель на останавливаемый процесс.

Возвращает

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

4.29.1.11 void scall_proc_self_stop (void * arg)

Останавливает вызывающий процесс.

Аргументы

arg не используется.

4.29.1.12 void proc_self_stop (void)

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

4.29.1.13 void scall_proc_yeld (void * arg)

Передает управление следующему процессу.

Аргументы

arg не используется.

4.29.1.14 index_t proc_yeld (void)

Передает управление следующему процессу, если такой процесс есть.

Возвращает

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

4.29.1.15 void scall_proc_terminate (void * arg)

Завершает выполенение процесса после выхода из pmain. Вызывает _proc_terminate.

Аргументы

arg указатель на процесс.

4.29.1.16 void proc_terminate (void)

4.29.1.17 void scall_proc_flag_stop (void * arg)

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

Аргументы

arg указатель на маску обнуления флагов процесса.

4.29.1.18 void proc_flag_stop (flag_t mask)

4.29.1.19 void scall_proc_reset_watchdog (void * arg)

Вызывает _proc_reset_watchdog.

Аргументы

arg не используется.

4.29.1.20 void proc_reset_watchdog (void)

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.

4.29.1.21 void scall_sig_init (void * arg)

Инициализирует сигнал, вызывает sig_init_isr.

Аргументы

arg указатель на сигнал.

4.29.1.22 void sig_init (sig_t * sig)

Аргументы

sig Указатель на сигнал.

4.29.1.23 void scall_sig_wait (void * arg)

Переводит вызваший процесс в состояние ожидания сигнала, вызывает _sig_wait_prologue.

Аргументы

arg указатель на сигнал.

4.29.1.24 void scall_sig_wakeup (void * arg)

4.29.1.25 void sig_wait (sig_t * sig)

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

Аргументы

sig Указатель на сигнал.

4.29.1.26 void scall_sig_signal (void * arg)

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

Предупреждения

На многопроцессорной не действует принцип FIFO при "пробуждении" процессов, вставленных в списки ожидания для разных процессоров!

Аргументы

arg указатель на сигнал.

4.29.1.27 void sig_signal (sig_t * sig)

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

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

Аргументы

sig Указатель на сигнал.

4.29.1.28 void scall_sig_broadcast (void * arg)

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

Аргументы

arg указатель на сигнал.

4.29.1.29 void sig_broadcast (sig_t * sig)

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

Аргументы

sig Указатель на сигнал.

4.29.1.30 void scall_sem_init (void * arg)

Инициирует семафор, вызывает sem_init_isr.

Аргументы

arg указатель на аргумент типа sem_init_arg_t.

4.29.1.31 void sem_init (sem_t * sem, count_t count)

Аргументы

sem Указатель на семафор.

count Начальное значение счетчика.

4.29.1.32 void scall_sem_lock (void * arg)

Вызывает _sem_lock.

Аргументы

arg указатель на аргумент типа sem_lock_arg_t.

4.29.1.33 bool_t sem_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

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

4.29.1.34 void scall_sem_try_lock (void * arg)

Вызывает _sem_try_lock.

Аргументы

arg указатель на аргумент типа sem_lock_arg_t.

4.29.1.35 bool_t sem_try_lock (sem_t * sem)

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.

Аргументы

sem Указатель на семафор.

Возвращает

1 если удалось захватить семафор, 0 если не удалось.

4.29.1.36 void scall_sem_unlock (void * arg)

Вызывает sem_unlock_isr.

Аргументы

arg указатель на семафор.

4.29.1.37 void sem_unlock (sem_t * sem)

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

Аргументы

sem Указатель на семафор.

4.29.1.38 void scall_mutex_init (void * arg)

Инициирует мьютекс, вызывает mutex_init_isr.

Аргументы

arg указатель на аргумент типа mutex_init_arg_t.

4.29.1.39 void mutex_init (mutex_t * mutex, prio_t prio)

Аргументы

mutex Указатель на мьютекс.

prio В случае использования CONFIG_USE_HIGHEST_LOCKER, - приоритет мьютекса.

4.29.1.40 void scall_mutex_lock (void * arg)

Вызывает _mutex_lock.

Аргументы

arg указатель на аргумент типа mutex_lock_arg_t.

4.29.1.41 bool_t mutex_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.29.1.42 void scall_mutex_try_lock (void * arg)

Вызывает _mutex_try_lock.

Аргументы

arg указатель на аргумент типа mutex_lock_arg_t.

4.29.1.43 bool_t mutex_try_lock (mutex_t * mutex)

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

Аргументы

mutex Указатель на мьютекс.

Возвращает

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

4.29.1.44 void scall_mutex_unlock (void * arg)

Вызывает _mutex_unlock.

Аргументы

arg указатель на мьютекс.

4.29.1.45 void mutex_unlock (mutex_t * mutex)

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.

Аргументы

mutex Указатель на мьютекс.

4.29.1.46 void scall_ipc_wait (void * arg)

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

Аргументы

arg указатель на хранилище для передачи данных.

4.29.1.47 ipc_data_t ipc_wait (void)

Возвращает

Данные.

4.29.1.48 void scall_ipc_send (void * arg)

Вызывает ipc_send_isr.

Аргументы

arg указатель на аргумент типа ipc_send_arg_t.

4.29.1.49 bool_t ipc_send (proc_t * proc, ipc_data_t ipc_data)

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

Аргументы

proc Указатель на процесс-адресат.

data Данные для передачи.

Возвращает

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

4.29.1.50 void scall_ipc_exchange (void * arg)

Вызывает _ipc_exchange.

Аргументы

arg указатель на аргумент типа ipc_send_arg_t.

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

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

Аргументы

proc Указатель на процесс-адресат.

send Данные для передачи.

receive указатель на хранилище данных для приема.

Возвращает

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

4.29.1.52 void scall_user (void * arg)

Вызывает пользовательскую функцию.

Аргументы

arg указатель на функцию пользователя.

Предупреждения

Осторожно! Параметр не проверяется!

4.29.2 Переменные

4.29.2.1 syscall_t syscall_num = (syscall_t)0

Зпускает обработчик системного вызова и передает ему аргумент.

Номер системного вызова.

4.29.2.2 void* syscall_arg = (void *)0

Аргумент системного вызова.

4.30 Файл bugurtos/kernel/timer.c

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

Функции

- 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.

4.30.1 Функции

4.30.1.1 void _clear_timer (timer_t * t)

Аргументы

t Указатель на таймер.

4.30.1.2 timer_t _timer (timer_t t)

Аргументы

t Значение таймера.

4.30.1.3 void wait_time (timer_t time)

Просто ждет в цикле пока пройдет время time.

Аргументы

time Время ожидания.

4.31 Файл bugurtos/kernel/xlist.c

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

Функции

- 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.

4.31.1 Функции

4.31.1.1 void xlist_init (xlist_t * xlist)

Аргументы

xlist Указатель на список.

4.31.1.2 item_t* xlist_head (xlist_t * xlist)

Аргументы

xlist Указатель на список.

Возвращает

Указатель на голову - самый приоритетный элемент в массиве указателей.

4.31.1.3 void xlist_switch (xlist_t * xlist, prio_t prio)

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

Аргументы

xlist Указатель на список.

prio Приоритет переключаемой части списка.

Предметный указатель