Robot OS's arbitrator Design Reference Manual

v0.0 "Exo"

Generated by Doxygen 1.5.3

Tue Dec 2 16:13:42 2008

Contents

1 Robot OS's arbitrator Design Data Structure Index **1**

1.1 Robot OS's arbitrator Design Data Structures 1

2 Robot OS's arbitrator Design File Index **3**

2.1 Robot OS's arbitrator Design File List 3

3 Robot OS's arbitrator Design Data Structure Documentation **5**

3.1 arbitrator_if Struct Reference 5

3.2 mrt_t Struct Reference 6

3.3 object_t Struct Reference 8

3.4 rd_t Struct Reference 10

3.5 ssd_t Struct Reference 12

4 Robot OS's arbitrator Design File Documentation **13**

4.1 include/sys/brain/arbitrator.h File Reference 13

4.2 sys/brain/arbitrator.c File Reference 14

Chapter 1

Robot OS's arbitrator Design Data Structure Index

1.1 Robot OS's arbitrator Design Data Structures

Here are the data structures with brief descriptions:

arbitrator_if 5
mrt_t 6
object_t 8
rd_t 10
ssd_t 12

Chapter 2

Robot OS's arbitrator Design File Index

2.1 Robot OS's arbitrator Design File List

Here is a list of all files with brief descriptions:

include/sys/brain/arbitrator.h 13
sys/brain/arbitrator.c 14

Chapter 3

Robot OS's arbitrator Design Data Structure Documentation

3.1 arbitrator_if Struct Reference

Data Fields

- struct object ∗(∗ create_task)(void(∗task_entry)(void), void ∗task_params)
- void(∗ sched_start)(void)
- void(∗ sched_stop)(void)
- void(∗ schedule)(void)

3.1.1 Detailed Description

Interface of the arbitrator (scheduler)

Definition at line 154 of file arbitrator.h.

3.1.2 Field Documentation

3.1.2.1 struct object∗(∗ arbitrator_if::create_task)(void(∗task_entry)(void), void ∗task_params) [read]

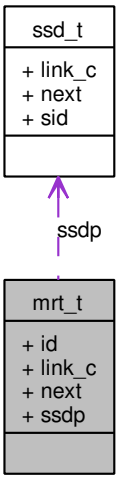
3.1.2.2 void(∗ arbitrator_if::sched_start)(void)

3.1.2.3 void(∗ arbitrator_if::sched_stop)(void)

3.1.2.4 void(∗ arbitrator_if::schedule)(void)

3.2 mrt_t Struct Reference

Collaboration diagram for mrt_t:



Data Fields

- char id [MAX_STRSIZE]
- uint32_t link_c
- struct mrt * next
- ssd_t * ssdp

3.2.1 Detailed Description

Message routing database

As we see in the described algorithm, eventually all processes will be destroyed because of their IO-counters will be decremented to zero. To prevent the process destruction, they need to increment their counters. To provide cooperative work the method of incrementing I/O counters is advised. The method is based on idea of indirect conformity between cooperative work and acts of IO.

Messaging subsystem is formed as a separate subsystem and must realize functions described below. The heart of messaging subsystem is a Message Routing Database (MRD) which is used by a subsystem to manipulate with messages. MRD is organized as an objective relational database and contains at least two types of tables: the source table and the subscribers table. It has system message queue where all messages are scheduled to be sent by messaging subsystem.

When all processes have finished their execution loop again the messaging subsystem takes messages from queue and processes them all before the next execution loop will start. The theoretical background of the technology was partially borrowed from "The Theory of Dissipative Systems", "The Theory of Open Systems", "Genetic Programming" and "Evolutionary algorythms".

Definition at line 112 of file arbitrator.h.

3.2.2 Field Documentation

3.2.2.1 char mrt_t::id[MAX_STRSIZE]

Definition at line 113 of file arbitrator.h.

3.2.2.2 uint32_t mrt_t::link_c

Definition at line 114 of file arbitrator.h.

3.2.2.3 struct mrt* mrt_t::next [read]

Definition at line 116 of file arbitrator.h.

3.2.2.4 ssd_t* mrt_t::ssdp

Definition at line 115 of file arbitrator.h.

3.3 object_t Struct Reference

Data Fields

- struct context ctx
- char id [MAX_STRSIZE]
- struct object * next
- char rid [MAX_STRSIZE]

3.3.1 Detailed Description

Basic process/thread structure

Each process in the system must be fine grained to give the following features: flexibility, adaptation, fault tolerance and security, and thus they must be soft linked with each other. It means that we do not permit direct interprocess communication (IPC). After execution, each process takes unique identifier, and registers its description in resource description database.

Each process has loop method which is the logical block of the process. The scheduler governs each running process and ensures one loop of each process per one cycle to provide dissipation. So the characteristic period (this is the time of dissipation) of time in such system will be described as follows

$$\tau = NXn = 0tn(2.1)$$

where N is the number of running processes.

Each process in the system has in common association with ordinary process descrip- tion fields also the information of its behavior and cooperative work. In this case the scheduler can make the decision about destroying process if the process behavior becomes asocial to the system. The decision is based on formal parameters that mirror the social links of the process. Eventually links are to be erased, so the process must take them in actual state to make this operation useful to other processes in any kind. Otherwise such process will lose 'social' links and will be destroyed by scheduler.

As said before each process must be fine grained. Ideally it must know nothing about other running processes in the system at process's initial state (when it loaded for the first time). So the main idea is in to make it an autonomous process. Each process has input and output information, on initial state the process tries to lookup needed resources to work with sending export and import requests. It should know only the description of needed resource to be independent of systems architecture (ex. filesystem type or filesystem structure). To make this possible in this invention the method for resource lookup is advised.

Definition at line 74 of file arbitrator.h.

3.3.2 Field Documentation

3.3.2.1 struct context object_t::ctx [read]

Definition at line 77 of file arbitrator.h.

Referenced by arbitrator_create_task().

3.3.2.2 char object_t::id[MAX_STRSIZE]

Definition at line 75 of file arbitrator.h.

3.3.2.3 struct object* object_t::next [read]

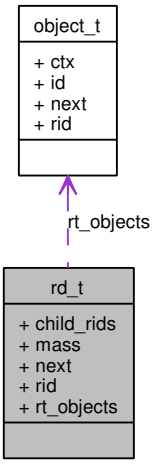
Definition at line 78 of file arbitrator.h.

3.3.2.4 char object_t::rid[MAX_STRSIZE]

Definition at line 76 of file arbitrator.h.

3.4 rd_t Struct Reference

Collaboration diagram for rd_t:



Data Fields

- char * child_rids
- uint32_t mass
- struct rd * next
- char rid [MAX_STRSIZE]
- object_t * rt_objects

3.4.1 Detailed Description

Resource description database

The operating system searches in the resource description database for requested processes and returns their ID's to the requestor. If no process found, the operating system starts search in process's repository and if finds similar description, loads and run found process. To make this possible each process must be registered in resource description database with unique ID when loading and must have description.

The request is very simple and contains data description keywords and if we translate it to human native language we can describe it as following:

Process says to OS: "I want data that is like IP packets, networking" and then OS searches for processes which are produce such a data, ex. "Network driver processes".

Non learned OS has only hardware drivers (low level alphabet).This drivers cannot be dropped or destroyed, and they produce initial location requests. By processing requests from drivers OS starts searching and loading other level processes.

Next is the resource descriptor structure

Definition at line 141 of file arbitrator.h.

3.4.2 Field Documentation

3.4.2.1 char* rd_t::child_rids

Definition at line 146 of file arbitrator.h.

3.4.2.2 uint32_t rd_t::mass

Definition at line 143 of file arbitrator.h.

3.4.2.3 struct rd* rd_t::next [read]

Definition at line 145 of file arbitrator.h.

3.4.2.4 char rd_t::rid[MAX_STRSIZE]

Definition at line 142 of file arbitrator.h.

3.4.2.5 object_t* rd_t::rt_objects

Definition at line 144 of file arbitrator.h.

3.5 ssd_t Struct Reference

Data Fields

- uint32_t link_c
- struct ssd * next
- char sid [MAX_STRSIZE]

3.5.1 Detailed Description

Subscriber's list definition structure

Definition at line 85 of file arbitrator.h.

3.5.2 Field Documentation

3.5.2.1 uint32_t ssd_t::link_c

Definition at line 87 of file arbitrator.h.

3.5.2.2 struct ssd* ssd_t::next [read]

Definition at line 88 of file arbitrator.h.

3.5.2.3 char ssd_t::sid[MAX_STRSIZE]

Definition at line 86 of file arbitrator.h.

Chapter 4

Robot OS's arbitrator Design File Documentation

4.1 include/sys/brain/arbitrator.h File Reference

Data Structures

- struct arbitrator_if
- struct mrt_t
- struct object_t
- struct rd_t
- struct ssd_t

Typedefs

- typedef struct arbitrator_if arbitrator_if_t

4.1.1 Typedef Documentation

4.1.1.1 typedef struct arbitrator_if arbitrator_if_t

Definition at line 161 of file arbitrator.h.

4.2 sys/brain/arbitrator.c File Reference

Defines

- #define DPRINTK(level,...) do { } while (0)
- #define MAX_TASKS 16

Functions

- struct object * arbitrator_create_task (void(*task_entry)(void), void *task_params)
- struct arbitrator_if * arbitrator_init (void)
- void arbitrator_schedule (void)
- void arbitrator_start (void)
- void arbitrator_stop (void)

Variables

- static struct arbitrator_if core_arbitrator
- int curid = 0
- static mrt_t * mrd = NULL
- static int sched_run = 0
- object_t * t_current = 0
- static unsigned char t_stack_base [MAX_TASKS][TASK_STACK_SIZE]
- static object_t taskarray [MAX_TASKS]
- static int taskcount = 1

4.2.1 Define Documentation

4.2.1.1 #define DPRINTK(level, ...) do {} while (0)

Definition at line 47 of file arbitrator.c.

Referenced by arbitrator_schedule().

4.2.1.2 #define MAX_TASKS 16

Definition at line 56 of file arbitrator.c.

Referenced by arbitrator_create_task().

4.2.2 Function Documentation

4.2.2.1 struct object * arbitrator_create_task (void(*)(void) *task_entry*, void * *task_params*) [read]

Creates and initializes new kernel task and places it to run queue.

Returns:

- new object, otherwise NULL.

Definition at line 139 of file arbitrator.c.

References object_t::ctx, MAX_TASKS, and taskcount.

4.2.2.2 struct arbitrator_if* arbitrator_init (void) [read]

Definition at line 187 of file arbitrator.c.

4.2.2.3 void arbitrator_schedule (void)

Decision making routine of arbitrator. Finds the currently running task and next task to run and calls to cpu context switching routine (see 'cpu_switch()' in atoms.S)

Returns:

- nothing (void function)

Definition at line 99 of file arbitrator.c.

References curid, DPRINTK, sched_run, and taskcount.

4.2.2.4 void arbitrator_start (void)

Definition at line 79 of file arbitrator.c.

References sched_run.

4.2.2.5 void arbitrator_stop (void)

Definition at line 85 of file arbitrator.c.

References sched_run.

4.2.3 Variable Documentation

4.2.3.1 struct arbitrator_if core_arbitrator [static]

Initial value:

```
 {
                .sched_start = arbitrator_start,
                .schedule = arbitrator_schedule,
                .create_task = arbitrator_create_task,
}
```

Definition at line 181 of file arbitrator.c.

4.2.3.2 int curid = 0

Definition at line 68 of file arbitrator.c.

Referenced by arbitrator_schedule().

4.2.3.3 mrt_t* mrd = NULL [static]

Message routing database core

Definition at line 54 of file arbitrator.c.

4.2.3.4 int sched_run = 0 [static]

Definition at line 58 of file arbitrator.c.

Referenced by arbitrator_schedule(), arbitrator_start(), and arbitrator_stop().

4.2.3.5 object_t* t_current = 0

The pointer to current running task

Definition at line 67 of file arbitrator.c.

4.2.3.6 unsigned char t_stack_base[MAX_TASKS][TASK_STACK_SIZE] [static]

kernel space thread stacks

Definition at line 76 of file arbitrator.c.

4.2.3.7 object_t taskarray[MAX_TASKS] [static]

Definition at line 62 of file arbitrator.c.

4.2.3.8 int taskcount = 1 [static]

Definition at line 61 of file arbitrator.c.

Referenced by arbitrator_create_task(), and arbitrator_schedule().

Index

arbitrator.c
    arbitrator_create_task, 14
    arbitrator_init, 15
    arbitrator_schedule, 15
    arbitrator_start, 15
    arbitrator_stop, 15
    core_arbitrator, 15
    curid, 15
    DPRINTK, 14
    MAX_TASKS, 14
    mrd, 15
    sched_run, 16
    t_current, 16
    t_stack_base, 16
    taskarray, 16
    taskcount, 16
arbitrator.h
    arbitrator_if_t, 13
arbitrator_create_task
    arbitrator.c, 14
arbitrator_if, 5
    create_task, 5
    sched_start, 5
    sched_stop, 5
    schedule, 5
arbitrator_if_t
    arbitrator.h, 13
arbitrator_init
    arbitrator.c, 15
arbitrator_schedule
    arbitrator.c, 15
arbitrator_start
    arbitrator.c, 15
arbitrator_stop
    arbitrator.c, 15

child_rids
    rd_t, 11
core_arbitrator
    arbitrator.c, 15
create_task
    arbitrator_if, 5
ctx
    object_t, 8
curid
    arbitrator.c, 15

DPRINTK
    arbitrator.c, 14

id
    mrt_t, 7
    object_t, 8
include/sys/brain/arbitrator.h, 13

link_c
    mrt_t, 7
    ssd_t, 12

mass
    rd_t, 11
MAX_TASKS
    arbitrator.c, 14
mrd
    arbitrator.c, 15
mrt_t, 6
    id, 7
    link_c, 7
    next, 7
    ssdp, 7

next
    mrt_t, 7
    object_t, 8
    rd_t, 11
    ssd_t, 12

object_t, 8
    ctx, 8
    id, 8
    next, 8
    rid, 9

rd_t, 10
    child_rids, 11
    mass, 11
    next, 11
    rid, 11
    rt_objects, 11
rid
    object_t, 9

rd_t, 11
rt_objects
rd_t, 11

sched_run
arbitrator.c, 16
sched_start
arbitrator_if, 5
sched_stop
arbitrator_if, 5
schedule
arbitrator_if, 5
sid
ssd_t, 12
ssd_t, 12
link_c, 12
next, 12
sid, 12
ssdp
mrt_t, 7
sys/brain/arbitrator.c, 14

t_current
arbitrator.c, 16
t_stack_base
arbitrator.c, 16
taskarray
arbitrator.c, 16
taskcount
arbitrator.c, 16