bsp-microej-async-worker

User Manual

Reference:	TLT-XXX-MAN-bsp-microej-async-worker-bsp-microej-async-worker
Version:	0.2.1
Revision:	XXX

Confidentiality & Intellectual Property

Table of Contents

Chapter 1. Data Structure Documentation

1.1. MICROEJ_ASYNC_WORKER_handle_t struct Reference

```
#include <microej_async_worker.h>
```

1.1.1. Data Fields

- int32_t job_count
- MICROEJ_ASYNC_WORKER_job_t * free_jobs
- void * params
- int32_t params_sizeof
- int32_t waiting_threads_length
- int32_t * waiting_threads
- uint16_t waiting_thread_offset
- uint16_t free_waiting_thread_offset
- OSAL_queue_handle_t jobs_queue
- OSAL_task_handle_t task
- OSAL_mutex_handle_t mutex

An async worker.

Detailed Description

Workers are declared using the `MICROEJ_ASYNC_WORKER_worker_declare()` macro and initialized with `MICROEJ_ASYNC_WORKER_initialize()`.

All the fields of this structure are internal data and must not be modified.

Definition at line 170 of file microej_async_worker.h

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

- microej_async_worker.h

1.1.2. Field Documentation

1.2. MICROEJ_ASYNC_WORKER_job struct Reference

```
#include <microej_async_worker.h>
```

1.2.1. Data Structures

1.2.2. Data Fields

- void * params

  Pointers to the parameters.

- MICROEJ_ASYNC_WORKER_action_t action
- int32_t thread_id
- MICROEJ_ASYNC_WORKER_job_t * next_free_job
- struct MICROEJ_ASYNC_WORKER_job::@0 _intern

  Structure internal data. Must not be modified.

A job to execute in a worker.

Detailed Description

Jobs are allocated using `MICROEJ_ASYNC_WORKER_allocate_job()` and freed using `MICROEJ_ASYNC_WORKER_free_job()`.

Definition at line 146 of file microej_async_worker.h

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

- microej_async_worker.h

1.2.3. Field Documentation

void* MICROEJ_ASYNC_WORKER_job::params

This pointer is initialized by the system and references an union of type `_param_type` (value given to `MICROEJ_ASYNC_WORKER_worker_declare()`). This pointer field must not be modified but the content of the referenced union can be modified.

Definition at line 153 of file microej_async_worker.h

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

- microej_async_worker.h

Chapter 2. File Documentation

2.1. microej_async_worker.h File Reference

```
#include <stdint.h>
```

```
#include "sni.h"
```

```
#include "osal.h"
```

2.1.1. Data Structures

- struct MICROEJ_ASYNC_WORKER_job

  A job to execute in a worker.

- struct MICROEJ_ASYNC_WORKER_handle_t

  An async worker.

2.1.2. Macros

- #define MICROEJ_ASYNC_WORKER_worker_declare _param_type _name ## _params[_job_count];\ MICROEJ_ASYNC_WORKER_job_t _name ## _jobs[_job_count];\ int32_t _name ## _waiting_threads[_waiting_list_size+1];\ MICROEJ_ASYNC_WORKER_handle_t _name = {\ .job_count = _job_count,\ .free_jobs = _name ## _jobs,\ .params = _name ## _params,\ .params_sizeof = sizeof(_param_type), \ .waiting_threads_length = _waiting_list_size+1,\ .waiting_threads = _name ## _waiting_threads,\ .waiting_thread_offset = 0,\ .free_waiting_thread_offset = 0\ }

  Declares a worker named <code>_name</code>.

2.1.3. Enumerations

- enum MICROEJ_ASYNC_WORKER_status_t {
  MICROEJ_ASYNC_WORKER_OK,
  MICROEJ_ASYNC_WORKER_ERROR,
  MICROEJ_ASYNC_WORKER_INVALID_ARGS
  }

  Return codes list.

2.1.4. Typedefs

- typedef struct MICROEJ_ASYNC_WORKER_job MICROEJ_ASYNC_WORKER_job_t

  See <code>struct MICROEJ_ASYNC_WORKER_job</code>.

- typedef void(* MICROEJ_ASYNC_WORKER_action_t

  Pointer to a function to call asynchronously.

2.1.5. Functions

- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_initialize (MICROEJ_ASYNC_WORKER_handle_t * async_worker, uint8_t * name, OSAL_task_stack_t stack, int32_t priority)

  Initializes and starts a worker previously declared with <code>MICROEJ_ASYNC_WORKER_worker_declare()</code> macro.

- MICROEJ_ASYNC_WORKER_job_t * MICROEJ_ASYNC_WORKER_allocate_job (MICROEJ_ASYNC_WORKER_handle_t * async_worker, SNI_callback sni_retry_callback)

  Allocates a new job for the given worker.

- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_free_job (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job)

  Frees a job previously allocated with MICROEJ_ASYNC_WORKER_allocate_job().

- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_async_exec (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job, MICROEJ_ASYNC_WORKER_action_t action, SNI_callback on_done_callback)

  Executes the given job asynchronously.

- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_async_exec_no_wait (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job, MICROEJ_ASYNC_WORKER_action_t action)

  Executes the given job asynchronously.

- MICROEJ_ASYNC_WORKER_job_t * MICROEJ_ASYNC_WORKER_get_job_done (void)

  Returns the job that has been executed.

Detailed Description

Asynchronous Worker API. This library helps writing SNI functions that must be executed asynchronously.

The execution of an SNI function prevents the other Java thread to be scheduled. That's why blocking native functions or long native functions should be executed asynchronously.

An async worker is declared statically using the `MICROEJ_ASYNC_WORKER_worker_declare()` macro and started with the `MICROEJ_ASYNC_WORKER_initialize()` function. Jobs are allocated using `MICROEJ_ASYNC_WORKER_allocate_job()` and scheduled with `MICROEJ_ASYNC_WORKER_async_exec()`.

Typical usage consists in declaring:

- for each SNI function, a structure that contains the parameters of the function,
- an union of all the previously declared structures,
- the async worker using `MICROEJ_ASYNC_WORKER_worker_declare()` macro.

For example, if we have to implement asynchronously the following two SNI functions :

```
int foo(int i, int j);
void bar(int i);
```

Then we will declare the async worker as following:

```
// foo() parameters structure
typedef struct {
        int i;
        int j;
        int result;
} foo_param_t;

// bar() parameters structure
typedef struct {
        int i;
} bar_param_t;

// union of all the parameters structures
typedef union {
        foo_param_t foo;
        bar_param_t bar;
        ...
} my_worker_param_t;

// Declare the worker and the task that will execute it
MICROEJ_ASYNC_WORKER_worker_declare(my_worker, MY_WORKER_JOB_COUNT, my_worker_param
OSAL_task_stack_declare(my_worker_stack, MY_WORKER_STACK_SIZE);

void initialize(){
    MICROEJ_ASYNC_WORKER_status_t status = MICROEJ_ASYNC_WORKER_initialize(&my_work
    if(status != MICROEJ_ASYNC_WORKER_OK){
        ...
    }
    ...
```

```
}

int foo(int i, int j){
    MICROEJ_ASYNC_WORKER_job_t* job = MICROEJ_ASYNC_WORKER_allocate_job(&my_worker,
    if(job != NULL){
        foo_param_t* params = (foo_param_t*)job->params;
        params->i = i;
        params->j = j;

        MICROEJ_ASYNC_WORKER_status_t status = MICROEJ_ASYNC_WORKER_async_exec(&my_
        if(status != MICROEJ_ASYNC_WORKER_OK){
            // Error
            MICROEJ_ASYNC_WORKER_free_job(&my_worker, job);
        }
    }
    // If an error occurred, then an exception is pending,
    // otherwise the current thread is suspended.
    // In any case, the returned value is not used.
    return -1;
}

void foo_action(MICROEJ_ASYNC_WORKER_job_t* job){
    foo_param_t* params = (foo_param_t*)job->params;

    int i = params->i;
    int j = params->j;

    ...

    params->result = result;
}

int foo_on_done(int i, int j){
    MICROEJ_ASYNC_WORKER_job_t* job = MICROEJ_ASYNC_WORKER_get_job_done();
    foo_param_t* params = (foo_param_t*)job->params;

    int result = params->result;
    MICROEJ_ASYNC_WORKER_free_job(&my_worker, job);

    return result;
}

...
```

Author: . MicroEJ Developer Team

Version: . 0.2.1

Date: . 13 November 2020

Definition in file C:/jenkins/workspace/M0124_CCO-Mic---4c994aea/bsp-microej-async-worker/target~/ccomponentWorking/bsp/util/inc/microej_async_worker.h

2.2. microej_async_worker.c File Reference

```
#include "microej_async_worker.h"
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <stdbool.h>
```

2.2.1. Functions

- static void * MICROEJ_ASYNC_WORKER_loop (void * args)
- static MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_async_exec_intern (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job, MICROEJ_ASYNC_WORKER_action_t action, SNI_callback on_done_callback, bool wait)
- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_initialize (MICROEJ_ASYNC_WORKER_handle_t * async_worker, uint8_t * name, OSAL_task_stack_t stack, int32_t priority)

  Initializes and starts a worker previously declared with <code>MICROEJ_ASYNC_WORKER_worker_declare()</code> macro.
- MICROEJ_ASYNC_WORKER_job_t * MICROEJ_ASYNC_WORKER_allocate_job (MICROEJ_ASYNC_WORKER_handle_t * async_worker, SNI_callback sni_retry_callback)

  Allocates a new job for the given worker.
- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_free_job (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job)

  Frees a job previously allocated with MICROEJ_ASYNC_WORKER_allocate_job().
- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_async_exec (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job, MICROEJ_ASYNC_WORKER_action_t action, SNI_callback on_done_callback)

  Executes the given job asynchronously.

- MICROEJ_ASYNC_WORKER_status_t MICROEJ_ASYNC_WORKER_async_exec_no_wait (MICROEJ_ASYNC_WORKER_handle_t * async_worker, MICROEJ_ASYNC_WORKER_job_t * job, MICROEJ_ASYNC_WORKER_action_t action)

  Executes the given job asynchronously.

- MICROEJ_ASYNC_WORKER_job_t * MICROEJ_ASYNC_WORKER_get_job_done (void)

  Returns the job that has been executed.

Detailed Description

Asynchronous Worker implementation.

Author: . MicroEJ Developer Team

Version: . 0.2.1

Date: . 13 November 2020

Definition in file C:/jenkins/workspace/M0124_CCO-Mic---4c994aea/bsp-microej-async-worker/target~/ccomponentWorking/bsp/util/src/microej_async_worker.c