

## User Mode thread Scheduling (kernel module)

Generated by Doxygen 1.9.3



|  |          |
|--|----------|
| <b>1 Data Structure Index</b>                    | <b>1</b> |
| 1.1 Data Structures                              | 1        |
| <b>2 File Index</b>                              | <b>3</b> |
| 2.1 File List                                    | 3        |
| <b>3 Data Structure Documentation</b>            | <b>5</b> |
| 3.1 completion_queue_descriptor Struct Reference | 5        |
| 3.1.1 Detailed Description                       | 5        |
| 3.1.2 Field Documentation                        | 5        |
| 3.1.2.1 completion_queue                         | 5        |
| 3.1.2.2 hlist                                    | 6        |
| 3.1.2.3 id                                       | 6        |
| 3.1.2.4 used_by_couter                           | 6        |
| 3.2 cq_proc Struct Reference                     | 6        |
| 3.2.1 Detailed Description                       | 6        |
| 3.2.2 Field Documentation                        | 6        |
| 3.2.2.1 cq_id                                    | 7        |
| 3.2.2.2 dir_entry                                | 7        |
| 3.2.2.3 hlist                                    | 7        |
| 3.2.2.4 link_to                                  | 7        |
| 3.2.2.5 name_dir                                 | 7        |
| 3.2.2.6 path                                     | 7        |
| 3.3 owner_proc Struct Reference                  | 7        |
| 3.3.1 Detailed Description                       | 8        |
| 3.3.2 Field Documentation                        | 8        |
| 3.3.2.1 dir_entry                                | 8        |
| 3.3.2.2 hlist                                    | 8        |
| 3.3.2.3 name_dir                                 | 8        |
| 3.3.2.4 pid                                      | 8        |
| 3.3.2.5 sched_entry                              | 9        |
| 3.4 ums_cq_param Struct Reference                | 9        |
| 3.4.1 Field Documentation                        | 9        |
| 3.4.1.1 completion_queue_id                      | 9        |
| 3.4.1.2 pids                                     | 9        |
| 3.5 ums_km_param Struct Reference                | 9        |
| 3.5.1 Field Documentation                        | 10       |
| 3.5.1.1 cq_id                                    | 10       |
| 3.5.1.2 owner_pid                                | 10       |
| 3.6 ums_proc Struct Reference                    | 10       |
| 3.6.1 Detailed Description                       | 10       |
| 3.6.2 Field Documentation                        | 11       |
| 3.6.2.1 cq_id                                    | 11       |

|          |  |    |
|----------|--|----|
| 3.6.2.2  | <a href="#">dir_entry</a>                      | 11 |
| 3.6.2.3  | <a href="#">hlist</a>                          | 11 |
| 3.6.2.4  | <a href="#">info_entry</a>                     | 11 |
| 3.6.2.5  | <a href="#">link_to</a>                        | 11 |
| 3.6.2.6  | <a href="#">name_dir</a>                       | 11 |
| 3.6.2.7  | <a href="#">path</a>                           | 11 |
| 3.6.2.8  | <a href="#">pid</a>                            | 12 |
| 3.6.2.9  | <a href="#">pid_owner</a>                      | 12 |
| 3.7      | <a href="#">ums_scheduler Struct Reference</a> | 12 |
| 3.7.1    | <a href="#">Detailed Description</a>           | 12 |
| 3.7.2    | <a href="#">Field Documentation</a>            | 12 |
| 3.7.2.1  | <a href="#">cq_list</a>                        | 13 |
| 3.7.2.2  | <a href="#">hlist</a>                          | 13 |
| 3.7.2.3  | <a href="#">last_wkt_runtime</a>               | 13 |
| 3.7.2.4  | <a href="#">list</a>                           | 13 |
| 3.7.2.5  | <a href="#">owner_pid</a>                      | 13 |
| 3.7.2.6  | <a href="#">pid</a>                            | 13 |
| 3.7.2.7  | <a href="#">pid_wkt_sched</a>                  | 13 |
| 3.7.2.8  | <a href="#">saved_fpu_regs</a>                 | 13 |
| 3.7.2.9  | <a href="#">saved_pt_regs</a>                  | 14 |
| 3.7.2.10 | <a href="#">state</a>                          | 14 |
| 3.7.2.11 | <a href="#">task_struct</a>                    | 14 |
| 3.7.2.12 | <a href="#">total_switch</a>                   | 14 |
| 3.7.2.13 | <a href="#">ums_cq_id</a>                      | 14 |
| 3.7.2.14 | <a href="#">wkt_sched_struct</a>               | 14 |
| 3.8      | <a href="#">worker_proc Struct Reference</a>   | 14 |
| 3.8.1    | <a href="#">Detailed Description</a>           | 15 |
| 3.8.2    | <a href="#">Field Documentation</a>            | 15 |
| 3.8.2.1  | <a href="#">dir_entry</a>                      | 15 |
| 3.8.2.2  | <a href="#">hlist</a>                          | 15 |
| 3.8.2.3  | <a href="#">info_entry</a>                     | 15 |
| 3.8.2.4  | <a href="#">link_to</a>                        | 15 |
| 3.8.2.5  | <a href="#">name_dir</a>                       | 15 |
| 3.8.2.6  | <a href="#">path</a>                           | 16 |
| 3.8.2.7  | <a href="#">pid</a>                            | 16 |
| 3.9      | <a href="#">worker_thread Struct Reference</a> | 16 |
| 3.9.1    | <a href="#">Detailed Description</a>           | 16 |
| 3.9.2    | <a href="#">Field Documentation</a>            | 16 |
| 3.9.2.1  | <a href="#">hlist</a>                          | 17 |
| 3.9.2.2  | <a href="#">list</a>                           | 17 |
| 3.9.2.3  | <a href="#">pid</a>                            | 17 |
| 3.9.2.4  | <a href="#">saved_fpu_regs</a>                 | 17 |

|  |           |
|--|-----------|
| 3.9.2.5 saved_pt_regs . . . . .                        | 17        |
| 3.9.2.6 scheduled_by . . . . .                         | 17        |
| 3.9.2.7 state . . . . .                                | 17        |
| 3.9.2.8 task_struct . . . . .                          | 17        |
| 3.9.2.9 time_at_switch . . . . .                       | 18        |
| 3.9.3 of switch that this work thread caused . . . . . | 18        |
| 3.9.3.1 total_runtime . . . . .                        | 18        |
| 3.9.3.2 total_switch . . . . .                         | 18        |
| 3.9.4 of switch that this work thread caused . . . . . | 18        |
| <b>4 File Documentation</b>                            | <b>19</b> |
| 4.1 device.c File Reference . . . . .                  | 19        |
| 4.1.1 Detailed Description . . . . .                   | 19        |
| 4.1.2 Function Documentation . . . . .                 | 19        |
| 4.1.2.1 try_start_device() . . . . .                   | 20        |
| 4.1.3 Variable Documentation . . . . .                 | 20        |
| 4.1.3.1 cmds . . . . .                                 | 20        |
| 4.2 device.h File Reference . . . . .                  | 20        |
| 4.2.1 Detailed Description . . . . .                   | 21        |
| 4.2.2 Function Documentation . . . . .                 | 21        |
| 4.2.2.1 try_start_device() . . . . .                   | 21        |
| 4.3 device.h . . . . .                                 | 21        |
| 4.4 module.c File Reference . . . . .                  | 22        |
| 4.4.1 Detailed Description . . . . .                   | 22        |
| 4.5 module.h File Reference . . . . .                  | 22        |
| 4.5.1 Detailed Description . . . . .                   | 23        |
| 4.6 module.h . . . . .                                 | 23        |
| 4.7 proc.c File Reference . . . . .                    | 23        |
| 4.7.1 Detailed Description . . . . .                   | 24        |
| 4.7.2 Function Documentation . . . . .                 | 24        |
| 4.7.2.1 clear_ums_proc() . . . . .                     | 24        |
| 4.7.2.2 Proc_Update_Cq_Created() . . . . .             | 24        |
| 4.7.2.3 Proc_Update_Cq_Deleted() . . . . .             | 25        |
| 4.7.2.4 Proc_Update_Ums_Created() . . . . .            | 25        |
| 4.7.2.5 Proc_Update_Ums_Ended() . . . . .              | 26        |
| 4.7.2.6 Proc_Update_Worker_Appended() . . . . .        | 26        |
| 4.7.2.7 Proc_Update_Worker_Created() . . . . .         | 26        |
| 4.7.2.8 Proc_Update_Worker_Ended() . . . . .           | 27        |
| 4.7.2.9 try_build_ums_proc() . . . . .                 | 27        |
| 4.8 proc.h File Reference . . . . .                    | 27        |
| 4.8.1 Detailed Description . . . . .                   | 29        |
| 4.8.2 Function Documentation . . . . .                 | 29        |

|                                       |    |
|---------------------------------------|----|
| 4.8.2.1 clear_ums_proc()              | 29 |
| 4.8.2.2 Proc_Update_Cq_Created()      | 29 |
| 4.8.2.3 Proc_Update_Cq_Deleted()      | 30 |
| 4.8.2.4 Proc_Update_Ums_Created()     | 30 |
| 4.8.2.5 Proc_Update_Ums_Ended()       | 31 |
| 4.8.2.6 Proc_Update_Worker_Appended() | 31 |
| 4.8.2.7 Proc_Update_Worker_Created()  | 31 |
| 4.8.2.8 Proc_Update_Worker_Ended()    | 32 |
| 4.8.2.9 try_build_ums_proc()          | 32 |
| 4.9 proc.h                            | 32 |
| 4.10 shared.h File Reference          | 34 |
| 4.10.1 Detailed Description           | 34 |
| 4.11 shared.h                         | 35 |
| 4.12 ums.c File Reference             | 35 |
| 4.12.1 Detailed Description           | 37 |
| 4.12.2 Function Documentation         | 37 |
| 4.12.2.1 append_to_cq()               | 37 |
| 4.12.2.2 DECLARE_BITMAP()             | 37 |
| 4.12.2.3 DEFINE_HASHTABLE() [1/3]     | 37 |
| 4.12.2.4 DEFINE_HASHTABLE() [2/3]     | 38 |
| 4.12.2.5 DEFINE_HASHTABLE() [3/3]     | 38 |
| 4.12.2.6 dequeue_cq()                 | 38 |
| 4.12.2.7 end_ums_scheduler()          | 38 |
| 4.12.2.8 end_worker_thread()          | 39 |
| 4.12.3 Implementation                 | 39 |
| 4.12.3.1 execute_wkt()                | 40 |
| 4.12.4 Implementation                 | 40 |
| 4.12.4.1 Get_UMS()                    | 40 |
| 4.12.4.2 Get_UMS_from_WKT()           | 41 |
| 4.12.4.3 Get_UMS_Info()               | 41 |
| 4.12.4.4 Get_WKT()                    | 41 |
| 4.12.4.5 Get_WKT_Info()               | 43 |
| 4.12.4.6 init_cq()                    | 43 |
| 4.12.5 Implementation                 | 43 |
| 4.12.5.1 init_ums_scheduler()         | 44 |
| 4.12.6 Implementation                 | 44 |
| 4.12.6.1 init_worker_thread()         | 44 |
| 4.12.7 Implementation                 | 45 |
| 4.12.7.1 try_build_ums_core()         | 45 |
| 4.12.7.2 ums_do_unwait()              | 45 |
| 4.12.7.3 ums_do_wait()                | 46 |
| 4.12.7.4 yield_to_ums()               | 46 |

|   |    |
|---|----|
| 4.12.8 Implementation . . . . .                       | 46 |
| 4.12.9 Variable Documentation . . . . .               | 47 |
| 4.12.9.1 state . . . . .                              | 47 |
| 4.13 ums.h File Reference . . . . .                   | 47 |
| 4.13.1 Detailed Description . . . . .                 | 49 |
| 4.13.2 Function Documentation . . . . .               | 50 |
| 4.13.2.1 append_to_cq() . . . . .                     | 50 |
| 4.13.2.2 dequeue_cq() . . . . .                       | 50 |
| 4.13.2.3 end_ums_scheduler() . . . . .                | 50 |
| 4.13.2.4 end_worker_thread() . . . . .                | 51 |
| 4.13.3 Implementation . . . . .                       | 51 |
| 4.13.3.1 execute_wkt() . . . . .                      | 52 |
| 4.13.4 Implementation . . . . .                       | 52 |
| 4.13.4.1 Get_UMS() . . . . .                          | 53 |
| 4.13.4.2 Get_UMS_from_WKT() . . . . .                 | 53 |
| 4.13.4.3 Get_UMS_Info() . . . . .                     | 53 |
| 4.13.4.4 Get_WKT() . . . . .                          | 54 |
| 4.13.4.5 Get_WKT_Info() . . . . .                     | 54 |
| 4.13.4.6 init_cq() . . . . .                          | 54 |
| 4.13.5 Implementation . . . . .                       | 54 |
| 4.13.5.1 init_ums_scheduler() . . . . .               | 55 |
| 4.13.6 Implementation . . . . .                       | 55 |
| 4.13.6.1 init_worker_thread() . . . . .               | 56 |
| 4.13.7 Implementation . . . . .                       | 56 |
| 4.13.7.1 try_build_ums_core() . . . . .               | 56 |
| 4.13.7.2 ums_do_unwait() . . . . .                    | 56 |
| 4.13.7.3 ums_do_wait() . . . . .                      | 57 |
| 4.13.7.4 yield_to_ums() . . . . .                     | 57 |
| 4.13.8 Implementation . . . . .                       | 57 |
| 4.14 ums.h . . . . .                                  | 58 |
| 4.15 utility.h File Reference . . . . .               | 60 |
| 4.15.1 Detailed Description . . . . .                 | 60 |
| 4.15.2 Macro Definition Documentation . . . . .       | 60 |
| 4.15.2.1 add_new_item_to_hlist . . . . .              | 61 |
| 4.15.2.2 add_new_item_to_list . . . . .               | 61 |
| 4.15.2.3 delete_completion_queue_descriptor . . . . . | 61 |
| 4.15.2.4 get_hlist_item_by_id . . . . .               | 62 |
| 4.15.2.5 retriive_from_hlist . . . . .                | 62 |
| 4.15.2.6 retriive_from_list . . . . .                 | 63 |
| 4.15.2.7 ums_delete_hlist . . . . .                   | 63 |
| 4.15.2.8 ums_delete_list . . . . .                    | 64 |
| 4.16 utility.h . . . . .                              | 64 |





# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

|   |   |    |
|---|---|----|
| <a href="#">completion_queue_descriptor</a> | Is the representation of the completion queue contain the id of the completion queue and the completion queue itself . . . . .                  | 5  |
| <a href="#">cq_proc</a>                     | Data of the completion queue directory that will contain the worker thread contained in the completion queue . . . . .                          | 6  |
| <a href="#">owner_proc</a>                  | Data of the ums owner directory that contain the scheduler dir with all the UMS thread created by the owner . . . . .                           | 7  |
| <a href="#">ums_cq_param</a>                | . . . . .   | 9  |
| <a href="#">ums_km_param</a>                | . . . . .   | 9  |
| <a href="#">ums_proc</a>                    | Data of the UMS thread directory that will contain the info file of the UMS and the directory of the completion queue used by the UMS . . . . . | 10 |
| <a href="#">ums_scheduler</a>               | This struct contain all the data related to a Ums scheduler . . . . .   | 12 |
| <a href="#">worker_proc</a>                 | Data of the worker thread directory that contain the info file of the worker . . . . .  | 14 |
| <a href="#">worker_thread</a>               | Struct with the representation of the worker thread . . . . .   | 16 |



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

|                           |   |    |
|---------------------------|---|----|
| <a href="#">device.c</a>  | Contain the main ioctl switch for the requested ops . . . . .   | 19 |
| <a href="#">device.h</a>  | The header of the device section of the module . . . . .  | 20 |
| <a href="#">module.c</a>  | This file contains the implementation of the kernel module . . . . .  | 22 |
| <a href="#">module.h</a>  | The header of the kernel module . . . . .   | 22 |
| <a href="#">proc.c</a>    | This file contains the functionalities for managing the exposure of the stats info using the procs  | 23 |
| <a href="#">proc.h</a>    | This file is the header of the <a href="#">proc.c</a> file . . . . .  | 27 |
| <a href="#">shared.h</a>  | This file contains definition shared between kernel module and user library . . . . .   | 34 |
| <a href="#">ums.c</a>     | This file contains main definiton and function for the ums it contains all the function for changing the context and for handle the completion queue and the worker and ums threads . . . . . | 35 |
| <a href="#">ums.h</a>     | This file is the header of <a href="#">ums.c</a> . . . . .  | 47 |
| <a href="#">utility.h</a> | This file contains utility macro definition . . . . .   | 60 |



## Chapter 3

# Data Structure Documentation

### 3.1 completion\_queue\_descriptor Struct Reference

Is the representation of the completion queue contain the id of the completion queue and the completion queue itself.

```
#include <ums.h>
```

#### Data Fields

- struct list\_head [completion\\_queue](#)
- int [id](#)
- unsigned int [used\\_by\\_couter](#)
- struct hlist\_node [hlist](#)

#### 3.1.1 Detailed Description

Is the representation of the completion queue contain the id of the completion queue and the completion queue itself.

#### 3.1.2 Field Documentation

##### 3.1.2.1 completion\_queue

```
struct list_head completion_queue_descriptor::completion_queue
```

list of all the worker thread in the cq

### 3.1.2.2 hlist

```
struct hlist_node completion_queue_descriptor::hlist
```

hlist node for the htable

### 3.1.2.3 id

```
int completion_queue_descriptor::id
```

id of the completion queue

### 3.1.2.4 used\_by\_couter

```
unsigned int completion_queue_descriptor::used_by_couter
```

conter of the ums that use this cq

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

- [ums.h](#)

## 3.2 cq\_proc Struct Reference

contains the data of the completion queue directory that will contain the worker thread contained in the completion queue.

```
#include <proc.h>
```

### Data Fields

- struct proc\_dir\_entry \* [dir\\_entry](#)
- struct proc\_dir\_entry \* [link\\_to](#)
- int [cq\\_id](#)
- char [name\\_dir](#) [NAME\_BUFF]
- char [path](#) [NAME\_BUFF]
- struct hlist\_node [hlist](#)

### 3.2.1 Detailed Description

contains the data of the completion queue directory that will contain the worker thread contained in the completion queue.

### 3.2.2 Field Documentation

### 3.2.2.1 cq\_id

```
int cq_proc::cq_id
```

id of the completion queue

### 3.2.2.2 dir\_entry

```
struct proc_dir_entry* cq_proc::dir_entry
```

proc\_dir\_entry of the directory for the completion queue

### 3.2.2.3 hlist

```
struct hlist_node cq_proc::hlist
```

hlist\_node used forhtable

### 3.2.2.4 link\_to

```
struct proc_dir_entry* cq_proc::link_to
```

will contain the proc\_dir\_entry of the link to the UMS scheduler that use this completion queue

### 3.2.2.5 name\_dir

```
char cq_proc::name_dir[NAME_BUFF]
```

name of the completion queue dir (id to string)

### 3.2.2.6 path

```
char cq_proc::path[NAME_BUFF]
```

path to reach the completion queue dir inside of the CQ\_ALL\_DIR

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

- [proc.h](#)

## 3.3 owner\_proc Struct Reference

contains the data of the ums owner directory that contain the scheduler dir with all the UMS thread created by the owner

```
#include <proc.h>
```

## Data Fields

- struct proc\_dir\_entry \* [dir\\_entry](#)
- struct proc\_dir\_entry \* [sched\\_entry](#)
- unsigned [pid](#)
- char [name\\_dir](#) [NAME\_BUFF]
- struct hlist\_node [hlist](#)

### 3.3.1 Detailed Description

contains the data of the ums owner directory that contain the scheduler dir with all the UMS thread created by the owner

### 3.3.2 Field Documentation

#### 3.3.2.1 [dir\\_entry](#)

```
struct proc_dir_entry* owner_proc::dir_entry
```

directory of the owner

#### 3.3.2.2 [hlist](#)

```
struct hlist_node owner_proc::hlist
```

hlist\_node used for htable

#### 3.3.2.3 [name\\_dir](#)

```
char owner_proc::name_dir[NAME_BUFF]
```

name of the owner's dir (pid to string)

#### 3.3.2.4 [pid](#)

```
unsigned owner_proc::pid
```

pid of the owner



### 3.3.2.5 sched\_entry

```
struct proc_dir_entry* owner_proc::sched_entry
```

directory that will contain all the ums schedulers

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

- [proc.h](#)

## 3.4 ums\_cq\_param Struct Reference

### Data Fields

- int [completion\\_queue\\_id](#)
- int [pids](#) [COMPLETION\_QUEUE\_BUFF]

### 3.4.1 Field Documentation

#### 3.4.1.1 completion\_queue\_id

```
int ums_cq_param::completion_queue_id
```

id of the completion queue

#### 3.4.1.2 pids

```
int ums_cq_param::pids[COMPLETION_QUEUE_BUFF]
```

array with pid of the worker threads

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

- [shared.h](#)

## 3.5 ums\_km\_param Struct Reference

### Data Fields

- int [cq\\_id](#)
- int [owner\\_pid](#)

### 3.5.1 Field Documentation

#### 3.5.1.1 cq\_id

```
int ums_km_param::cq_id
```

id of the completion queue that will be used by the ums

#### 3.5.1.2 owner\_pid

```
int ums_km_param::owner_pid
```

process that has request the creation of the ums

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

- [shared.h](#)

## 3.6 ums\_proc Struct Reference

contains the data of the UMS thread directory that will contain the info file of the UMS and the directory of the completion queue used by the UMS.

```
#include <proc.h>
```

### Data Fields

- struct proc\_dir\_entry \* [dir\\_entry](#)
- struct proc\_dir\_entry \* [info\\_entry](#)
- struct proc\_dir\_entry \* [link\\_to](#)
- unsigned [pid\\_owner](#)
- unsigned [pid](#)
- int [cq\\_id](#)
- char [name\\_dir](#) [NAME\_BUFF]
- char [path](#) [NAME\_BUFF]
- struct hlist\_node [hlist](#)

#### 3.6.1 Detailed Description

contains the data of the UMS thread directory that will contain the info file of the UMS and the directory of the completion queue used by the UMS.

## 3.6.2 Field Documentation

### 3.6.2.1 cq\_id

```
int ums_proc::cq_id
```

id of the completion queue used by the UMS thread

### 3.6.2.2 dir\_entry

```
struct proc_dir_entry* ums_proc::dir_entry
```

proc\_dir\_entry of the directory for the UMS thread

### 3.6.2.3 hlist

```
struct hlist_node ums_proc::hlist
```

hlist\_node used for htable

### 3.6.2.4 info\_entry

```
struct proc_dir_entry* ums_proc::info_entry
```

proc\_dir\_entry of the file containing the info of the UMS thread

### 3.6.2.5 link\_to

```
struct proc_dir_entry* ums_proc::link_to
```

will contain the proc\_dir\_entry of the link to the owner dir

### 3.6.2.6 name\_dir

```
char ums_proc::name_dir[NAME_BUFF]
```

name of the UMS thread dir (pid to string)

### 3.6.2.7 path

```
char ums_proc::path[NAME_BUFF]
```

path to reach the UMS thread dir inside of the UMS\_ALL\_DIR

### 3.6.2.8 pid

```
unsigned ums_proc::pid
```

pid of the UMD thread

### 3.6.2.9 pid\_owner

```
unsigned ums_proc::pid_owner
```

pid of the owner of the UMS thread

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

- [proc.h](#)

## 3.7 ums\_scheduler Struct Reference

This struct contain all the data related to a Ums scheduler.

```
#include <ums.h>
```

### Data Fields

- int [pid](#)
- struct task\_struct \* [task\\_struct](#)
- int [owner\\_pid](#)
- int [ums\\_cq\\_id](#)
- struct list\_head \* [cq\\_list](#)
- int [pid\\_wkt\\_sched](#)
- worker\_thread\_t \* [wkt\\_sched\\_struct](#)
- struct pt\_regs [saved\\_pt\\_regs](#)
- struct fpu [saved\\_fpu\\_regs](#)
- unsigned long [total\\_switch](#)
- unsigned long [last\\_wkt\\_runtime](#)
- long [state](#)
- struct list\_head [list](#)
- struct hlist\_node [hlist](#)

### 3.7.1 Detailed Description

This struct contain all the data related to a Ums scheduler.

### 3.7.2 Field Documentation

### 3.7.2.1 cq\_list

```
struct list_head* ums_scheduler::cq_list
```

completion\_queue of this scheduler

### 3.7.2.2 hlist

```
struct hlist_node ums_scheduler::hlist
```

for implementing the master\_ums\_hashlist

### 3.7.2.3 last\_wkt\_runtime

```
unsigned long ums_scheduler::last_wkt_runtime
```

time needed for the last worker thread switch

### 3.7.2.4 list

```
struct list_head ums_scheduler::list
```

all the scheduler are into a global list

### 3.7.2.5 owner\_pid

```
int ums_scheduler::owner_pid
```

pid of the parent process that has generated the ums

### 3.7.2.6 pid

```
int ums_scheduler::pid
```

pid of the ums thread (is also its id)

### 3.7.2.7 pid\_wkt\_sched

```
int ums_scheduler::pid_wkt_sched
```

pid of the worker scheduled or -1

### 3.7.2.8 saved\_fpu\_regs

```
struct fpu ums_scheduler::saved_fpu_regs
```

saved fpu register

### 3.7.2.9 saved\_pt\_regs

```
struct pt_regs ums_scheduler::saved_pt_regs
```

current context of the cpu registers

### 3.7.2.10 state

```
long ums_scheduler::state
```

current state of the scheduler IDLE | RUNNING

### 3.7.2.11 task\_struct

```
struct task_struct* ums_scheduler::task_struct
```

task\_struct of the ums thread

### 3.7.2.12 total\_switch

```
unsigned long ums_scheduler::total_switch
```

total number of switch done by this scheduler

### 3.7.2.13 ums\_cq\_id

```
int ums_scheduler::ums_cq_id
```

id of the completion queue used by this ums

### 3.7.2.14 wkt\_sched\_struct

```
worker_thread_t* ums_scheduler::wkt_sched_struct
```

datastructure of the wkt scheduled

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

- [ums.h](#)

## 3.8 worker\_proc Struct Reference

contains the data of the worker thread directory that contain the info file of the worker.

```
#include <proc.h>
```

## Data Fields

- struct proc\_dir\_entry \* [dir\\_entry](#)
- struct proc\_dir\_entry \* [info\\_entry](#)
- struct proc\_dir\_entry \* [link\\_to](#)
- unsigned [pid](#)
- char [name\\_dir](#) [NAME\_BUFF]
- char [path](#) [NAME\_BUFF]
- struct hlist\_node [hlist](#)

### 3.8.1 Detailed Description

contains the data of the worker thread directory that contain the info file of the worker.

### 3.8.2 Field Documentation

#### 3.8.2.1 dir\_entry

```
struct proc_dir_entry* worker_proc::dir_entry
```

proc\_dir\_entry of the directory of the worker thread

#### 3.8.2.2 hlist

```
struct hlist_node worker_proc::hlist
```

hlist\_node used for htable

#### 3.8.2.3 info\_entry

```
struct proc_dir_entry* worker_proc::info_entry
```

proc\_dir\_entry of the file containing the info of the worker thread

#### 3.8.2.4 link\_to

```
struct proc_dir_entry* worker_proc::link_to
```

will contain the proc\_dir\_entry of the link to the completion queue dir

#### 3.8.2.5 name\_dir

```
char worker_proc::name_dir[NAME_BUFF]
```

name of the worker dir (pid to string)

### 3.8.2.6 path

```
char worker_proc::path[NAME_BUFF]
```

path to reach the worker dir inside of the WKT\_ALL\_DIR

### 3.8.2.7 pid

```
unsigned worker_proc::pid
```

pid of the worker thread

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

- [proc.h](#)

## 3.9 worker\_thread Struct Reference

Struct with the representation of the worker thread.

```
#include <ums.h>
```

### Data Fields

- int [pid](#)
- struct task\_struct \* [task\\_struct](#)
- struct pt\_regs [saved\\_pt\\_regs](#)
- struct fpu [saved\\_fpu\\_regs](#)
- int [scheduled\\_by](#)
- unsigned long [time\\_at\\_switch](#)
- unsigned long [total\\_switch](#)
- unsigned long [total\\_runtime](#)
- long [state](#)
- struct list\_head [list](#)
- struct hlist\_node [hlist](#)

### 3.9.1 Detailed Description

Struct with the representation of the worker thread.

### 3.9.2 Field Documentation



### 3.9.2.1 hlist

```
struct hlist_node worker_thread::hlist
```

for implementing the master\_wt\_hashlist

### 3.9.2.2 list

```
struct list_head worker_thread::list
```

for implementing the ready queue

### 3.9.2.3 pid

```
int worker_thread::pid
```

pid of the worker thread

### 3.9.2.4 saved\_fpu\_regs

```
struct fpu worker_thread::saved_fpu_regs
```

saved fpu register

### 3.9.2.5 saved\_pt\_regs

```
struct pt_regs worker_thread::saved_pt_regs
```

current context of the cpu registers

### 3.9.2.6 scheduled\_by

```
int worker_thread::scheduled_by
```

pid of the scheduler that has scheduled this thread

### 3.9.2.7 state

```
long worker_thread::state
```

current state of the worker

### 3.9.2.8 task\_struct

```
struct task_struct* worker_thread::task_struct
```

task\_struct of the worker thread

### 3.9.2.9 time\_at\_switch

```
unsigned long worker_thread::time_at_switch
```

## 3.9.3 of switch that this work thread caused

### 3.9.3.1 total\_runtime

```
unsigned long worker_thread::total_runtime
```

total running time of the thread

### 3.9.3.2 total\_switch

```
unsigned long worker_thread::total_switch
```

## 3.9.4 of switch that this work thread caused

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

- [ums.h](#)

## Chapter 4

# File Documentation

### 4.1 device.c File Reference

contain the main ioctl switch for the requested ops

```
#include "device.h"
```

#### Functions

- **DEFINE\_SPINLOCK** (ioctl\_spinlock)
- int [try\\_start\\_device](#) (void)  
*try to register the char device*
- void **stop\_device** (void)  
*deregister the device*

#### Variables

- char \* [cmds](#) [UMS\_IOC\_MAXNR+1]

#### 4.1.1 Detailed Description

contain the main ioctl switch for the requested ops

Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

#### 4.1.2 Function Documentation

#### 4.1.2.1 try\_start\_device()

```
int try_start_device (
    void )
```

try to register the char device

return 0 if registration ok return -1 if registration no ok

##### Returns

int

### 4.1.3 Variable Documentation

#### 4.1.3.1 cmds

```
char* cmds[UMS_IOC_MAXNR+1]
```

##### Initial value:

```
= {
    "RESET",
    "UMS_IOC_THREAD_YIELD",
    "UMS_IOC_THREAD_EXECUTE",
    "UMS_IOC_ENTER_UMS_SCHEDULING_MODE",
    "UMS_IOC_END_UMS_SCHEDULER",
    "UMS_IOC_INIT_WORKER_THREAD",
    "UMS_IOC_END_WORKER_THREAD",
    "UMS_IOC_INIT_COMPLETION_QUEUE",
    "UMS_IOC_APPEND_TO_COMPLETION_QUEUE",
    "UMS_IOC_DEQUEUE_COMPLETION_LIST"
}
```

## 4.2 device.h File Reference

The header of the device section of the module.

```
#include "shared.h"
#include "module.h"
#include "ums.h"
#include <asm/uaccess.h>
#include <asm/current.h>
#include <linux/ioctl.h>
#include <linux/miscdevice.h>
#include <linux/kernel.h>
#include <linux/cdev.h>
#include <linux/fs.h>
```

### Macros

- #define **MODULE\_DEV\_LOG** "UMS\_DEV: "

## Functions

- `int try\_start\_device (void)`  
*try to register the char device*
- `void stop\_device (void)`  
*deregister the device*

### 4.2.1 Detailed Description

The header of the device section of the module.

#### Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

### 4.2.2 Function Documentation

#### 4.2.2.1 `try_start_device()`

```
int try_start_device (
    void )
```

try to register the char device

return 0 if registration ok return -1 if registration no ok

#### Returns

int

## 4.3 device.h

[Go to the documentation of this file.](#)

```
1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
26 #ifndef __DEVICE_HEADER
27 #define __DEVICE_HEADER
28
29 #include "shared.h"
30 #include "module.h"
```

```
31 #include "ums.h"
32
33 #include <asm/uaccess.h>
34 #include <asm/current.h>
35 #include <linux/ioctl.h>
36 #include <linux/miscdevice.h>
37 #include <linux/kernel.h>
38 #include <linux/cdev.h>
39 #include <linux/fs.h>
40
41 #define MODULE_DEV_LOG "UMS_DEV: "
42
43 // #define UMS_DEV_DEBUG
44
45 int  try_start_device(void);
46 void stop_device(void);
47
48
49 #endif
```

## 4.4 module.c File Reference

This file contains the implementation of the kernel module.

```
#include "module.h"
```

### Functions

- **MODULE\_LICENSE** ("GPL")
- **MODULE\_AUTHOR** ("Tiziano Colagrossi <tiziano.colagrossi@gmail.com>")
- **MODULE\_DESCRIPTION** ("User Module Thread Scheduling kernel module")
- **MODULE\_VERSION** ("1.0.0")
- **module\_init** (ums\_module\_init)
- **module\_exit** (ums\_module\_exit)

### 4.4.1 Detailed Description

This file contains the implementation of the kernel module.

Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

## 4.5 module.h File Reference

The header of the kernel module.

```
#include "shared.h"
#include "device.h"
#include "ums.h"
#include <linux/init.h>
```

## Macros

- `#define MODULE_LOG "UMS: "`
- `#define UMS_MOD_DEBUG`

## Functions

- `int init_core (void)`
- `void destroy_core (void)`

### 4.5.1 Detailed Description

The header of the kernel module.

#### Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

## 4.6 module.h

[Go to the documentation of this file.](#)

```

1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
18 #ifndef __MODULE_HEADER
19 #define __MODULE_HEADER
20
21 #include "shared.h"
22 #include "device.h"
23 #include "ums.h"
24
25 #include <linux/init.h>
26
27 #define MODULE_LOG "UMS: "
28
29 #define UMS_MOD_DEBUG
30
31 int init_core(void);
32 void destroy_core(void);
33
34 #endif

```

## 4.7 proc.c File Reference

This file contains the functionalities for managing the exposure of the stats info using the procs.

```

#include "proc.h"
#include "shared.h"

```

## Functions

- **DEFINE\_HASHTABLE** (own\_htable, HASH\_KEY\_SIZE)
- **DEFINE\_HASHTABLE** (ums\_htable, HASH\_KEY\_SIZE)
- **DEFINE\_HASHTABLE** (wkt\_htable, HASH\_KEY\_SIZE)
- **DEFINE\_HASHTABLE** (cq\_htable, HASH\_KEY\_SIZE)
- int **Proc\_Update\_Worker\_Created** (int wkt\_pid)  
*Insert the created worker thread into the procs.*
- int **Proc\_Update\_Worker\_Ended** (int wkt\_pid)  
*Remove the worker that has ended from the procs.*
- int **Proc\_Update\_Worker\_Appended** (int wkt\_pid, int id)  
*link a worker to its completion queue*
- int **Proc\_Update\_Ums\_Created** (int ums\_pid, int owner\_pid, int id)  
*Insert the new UMS into the proc fs and link the completion queue path to the UMS folder.*
- int **Proc\_Update\_Ums\_Ended** (int ums\_pid)  
*Remove the UMS directory from the proc fs.*
- int **Proc\_Update\_Cq\_Created** (int id)  
*create a completion queue folder into the proc fs*
- int **Proc\_Update\_Cq\_Deleted** (int id)  
*remove the completion queue folder from the proc fs*
- int **try\_build\_ums\_proc** (void)  
*initialize the main data structures for the proc part for the ums*
- int **clear\_ums\_proc** (void)  
*clear the data structures allocated from this part of the UMS kernel module*

### 4.7.1 Detailed Description

This file contains the functionalities for managing the exposure of the stats info using the procs.

Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

### 4.7.2 Function Documentation

#### 4.7.2.1 clear\_ums\_proc()

```
int clear_ums_proc (
    void )
```

clear the data structures allocated from this part of the UMS kernel module

Returns

int

#### 4.7.2.2 Proc\_Update\_Cq\_Created()

```
int Proc_Update_Cq_Created (
    int id )
```

create a completion queue folder into the proc fs



**Parameters**

|           |                     |
|-----------|---------------------|
| <i>id</i> | completion queue id |
|-----------|---------------------|

**Returns**

int

**4.7.2.3 Proc\_Update\_Cq\_Deleted()**

```
int Proc_Update_Cq_Deleted (
    int id )
```

remove the completion queue folder from the proc fs

**Parameters**

|           |                     |
|-----------|---------------------|
| <i>id</i> | completion queue id |
|-----------|---------------------|

**Returns**

int

**4.7.2.4 Proc\_Update\_Ums\_Created()**

```
int Proc_Update_Ums_Created (
    int ums_pid,
    int owner_pid,
    int id )
```

Insert the new UMS into the proc fs and link the completion queue path to the UMS folder.

**Parameters**

|                  |                                     |
|------------------|-------------------------------------|
| <i>ums_pid</i>   | pid of the UMS                      |
| <i>owner_pid</i> | pid of the owner process of the UMS |
| <i>id</i>        | completion queue id of the UMS      |

**Returns**

int

#### 4.7.2.5 Proc\_Update\_Ums\_Ended()

```
int Proc_Update_Ums_Ended (
    int ums_pid )
```

Remove the UMS directory from the proc fs.

##### Parameters

|                |                |
|----------------|----------------|
| <i>ums_pid</i> | pid of the ums |
|----------------|----------------|

##### Returns

int

#### 4.7.2.6 Proc\_Update\_Worker\_Appended()

```
int Proc_Update_Worker_Appended (
    int wkt_pid,
    int id )
```

link a worker to its completion queue

##### Parameters

|                |   |
|----------------|---|
| <i>wkt_pid</i> | pid of the worker                                     |
| <i>id</i>      | completion queue where the worker need to be appended |

##### Returns

int

#### 4.7.2.7 Proc\_Update\_Worker\_Created()

```
int Proc_Update_Worker_Created (
    int wkt_pid )
```

Insert the created worker thread into the procs.

##### Parameters

|                |                   |
|----------------|-------------------|
| <i>wkt_pid</i> | pid of the worker |
|----------------|-------------------|

**Returns**

int

**4.7.2.8 Proc\_Update\_Worker\_Ended()**

```
int Proc_Update_Worker_Ended (
    int wkt_pid )
```

Remove the worker that has ended from the procs.

**Parameters**

|                |                   |
|----------------|-------------------|
| <i>wkt_pid</i> | pid of the worker |
|----------------|-------------------|

**Returns**

int

**4.7.2.9 try\_build\_ums\_proc()**

```
int try_build_ums_proc (
    void )
```

initialize the main data structures for the proc part for the ums

**Returns**

int

## 4.8 proc.h File Reference

This file is the header of the [proc.c](#) file.

```
#include "utility.h"
#include "ums.h"
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <linux/hashtable.h>
```

## Data Structures

- struct [owner\\_proc](#)  
*contains the data of the ums owner directory that contain the scheduler dir with all the UMS thread created by the owner*
- struct [worker\\_proc](#)  
*contains the data of the worker thread directory that contain the info file of the worker.*
- struct [ums\\_proc](#)  
*contains the data of the UMS thread directory that will contain the info file of the UMS and the directory of the completion queue used by the UMS.*
- struct [cq\\_proc](#)  
*contains the data of the completion queue directory that will contain the worker thread contained in the completion queue.*

## Macros

- #define **MODULE\_PROC\_LOG** "UMS\_PROC: "
- #define **UMS\_PROC\_DEBUG**
- #define **HASH\_KEY\_SIZE** 10
- #define **NAME\_BUFF** 30
- #define **PROC\_DIR** "ums"
- #define **SCHED\_DIR** "schedulers"
- #define **WKT\_DIR** "workers"
- #define **UMS\_INFO** "ums\_info"
- #define **WKT\_INFO** "wkt\_info"
- #define **UMS\_ALL\_DIR** ".all\_ums"
- #define **WKT\_ALL\_DIR** ".all\_wkt"
- #define **CQ\_ALL\_DIR** ".all\_cq"

## Typedefs

- typedef struct [owner\\_proc](#) **owner\_proc\_t**  
*contains the data of the ums owner directory that contain the scheduler dir with all the UMS thread created by the owner*
- typedef struct [worker\\_proc](#) **worker\_proc\_t**  
*contains the data of the worker thread directory that contain the info file of the worker.*
- typedef struct [ums\\_proc](#) **ums\_proc\_t**  
*contains the data of the UMS thread directory that will contain the info file of the UMS and the directory of the completion queue used by the UMS.*
- typedef struct [cq\\_proc](#) **cq\_proc\_t**  
*contains the data of the completion queue directory that will contain the worker thread contained in the completion queue.*

## Functions

- int [Proc\\_Update\\_Worker\\_Created](#) (int pid)  
*Insert the created worker thread into the procs.*
- int [Proc\\_Update\\_Worker\\_Ended](#) (int wkt\_pid)  
*Remove the worker that has ended from the procs.*
- int [Proc\\_Update\\_Worker\\_Appended](#) (int wkt\_pid, int id)  
*link a worker to its completion queue*
- int [Proc\\_Update\\_Ums\\_Created](#) (int pid, int pid\_owner, int id)  
*Insert the new UMS into the proc fs and link the completion queue path to the UMS folder.*
- int [Proc\\_Update\\_Ums\\_Ended](#) (int ums\_pid)  
*Remove the UMS directory from the proc fs.*
- int [Proc\\_Update\\_Cq\\_Created](#) (int id)  
*create a completion queue folder into the proc fs*
- int [Proc\\_Update\\_Cq\\_Deleted](#) (int id)  
*remove the completion queue folder from the proc fs*
- int [try\\_build\\_ums\\_proc](#) (void)  
*initialize the main data structures for the proc part for the ums*
- int [clear\\_ums\\_proc](#) (void)  
*clear the data structures allocated from this part of the UMS kernel module*

### 4.8.1 Detailed Description

This file is the header of the [proc.c](#) file.

#### Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

### 4.8.2 Function Documentation

#### 4.8.2.1 [clear\\_ums\\_proc\(\)](#)

```
int clear_ums_proc (
    void )
```

clear the data structures allocated from this part of the UMS kernel module

#### Returns

int

#### 4.8.2.2 [Proc\\_Update\\_Cq\\_Created\(\)](#)

```
int Proc_Update_Cq_Created (
    int id )
```

create a completion queue folder into the proc fs

**Parameters**

|           |                     |
|-----------|---------------------|
| <i>id</i> | completion queue id |
|-----------|---------------------|

**Returns**

int

**4.8.2.3 Proc\_Update\_Cq\_Deleted()**

```
int Proc_Update_Cq_Deleted (
    int id )
```

remove the completion queue folder from the proc fs

**Parameters**

|           |                     |
|-----------|---------------------|
| <i>id</i> | completion queue id |
|-----------|---------------------|

**Returns**

int

**4.8.2.4 Proc\_Update\_Ums\_Created()**

```
int Proc_Update_Ums_Created (
    int ums_pid,
    int owner_pid,
    int id )
```

Insert the new UMS into the proc fs and link the completion queue path to the UMS folder.

**Parameters**

|                  |                                     |
|------------------|-------------------------------------|
| <i>ums_pid</i>   | pid of the UMS                      |
| <i>owner_pid</i> | pid of the owner process of the UMS |
| <i>id</i>        | completion queue id of the UMS      |

**Returns**

int

#### 4.8.2.5 Proc\_Update\_Ums\_Ended()

```
int Proc_Update_Ums_Ended (
    int ums_pid )
```

Remove the UMS directory from the proc fs.

##### Parameters

|                |                |
|----------------|----------------|
| <i>ums_pid</i> | pid of the ums |
|----------------|----------------|

##### Returns

int

#### 4.8.2.6 Proc\_Update\_Worker\_Appended()

```
int Proc_Update_Worker_Appended (
    int wkt_pid,
    int id )
```

link a worker to its completion queue

##### Parameters

|                |   |
|----------------|---|
| <i>wkt_pid</i> | pid of the worker                                     |
| <i>id</i>      | completion queue where the worker need to be appended |

##### Returns

int

#### 4.8.2.7 Proc\_Update\_Worker\_Created()

```
int Proc_Update_Worker_Created (
    int wkt_pid )
```

Insert the created worker thread into the procs.

##### Parameters

|                |                   |
|----------------|-------------------|
| <i>wkt_pid</i> | pid of the worker |
|----------------|-------------------|

**Returns**

int

**4.8.2.8 Proc\_Update\_Worker\_Ended()**

```
int Proc_Update_Worker_Ended (
    int wkt_pid )
```

Remove the worker that has ended from the procs.

**Parameters**

|                |                   |
|----------------|-------------------|
| <i>wkt_pid</i> | pid of the worker |
|----------------|-------------------|

**Returns**

int

**4.8.2.9 try\_build\_ums\_proc()**

```
int try_build_ums_proc (
    void )
```

initialize the main data structures for the proc part for the ums

**Returns**

int

**4.9 proc.h**

[Go to the documentation of this file.](#)

```
1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
26 #ifndef __PROC_H
27 #define __PROC_H
```



```

28
29 #include "utility.h"
30 #include "ums.h"
31
32 #include <linux/kernel.h>
33 #include <linux/module.h>
34 #include <linux/slab.h>
35 #include <linux/proc_fs.h>
36 #include <linux/seq_file.h>
37 #include <linux/hashtable.h>
38
39 #define MODULE_PROC_LOG "UMS_PROC: "
40
41 #define UMS_PROC_DEBUG
42
43 #define HASH_KEY_SIZE 10
44 #define NAME_BUFF 30
45
46 #define PROC_DIR "ums"
47 #define SCHED_DIR "schedulers"
48 #define WKT_DIR "workers"
49 #define UMS_INFO "ums_info"
50 #define WKT_INFO "wkt_info"
51
52 #define UMS_ALL_DIR ".all_ums"
53 #define WKT_ALL_DIR ".all_wkt"
54 #define CQ_ALL_DIR ".all_cq"
55
56
57 typedef struct owner_proc{
58     struct proc_dir_entry * dir_entry;
59     struct proc_dir_entry * sched_entry;
60     unsigned pid;
61     char name_dir[NAME_BUFF];
62     struct hlist_node hlist;
63 }owner_proc_t;
64
65 typedef struct worker_proc{
66     struct proc_dir_entry * dir_entry;
67     struct proc_dir_entry * info_entry;
68     struct proc_dir_entry * link_to;
69     unsigned pid;
70     char name_dir[NAME_BUFF];
71     char path[NAME_BUFF];
72     struct hlist_node hlist;
73 }worker_proc_t;
74
75 typedef struct ums_proc{
76     struct proc_dir_entry * dir_entry;
77     struct proc_dir_entry * info_entry;
78     struct proc_dir_entry * link_to;
79     unsigned pid_owner;
80     unsigned pid;
81     int cq_id;
82     char name_dir[NAME_BUFF];
83     char path[NAME_BUFF];
84     struct hlist_node hlist;
85 }ums_proc_t;
86
87 typedef struct cq_proc{
88     struct proc_dir_entry * dir_entry;
89     struct proc_dir_entry * link_to;
90     int cq_id;
91     char name_dir[NAME_BUFF];
92     char path[NAME_BUFF];
93     struct hlist_node hlist;
94 }cq_proc_t;
95
96 /*
97 * Function exported for create file and directory inside the procs
98 */
99 int Proc_Update_Worker_Created (int pid );
100 int Proc_Update_Worker_Ended (int wkt_pid);
101 int Proc_Update_Worker_Appended (int wkt_pid , int id );
102 int Proc_Update_Ums_Created (int pid , int pid_owner, int id);
103 int Proc_Update_Ums_Ended (int ums_pid);
104 int Proc_Update_Cq_Created (int id );
105 int Proc_Update_Cq_Deleted (int id );
106
107 /*
108 * Function used to initialize this part of the UMS kernel module
109 */
110 int try_build_ums_proc (void);
111 int clear_ums_proc (void);
112
113 #endif

```

## 4.10 shared.h File Reference

This file contains definition shared between kernel module and user library.

```
#include <linux/ioctl.h>
```

### Data Structures

- struct [ums\\_km\\_param](#)
- struct [ums\\_cq\\_param](#)

### Macros

- #define **MODULE\_NAME** "ums"
- #define **MODULE\_PATH** "/dev/ums"
- #define **EXIT\_SUCCESS** 0
- #define **EXIT\_FAILURE** -1
- #define **THREAD\_RUNNING** -3
- #define **CQ\_FULL** -4
- #define **UMS\_IOC\_MAGIC** 0xF4
- #define **UMS\_IOCRESET\_IO** (UMS\_IOC\_MAGIC, 0)
- #define **UMS\_IOC\_THREAD\_YIELD\_IO** (UMS\_IOC\_MAGIC, 1)
- #define **UMS\_IOC\_THREAD\_EXECUTE\_IOW** (UMS\_IOC\_MAGIC, 2, unsigned)
- #define **UMS\_IOC\_ENTER\_UMS\_SCHEDULING\_MODE\_IOW** (UMS\_IOC\_MAGIC, 3, unsigned long)
- #define **UMS\_IOC\_END\_UMS\_SCHEDULER\_IO** (UMS\_IOC\_MAGIC, 4)
- #define **UMS\_IOC\_INIT\_WORKER\_THREAD\_IO** (UMS\_IOC\_MAGIC, 5)
- #define **UMS\_IOC\_END\_WORKER\_THREAD\_IO** (UMS\_IOC\_MAGIC, 6)
- #define **UMS\_IOC\_INIT\_COMPLETION\_QUEUE\_IOR** (UMS\_IOC\_MAGIC, 7, unsigned long)
- #define **UMS\_IOC\_APPEND\_TO\_COMPLETION\_QUEUE\_IOW** (UMS\_IOC\_MAGIC, 8, int)
- #define **UMS\_IOC\_DEQUEUE\_COMPLETION\_LIST\_IOR** (UMS\_IOC\_MAGIC, 9, unsigned long)
- #define **UMS\_IOC\_MAXNR** 9
- #define **COMPLETION\_QUEUE\_BUFF** 100

### Typedefs

- typedef void \*(\* **worker\_job\_t**) (void \*)
- typedef struct [ums\\_km\\_param](#) **ums\_km\_param\_t**
- typedef struct [ums\\_cq\\_param](#) **ums\_cq\_param\_t**

#### 4.10.1 Detailed Description

This file contains definition shared between kernel module and user library.

Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

## 4.11 shared.h

[Go to the documentation of this file.](#)

```

1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
18 #ifndef __SHARED_HEADER
19 #define __SHARED_HEADER
20
21 #include <linux/ioctl.h>
22
23 #define MODULE_NAME "ums"
24 #define MODULE_PATH "/dev/ums"
25
26 #define EXIT_SUCCESS 0
27 #define EXIT_FAILURE -1
28 #define THREAD_RUNNING -3
29 #define CQ_FULL -4
30
31 // Define ioctl command
32 #define UMS_IOC_MAGIC 0xF4
33
34 #define UMS_IOCRESET _IO (UMS_IOC_MAGIC, 0)
35 #define UMS_IOC_THREAD_YIELD _IO (UMS_IOC_MAGIC, 1)
36 #define UMS_IOC_THREAD_EXECUTE _IOW (UMS_IOC_MAGIC, 2, unsigned)
37 #define UMS_IOC_ENTER_UMS_SCHEDULING_MODE _IOW (UMS_IOC_MAGIC, 3, unsigned long)
38 #define UMS_IOC_END_UMS_SCHEDULER _IO (UMS_IOC_MAGIC, 4)
39 #define UMS_IOC_INIT_WORKER_THREAD _IO (UMS_IOC_MAGIC, 5)
40 #define UMS_IOC_END_WORKER_THREAD _IO (UMS_IOC_MAGIC, 6)
41 #define UMS_IOC_INIT_COMPLETION_QUEUE _IOR (UMS_IOC_MAGIC, 7, unsigned long)
42 #define UMS_IOC_APPEND_TO_COMPLETION_QUEUE _IOW (UMS_IOC_MAGIC, 8, int)
43 #define UMS_IOC_DEQUEUE_COMPLETION_LIST _IOR (UMS_IOC_MAGIC, 9, unsigned long)
44
45 #define UMS_IOC_MAXNR 9
46 #define COMPLETION_QUEUE_BUFF 100
47
48 typedef void (*worker_job_t)(void *);
49
50 typedef struct ums_km_param{
51     int cq_id;
52     int owner_pid;
53 }ums_km_param_t;
54
55 typedef struct ums_cq_param{
56     int completion_queue_id;
57     int pids[COMPLETION_QUEUE_BUFF];
58 }ums_cq_param_t;
59
60 #endif

```

## 4.12 ums.c File Reference

This file contains main definition and function for the ums it contains all the function for changing the context and for handle the completion queue and the worker and ums threads.

```
#include "ums.h"
```

## Functions

- [DECLARE\\_BITMAP](#) (cq\_id\_bitmap, BITMAP\_CQ\_SIZE)
- [DEFINE\\_HASHTABLE](#) (master\_wkt\_hashlist, HASH\_KEY\_SIZE)
- [DEFINE\\_HASHTABLE](#) (master\_cq\_hashlist, HASH\_KEY\_SIZE)
- [DEFINE\\_HASHTABLE](#) (master\_ums\_hashlist, HASH\_KEY\_SIZE)
- int [yield\\_to\\_ums](#) (spinlock\_t ioctl\_lock)
  - perform a context switch from wkt to ums that host it*
- int [execute\\_wkt](#) (spinlock\_t ioctl\_lock, unsigned \*u\_wkt\_pid)
  - perform a context switch from ums to wkt selected*
- int [init\\_cq](#) (spinlock\_t ioctl\_lock, void \*cq\_id\_u\_ptr)
  - Initialize the struct for a new completion queue.*
- int [append\\_to\\_cq](#) (spinlock\_t ioctl\_lock, [ums\\_cq\\_param\\_t](#) \*args)
  - Append worker threads to a completion queue.*
- int [dequeue\\_cq](#) (spinlock\_t ioctl\_lock, [ums\\_cq\\_param\\_t](#) \*ret\_cq)
  - return the first COMPLETION\_QUEUE\_BUFF size of worker thread id in the completion queue*
- int [init\\_ums\\_scheduler](#) (spinlock\_t ioctl\_lock, [ums\\_km\\_param\\_t](#) \*args)
  - convert a standard pthread into an UmsSchedulerThread*
- int [end\\_ums\\_scheduler](#) (spinlock\_t ioctl\_lock)
  - clear the data structure used by the ums*
- int [init\\_worker\\_thread](#) (spinlock\_t ioctl\_lock)
  - block at startup the [worker\\_thread](#) to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS*
- int [end\\_worker\\_thread](#) (spinlock\_t ioctl\_lock)
  - if SWITCH\_PT\_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task\_struct. Also remove the worker\_thread\_struct from the completion queue and the hashtable of the worker thread. if SWITCH\_PT\_REGS is not defined wake up the UMS*
- [worker\\_thread\\_t](#) \* [Get\\_WKT](#) (int wkt\_pid)
  - retrive a pointer to worker\_thread\_t struct*
- [ums\\_scheduler\\_t](#) \* [Get\\_UMS](#) (int ums\_pid)
  - retrive a pointer to ums\_scheduler\_t struct*
- [ums\\_scheduler\\_t](#) \* [Get\\_UMS\\_from\\_WKT](#) (int wkt\_pid)
  - retrive a pointer to ums\_scheduler\_t struct*
- char \* [Get\\_UMS\\_Info](#) (int ums\_pid)
  - retrive a char \* with the info of the UMS in a human readable fashion*
- char \* [Get\\_WKT\\_Info](#) (int wkt\_pid)
  - retrive a char \* with the info of the worker in a human readable fashion*
- void [ums\\_do\\_wait](#) ([worker\\_thread\\_t](#) \*from\_wkt, [ums\\_scheduler\\_t](#) \*to\_ums)
  - handler used in the case that a worker thread is put on wait*
- void [ums\\_do\\_unwait](#) ([worker\\_thread\\_t](#) \*from\_wkt, struct task\_struct \*p)
  - handler used in the case that a worker thread previously put on wait is awakened*
- int [try\\_build\\_ums\\_core](#) (void)
  - Initialize the data structure needed for the UMS.*
- void [clear\\_ums\\_core](#) (void)
  - Destroy the ums data structure.*

## Variables

- char \* [state](#) [5]

### 4.12.1 Detailed Description

This file contains main definition and function for the ums it contains all the function for changing the context and for handle the completion queue and the worker and ums threads.

#### Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

### 4.12.2 Function Documentation

#### 4.12.2.1 `append_to_cq()`

```
int append_to_cq (
    spinlock_t ioctl_lock,
    ums_cq_param_t * args )
```

Append worker threads to a completion queue.

It cycle over the completion queue buffer. The buffer is initialized to -1 so if reach -1 before that cycle over all the buffer means that it has appended all the worker

#### Parameters

|                   |   |
|-------------------|---|
| <i>ioctl_lock</i> |   |
| <i>args</i>       | pointer to the ums_cq_param_t struct placed in the user space |

#### Returns

int

#### 4.12.2.2 `DECLARE_BITMAP()`

```
DECLARE_BITMAP (
    cq_id_bitmap ,
    BITMAP_CQ_SIZE )
```

bitmap to keep track of the completion queue id

#### 4.12.2.3 `DEFINE_HASHTABLE()` [1/3]

```
DEFINE_HASHTABLE (
    master_cq_hashlist ,
    HASH_KEY_SIZE )
```

master hash table key:id data:ums\_completion\_queue\_list\_t

#### 4.12.2.4 DEFINE\_HASHTABLE() [2/3]

```
DEFINE_HASHTABLE (
    master_ums_hashlist ,
    HASH_KEY_SIZE )
```

hashtable of all created [ums\\_scheduler](#)

#### 4.12.2.5 DEFINE\_HASHTABLE() [3/3]

```
DEFINE_HASHTABLE (
    master_wkt_hashlist ,
    HASH_KEY_SIZE )
```

master hash table key:pid data:worker\_thread\_t

#### 4.12.2.6 dequeue\_cq()

```
int dequeue_cq (
    spinlock_t ioctl_lock,
    ums_cq_param_t * ret_cq )
```

return the first COMPLETION\_QUEUE\_BUFF size of worker thread id in the completion queue

##### Parameters

|                   |   |
|-------------------|---|
| <i>ioctl_lock</i> |   |
| <i>ret_cq</i>     | pointer to the ums_cq_param_t in the user space used to return the worker thread pid in the cq that are ready |

##### Returns

int

#### 4.12.2.7 end\_ums\_scheduler()

```
int end_ums_scheduler (
    spinlock_t ioctl_lock )
```

clear the data structure used by the ums

##### Parameters

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
|-------------------|--|

**Returns**

int

**4.12.2.8 end\_worker\_thread()**

```
int end_worker_thread (
    spinlock_t ioctl_lock )
```

if SWITCH\_PT\_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task\_struct. Also remove the worker\_thread\_struct from the completion queue and the hashtable of the worker thread. if SWITCH\_PT\_REGS is not defined wake up the UMS

**4.12.3 Implementation**

Check done by the function:

- The current pid needs to be a UMS pid if SWITCH\_PT\_REGS is defined else the current pid represent the worker scheduled
- if SWITCH\_PT\_REGS is defined, try to retrieve the worker\_thread\_t struct else try to retrieve the UMS using the worker pid

After this check perform:

- Restore context of the worker into his own task\_struct if SWITCH\_PT\_REGS is defined
- Restore ums context if SWITCH\_PT\_REGS is defined
- Remove the worker thread from the completion queue and from the hashtable and free the memory of the worker\_thread\_t struct
- Finally resume the worker thread
- If SWITCH\_PT\_REGS is not defined wake up also the UMS

**Parameters**

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
|-------------------|--|

**Returns**

int 0 if all went ok, else -1

#### 4.12.3.1 execute\_wkt()

```
int execute_wkt (
    spinlock_t ioctl_lock,
    unsigned * u_wkt_pid )
```

perform a context switch from ums to wkt selected

#### 4.12.4 Implementation

Initially the current check are done:

- Try to copy from user the value passed to the module by ioctl
- The current need to be an ums pid because only an UMS can schedule a worker thread.
- I try to retrieve the worker\_thread\_t from the cq\_list using the wkt\_pid\_to\_switch passed by the user arg
- If the state of the worker selected is W\_RUNNING the function end because the thread is scheduled by another UMS

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH\_PT\_REGS is defined, by saving the current state (pt\_regs and fxregs) into the ums\_scheduler\_t struct and restoring the previously saved state of the worker thread from the worker\_thread\_t struct. if SWITCH\_PT\_REGS is not defined, by stop the UMS and waking up the worker thread.

##### Parameters

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
| <i>u_wkt_pid</i>  | pointer to the user space mem of the id of the worker thread choose to be executed |

##### Returns

int 0 if all went ok, else -1

#### 4.12.4.1 Get\_UMS()

```
ums_scheduler_t * Get_UMS (
    int ums_pid )
```

retrieve a pointer to ums\_scheduler\_t struct



**Parameters**

|                |                               |
|----------------|-------------------------------|
| <i>ums_pid</i> | pid of the ums thread to find |
|----------------|-------------------------------|

**Returns**

ums\_scheduler\_t\*

**4.12.4.2 Get\_UMS\_from\_WKT()**

```
ums_scheduler_t * Get_UMS_from_WKT (
    int wkt_pid )
```

retrive a pointer to ums\_scheduler\_t struct

**Parameters**

|                |   |
|----------------|---|
| <i>wkt_pid</i> | pid of the worker actual scheduled by the ums |
|----------------|---|

**Returns**

ums\_scheduler\_t\*

**4.12.4.3 Get\_UMS\_Info()**

```
char * Get_UMS_Info (
    int ums_pid )
```

retrive a char \* with the info of the UMS in a human readable fashion

**Parameters**

|                |                |
|----------------|----------------|
| <i>ums_pid</i> | pid of the ums |
|----------------|----------------|

**Returns**

char\*

**4.12.4.4 Get\_WKT()**

```
worker_thread_t * Get_WKT (
    int wkt_pid )
```

retrive a pointer to worker\_thread\_t struct

**Parameters**

|                |                                  |
|----------------|----------------------------------|
| <i>wkt_pid</i> | pid of the worker thread to find |
|----------------|----------------------------------|

**Returns**

worker\_thread\_t\*

**4.12.4.5 Get\_WKT\_Info()**

```
char * Get_WKT_Info (
    int wkt_pid )
```

retrieves a char \* with the info of the worker in a human readable fashion

**Parameters**

|                |                          |
|----------------|--------------------------|
| <i>wkt_pid</i> | oid of the worker thread |
|----------------|--------------------------|

**Returns**

char\*

**4.12.4.6 init\_cq()**

```
int init_cq (
    spinlock_t ioctl_lock,
    void * cq_id_u_ptr )
```

Initialize the struct for a new completion queue.

**4.12.5 Implementation**

- Find id for a new completion queue from the bitmap and set the bit.
- Create a new completion\_queue\_descriptor\_t and initialize it
- Try to return the completion queue id

**Parameters**

|                    |                                    |
|--------------------|------------------------------------|
| <i>ioctl_lock</i>  |                                    |
| <i>cq_id_u_ptr</i> | pointer to the cq_id in user space |

**Returns**

int 0 if all went ok, else -1

**4.12.5.1 init\_ums\_scheduler()**

```
int init_ums_scheduler (
    spinlock_t ioctl_lock,
    ums_km_param_t * args )
```

convert a standard pthread into an UmsSchedulerThread

converts a standard pthread in a UMS Scheduler thread, the function takes as input a completion list of worker threads and a entry point function

**4.12.6 Implementation**

Initially the current check are done:

- Try to copy from user the value passed to the module by ioctl
- Check if the completion queue id is a valid id

After this checks the function populate the structs in the kernel:

- Create a new ums\_scheduler\_t struct for the new UMS created
- Initialize the struct with default value
- Link the completion queue to this UMS and update the used\_by\_couter entry in the completion\_queue\_descriptor\_t descriptor cause the completion queue can be shared among multiple UMS

**Parameters**

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
| <i>args</i>       | pointer to the ums_km_param_t struct in user space |

**Returns**

int 0 if all went ok, else -1

**4.12.6.1 init\_worker\_thread()**

```
int init_worker_thread (
    spinlock_t ioctl_lock )
```

block at startup the [worker\\_thread](#) to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS

### 4.12.7 Implementation

- Create a new `worker_thread_t` struct and initialize it by save the current pid of the real worker thread into its own struct, and save the pointer to the worker task\_struct into `task_struct`. And set the worker state to `W_READY`.
- If `SWITCH_PT_REGS` is defined, save the worker current state into the `worker_thread_t` struct
- Put on wait worker thread

#### Parameters

|                         |  |
|-------------------------|--|
| <code>ioctl_lock</code> |  |
|-------------------------|--|

#### Returns

int 0 if all went ok, else -1

#### 4.12.7.1 try\_build\_ums\_core()

```
int try_build_ums_core (
    void )
```

Initialize the data structure needed for the UMS.

It initialize the following hashtables:

- `master_wkt_hashlist` (key:pid , data:worker\_thread\_t)
- `master_cq_hashlist` (key:id , data:completion\_queue\_descriptor\_t)
- `master_ums_hashlist` (key:pid , data:ums\_scheduler\_t)
- `cq_id_bitmap` (is used to check the available id for the completion queues)

#### Returns

int

#### 4.12.7.2 ums\_do\_unwait()

```
void ums_do_unwait (
    worker_thread_t * from_wkt,
    struct task_struct * p )
```

handler used in the case that a worker thread previously put on wait is awakened

## Parameters

|                 |  |
|-----------------|--|
| <i>from_wkt</i> | pointer to worker_thread_t struct of the worker awakened |
| <i>p</i>        | task_struct of the worker thread awakened                |

**4.12.7.3 ums\_do\_wait()**

```
void ums_do_wait (
    worker_thread_t * from_wkt,
    ums_scheduler_t * to_ums )
```

handler used in the case that a worker thread is put on wait

## Parameters

|                 |  |
|-----------------|--|
| <i>from_wkt</i> | pointer to worker_thread_t struct of the worker thread that will put in wait |
| <i>to_ums</i>   | pointer to ums_scheduler_t struct if the ums thread that will wakeup         |

**4.12.7.4 yield\_to\_ums()**

```
int yield_to_ums (
    spinlock_t ioctl_lock )
```

perform a context switch from wkt to ums that host it

**4.12.8 Implementation**

Initially the current check are done:

- Try to retrieve the UMS that currently host the execution of the worker thread that has required the yield.
- Try to retrieve the worker thread struct scheduled saved inside the UMS struct.

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH\_PT\_REGS is defined, by saving the current state (pt\_regs and fxregs) into the worker\_thread\_t struct and restoring the previously saved state of the ums from the ums\_scheduler\_t struct. if SWITCH\_PT\_REGS is not defined, by stop the worker thread and waking up the UMS scheduler that previously has scheduled the worker.

## Parameters

|                         |  |
|-------------------------|--|
| <code>ioctl_lock</code> |  |
|-------------------------|--|

## Returns

int 0 if all went ok, else -1

## 4.12.9 Variable Documentation

### 4.12.9.1 state

```
char* state[5]
```

## Initial value:

```
= { "RUNNING"
    , "IDLE"
    , "READY", ""
    , "WAITING"
  }
```

## 4.13 ums.h File Reference

This file is the header of [ums.c](#).

```
#include "shared.h"
#include "utility.h"
#include "proc.h"
#include <asm/fpu/internal.h>
#include <asm/fpu/types.h>
#include <asm/ptrace.h>
#include <asm/uaccess.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/list.h>
#include <linux/hashtable.h>
#include <linux/sched.h>
#include <linux/ktime.h>
#include <linux/timekeeping.h>
#include <linux/bitmap.h>
```

## Data Structures

- struct [completion\\_queue\\_descriptor](#)  
*Is the representation of the completion queue contain the id of the completion queue and the completion queue itself.*
- struct [ums\\_scheduler](#)  
*This struct contain all the data related to a Ums scheduler.*
- struct [worker\\_thread](#)  
*Struct with the representation of the worker thread.*

## Macros

- #define **MODULE\_UMS\_LOG** "UMS\_MASTER\_FUNCS: "
- #define **F\_DEQUEUE** "DEQUEUE: "
- #define **F\_APPEND** "APPEND: "
- #define **F\_DESTROY\_CQ** "DESTROY\_CQ: "
- #define **F\_INIT\_CQ** "INIT\_CQ: "
- #define **F\_INIT\_WORKER** "INIT\_WORKER: "
- #define **F\_END\_WORKER** "END\_WORKER: "
- #define **F\_INIT\_UMS** "INIT\_UMS: "
- #define **F\_SCHED\_WRK** "SCHED\_WKT: "
- #define **F\_SCHED\_UMS** "SCHED\_UMS: "
- #define **HASH\_KEY\_SIZE** 10
- #define **BITMAP\_CQ\_SIZE** 128
- #define **STATE\_RUNNING** 0x0000
- #define **STATE\_IDLE** 0x0001
- #define **STATE\_READY** 0x0002
- #define **STATE\_WAITING** 0x0004
- #define **ums\_is\_idle**(ums) ((ums->state & STATE\_IDLE) != 0)
- #define **ums\_is\_running**(ums) ((ums->state & STATE\_RUNNING) != 0)
- #define **wkt\_is\_running**(wkt) ((wkt->state & STATE\_RUNNING) != 0)
- #define **wkt\_is\_ready**(wkt) ((wkt->state & STATE\_READY) != 0)
- #define **wkt\_is\_waiting**(wkt) ((wkt->state & STATE\_WAITING) != 0)
- #define **fxsave**(fpu) copy\_fxregs\_to\_kernel(fpu)
- #define **fxrestore**(fpu) copy\_kernel\_to\_fxregs(&((fpu)->state.fxsave))
- #define **UMS\_CORE\_DEBUG**
- #define **SWITCH\_PT\_REGS**

## Typedefs

- typedef struct [ums\\_scheduler](#) **ums\_scheduler\_t**  
*This struct contain all the data related to a Ums scheduler.*
- typedef struct [worker\\_thread](#) **worker\_thread\_t**  
*Struct with the representation of the worker thread.*
- typedef enum [direction](#) **direction\_t**  
*This enum represent the possible direction for update the stats.*
- typedef struct [completion\\_queue\\_descriptor](#) **completion\_queue\_descriptor\_t**  
*Is the representation of the completion queue contain the id of the completion queue and the completion queue itself.*

## Enumerations

- enum [direction](#) { **YIELD** , **EXECUTE** , **WAIT** }  
*This enum represent the possible direction for update the stats.*



## Functions

- int [yield\\_to\\_ums](#) (spinlock\_t ioctl\_lock)  
*perform a context switch from wkt to ums that host it*
- int [execute\\_wkt](#) (spinlock\_t ioctl\_lock, unsigned \*u\_wkt\_pid)  
*perform a context switch from ums to wkt selected*
- int [init\\_cq](#) (spinlock\_t ioctl\_lock, void \*cq\_id\_u\_ptr)  
*Initialize the struct for a new completion queue.*
- int [append\\_to\\_cq](#) (spinlock\_t ioctl\_lock, [ums\\_cq\\_param\\_t](#) \*args)  
*Append worker threads to a completion queue.*
- int [dequeue\\_cq](#) (spinlock\_t ioctl\_lock, [ums\\_cq\\_param\\_t](#) \*ret\_cq)  
*return the first COMPLETION\_QUEUE\_BUFF size of worker thread id in the completion queue*
- int [init\\_ums\\_scheduler](#) (spinlock\_t ioctl\_lock, [ums\\_km\\_param\\_t](#) \*args)  
*convert a standard pthread into an UmsSchedulerThread*
- int [end\\_ums\\_scheduler](#) (spinlock\_t ioctl\_lock)  
*clear the data structure used by the ums*
- int [init\\_worker\\_thread](#) (spinlock\_t ioctl\_lock)  
*block at startup the [worker\\_thread](#) to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS*
- int [end\\_worker\\_thread](#) (spinlock\_t ioctl\_lock)  
*if SWITCH\_PT\_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task\_struct. Also remove the worker\_thread\_struct from the completion queue and the hashtable of the worker thread. if SWITCH\_PT\_REGS is not defined wake up the UMS*
- [worker\\_thread\\_t](#) \* [Get\\_WKT](#) (int wkt\_pid)  
*retrive a pointer to worker\_thread\_t struct*
- [ums\\_scheduler\\_t](#) \* [Get\\_UMS](#) (int ums\_pid)  
*retrive a pointer to ums\_scheduler\_t struct*
- [ums\\_scheduler\\_t](#) \* [Get\\_UMS\\_from\\_WKT](#) (int wkt\_pid)  
*retrive a pointer to ums\_scheduler\_t struct*
- char \* [Get\\_UMS\\_Info](#) (int ums\_pid)  
*retrive a char \* with the info of the UMS in a human readable fashion*
- char \* [Get\\_WKT\\_Info](#) (int wkt\_pid)  
*retrive a char \* with the info of the worker in a human readable fashion*
- void [ums\\_do\\_wait](#) ([worker\\_thread\\_t](#) \*from\_wkt, [ums\\_scheduler\\_t](#) \*to\_ums)  
*handler used in the case that a worker thread is put on wait*
- void [ums\\_do\\_unwait](#) ([worker\\_thread\\_t](#) \*from\_wkt, struct task\_struct \*p)  
*handler used in the case that a worker thread previously put on wait is awakened*
- int [try\\_build\\_ums\\_core](#) (void)  
*Initialize the data structure needed for the UMS.*
- void [clear\\_ums\\_core](#) (void)  
*Destroy the ums data structure.*

### 4.13.1 Detailed Description

This file is the header of [ums.c](#).

Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

## 4.13.2 Function Documentation

### 4.13.2.1 `append_to_cq()`

```
int append_to_cq (
    spinlock_t ioctl_lock,
    ums_cq_param_t * args )
```

Append worker threads to a completion queue.

It cycle over the cpmpletion queue buffer. The buffer is initialized to -1 so if reach -1 before that cycle over all the buffer means that it has appended all the worker

#### Parameters

|                   |   |
|-------------------|---|
| <i>ioctl_lock</i> |   |
| <i>args</i>       | pointer to the ums_cq_param_t struct placed in the user space |

#### Returns

int

### 4.13.2.2 `dequeue_cq()`

```
int dequeue_cq (
    spinlock_t ioctl_lock,
    ums_cq_param_t * ret_cq )
```

return the first COMPLETION\_QUEUE\_BUFF size of worker thread id in the completion queue

#### Parameters

|                   |   |
|-------------------|---|
| <i>ioctl_lock</i> |   |
| <i>ret_cq</i>     | pointer to the ums_cq_param_t in the user space used to return the worker thread pid in the cq that are ready |

#### Returns

int

### 4.13.2.3 `end_ums_scheduler()`

```
int end_ums_scheduler (
    spinlock_t ioctl_lock )
```

clear the data structure used by the ums

#### Parameters

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
|-------------------|--|

#### Returns

int

#### 4.13.2.4 end\_worker\_thread()

```
int end_worker_thread (
    spinlock_t ioctl_lock )
```

if SWITCH\_PT\_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task\_struct. Also remove the worker\_thread\_struct from the completion queue and the hashtable of the worker thread. if SWITCH\_PT\_REGS is not defined wake up the UMS

### 4.13.3 Implementation

Check done by the function:

- The current pid needs to be a UMS pid if SWITCH\_PT\_REGS is defined else the current pid represent the worker scheduled
- if SWITCH\_PT\_REGS is defined, try to retrieve the worker\_thread\_t struct else try to retrieve the UMS using the worker pid

After this check perform:

- Restore context of the worker into his own task\_struct if SWITCH\_PT\_REGS is defined
- Restore ums context if SWITCH\_PT\_REGS is defined
- Remove the worker thread from the completion queue and from the hashtable and free the memory of the worker\_thread\_t struct
- Finally resume the worker thread
- If SWITCH\_PT\_REGS is not defined wake up also the UMS

#### Parameters

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
|-------------------|--|

**Returns**

int 0 if all went ok, else -1

**4.13.3.1 execute\_wkt()**

```
int execute_wkt (
    spinlock_t ioctl_lock,
    unsigned * u_wkt_pid )
```

perform a context switch from ums to wkt selected

**4.13.4 Implementation**

Initially the current check are done:

- Try to copy from user the value passed to the module by ioctl
- The current need to be an ums pid because only an UMS can schedule a worker thread.
- I try to retrieve the worker\_thread\_t from the cq\_list using the wkt\_pid\_to\_switch passed by the user arg
- If the state of the worker selected is W\_RUNNING the function end because the thread is scheduled by another UMS

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH\_PT\_REGS is defined, by saving the current state (pt\_↔ regs and fxregs) into the ums\_scheduler\_t struct and restoring the previously saved state of the worker thread from the worker\_thread\_t struct. if SWITCH\_PT\_REGS is not defined, by stop the UMS and waking up the worker thread.

**Parameters**

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
| <i>u_wkt_pid</i>  | pointer to the user space mem of the id of the worker thread choose to be executed |

**Returns**

int 0 if all went ok, else -1

#### 4.13.4.1 Get\_UMS()

```
ums_scheduler_t * Get_UMS (
    int ums_pid )
```

retrieves a pointer to ums\_scheduler\_t struct

##### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>ums_pid</i> | pid of the ums thread to find |
|----------------|-------------------------------|

##### Returns

ums\_scheduler\_t\*

#### 4.13.4.2 Get\_UMS\_from\_WKT()

```
ums_scheduler_t * Get_UMS_from_WKT (
    int wkt_pid )
```

retrieves a pointer to ums\_scheduler\_t struct

##### Parameters

|                |   |
|----------------|---|
| <i>wkt_pid</i> | pid of the worker actually scheduled by the ums |
|----------------|---|

##### Returns

ums\_scheduler\_t\*

#### 4.13.4.3 Get\_UMS\_Info()

```
char * Get_UMS_Info (
    int ums_pid )
```

retrieves a char \* with the info of the UMS in a human readable fashion

##### Parameters

|                |                |
|----------------|----------------|
| <i>ums_pid</i> | pid of the ums |
|----------------|----------------|

##### Returns

char\*

#### 4.13.4.4 Get\_WKT()

```
worker_thread_t * Get_WKT (
    int wkt_pid )
```

retrive a pointer to worker\_thread\_t struct

##### Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>wkt_pid</i> | pid of the worker thread to find |
|----------------|----------------------------------|

##### Returns

worker\_thread\_t\*

#### 4.13.4.5 Get\_WKT\_Info()

```
char * Get_WKT_Info (
    int wkt_pid )
```

retrive a char \* with the info of the worker in a human readable fashion

##### Parameters

|                |                          |
|----------------|--------------------------|
| <i>wkt_pid</i> | oid of the worker thread |
|----------------|--------------------------|

##### Returns

char\*

#### 4.13.4.6 init\_cq()

```
int init_cq (
    spinlock_t ioctl_lock,
    void * cq_id_u_ptr )
```

Initialize the struct for a new completion queue.

### 4.13.5 Implementation

- Find id for a new completion queue from the bitmap and set the bit.
- Create a new completion\_queue\_descriptor\_t and initialize it
- Try to return the completion queue id

## Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>ioctl_lock</i>  |                                    |
| <i>cq_id_u_ptr</i> | pointer to the cq_id in user space |

## Returns

int 0 if all went ok, else -1

#### 4.13.5.1 init\_ums\_scheduler()

```
int init_ums_scheduler (
    spinlock_t ioctl_lock,
    ums_km_param_t * args )
```

convert a standard pthread into an UmsSchedulerThread

converts a standard pthread in a UMS Scheduler thread, the function takes as input a completion list of worker threads and a entry point function

#### 4.13.6 Implementation

Initially the current check are done:

- Try to copy from user the value passed to the module by ioctl
- Check if the completion queue id is a valid id

After this checks the function populate the structs in the kernel:

- Create a new ums\_scheduler\_t struct for the new UMS created
- Initialize the struct with default value
- Link the completion queue to this UMS and update the used\_by\_couter entry in the completion\_queue\_↔ descriptor\_t descriptor cause the completion queue can be shared among multiple UMS

## Parameters

|                   |  |
|-------------------|--|
| <i>ioctl_lock</i> |  |
| <i>args</i>       | pointer to the ums_km_param_t struct in user space |

## Returns

int 0 if all went ok, else -1

#### 4.13.6.1 init\_worker\_thread()

```
int init_worker_thread (
    spinlock_t ioctl_lock )
```

block at startup the [worker\\_thread](#) to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS

#### 4.13.7 Implementation

- Create a new `worker_thread_t` struct and initialize it by save the current pid of the real worker thread into its own struct, and save the pointer to the worker task\_struct into `task_struct`. And set the worker state to `W_READY`.
- If `SWITCH_PT_REGS` is defined, save the worker current state into the `worker_thread_t` struct
- Put on wait worker thread

##### Parameters

|                         |  |
|-------------------------|--|
| <code>ioctl_lock</code> |  |
|-------------------------|--|

##### Returns

int 0 if all went ok, else -1

#### 4.13.7.1 try\_build\_ums\_core()

```
int try_build_ums_core (
    void )
```

Initialize the data structure needed for the UMS.

It initialize the following hashtables:

- `master_wkt_hashlist` (key:pid , data:worker\_thread\_t)
- `master_cq_hashlist` (key:id , data:completion\_queue\_descriptor\_t)
- `master_ums_hashlist` (key:pid , data:ums\_scheduler\_t)
- `cq_id_bitmap` (is used to check the avaiable id for the completion queues)

##### Returns

int

#### 4.13.7.2 ums\_do\_unwait()

```
void ums_do_unwait (
    worker_thread_t * from_wkt,
    struct task_struct * p )
```

handler used in the case that a worker thread previously put on wait is awakened



## Parameters

|                 |  |
|-----------------|--|
| <i>from_wkt</i> | pointer to worker_thread_t struct of the worker awakened |
| <i>p</i>        | task_struct of the worker thread awakened                |

**4.13.7.3 ums\_do\_wait()**

```
void ums_do_wait (
    worker_thread_t * from_wkt,
    ums_scheduler_t * to_ums )
```

handler used in the case that a worker thread is put on wait

## Parameters

|                 |  |
|-----------------|--|
| <i>from_wkt</i> | pointer to worker_thread_t struct of the worker thread that will put in wait |
| <i>to_ums</i>   | pointer to ums_scheduler_t struct if the ums thread that will wakeup         |

**4.13.7.4 yield\_to\_ums()**

```
int yield_to_ums (
    spinlock_t ioctl_lock )
```

perform a context switch from wkt to ums that host it

**4.13.8 Implementation**

Initially the current check are done:

- Try to retrieve the UMS that currently host the execution of the worker thread that has required the yield.
- Try to retrieve the worker thread struct scheduled saved inside the UMS struct.

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH\_PT\_REGS is defined, by saving the current state (pt\_regs and fxregs) into the worker\_thread\_t struct and restoring the previously saved state of the ums from the ums\_scheduler\_t struct. if SWITCH\_PT\_REGS is not defined, by stop the worker thread and waking up the UMS scheduler that previously has scheduled the worker.

## Parameters

|                         |  |
|-------------------------|--|
| <code>ioctl_lock</code> |  |
|-------------------------|--|

## Returns

int 0 if all went ok, else -1

## 4.14 ums.h

[Go to the documentation of this file.](#)

```

1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
25 #ifndef __UMS_HEADER
26 #define __UMS_HEADER
27
28 #include "shared.h"
29 #include "utility.h"
30 #include "proc.h"
31
32 #include <asm/fpu/internal.h>
33 #include <asm/fpu/types.h>
34 #include <asm/ptrace.h>
35 #include <asm/uaccess.h>
36 #include <linux/init.h>
37 #include <linux/module.h>
38 #include <linux/list.h>
39 #include <linux/hashtable.h>
40 #include <linux/sched.h>
41 #include <linux/ktime.h>
42 #include <linux/timekeeping.h>
43 #include <linux/bitmap.h>
44
45
46 #define MODULE_UMS_LOG "UMS_MASTER_FUNCS: "
47
48 #define F_DEQUEUE "DEQUEUE: "
49 #define F_APPEND "APPEND: "
50 #define F_DESTROY_CQ "DESTROY_CQ: "
51 #define F_INIT_CQ "INIT_CQ: "
52 #define F_INIT_WORKER "INIT_WORKER: "
53 #define F_END_WORKER "END_WORKER: "
54 #define F_INIT_UMS "INIT_UMS: "
55 #define F_SCHED_WRK "SCHED_WKT: "
56 #define F_SCHED_UMS "SCHED_UMS: "
57
58 #define HASH_KEY_SIZE 10
59 #define BITMAP_CQ_SIZE 128
60
61 #define STATE_RUNNING 0x0000
62 #define STATE_IDLE 0x0001
63 #define STATE_READY 0x0002
64 #define STATE_WAITING 0x0004
65
66 #define ums_is_idle(ums) ((ums->state & STATE_IDLE) != 0)
67 #define ums_is_running(ums) ((ums->state & STATE_RUNNING) != 0)
68 #define wkt_is_running(wkt) ((wkt->state & STATE_RUNNING) != 0)
69 #define wkt_is_ready(wkt) ((wkt->state & STATE_READY) != 0)
70 #define wkt_is_waiting(wkt) ((wkt->state & STATE_WAITING) != 0)
71
72 #define fxsave(fpu) copy_fxregs_to_kernel(fpu)

```

```

73 #define fxrestore(fpu)      copy_kernel_to_fxregs(&((fpu)->state.fxsave))
74
75
76 #define UMS_CORE_DEBUG
77
78 #define SWITCH_PT_REGS
79
80 typedef struct ums_scheduler ums_scheduler_t;
81 typedef struct worker_thread worker_thread_t;
82
83
84
88 typedef enum direction{
89     YIELD,
90     EXECUTE,
91     WAIT
92 } direction_t;
93
94
95
102 typedef struct completion_queue_descriptor{
103     struct list_head completion_queue;
104     int id;
105     unsigned int used_by_couter;
106     struct hlist_node hlist;
107 } completion_queue_descriptor_t;
108
109
110
115 typedef struct ums_scheduler {
116     int pid;
117     struct task_struct * task_struct;
118     int owner_pid;
119     int ums_cq_id;
120     struct list_head * cq_list;
121     int pid_wkt_sched;
122     worker_thread_t * wkt_sched_struct;
123 #ifdef SWITCH_PT_REGS
124     struct pt_regs saved_pt_regs;
125     struct fpu saved_fpu_regs;
126 #endif
127     unsigned long total_switch;
128     unsigned long last_wkt_runtime;
129     long state;
130     struct list_head list;
131     struct hlist_node hlist;
132 } ums_scheduler_t;
133
138 typedef struct worker_thread {
139     int pid;
140     struct task_struct * task_struct;
141 #ifdef SWITCH_PT_REGS
142     struct pt_regs saved_pt_regs;
143     struct fpu saved_fpu_regs;
144 #endif
145     int scheduled_by;
146     unsigned long time_at_switch;
147     unsigned long total_switch;
148     unsigned long total_runtime;
149     long state;
150     struct list_head list;
151     struct hlist_node hlist;
152 } worker_thread_t;
153
154 /*
155  * IOCTL exposed handler
156  */
157 int yield_to_ums (spinlock_t ioclt_lock);
158 int execute_wkt (spinlock_t ioclt_lock , unsigned * u_wkt_pid);
159 int init_cq (spinlock_t ioclt_lock , void * cq_id_u_ptr);
160 int append_to_cq (spinlock_t ioclt_lock , ums_cq_param_t * args);
161 int dequeue_cq (spinlock_t ioclt_lock , ums_cq_param_t * ret_cq);
162 int init_ums_scheduler (spinlock_t ioclt_lock , ums_km_param_t * args);
163 int end_ums_scheduler (spinlock_t ioclt_lock);
164 int init_worker_thread (spinlock_t ioclt_lock);
165 int end_worker_thread (spinlock_t ioclt_lock);
166
167 /*
168  * Function exposed for proc.c
169  */
170 worker_thread_t * Get_WKT(int wkt_pid);
171 ums_scheduler_t * Get_UMS(int ums_pid);
172 ums_scheduler_t * Get_UMS_from_WKT(int wkt_pid);
173
174 char * Get_UMS_Info(int ums_pid);
175 char * Get_WKT_Info(int wkt_pid);
176
177 /*

```

```

178  * Function exposed for wait_trace.c
179  */
180 void ums_do_wait ( worker_thread_t * from_wkt, ums_scheduler_t * to_ums);
181 void ums_do_unwait( worker_thread_t * from_wkt, struct task_struct * p);
182
183 /*
184  * Constructor destructor of this part
185  */
186 int  try_build_ums_core(void);
187 void clear_ums_core(void);
188
189 #endif

```

## 4.15 utility.h File Reference

This file contains utility macro definition.

### Macros

- #define `add_new_item_to_list`(new\_item\_pointer, list\_head, member, item\_type)  
*allocate and append to tail a new item to a list*
- #define `add_new_item_to_hlist`(new\_item\_pointer, hashtable, node, item\_type, identifier)  
*allocate and append to tail a new item to a hlist*
- #define `get_hlist_item_by_id`(getted\_item\_pointer, hashtable, node, member\_identifier, identifier)  
*retrive the item from an hashtable by the key*
- #define `delete_completion_queue_descriptor`(cq\_desc\_to\_delete, node)  
*delete the completion queue descriptor from its hashtable and free it*
- #define `retrive_from_hlist`(item\_select, hashtable, node, member\_identifier, identifier)  
*retrive a node in an hlist by its identifier*
- #define `retrive_from_list`(item\_select, head, member, member\_identifier, identifier)  
*retrive a node in an hlist by its identifier*
- #define `ums_delete_list`(entry\_cursor, entry\_cursor\_safe, head, member)  
*delete all the item from a list*
- #define `ums_delete_hlist`(cursor, node\_ptr, bucket, hashtable, node)  
*delete all the item from an hlist*

### 4.15.1 Detailed Description

This file contains utility macro definition.

#### Author

Tiziano Colagrossi [tiziano.colagrossi@gmail.com](mailto:tiziano.colagrossi@gmail.com)

### 4.15.2 Macro Definition Documentation

#### 4.15.2.1 add\_new\_item\_to\_hlist

```
#define add_new_item_to_hlist(  
    new_item_pointer,  
    hashtable,  
    node,  
    item_type,  
    identifier )
```

**Value:**

```
new_item_pointer = (item_type *) kmalloc(sizeof(item_type), GFP_KERNEL);  
hash_add(hashtable, &(new_item_pointer->node), identifier) \
```

allocate and append to tail a new item to a hlist

**Parameters**

|                         |  |
|-------------------------|--|
| <i>new_item_pointer</i> | will be filled with the new item address.        |
| <i>hashtable</i>        | hashtable to add to                              |
| <i>node</i>             | the &struct hlist_node of the object to be added |
| <i>item_type</i>        | the item type of the new element                 |
| <i>identifier</i>       | the key of the object to be added                |

#### 4.15.2.2 add\_new\_item\_to\_list

```
#define add_new_item_to_list(  
    new_item_pointer,  
    list_head,  
    member,  
    item_type )
```

**Value:**

```
new_item_pointer = (item_type *) kmalloc(sizeof(item_type), GFP_KERNEL);  
list_add_tail(&(new_item_pointer->member), list_head) \
```

allocate and append to tail a new item to a list

**Parameters**

|                         |   |
|-------------------------|---|
| <i>new_item_pointer</i> | will be filled with the new item address.   |
| <i>list_head</i>        | list head to add it before                  |
| <i>member</i>           | the name of the list_head within the struct |
| <i>item_type</i>        | the item type of the new element            |

#### 4.15.2.3 delete\_completion\_queue\_descriptor

```
#define delete_completion_queue_descriptor(  
    cq_desc_to_delete,  
    node )
```

**Value:**

```
hash_del(&cq_desc_to_delete->node);
kfree(cq_desc_to_delete)
```

delete the completion queue descriptor from its hashtable and free it

**Parameters**

|                          |   |
|--------------------------|---|
| <i>cq_desc_to_delete</i> | will be filled with the retrived node.              |
| <i>node</i>              | the &struct hlist_node of the object to be retrived |

**4.15.2.4 get\_hlist\_item\_by\_id**

```
#define get_hlist_item_by_id(
    getted_item_pointer,
    hashtable,
    node,
    member_identifier,
    identifier )
```

**Value:**

```
hash_for_each_possible(hashtable, getted_item_pointer, node, identifier) {
    if (getted_item_pointer->member_identifier != identifier) continue;
    break;
}
```

retrive the item from an hashtable by the key

**Parameters**

|                            |   |
|----------------------------|---|
| <i>getted_item_pointer</i> | will be filled with the retrived node.              |
| <i>hashtable</i>           | hashtable where to search                           |
| <i>node</i>                | the &struct hlist_node of the object to be retrived |
| <i>member_identifier</i>   | the name of the identifier within the struct        |
| <i>identifier</i>          | the key of the object to find                       |

**4.15.2.5 retrive\_from\_hlist**

```
#define retrive_from_hlist(
    item_select,
    hashtable,
    node,
    member_identifier,
    identifier )
```

**Value:**

```
hash_for_each_possible(hashtable, item_select, node, identifier) {
    if (item_select->member_identifier != identifier) continue;
    break;
}
```

retrive a node in an hlist by its identifier

## Parameters

|                          |   |
|--------------------------|---|
| <i>hashtable</i>         | hashtable where to search                           |
| <i>item_select</i>       | will be filled with the retrived node               |
| <i>node</i>              | the &struct hlist_node of the object to be retrived |
| <i>member_identifier</i> | the name of the identifier within the struct        |
| <i>identifier</i>        | the key of the object to find                       |

## 4.15.2.6 retrieve\_from\_list

```
#define retrieve_from_list(  
    item_select,  
    head,  
    member,  
    member_identifier,  
    identifier )
```

## Value:

```
list_for_each_entry(item_select, head, member) {  
    if (item_select->member_identifier != identifier) continue;  
    break;  
}
```

retrieve a node in an hlist by its identifier

## Parameters

|                          |  |
|--------------------------|--|
| <i>item_select</i>       | will be filled with the retrived node              |
| <i>head</i>              | the head of the list                               |
| <i>member</i>            | the &struct list_head of the object to be retrived |
| <i>member_identifier</i> | the name of the identifier within the struct       |
| <i>identifier</i>        | the key of the object to find                      |

## 4.15.2.7 ums\_delete\_hlist

```
#define ums_delete_hlist(  
    cursor,  
    node_ptr,  
    bucket,  
    hashtable,  
    node )
```

## Value:

```
hash_for_each_safe(hashtable, bucket, node_ptr, cursor, node){  
    hlist_del(&cursor->node);  
    kfree(cursor);  
}
```

delete all the item from an hlist

## Parameters

|                  |   |
|------------------|---|
| <i>cursor</i>    | the type * to use as a loop cursor.                 |
| <i>bucket</i>    | integer to use as bucket loop cursor                |
| <i>hashtable</i> | hashtable where to search                           |
| <i>node</i>      | the &struct hlist_node of the object to be retrived |

## 4.15.2.8 ums\_delete\_list

```
#define ums_delete_list(
    entry_cursor,
    entry_cursor_safe,
    head,
    member )
```

## Value:

```
list_for_each_entry_safe(entry_cursor, entry_cursor_safe, head, member ){
    \
    list_del(&entry_cursor->member);
    kfree(entry_cursor);
}
```

delete all the item from a list

## Parameters

|                          |  |
|--------------------------|--|
| <i>entry_cursor</i>      | the type * to use as a loop cursor           |
| <i>entry_cursor_safe</i> | another type * to use as temporary storage   |
| <i>head</i>              | the head for your list.                      |
| <i>member</i>            | the name of the list_head within the struct. |

## 4.16 utility.h

[Go to the documentation of this file.](#)

```
1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10 * WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17
26 #ifndef __UTILITY_HEADER
27 #define __UTILITY_HEADER
28
37 #define add_new_item_to_list(new_item_pointer, list_head, member, item_type)
38     new_item_pointer = (item_type *) kmalloc(sizeof(item_type), GFP_KERNEL);
```



```

39     list_add_tail(&(new_item_pointer->member), list_head)
40
50 #define add_new_item_to_hlist(new_item_pointer, hashtable , node, item_type, identifier)
51     \
52     new_item_pointer = (item_type *) kmalloc(sizeof(item_type), GFP_KERNEL);
53     hash_add(hashtable, &(new_item_pointer->node), identifier)
54
63 #define get_hlist_item_by_id(getted_item_pointer, hashtable , node, member_identifier, identifier)
64     \
65     hash_for_each_possible(hashtable, getted_item_pointer, node, identifier) {
66         if (getted_item_pointer->member_identifier != identifier) continue;
67         break;
68     }
69
75 #define delete_completion_queue_descriptor(cq_desc_to_delete, node)
76     \
77     hash_del(&cq_desc_to_delete->node);
78     kfree(cq_desc_to_delete)
79
88 #define retrieve_from_hlist(item_select, hashtable, node, member_identifier, identifier)
89     \
90     hash_for_each_possible(hashtable, item_select, node, identifier) {
91         if (item_select->member_identifier != identifier) continue;
92         break;
93     }
94
103 #define retrieve_from_list(item_select, head, member, member_identifier, identifier)
104     \
105     list_for_each_entry(item_select, head, member) {
106         if (item_select->member_identifier != identifier) continue;
107         break;
108     }
109
118 #define ums_delete_list(entry_cursor, entry_cursor_safe, head, member)
119     \
120     list_for_each_entry_safe(entry_cursor, entry_cursor_safe, head, member ){
121         list_del(&entry_cursor->member);
122         kfree(entry_cursor);
123     }
124
132 #define ums_delete_hlist(cursor, node_ptr, bucket, hashtable, node)
133     \
134     hash_for_each_safe(hashtable, bucket, node_ptr, cursor, node){
135         hlist_del(&cursor->node);
136         kfree(cursor);
137     }
138 #endif

```



# Index

- add\_new\_item\_to\_hlist
  - utility.h, [60](#)
- add\_new\_item\_to\_list
  - utility.h, [61](#)
- append\_to\_cq
  - ums.c, [37](#)
  - ums.h, [50](#)
- clear\_ums\_proc
  - proc.c, [24](#)
  - proc.h, [29](#)
- cmds
  - device.c, [20](#)
- completion\_queue
  - completion\_queue\_descriptor, [5](#)
- completion\_queue\_descriptor, [5](#)
  - completion\_queue, [5](#)
  - hlist, [5](#)
  - id, [6](#)
  - used\_by\_couter, [6](#)
- completion\_queue\_id
  - ums\_cq\_param, [9](#)
- cq\_id
  - cq\_proc, [6](#)
  - ums\_km\_param, [10](#)
  - ums\_proc, [11](#)
- cq\_list
  - ums\_scheduler, [12](#)
- cq\_proc, [6](#)
  - cq\_id, [6](#)
  - dir\_entry, [7](#)
  - hlist, [7](#)
  - link\_to, [7](#)
  - name\_dir, [7](#)
  - path, [7](#)
- DECLARE\_BITMAP
  - ums.c, [37](#)
- DEFINE\_HASHTABLE
  - ums.c, [37](#), [38](#)
- delete\_completion\_queue\_descriptor
  - utility.h, [61](#)
- dequeue\_cq
  - ums.c, [38](#)
  - ums.h, [50](#)
- device.c, [19](#)
  - cmds, [20](#)
  - try\_start\_device, [19](#)
- device.h, [20](#)
  - try\_start\_device, [21](#)
- dir\_entry
  - cq\_proc, [7](#)
  - owner\_proc, [8](#)
  - ums\_proc, [11](#)
  - worker\_proc, [15](#)
- end\_ums\_scheduler
  - ums.c, [38](#)
  - ums.h, [50](#)
- end\_worker\_thread
  - ums.c, [39](#)
  - ums.h, [51](#)
- execute\_wkt
  - ums.c, [39](#)
  - ums.h, [52](#)
- get\_hlist\_item\_by\_id
  - utility.h, [62](#)
- Get\_UMS
  - ums.c, [40](#)
  - ums.h, [52](#)
- Get\_UMS\_from\_WKT
  - ums.c, [41](#)
  - ums.h, [53](#)
- Get\_UMS\_Info
  - ums.c, [41](#)
  - ums.h, [53](#)
- Get\_WKT
  - ums.c, [41](#)
  - ums.h, [54](#)
- Get\_WKT\_Info
  - ums.c, [43](#)
  - ums.h, [54](#)
- hlist
  - completion\_queue\_descriptor, [5](#)
  - cq\_proc, [7](#)
  - owner\_proc, [8](#)
  - ums\_proc, [11](#)
  - ums\_scheduler, [13](#)
  - worker\_proc, [15](#)
  - worker\_thread, [16](#)
- id
  - completion\_queue\_descriptor, [6](#)
- info\_entry
  - ums\_proc, [11](#)
  - worker\_proc, [15](#)
- init\_cq
  - ums.c, [43](#)

- ums.h, 54
- init\_ums\_scheduler
  - ums.c, 44
  - ums.h, 55
- init\_worker\_thread
  - ums.c, 44
  - ums.h, 55
- last\_wkt\_runtime
  - ums\_scheduler, 13
- link\_to
  - cq\_proc, 7
  - ums\_proc, 11
  - worker\_proc, 15
- list
  - ums\_scheduler, 13
  - worker\_thread, 17
- module.c, 22
- module.h, 22
- name\_dir
  - cq\_proc, 7
  - owner\_proc, 8
  - ums\_proc, 11
  - worker\_proc, 15
- owner\_pid
  - ums\_km\_param, 10
  - ums\_scheduler, 13
- owner\_proc, 7
  - dir\_entry, 8
  - hlist, 8
  - name\_dir, 8
  - pid, 8
  - sched\_entry, 8
- path
  - cq\_proc, 7
  - ums\_proc, 11
  - worker\_proc, 15
- pid
  - owner\_proc, 8
  - ums\_proc, 11
  - ums\_scheduler, 13
  - worker\_proc, 16
  - worker\_thread, 17
- pid\_owner
  - ums\_proc, 12
- pid\_wkt\_sched
  - ums\_scheduler, 13
- pids
  - ums\_cq\_param, 9
- proc.c, 23
  - clear\_ums\_proc, 24
  - Proc\_Update\_Cq\_Created, 24
  - Proc\_Update\_Cq\_Deleted, 25
  - Proc\_Update\_Ums\_Created, 25
  - Proc\_Update\_Ums\_Ended, 25
- Proc\_Update\_Worker\_Appended, 26
- Proc\_Update\_Worker\_Created, 26
- Proc\_Update\_Worker\_Ended, 27
- try\_build\_ums\_proc, 27
- proc.h, 27
  - clear\_ums\_proc, 29
  - Proc\_Update\_Cq\_Created, 29
  - Proc\_Update\_Cq\_Deleted, 30
  - Proc\_Update\_Ums\_Created, 30
  - Proc\_Update\_Ums\_Ended, 30
  - Proc\_Update\_Worker\_Appended, 31
  - Proc\_Update\_Worker\_Created, 31
  - Proc\_Update\_Worker\_Ended, 32
  - try\_build\_ums\_proc, 32
- Proc\_Update\_Cq\_Created
  - proc.c, 24
  - proc.h, 29
- Proc\_Update\_Cq\_Deleted
  - proc.c, 25
  - proc.h, 30
- Proc\_Update\_Ums\_Created
  - proc.c, 25
  - proc.h, 30
- Proc\_Update\_Ums\_Ended
  - proc.c, 25
  - proc.h, 30
- Proc\_Update\_Worker\_Appended
  - proc.c, 26
  - proc.h, 31
- Proc\_Update\_Worker\_Created
  - proc.c, 26
  - proc.h, 31
- Proc\_Update\_Worker\_Ended
  - proc.c, 27
  - proc.h, 32
- retrive\_from\_hlist
  - utility.h, 62
- retrive\_from\_list
  - utility.h, 63
- saved\_fpu\_regs
  - ums\_scheduler, 13
  - worker\_thread, 17
- saved\_pt\_regs
  - ums\_scheduler, 13
  - worker\_thread, 17
- sched\_entry
  - owner\_proc, 8
- scheduled\_by
  - worker\_thread, 17
- shared.h, 34
- state
  - ums.c, 47
  - ums\_scheduler, 14
  - worker\_thread, 17
- task\_struct
  - ums\_scheduler, 14

- worker\_thread, 17
- time\_at\_switch
  - worker\_thread, 17
- total\_runtime
  - worker\_thread, 18
- total\_switch
  - ums\_scheduler, 14
  - worker\_thread, 18
- try\_build\_ums\_core
  - ums.c, 45
  - ums.h, 56
- try\_build\_ums\_proc
  - proc.c, 27
  - proc.h, 32
- try\_start\_device
  - device.c, 19
  - device.h, 21
- ums.c, 35
  - append\_to\_cq, 37
  - DECLARE\_BITMAP, 37
  - DEFINE\_HASHTABLE, 37, 38
  - dequeue\_cq, 38
  - end\_ums\_scheduler, 38
  - end\_worker\_thread, 39
  - execute\_wkt, 39
  - Get\_UMS, 40
  - Get\_UMS\_from\_WKT, 41
  - Get\_UMS\_Info, 41
  - Get\_WKT, 41
  - Get\_WKT\_Info, 43
  - init\_cq, 43
  - init\_ums\_scheduler, 44
  - init\_worker\_thread, 44
  - state, 47
  - try\_build\_ums\_core, 45
  - ums\_do\_unwait, 45
  - ums\_do\_wait, 46
  - yield\_to\_ums, 46
- ums.h, 47
  - append\_to\_cq, 50
  - dequeue\_cq, 50
  - end\_ums\_scheduler, 50
  - end\_worker\_thread, 51
  - execute\_wkt, 52
  - Get\_UMS, 52
  - Get\_UMS\_from\_WKT, 53
  - Get\_UMS\_Info, 53
  - Get\_WKT, 54
  - Get\_WKT\_Info, 54
  - init\_cq, 54
  - init\_ums\_scheduler, 55
  - init\_worker\_thread, 55
  - try\_build\_ums\_core, 56
  - ums\_do\_unwait, 56
  - ums\_do\_wait, 57
  - yield\_to\_ums, 57
- ums\_cq\_id
  - ums\_scheduler, 14
- ums\_cq\_param, 9
  - completion\_queue\_id, 9
  - pids, 9
- ums\_delete\_hlist
  - utility.h, 63
- ums\_delete\_list
  - utility.h, 64
- ums\_do\_unwait
  - ums.c, 45
  - ums.h, 56
- ums\_do\_wait
  - ums.c, 46
  - ums.h, 57
- ums\_km\_param, 9
  - cq\_id, 10
  - owner\_pid, 10
- ums\_proc, 10
  - cq\_id, 11
  - dir\_entry, 11
  - hlist, 11
  - info\_entry, 11
  - link\_to, 11
  - name\_dir, 11
  - path, 11
  - pid, 11
  - pid\_owner, 12
- ums\_scheduler, 12
  - cq\_list, 12
  - hlist, 13
  - last\_wkt\_runtime, 13
  - list, 13
  - owner\_pid, 13
  - pid, 13
  - pid\_wkt\_sched, 13
  - saved\_fpu\_regs, 13
  - saved\_pt\_regs, 13
  - state, 14
  - task\_struct, 14
  - total\_switch, 14
  - ums\_cq\_id, 14
  - wkt\_sched\_struct, 14
- used\_by\_couter
  - completion\_queue\_descriptor, 6
- utility.h, 60
  - add\_new\_item\_to\_hlist, 60
  - add\_new\_item\_to\_list, 61
  - delete\_completion\_queue\_descriptor, 61
  - get\_hlist\_item\_by\_id, 62
  - retrive\_from\_hlist, 62
  - retrive\_from\_list, 63
  - ums\_delete\_hlist, 63
  - ums\_delete\_list, 64
- wkt\_sched\_struct
  - ums\_scheduler, 14
- worker\_proc, 14
  - dir\_entry, 15
  - hlist, 15
  - info\_entry, 15

- link\_to, [15](#)
- name\_dir, [15](#)
- path, [15](#)
- pid, [16](#)
- worker\_thread, [16](#)
  - hlist, [16](#)
  - list, [17](#)
  - pid, [17](#)
  - saved\_fpu\_regs, [17](#)
  - saved\_pt\_regs, [17](#)
  - scheduled\_by, [17](#)
  - state, [17](#)
  - task\_struct, [17](#)
  - time\_at\_switch, [17](#)
  - total\_runtime, [18](#)
  - total\_switch, [18](#)
- yield\_to\_ums
  - ums.c, [46](#)
  - ums.h, [57](#)