

## UMS module

Generated by Doxygen 1.9.2



<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 add_wt_params Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 completion_list_id	5
3.1.2.2 worker_thread_id	5
3.2 cl_list Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Field Documentation	6
3.2.2.1 cl_count	6
3.3 completion_list Struct Reference	6
3.3.1 Detailed Description	7
3.3.2 Field Documentation	7
3.3.2.1 id	7
3.3.2.2 list	7
3.3.2.3 worker_thread_count	7
3.3.2.4 wt_list	7
3.4 process Struct Reference	7
3.4.1 Detailed Description	8
3.4.2 Field Documentation	8
3.4.2.1 cl_list	8
3.4.2.2 pid	8
3.4.2.3 ums_thread_list	8
3.4.2.4 worker_thread_list	9
3.5 process_entry Struct Reference	9
3.5.1 Detailed Description	9
3.5.2 Field Documentation	9
3.5.2.1 entry	9
3.5.2.2 pid	9
3.5.2.3 schedulers_entry	10
3.6 process_entry_list Struct Reference	10
3.6.1 Detailed Description	10
3.6.2 Field Documentation	10
3.6.2.1 process_entry_count	10
3.7 process_list Struct Reference	10
3.7.1 Detailed Description	11
3.7.2 Field Documentation	11

3.7.2.1 process_count	11
3.8 ums_thread_context Struct Reference	11
3.8.1 Detailed Description	12
3.8.2 Field Documentation	12
3.8.2.1 avg_switching_time	12
3.8.2.2 cl_id	12
3.8.2.3 created_by	12
3.8.2.4 entry_point	12
3.8.2.5 fpu_regs	12
3.8.2.6 id	12
3.8.2.7 last_switch_time	13
3.8.2.8 regs	13
3.8.2.9 ret_regs	13
3.8.2.10 run_by	13
3.8.2.11 state	13
3.8.2.12 switch_count	13
3.8.2.13 switching_time	13
3.8.2.14 wt_id	14
3.9 ums_thread_entry Struct Reference	14
3.9.1 Detailed Description	14
3.9.2 Field Documentation	14
3.9.2.1 created_by	14
3.9.2.2 entry	14
3.9.2.3 id	15
3.9.2.4 info_entry	15
3.9.2.5 workers_entry	15
3.10 ums_thread_entry_list Struct Reference	15
3.10.1 Detailed Description	15
3.10.2 Field Documentation	15
3.10.2.1 ums_thread_entry_count	16
3.11 ums_thread_list Struct Reference	16
3.11.1 Detailed Description	16
3.11.2 Field Documentation	16
3.11.2.1 ums_thread_count	16
3.12 ums_thread_params Struct Reference	16
3.12.1 Detailed Description	17
3.12.2 Field Documentation	17
3.12.2.1 completion_list_id	17
3.12.2.2 function	17
3.13 worker_thread_context Struct Reference	17
3.13.1 Detailed Description	18
3.13.2 Field Documentation	18

3.13.2.1 cl_id . . . . .	18
3.13.2.2 created_by . . . . .	18
3.13.2.3 entry_point . . . . .	18
3.13.2.4 fpu_regs . . . . .	18
3.13.2.5 id . . . . .	18
3.13.2.6 last_switch_time . . . . .	18
3.13.2.7 list . . . . .	19
3.13.2.8 regs . . . . .	19
3.13.2.9 run_by . . . . .	19
3.13.2.10 running_time . . . . .	19
3.13.2.11 state . . . . .	19
3.13.2.12 switch_count . . . . .	19
3.13.2.13 wt_list . . . . .	19
3.14 worker_thread_entry Struct Reference . . . . .	20
3.14.1 Detailed Description . . . . .	20
3.14.2 Field Documentation . . . . .	20
3.14.2.1 entry . . . . .	20
3.14.2.2 id . . . . .	20
3.15 worker_thread_entry_list Struct Reference . . . . .	20
3.15.1 Detailed Description . . . . .	21
3.15.2 Field Documentation . . . . .	21
3.15.2.1 worker_thread_entry_count . . . . .	21
3.16 worker_thread_list Struct Reference . . . . .	21
3.16.1 Detailed Description . . . . .	21
3.16.2 Field Documentation . . . . .	21
3.16.2.1 worker_thread_count . . . . .	22
3.17 worker_thread_params Struct Reference . . . . .	22
3.17.1 Detailed Description . . . . .	22
3.17.2 Field Documentation . . . . .	22
3.17.2.1 function . . . . .	22
3.17.2.2 function_args . . . . .	22
3.17.2.3 stack_address . . . . .	22
3.17.2.4 stack_size . . . . .	22
<b>4 File Documentation</b> . . . . .	<b>23</b>
4.1 device.c File Reference . . . . .	23
4.1.1 Detailed Description . . . . .	23
4.1.2 Function Documentation . . . . .	24
4.1.2.1 init_device() . . . . .	24
4.2 device.h File Reference . . . . .	24
4.2.1 Detailed Description . . . . .	25
4.2.2 Function Documentation . . . . .	25

4.2.2.1 init_device()	25
4.3 device.h	25
4.4 device_shared.h File Reference	26
4.4.1 Detailed Description	27
4.4.2 Enumeration Type Documentation	27
4.4.2.1 yield_reason	27
4.5 device_shared.h	28
4.6 module.c File Reference	28
4.6.1 Detailed Description	29
4.7 module.h File Reference	29
4.7.1 Detailed Description	29
4.8 module.h	30
4.9 proc.c File Reference	30
4.9.1 Detailed Description	31
4.9.2 Function Documentation	31
4.9.2.1 create_process_entry()	31
4.9.2.2 create_umst_entry()	31
4.9.2.3 create_wt_entry()	32
4.9.2.4 exit_proc()	32
4.9.2.5 get_process_entry_with_pid()	32
4.9.2.6 get_ums_thread_entry_with_id()	33
4.9.2.7 init_proc()	33
4.9.3 Variable Documentation	33
4.9.3.1 process_entry_list	33
4.9.3.2 ums_thread_entry_list	34
4.9.3.3 worker_thread_entry_list	34
4.10 proc.h File Reference	34
4.10.1 Detailed Description	35
4.10.2 Typedef Documentation	36
4.10.2.1 process_entry_list_t	36
4.10.2.2 process_entry_t	36
4.10.2.3 ums_thread_entry_list_t	36
4.10.2.4 ums_thread_entry_t	36
4.10.2.5 worker_thread_entry_list_t	36
4.10.2.6 worker_thread_entry_t	37
4.10.3 Function Documentation	37
4.10.3.1 create_process_entry()	37
4.10.3.2 create_umst_entry()	37
4.10.3.3 create_wt_entry()	38
4.10.3.4 exit_proc()	38
4.10.3.5 get_process_entry_with_pid()	38
4.10.3.6 get_ums_thread_entry_with_id()	39

4.10.3.7 init_proc()	39
4.11 proc.h	39
4.12 ums.c File Reference	40
4.12.1 Detailed Description	42
4.12.2 Function Documentation	42
4.12.2.1 add_to_completion_list()	42
4.12.2.2 convert_from_ums_thread()	43
4.12.2.3 convert_to_ums_thread()	43
4.12.2.4 create_completion_list()	44
4.12.2.5 create_ums_thread()	44
4.12.2.6 create_worker_thread()	45
4.12.2.7 dequeue_completion_list_items()	46
4.12.2.8 exit_ums()	46
4.12.2.9 exit_ums_process()	46
4.12.2.10 free_process()	47
4.12.2.11 free_process_cl_list()	47
4.12.2.12 free_process_ums_thread_list()	47
4.12.2.13 free_process_worker_thread_list()	48
4.12.2.14 get_cl_with_id()	48
4.12.2.15 get_process_with_pid()	49
4.12.2.16 get_ready_wt_list()	49
4.12.2.17 get_umst_run_by_pid()	49
4.12.2.18 get_umst_switching_time()	50
4.12.2.19 get_umst_with_id()	50
4.12.2.20 get_wt_run_by_umst_id()	50
4.12.2.21 get_wt_running_time()	51
4.12.2.22 get_wt_with_id()	51
4.12.2.23 init_ums_process()	52
4.12.2.24 switch_back_to_ums_thread()	52
4.12.2.25 switch_to_worker_thread()	53
4.12.3 Variable Documentation	54
4.12.3.1 process_list	54
4.13 ums.h File Reference	54
4.13.1 Detailed Description	57
4.13.2 Typedef Documentation	57
4.13.2.1 cl_list_t	57
4.13.2.2 completion_list_t	57
4.13.2.3 process_list_t	57
4.13.2.4 process_t	58
4.13.2.5 ums_thread_context_t	58
4.13.2.6 ums_thread_list_t	58
4.13.2.7 worker_thread_context_t	58

4.13.2.8 worker_thread_list_t . . . . .	58
4.13.3 Enumeration Type Documentation . . . . .	58
4.13.3.1 ums_state . . . . .	58
4.13.3.2 worker_state . . . . .	59
4.13.4 Function Documentation . . . . .	59
4.13.4.1 add_to_completion_list() . . . . .	59
4.13.4.2 convert_from_ums_thread() . . . . .	60
4.13.4.3 convert_to_ums_thread() . . . . .	60
4.13.4.4 create_completion_list() . . . . .	61
4.13.4.5 create_ums_thread() . . . . .	61
4.13.4.6 create_worker_thread() . . . . .	62
4.13.4.7 dequeue_completion_list_items() . . . . .	63
4.13.4.8 exit_ums() . . . . .	63
4.13.4.9 exit_ums_process() . . . . .	63
4.13.4.10 free_process() . . . . .	64
4.13.4.11 free_process_cl_list() . . . . .	64
4.13.4.12 free_process_ums_thread_list() . . . . .	64
4.13.4.13 free_process_worker_thread_list() . . . . .	65
4.13.4.14 get_cl_with_id() . . . . .	65
4.13.4.15 get_process_with_pid() . . . . .	66
4.13.4.16 get_ready_wt_list() . . . . .	66
4.13.4.17 get_umst_run_by_pid() . . . . .	66
4.13.4.18 get_umst_switching_time() . . . . .	67
4.13.4.19 get_umst_with_id() . . . . .	67
4.13.4.20 get_wt_run_by_umst_id() . . . . .	67
4.13.4.21 get_wt_running_time() . . . . .	68
4.13.4.22 get_wt_with_id() . . . . .	68
4.13.4.23 init_ums_process() . . . . .	69
4.13.4.24 switch_back_to_ums_thread() . . . . .	69
4.13.4.25 switch_to_worker_thread() . . . . .	70
4.14 ums.h . . . . .	71



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">add_wt_params</a>	Parameters passed by the library and user for adding worker thread to completion list . . . . .	5
<a href="#">cl_list</a>	The list of completion lists . . . . .	6
<a href="#">completion_list</a>	The completion list of worker threads . . . . .	6
<a href="#">process</a>	The process that initialized/enabled UMS mechanism . . . . .	7
<a href="#">process_entry</a>	The structure for process entry /proc/ums/<PID> . . . . .	9
<a href="#">process_entry_list</a>	The list of process entries in /proc/ums . . . . .	10
<a href="#">process_list</a>	The list of processes that initialized/enabled UMS mechanism . . . . .	10
<a href="#">ums_thread_context</a>	The ums thread(scheduler) . . . . .	11
<a href="#">ums_thread_entry</a>	The structure for scheduler entry /proc/ums/<PID>/schedulers/<ID> . . . . .	14
<a href="#">ums_thread_entry_list</a>	The list of scheduler entries in /proc/ums/<PID>/schedulers . . . . .	15
<a href="#">ums_thread_list</a>	The list of ums threads(schedulers) . . . . .	16
<a href="#">ums_thread_params</a>	Parameters passed by the library and user for ums thread(scheduler) creation . . . . .	16
<a href="#">worker_thread_context</a>	The worker thread . . . . .	17
<a href="#">worker_thread_entry</a>	The structure for worker thread entry /proc/ums/<PID>/schedulers/<ID>/workers/<ID> . . . . .	20
<a href="#">worker_thread_entry_list</a>	The list of worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers . . . . .	20
<a href="#">worker_thread_list</a>	The list of worker threads . . . . .	21
<a href="#">worker_thread_params</a>	Parameters passed by the library and user for worker thread creation . . . . .	22



## Chapter 2

# File Index

### 2.1 File List

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

<a href="#">device.c</a>	This file contains the implementation of the functions of the char device part . . . . .	23
<a href="#">device.h</a>	This file is a header of the char device part of the module . . . . .	24
<a href="#">device_shared.h</a>	This file contains definitions of ioctl commands . . . . .	26
<a href="#">module.c</a>	This file contains the entry point of the module . . . . .	28
<a href="#">module.h</a>	This file is a header of the module entry point . . . . .	29
<a href="#">proc.c</a>	This file contains the implementation of the functions of the /proc part . . . . .	30
<a href="#">proc.h</a>	This file is a header of the /proc part of the module . . . . .	34
<a href="#">ums.c</a>	This file contains the implementation of all main functions of the module . . . . .	40
<a href="#">ums.h</a>	This file is a header of the main module functionality . . . . .	54



## Chapter 3

# Data Structure Documentation

### 3.1 add\_wt\_params Struct Reference

Parameters passed by the library and user for adding worker thread to completion list.

```
#include <device_shared.h>
```

#### Data Fields

- unsigned int [completion\\_list\\_id](#)
- unsigned int [worker\\_thread\\_id](#)

#### 3.1.1 Detailed Description

Parameters passed by the library and user for adding worker thread to completion list.

#### 3.1.2 Field Documentation

##### 3.1.2.1 completion\_list\_id

```
unsigned int add_wt_params::completion_list_id
```

The id of the completion list to which is added

##### 3.1.2.2 worker\_thread\_id

```
unsigned int add_wt_params::worker_thread_id
```

The id of the worker thread which is being added

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

- [device\\_shared.h](#)

## 3.2 cl\_list Struct Reference

The list of completion lists.

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

### Data Fields

- struct list\_head **list**
- unsigned int [cl\\_count](#)

### 3.2.1 Detailed Description

The list of completion lists.

The purpose of this list is to store all completion lists created by the process

### 3.2.2 Field Documentation

#### 3.2.2.1 cl\_count

```
unsigned int cl_list::cl_count
```

The number of elements(completion lists) in the list

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

- [ums.h](#)

## 3.3 completion\_list Struct Reference

The completion list of worker threads.

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

### Data Fields

- struct list\_head [list](#)
- struct list\_head [wt\\_list](#)
- unsigned int [id](#)
- unsigned int [worker\\_thread\\_count](#)

### 3.3.1 Detailed Description

The completion list of worker threads.

This is a node in the [process::cl\\_list](#). This is a description of the completion list.

### 3.3.2 Field Documentation

#### 3.3.2.1 id

```
unsigned int completion_list::id
```

Unique id of the completion list

#### 3.3.2.2 list

```
struct list_head completion_list::list
```

This list structure is related to the list of completion lists in the process

#### 3.3.2.3 worker\_thread\_count

```
unsigned int completion_list::worker_thread_count
```

The number of worker threads in this completion list

#### 3.3.2.4 wt\_list

```
struct list_head completion_list::wt_list
```

This list structure is related to the list of worker threads that it contains

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

- [ums.h](#)

## 3.4 process Struct Reference

The process that initialized/enabled UMS mechanism.

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

## Data Fields

- `pid_t` `pid`
- `cl_list_t` `cl_list`
- `worker_thread_list_t` `worker_thread_list`
- `ums_thread_list_t` `ums_thread_list`
- `struct list_head` `list`

### 3.4.1 Detailed Description

The process that initialized/enabled UMS mechanism.

This is a node in the `process_list`. This is a process that initialized/enabled UMS mechanism. Each such process has 3 lists:

- list of completion lists
- list of ums threads(schedulers)
- list of worker threads

### 3.4.2 Field Documentation

#### 3.4.2.1 `cl_list`

```
cl_list_t process::cl_list
```

A list of completion list created in this process environment

#### 3.4.2.2 `pid`

```
pid_t process::pid
```

The PID of the main thread, hence the TGID of every thread of the process

#### 3.4.2.3 `ums_thread_list`

```
ums_thread_list_t process::ums_thread_list
```

A list of ums thread(schedulers) created in this process environment



#### 3.4.2.4 worker\_thread\_list

```
worker_thread_list_t process::worker_thread_list
```

A list of worker thread created in this process environment

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

- [ums.h](#)

## 3.5 process\_entry Struct Reference

The structure for process entry /proc/ums/<PID>

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

### Data Fields

- pid\_t [pid](#)
- struct list\_head [list](#)
- struct proc\_dir\_entry \* [entry](#)
- struct proc\_dir\_entry \* [schedulers\\_entry](#)

### 3.5.1 Detailed Description

The structure for process entry /proc/ums/<PID>

This is a node in the [process\\_entry\\_list](#). This is a description of the process entry.

### 3.5.2 Field Documentation

#### 3.5.2.1 entry

```
struct proc_dir_entry* process_entry::entry
```

The entry of the process

#### 3.5.2.2 pid

```
pid_t process_entry::pid
```

The PID of the process

### 3.5.2.3 schedulers\_entry

```
struct proc_dir_entry* process_entry::schedulers_entry
```

The entry of schedulers, child of [process\\_entry::entry](#)

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

- [proc.h](#)

## 3.6 process\_entry\_list Struct Reference

The list of process entries in /proc/ums.

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

### Data Fields

- struct list\_head **list**
- unsigned int [process\\_entry\\_count](#)

### 3.6.1 Detailed Description

The list of process entries in /proc/ums.

The purpose of this list is to store all process entries in /proc/ums/<PID>

### 3.6.2 Field Documentation

#### 3.6.2.1 process\_entry\_count

```
unsigned int process_entry_list::process_entry_count
```

The number of process entries in the list

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

- [proc.h](#)

## 3.7 process\_list Struct Reference

The list of processes that initialized/enabled UMS mechanism.

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

## Data Fields

- struct list\_head **list**
- unsigned int [process\\_count](#)

### 3.7.1 Detailed Description

The list of processes that initialized/enabled UMS mechanism.

The purpose of this list is to store all processes that initialized/enabled UMS mechanism

### 3.7.2 Field Documentation

#### 3.7.2.1 process\_count

```
unsigned int process_list::process_count
```

The number of elements(processes) in the list

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

- [ums.h](#)

## 3.8 ums\_thread\_context Struct Reference

The ums thread(scheduler)

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

## Data Fields

- unsigned int [id](#)
- struct list\_head **list**
- unsigned long [entry\\_point](#)
- unsigned int [cl\\_id](#)
- unsigned int [wt\\_id](#)
- pid\_t [created\\_by](#)
- pid\_t [run\\_by](#)
- [ums\\_state\\_t](#) state
- unsigned int [switch\\_count](#)
- struct timespec64 [last\\_switch\\_time](#)
- unsigned long [switching\\_time](#)
- unsigned long [avg\\_switching\\_time](#)
- struct pt\_regs [regs](#)
- struct fpu [fpu\\_regs](#)
- struct pt\_regs [ret\\_regs](#)

### 3.8.1 Detailed Description

The ums thread(scheduler)

This is a node in the [process::ums\\_thread\\_list](#). This is a description of ums thread(scheduler).

### 3.8.2 Field Documentation

#### 3.8.2.1 avg\_switching\_time

```
unsigned long ums_thread_context::avg_switching_time
```

The average switching time of the ums thread(scheduler)

#### 3.8.2.2 cl\_id

```
unsigned int ums_thread_context::cl_id
```

The id of the completion list associated to the ums thread(scheduler)

#### 3.8.2.3 created\_by

```
pid_t ums_thread_context::created_by
```

The PID of the process that created the ums thread(scheduler)

#### 3.8.2.4 entry\_point

```
unsigned long ums_thread_context::entry_point
```

The starting function of the ums thread(scheduler), i.e. scheduling function

#### 3.8.2.5 fpu\_regs

```
struct fpu ums_thread_context::fpu_regs
```

The current snapshot of fpu registers

#### 3.8.2.6 id

```
unsigned int ums_thread_context::id
```

Unique id of the ums thread(scheduler)

### 3.8.2.7 last\_switch\_time

```
struct timespec64 ums_thread_context::last_switch_time
```

The time of the last switch to the ums thread(scheduler)

### 3.8.2.8 regs

```
struct pt_regs ums_thread_context::regs
```

The current snapshot of cpu registers

### 3.8.2.9 ret\_regs

```
struct pt_regs ums_thread_context::ret_regs
```

The snapshot of cpu registers of the pthread that switched to the ums thread(scheduler). This is needed when thread exits UMS scheduling mode and converts back to the pthread

### 3.8.2.10 run\_by

```
pid_t ums_thread_context::run_by
```

The PID of the thread that is currently running the ums thread(scheduler)

### 3.8.2.11 state

```
ums_state_t ums_thread_context::state
```

The current state of the ums thread(scheduler)

### 3.8.2.12 switch\_count

```
unsigned int ums_thread_context::switch_count
```

The number of switches to the ums thread(scheduler)

### 3.8.2.13 switching\_time

```
unsigned long ums_thread_context::switching_time
```

The total switching time of the ums thread(scheduler)

### 3.8.2.14 wt\_id

```
unsigned int ums_thread_context::wt_id
```

The id of the worker thread that is currently run by the ums thread(scheduler)

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

- [ums.h](#)

## 3.9 ums\_thread\_entry Struct Reference

The structure for scheduler entry /proc/ums/<PID>/schedulers/<ID>

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

### Data Fields

- unsigned int [id](#)
- pid\_t [created\\_by](#)
- struct list\_head [list](#)
- struct proc\_dir\_entry \* [entry](#)
- struct proc\_dir\_entry \* [workers\\_entry](#)
- struct proc\_dir\_entry \* [info\\_entry](#)

### 3.9.1 Detailed Description

The structure for scheduler entry /proc/ums/<PID>/schedulers/<ID>

This is a node in the [ums\\_thread\\_entry\\_list](#). This is a description of the scheduler entry.

### 3.9.2 Field Documentation

#### 3.9.2.1 created\_by

```
pid_t ums_thread_entry::created_by
```

The PID of the process associated with scheduler entry

#### 3.9.2.2 entry

```
struct proc_dir_entry* ums_thread_entry::entry
```

The entry of the scheduler

### 3.9.2.3 id

```
unsigned int ums_thread_entry::id
```

Unique id of the scheduler

### 3.9.2.4 info\_entry

```
struct proc_dir_entry* ums_thread_entry::info_entry
```

The entry containing statistics about scheduler, child of [ums\\_thread\\_entry::entry](#)

### 3.9.2.5 workers\_entry

```
struct proc_dir_entry* ums_thread_entry::workers_entry
```

The entry of workers, child of [ums\\_thread\\_entry::entry](#)

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

- [proc.h](#)

## 3.10 ums\_thread\_entry\_list Struct Reference

The list of scheduler entries in /proc/ums/<PID>/schedulers.

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

### Data Fields

- struct list\_head **list**
- unsigned int [ums\\_thread\\_entry\\_count](#)

### 3.10.1 Detailed Description

The list of scheduler entries in /proc/ums/<PID>/schedulers.

The purpose of this list is to store all scheduler entries in /proc/ums/<PID>/schedulers

### 3.10.2 Field Documentation

### 3.10.2.1 ums\_thread\_entry\_count

```
unsigned int ums_thread_entry_list::ums_thread_entry_count
```

The number of scheduler entries in the list

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

- [proc.h](#)

## 3.11 ums\_thread\_list Struct Reference

The list of ums threads(schedulers)

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

### Data Fields

- struct list\_head **list**
- unsigned int [ums\\_thread\\_count](#)

### 3.11.1 Detailed Description

The list of ums threads(schedulers)

The purpose of this list is to store all ums threads(schedulers) created by the process

### 3.11.2 Field Documentation

#### 3.11.2.1 ums\_thread\_count

```
unsigned int ums_thread_list::ums_thread_count
```

The number of elements(ums threads) in the list

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

- [ums.h](#)

## 3.12 ums\_thread\_params Struct Reference

Parameters passed by the library and user for ums thread(scheduler) creation.

```
#include <device_shared.h>
```



## Data Fields

- unsigned long [function](#)
- unsigned int [completion\\_list\\_id](#)

### 3.12.1 Detailed Description

Parameters passed by the library and user for ums thread(scheduler) creation.

### 3.12.2 Field Documentation

#### 3.12.2.1 completion\_list\_id

```
unsigned int ums_thread_params::completion_list_id
```

The id of the completion list associated to ums thread(scheduler)

#### 3.12.2.2 function

```
unsigned long ums_thread_params::function
```

The pointer to starting function of the ums thread(scheduler) passed by the user

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

- [device\\_shared.h](#)

## 3.13 worker\_thread\_context Struct Reference

The worker thread.

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

## Data Fields

- unsigned int [id](#)
- struct list\_head [list](#)
- struct list\_head [wt\\_list](#)
- unsigned long [entry\\_point](#)
- unsigned int [cl\\_id](#)
- pid\_t [created\\_by](#)
- unsigned int [run\\_by](#)
- [worker\\_state\\_t](#) [state](#)
- unsigned long [running\\_time](#)
- unsigned int [switch\\_count](#)
- struct timespec64 [last\\_switch\\_time](#)
- struct pt\_regs [regs](#)
- struct fpu [fpu\\_regs](#)

### 3.13.1 Detailed Description

The worker thread.

This is a node in the [process::worker\\_thread\\_list](#). This is a description of worker thread.

### 3.13.2 Field Documentation

#### 3.13.2.1 cl\_id

```
unsigned int worker_thread_context::cl_id
```

The id of the completion list which contains the worker thread

#### 3.13.2.2 created\_by

```
pid_t worker_thread_context::created_by
```

The PID of the process that created the worker thread

#### 3.13.2.3 entry\_point

```
unsigned long worker_thread_context::entry_point
```

The starting function of the worker thread

#### 3.13.2.4 fpu\_regs

```
struct fpu worker_thread_context::fpu_regs
```

The current snapshot of fpu registers

#### 3.13.2.5 id

```
unsigned int worker_thread_context::id
```

Unique id of the worker thread

#### 3.13.2.6 last\_switch\_time

```
struct timespec64 worker_thread_context::last_switch_time
```

The time of the last switch to the worker thread

### 3.13.2.7 list

```
struct list_head worker_thread_context::list
```

This list structure is related to the list of worker threads in the process

### 3.13.2.8 regs

```
struct pt_regs worker_thread_context::regs
```

The current snapshot of cpu registers

### 3.13.2.9 run\_by

```
unsigned int worker_thread_context::run_by
```

The id of the ums thread(scheduler) that is currently running the worker thread

### 3.13.2.10 running\_time

```
unsigned long worker_thread_context::running_time
```

The total running time of the worker thread

### 3.13.2.11 state

```
worker_state_t worker_thread_context::state
```

The current state of the worker thread

### 3.13.2.12 switch\_count

```
unsigned int worker_thread_context::switch_count
```

The number of switches to the worker thread

### 3.13.2.13 wt\_list

```
struct list_head worker_thread_context::wt_list
```

This list structure is related to the list of worker threads in completion list

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

- [ums.h](#)

## 3.14 worker\_thread\_entry Struct Reference

The structure for worker thread entry /proc/ums/<PID>/schedulers/<ID>/workers/<ID>

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

### Data Fields

- unsigned int [id](#)
- struct list\_head [list](#)
- struct proc\_dir\_entry \* [entry](#)

### 3.14.1 Detailed Description

The structure for worker thread entry /proc/ums/<PID>/schedulers/<ID>/workers/<ID>

This is a node in the [worker\\_thread\\_entry\\_list](#). This is a description of the worker thread entry.

### 3.14.2 Field Documentation

#### 3.14.2.1 entry

```
struct proc_dir_entry* worker_thread_entry::entry
```

The entry of the worker thread, which contains statistics about it

#### 3.14.2.2 id

```
unsigned int worker_thread_entry::id
```

Unique id of the worker thread

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

- [proc.h](#)

## 3.15 worker\_thread\_entry\_list Struct Reference

The list of worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers.

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

## Data Fields

- struct list\_head **list**
- unsigned int [worker\\_thread\\_entry\\_count](#)

### 3.15.1 Detailed Description

The list of worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers.

The purpose of this list is to store all worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers

### 3.15.2 Field Documentation

#### 3.15.2.1 worker\_thread\_entry\_count

```
unsigned int worker_thread_entry_list::worker_thread_entry_count
```

The number of elements(completion lists) in the list

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

- [proc.h](#)

## 3.16 worker\_thread\_list Struct Reference

The list of worker threads.

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

## Data Fields

- struct list\_head **list**
- unsigned int [worker\\_thread\\_count](#)

### 3.16.1 Detailed Description

The list of worker threads.

The purpose of this list is to store all worker threads created by the process

### 3.16.2 Field Documentation

### 3.16.2.1 worker\_thread\_count

```
unsigned int worker_thread_list::worker_thread_count
```

The number of elements(worker threads) in the list

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

- [ums.h](#)

## 3.17 worker\_thread\_params Struct Reference

Parameters passed by the library and user for worker thread creation.

```
#include <device_shared.h>
```

### Data Fields

- unsigned long [function](#)
- unsigned long [function\\_args](#)
- unsigned long [stack\\_address](#)
- unsigned long [stack\\_size](#)

### 3.17.1 Detailed Description

Parameters passed by the library and user for worker thread creation.

### 3.17.2 Field Documentation

#### 3.17.2.1 function

```
unsigned long worker_thread_params::function
```

The pointer to starting function of the worker thread passed by the user

#### 3.17.2.2 function\_args

```
unsigned long worker_thread_params::function_args
```

The pointer to arguments for function

#### 3.17.2.3 stack\_address

```
unsigned long worker_thread_params::stack_address
```

The starting address of the stack allocated by the library

#### 3.17.2.4 stack\_size

```
unsigned long worker_thread_params::stack_size
```

The size of the allocated stack

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

- [device\\_shared.h](#)

# Chapter 4

## File Documentation

### 4.1 device.c File Reference

This file contains the implementation of the functions of the char device part.

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

#### Functions

- **DEFINE\_SPINLOCK** (spinlock)
- int **init\_device** (void)  
*Init/register the UMS device.*
- void **exit\_device** (void)  
*Exit/deregister the UMS device.*

#### Variables

- spinlock\_t **spinlock**
- unsigned long **sl\_irq\_flags**

#### 4.1.1 Detailed Description

This file contains the implementation of the functions of the char device part.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

## 4.1.2 Function Documentation

### 4.1.2.1 `init_device()`

```
int init_device (
    void )
```

Init/register the UMS device.

#### Returns

`int` exit code 0 for success, otherwise a corresponding error code

## 4.2 `device.h` File Reference

This file is a header of the char device part of the module.

```
#include <asm/uaccess.h>
#include <asm-generic/errno-base.h>
#include <linux/cdev.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/semaphore.h>
#include "ums.h"
#include "device_shared.h"
```

## Macros

- `#define UMS_DEVICE_NAME "umsdevice"`
- `#define UMS_DEVICE_LOG "UMS device: "`

## Functions

- `int` `init_device` (void)  
*Init/register the UMS device.*
- `void` `exit_device` (void)  
*Exit/deregister the UMS device.*



### 4.2.1 Detailed Description

This file is a header of the char device part of the module.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

This file contains the macros and function declarations of the char device part

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

### 4.2.2 Function Documentation

#### 4.2.2.1 init\_device()

```
int init_device (
    void )
```

Init/register the UMS device.

#### Returns

int exit code 0 for success, otherwise a corresponding error code

## 4.3 device.h

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

```
1
30 #pragma once
31
32 #include <asm/uaccess.h>
33 #include <asm-generic/errno-base.h>
34 #include <linux/cdev.h>
35 #include <linux/kernel.h>
36 #include <linux/miscdevice.h>
37 #include <linux/module.h>
38 #include <linux/fs.h>
39 #include <linux/semaphore.h>
40
41 #include "ums.h"
42 #include "device_shared.h"
43
44 #define UMS_DEVICE_NAME "umsdevice"
45 #define UMS_DEVICE_LOG "UMS device: "
46
47 /*
48  * init and exit device functions
49  */
50 int init_device(void);
51 void exit_device(void);
```

## 4.4 device\_shared.h File Reference

This file contains definitions of ioctl commands.

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

### Data Structures

- struct [worker\\_thread\\_params](#)  
*Parameters passed by the library and user for worker thread creation.*
- struct [add\\_wt\\_params](#)  
*Parameters passed by the library and user for adding worker thread to completion list.*
- struct [ums\\_thread\\_params](#)  
*Parameters passed by the library and user for ums thread(scheduler) creation.*

### Macros

- #define **UMS\_DEV\_IOCTL\_MAGIC** 'R'
- #define **UMS\_DEV\_INIT\_UMS\_PROCESS** \_IO(UMS\_DEV\_IOCTL\_MAGIC, 0)
- #define **UMS\_DEV\_EXIT\_UMS\_PROCESS** \_IO(UMS\_DEV\_IOCTL\_MAGIC, 1)
- #define **UMS\_DEV\_CREATE\_COMPLETION\_LIST** \_IO(UMS\_DEV\_IOCTL\_MAGIC, 2)
- #define **UMS\_DEV\_CREATE\_WORKER\_THREAD** \_IOW(UMS\_DEV\_IOCTL\_MAGIC, 3, [worker\\_thread\\_params\\_t](#) \*)
- #define **UMS\_DEV\_ADD\_TO\_COMPLETION\_LIST** \_IOW(UMS\_DEV\_IOCTL\_MAGIC, 4, [add\\_wt\\_params\\_t](#) \*)
- #define **UMS\_DEV\_CREATE\_UMS\_THREAD** \_IOW(UMS\_DEV\_IOCTL\_MAGIC, 5, [ums\\_thread\\_params\\_t](#) \*)
- #define **UMS\_DEV\_CONVERT\_TO\_UMS\_THREAD** \_IOW(UMS\_DEV\_IOCTL\_MAGIC, 6, int)
- #define **UMS\_DEV\_CONVERT\_FROM\_UMS\_THREAD** \_IO(UMS\_DEV\_IOCTL\_MAGIC, 7)
- #define **UMS\_DEV\_DEQUEUE\_CL\_ITEMS** \_IOWR(UMS\_DEV\_IOCTL\_MAGIC, 8, int \*)
- #define **UMS\_DEV\_SWITCH\_TO\_WORKER\_THREAD** \_IOW(UMS\_DEV\_IOCTL\_MAGIC, 9, int)
- #define **UMS\_DEV\_SWITCH\_BACK\_TO\_UMS\_THREAD** \_IOW(UMS\_DEV\_IOCTL\_MAGIC, 10, [yield\\_reason\\_t](#))

### Typedefs

- typedef struct [worker\\_thread\\_params](#) **worker\_thread\_params\_t**  
*Parameters passed by the library and user for worker thread creation.*
- typedef struct [add\\_wt\\_params](#) **add\_wt\_params\_t**  
*Parameters passed by the library and user for adding worker thread to completion list.*
- typedef struct [ums\\_thread\\_params](#) **ums\_thread\_params\_t**  
*Parameters passed by the library and user for ums thread(scheduler) creation.*
- typedef enum [yield\\_reason](#) **yield\_reason\_t**  
*Parameters passed by the library and user for worker thread yield function.*

### Enumerations

- enum [yield\\_reason](#) { **FINISH** , **PAUSE** }  
*Parameters passed by the library and user for worker thread yield function.*

### 4.4.1 Detailed Description

This file contains definitions of ioctl commands.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

This file is also included in the library Defined ioctl commands are mapped with the library functions:

- UMS\_DEV\_INIT\_UMS\_PROCESS <-> `init_ums()`
- UMS\_DEV\_EXIT\_UMS\_PROCESS <-> `exit_ums()`
- UMS\_DEV\_CREATE\_COMPLETION\_LIST <-> `create_completion_list()`
- UMS\_DEV\_CREATE\_WORKER\_THREAD <-> `create_worker_thread()`
- UMS\_DEV\_ADD\_TO\_COMPLETION\_LIST <-> `add_worker_thread()`
- UMS\_DEV\_CREATE\_UMS\_THREAD <-> `enter_ums_scheduling_mode()`
- UMS\_DEV\_CONVERT\_TO\_UMS\_THREAD <-> `convert_to_ums_thread()`
- UMS\_DEV\_CONVERT\_FROM\_UMS\_THREAD <-> `exit_ums_scheduling_mode()`
- UMS\_DEV\_DEQUEUE\_CL\_ITEMS <-> `dequeue_completion_list_items()`
- UMS\_DEV\_SWITCH\_TO\_WORKER\_THREAD <-> `execute_worker_thread()`
- UMS\_DEV\_SWITCH\_BACK\_TO\_UMS\_THREAD <-> `worker_thread_yield()`

Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

### 4.4.2 Enumeration Type Documentation

#### 4.4.2.1 yield\_reason

enum `yield_reason`

Parameters passed by the library and user for worker thread yield function.

## Enumerator

FINISH	The yield reason is FINISH when worker thread has finished its task
PAUSE	The yield reason is PAUSE when worker thread hasn't finished its task but for some reason is requested to be blocked/paused until next invocation

## 4.5 device\_shared.h

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

```

1
42 #pragma once
43
44 #include <linux/ioctl.h>
45
46 #define UMS_DEV_IOCTL_MAGIC 'R'
47
52 typedef struct worker_thread_params {
53     unsigned long function;
54     unsigned long function_args;
55     unsigned long stack_address;
56     unsigned long stack_size;
57 } worker_thread_params_t;
58
63 typedef struct add_wt_params {
64     unsigned int completion_list_id;
65     unsigned int worker_thread_id;
66 } add_wt_params_t;
67
72 typedef struct ums_thread_params {
73     unsigned long function;
74     unsigned int completion_list_id;
75 } ums_thread_params_t;
76
81 typedef enum yield_reason {
82     FINISH,
83     PAUSE
84 } yield_reason_t;
85
86
87 #define UMS_DEV_INIT_UMS_PROCESS _IO(UMS_DEV_IOCTL_MAGIC, 0)
88 #define UMS_DEV_EXIT_UMS_PROCESS _IO(UMS_DEV_IOCTL_MAGIC, 1)
89 #define UMS_DEV_CREATE_COMPLETION_LIST _IO(UMS_DEV_IOCTL_MAGIC, 2)
90 #define UMS_DEV_CREATE_WORKER_THREAD _IOW(UMS_DEV_IOCTL_MAGIC, 3, worker_thread_params_t *)
91 #define UMS_DEV_ADD_TO_COMPLETION_LIST _IOW(UMS_DEV_IOCTL_MAGIC, 4, add_wt_params_t *)
92 #define UMS_DEV_CREATE_UMS_THREAD _IOW(UMS_DEV_IOCTL_MAGIC, 5, ums_thread_params_t *)
93 #define UMS_DEV_CONVERT_TO_UMS_THREAD _IOW(UMS_DEV_IOCTL_MAGIC, 6, int)
94 #define UMS_DEV_CONVERT_FROM_UMS_THREAD _IO(UMS_DEV_IOCTL_MAGIC, 7)
95 #define UMS_DEV_DEQUEUE_CL_ITEMS _IOWR(UMS_DEV_IOCTL_MAGIC, 8, int *)
96 #define UMS_DEV_SWITCH_TO_WORKER_THREAD _IOW(UMS_DEV_IOCTL_MAGIC, 9, int)
97 #define UMS_DEV_SWITCH_BACK_TO_UMS_THREAD _IOW(UMS_DEV_IOCTL_MAGIC, 10, yield_reason_t)

```

## 4.6 module.c File Reference

This file contains the entry point of the module.

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

### Functions

- **MODULE\_AUTHOR** ("Sultan Umarbaev <umarbaev.1954544@studenti.uniroma1.it>")
- **MODULE\_DESCRIPTION** ("UMS module")
- **MODULE\_LICENSE** ("GPL")
- **MODULE\_VERSION** ("1.0.0")
- **module\_init** (ums\_module\_init)
- **module\_exit** (ums\_module\_exit)

### 4.6.1 Detailed Description

This file contains the entry point of the module.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

## 4.7 module.h File Reference

This file is a header of the module entry point.

```
#include <linux/kernel.h>
#include <linux/module.h>
#include "device.h"
#include "proc.h"
```

### 4.7.1 Detailed Description

This file is a header of the module entry point.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

## 4.8 module.h

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

```
1
28 #pragma once
29
30 #include <linux/kernel.h>
31 #include <linux/module.h>
32
33 #include "device.h"
34 #include "proc.h"
```

## 4.9 proc.c File Reference

This file contains the implementation of the functions of the /proc part.

```
#include "proc.h"
```

### Functions

- int [init\\_proc](#) (void)  
*Init initial proc files for the module.*
- void [exit\\_proc](#) (void)  
*Exit/delete allocated structures and entries of the proc part of the module.*
- int [create\\_process\\_entry](#) (pid\_t pid)  
*Create /proc/ums/<PID> entry.*
- int [create\\_umst\\_entry](#) (pid\_t pid, unsigned int umst\_id)  
*Create /proc/ums/<PID>/schedulers/<ID> entry.*
- int [create\\_wt\\_entry](#) (unsigned int umst\_id, unsigned int wt\_id)  
*Create /proc/ums/<PID>/schedulers/<ID>/workers/<ID> entry.*
- [process\\_entry\\_t](#) \* [get\\_process\\_entry\\_with\\_pid](#) (pid\_t req\_pid)  
*Get process entry structure from [process\\_entry\\_list](#) with specific PID.*
- [ums\\_thread\\_entry\\_t](#) \* [get\\_ums\\_thread\\_entry\\_with\\_id](#) (pid\_t req\_pid, unsigned int id)  
*Get scheduler entry structure from [ums\\_thread\\_entry\\_list](#) with specific id.*

### Variables

- [process\\_entry\\_list\\_t](#) [process\\_entry\\_list](#)
- [worker\\_thread\\_entry\\_list\\_t](#) [worker\\_thread\\_entry\\_list](#)
- [ums\\_thread\\_entry\\_list\\_t](#) [ums\\_thread\\_entry\\_list](#)

### 4.9.1 Detailed Description

This file contains the implementation of the functions of the /proc part.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

### 4.9.2 Function Documentation

#### 4.9.2.1 create\_process\_entry()

```
int create_process_entry (
    pid_t pid )
```

Create /proc/ums/<PID> entry.

First, we check if the process entry already exists. Create [process\\_entry\\_t](#) and add it to [process\\_entry\\_list](#) that enabled UMS mechanism. Create /proc/ums/<PID> and /proc/ums/<PID>/schedulers entries.

Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.9.2.2 create\_umst\_entry()

```
int create_umst_entry (
    pid_t pid,
    unsigned int umst_id )
```

Create /proc/ums/<PID>/schedulers/<ID> entry.

First, we check if current process has process entry in /proc/ums. Create [ums\\_thread\\_entry\\_t](#) and add it to [ums\\_thread\\_entry\\_list](#). Create /proc/ums/<PID>/schedulers/<ID>, /proc/ums/<PID>/schedulers/<ID>/workers and /proc/ums/<PID>/schedulers/<ID>/info entries.

Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.9.2.3 create\_wt\_entry()

```
int create_wt_entry (
    unsigned int umst_id,
    unsigned int wt_id )
```

Create /proc/ums/<PID>/schedulers/<ID>/workers/<ID> entry.

First, we check if current process has process entry in /proc/ums. Create [worker\\_thread\\_entry\\_t](#) and add it to [worker\\_thread\\_entry\\_list](#). Create /proc/ums/<PID>/schedulers/<ID>/workers/<ID> entry.

##### Returns

int exit code 0 for success, otherwise a corresponding error code

#### 4.9.2.4 exit\_proc()

```
void exit_proc (
    void )
```

Exit/delete allocated structures and entries of the proc part of the module.

Remove /proc/ums entry. Clean up memory allocated for the data structures that are associated in proc part:

- [process\\_entry\\_list](#)
- [ums\\_thread\\_entry\\_list](#)
- [worker\\_thread\\_entry\\_list](#)

#### 4.9.2.5 get\_process\_entry\_with\_pid()

```
process_entry_t * get_process_entry_with_pid (
    pid_t req_pid )
```

Get process entry structure from [process\\_entry\\_list](#) with specific PID.

##### Parameters

<i>req_pid</i>	the PID of the process
----------------	------------------------

##### Returns

process\_entry\_t the pointer to process entry structure



#### 4.9.2.6 get\_ums\_thread\_entry\_with\_id()

```
ums_thread_entry_t * get_ums_thread_entry_with_id (
    pid_t req_pid,
    unsigned int id )
```

Get scheduler entry structure from [ums\\_thread\\_entry\\_list](#) with specific id.

##### Parameters

<i>req_pid</i>	the pid that created the scheduler
<i>id</i>	the id of the scheduler

##### Returns

`ums_thread_entry_t` the pointer to scheduler entry structure

#### 4.9.2.7 init\_proc()

```
int init_proc (
    void )
```

Init initial proc files for the module.

Create /proc/ums entry at the start of the module

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

### 4.9.3 Variable Documentation

#### 4.9.3.1 process\_entry\_list

```
process_entry_list_t process_entry_list
```

##### Initial value:

```
= {
    .list = LIST_HEAD_INIT(process_entry_list.list),
    .process_entry_count = 0
}
```

#### 4.9.3.2 ums\_thread\_entry\_list

```
ums_thread_entry_list_t ums_thread_entry_list
```

##### Initial value:

```
= {
    .list = LIST_HEAD_INIT(ums_thread_entry_list.list),
    .ums_thread_entry_count = 0
}
```

#### 4.9.3.3 worker\_thread\_entry\_list

```
worker_thread_entry_list_t worker_thread_entry_list
```

##### Initial value:

```
= {
    .list = LIST_HEAD_INIT(worker_thread_entry_list.list),
    .worker_thread_entry_count = 0
}
```

## 4.10 proc.h File Reference

This file is a header of the /proc part of the module.

```
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include "ums.h"
```

## Data Structures

- struct [process\\_entry\\_list](#)  
*The list of process entries in /proc/ums.*
- struct [ums\\_thread\\_entry\\_list](#)  
*The list of scheduler entries in /proc/ums/<PID>/schedulers.*
- struct [worker\\_thread\\_entry\\_list](#)  
*The list of worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers.*
- struct [process\\_entry](#)  
*The structure for process entry /proc/ums/<PID>*
- struct [ums\\_thread\\_entry](#)  
*The structure for scheduler entry /proc/ums/<PID>/schedulers/<ID>*
- struct [worker\\_thread\\_entry](#)  
*The structure for worker thread entry /proc/ums/<PID>/schedulers/<ID>/workers/<ID>*

## Macros

- #define **UMS\_PROC\_LOG** "UMS proc: "
- #define **ERROR\_PROC\_FAIL** 1
- #define **ERROR\_PROCESS\_ENTRY\_NOT\_FOUND** 601
- #define **ERROR\_PROCESS\_ENTRY\_ALREADY\_EXISTS** 602
- #define **ERROR\_UMST\_ENTRY\_NOT\_FOUND** 603

## Typedefs

- typedef struct [process\\_entry\\_list](#) [process\\_entry\\_list\\_t](#)  
*The list of process entries in /proc/ums.*
- typedef struct [ums\\_thread\\_entry\\_list](#) [ums\\_thread\\_entry\\_list\\_t](#)  
*The list of scheduler entries in /proc/ums/<PID>/schedulers.*
- typedef struct [worker\\_thread\\_entry\\_list](#) [worker\\_thread\\_entry\\_list\\_t](#)  
*The list of worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers.*
- typedef struct [process\\_entry](#) [process\\_entry\\_t](#)  
*The structure for process entry /proc/ums/<PID>*
- typedef struct [ums\\_thread\\_entry](#) [ums\\_thread\\_entry\\_t](#)  
*The structure for scheduler entry /proc/ums/<PID>/schedulers/<ID>*
- typedef struct [worker\\_thread\\_entry](#) [worker\\_thread\\_entry\\_t](#)  
*The structure for worker thread entry /proc/ums/<PID>/schedulers/<ID>/workers/<ID>*

## Functions

- int [init\\_proc](#) (void)  
*Init initial proc files for the module.*
- void [exit\\_proc](#) (void)  
*Exit/delete allocated structures and entries of the proc part of the module.*
- int [create\\_process\\_entry](#) (pid\_t pid)  
*Create /proc/ums/<PID> entry.*
- int [create\\_umst\\_entry](#) (pid\_t pid, unsigned int umst\_id)  
*Create /proc/ums/<PID>/schedulers/<ID> entry.*
- int [create\\_wt\\_entry](#) (unsigned int umst\_id, unsigned int wt\_id)  
*Create /proc/ums/<PID>/schedulers/<ID>/workers/<ID> entry.*
- [process\\_entry\\_t](#) \* [get\\_process\\_entry\\_with\\_pid](#) (pid\_t req\_pid)  
*Get process entry structure from [process\\_entry\\_list](#) with specific PID.*
- [ums\\_thread\\_entry\\_t](#) \* [get\\_ums\\_thread\\_entry\\_with\\_id](#) (pid\_t req\_pid, unsigned int id)  
*Get scheduler entry structure from [ums\\_thread\\_entry\\_list](#) with specific id.*

### 4.10.1 Detailed Description

This file is a header of the /proc part of the module.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

This file contains all data structures, macros, error codes and function declarations of the /proc part

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

## 4.10.2 Typedef Documentation

### 4.10.2.1 process\_entry\_list\_t

```
typedef struct process_entry_list process_entry_list_t
```

The list of process entries in /proc/ums.

The purpose of this list is to store all process entries in /proc/ums/<PID>

### 4.10.2.2 process\_entry\_t

```
typedef struct process_entry process_entry_t
```

The structure for process entry /proc/ums/<PID>

This is a node in the [process\\_entry\\_list](#). This is a description of the process entry.

### 4.10.2.3 ums\_thread\_entry\_list\_t

```
typedef struct ums_thread_entry_list ums_thread_entry_list_t
```

The list of scheduler entries in /proc/ums/<PID>/schedulers.

The purpose of this list is to store all scheduler entries in /proc/ums/<PID>/schedulers

### 4.10.2.4 ums\_thread\_entry\_t

```
typedef struct ums_thread_entry ums_thread_entry_t
```

The structure for scheduler entry /proc/ums/<PID>/schedulers/<ID>

This is a node in the [ums\\_thread\\_entry\\_list](#). This is a description of the scheduler entry.

### 4.10.2.5 worker\_thread\_entry\_list\_t

```
typedef struct worker_thread_entry_list worker_thread_entry_list_t
```

The list of worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers.

The purpose of this list is to store all worker thread entries in /proc/ums/<PID>/schedulers/<ID>/workers

#### 4.10.2.6 worker\_thread\_entry\_t

```
typedef struct worker_thread_entry worker_thread_entry_t
```

The structure for worker thread entry /proc/ums/<PID>/schedulers/<ID>/workers/<ID>

This is a node in the [worker\\_thread\\_entry\\_list](#). This is a description of the worker thread entry.

### 4.10.3 Function Documentation

#### 4.10.3.1 create\_process\_entry()

```
int create_process_entry (  
    pid_t pid )
```

Create /proc/ums/<PID> entry.

First, we check if the process entry already exists. Create [process\\_entry\\_t](#) and add it to [process\\_entry\\_list](#) that enabled UMS mechanism. Create /proc/ums/<PID> and /proc/ums/<PID>/schedulers entries.

##### Returns

int exit code 0 for success, otherwise a corresponding error code

#### 4.10.3.2 create\_umst\_entry()

```
int create_umst_entry (  
    pid_t pid,  
    unsigned int umst_id )
```

Create /proc/ums/<PID>/schedulers/<ID> entry.

First, we check if current process has process entry in /proc/ums. Create [ums\\_thread\\_entry\\_t](#) and add it to [ums\\_thread\\_entry\\_list](#). Create /proc/ums/<PID>/schedulers/<ID>, /proc/ums/<PID>/schedulers/<ID>/workers and /proc/ums/<PID>/schedulers/<ID>/info entries.

##### Returns

int exit code 0 for success, otherwise a corresponding error code

#### 4.10.3.3 create\_wt\_entry()

```
int create_wt_entry (
    unsigned int umst_id,
    unsigned int wt_id )
```

Create /proc/ums/<PID>/schedulers/<ID>/workers/<ID> entry.

First, we check if current process has process entry in /proc/ums. Create [worker\\_thread\\_entry\\_t](#) and add it to [worker\\_thread\\_entry\\_list](#). Create /proc/ums/<PID>/schedulers/<ID>/workers/<ID> entry.

##### Returns

int exit code 0 for success, otherwise a corresponding error code

#### 4.10.3.4 exit\_proc()

```
void exit_proc (
    void )
```

Exit/delete allocated structures and entries of the proc part of the module.

Remove /proc/ums entry. Clean up memory allocated for the data structures that are associated in proc part:

- [process\\_entry\\_list](#)
- [ums\\_thread\\_entry\\_list](#)
- [worker\\_thread\\_entry\\_list](#)

#### 4.10.3.5 get\_process\_entry\_with\_pid()

```
process_entry_t * get_process_entry_with_pid (
    pid_t req_pid )
```

Get process entry structure from [process\\_entry\\_list](#) with specific PID.

##### Parameters

<i>req_pid</i>	the PID of the process
----------------	------------------------

##### Returns

process\_entry\_t the pointer to process entry structure

#### 4.10.3.6 get\_ums\_thread\_entry\_with\_id()

```
ums_thread_entry_t * get_ums_thread_entry_with_id (
    pid_t req_pid,
    unsigned int id )
```

Get scheduler entry structure from [ums\\_thread\\_entry\\_list](#) with specific id.

##### Parameters

<i>req_pid</i>	the pid that created the scheduler
<i>id</i>	the id of the scheduler

##### Returns

ums\_thread\_entry\_t the pointer to scheduler entry structure

#### 4.10.3.7 init\_proc()

```
int init_proc (
    void )
```

Init initial proc files for the module.

Create /proc/ums entry at the start of the module

##### Returns

int exit code 0 for success, otherwise a corresponding error code

## 4.11 proc.h

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

```
1
31 #pragma once
32
33 #include <linux/proc_fs.h>
34 #include <linux/seq_file.h>
35
36 #include "ums.h"
37
38 #define UMS_PROC_LOG "UMS proc: "
39
40 #define ERROR_PROC_FAIL 1
41 #define ERROR_PROCESS_ENTRY_NOT_FOUND 601
42 #define ERROR_PROCESS_ENTRY_ALREADY_EXISTS 602
43 #define ERROR_UMST_ENTRY_NOT_FOUND 603
44
45 /*
46  * Structs
47  */
48
49 typedef struct process_entry_list {
50     struct list_head list;
51     unsigned int process_entry_count;
52 } process_entry_list_t;
53
54 typedef struct ums_thread_entry_list {
```

```

67     struct list_head list;
68     unsigned int ums_thread_entry_count;
69 } ums_thread_entry_list_t;
70
71 typedef struct worker_thread_entry_list {
72     struct list_head list;
73     unsigned int worker_thread_entry_count;
74 } worker_thread_entry_list_t;
75
76 typedef struct process_entry {
77     pid_t pid;
78     struct list_head list;
79     struct proc_dir_entry *entry;
80     struct proc_dir_entry *schedulers_entry;
81 } process_entry_t;
82
83 typedef struct ums_thread_entry {
84     unsigned int id;
85     pid_t created_by;
86     struct list_head list;
87     struct proc_dir_entry *entry;
88     struct proc_dir_entry *workers_entry;
89     struct proc_dir_entry *info_entry;
90 } ums_thread_entry_t;
91
92 typedef struct worker_thread_entry {
93     unsigned int id;
94     struct list_head list;
95     struct proc_dir_entry *entry;
96 } worker_thread_entry_t;
97
98 /*
99  * Functions
100 */
101 int init_proc(void);
102 void exit_proc(void);
103 int create_process_entry(pid_t pid);
104 int create_umst_entry(pid_t pid, unsigned int umst_id);
105 int create_wt_entry(unsigned int umst_id, unsigned int wt_id);
106
107 /*
108  * Auxiliary functions
109 */
110 process_entry_t *get_process_entry_with_pid(pid_t req_pid);
111 ums_thread_entry_t *get_ums_thread_entry_with_id(pid_t req_pid, unsigned int id);

```

## 4.12 ums.c File Reference

This file contains the implementation of all main functions of the module.

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

### Functions

- int [init\\_ums\\_process](#) (void)  
*Initialize/enable UMS in the process.*
- int [exit\\_ums\\_process](#) (void)  
*Delete current process that enabled UMS.*
- void [exit\\_ums](#) (void)  
*Clean/free the list of processes [process\\_list](#).*
- int [create\\_completion\\_list](#) (void)  
*Create completion list.*
- int [create\\_worker\\_thread](#) ([worker\\_thread\\_params\\_t](#) \*params)  
*Create worker thread.*
- int [add\\_to\\_completion\\_list](#) ([add\\_wt\\_params\\_t](#) \*params)  
*Add worker thread to completion list.*



- int [create\\_ums\\_thread](#) ([ums\\_thread\\_params\\_t](#) \*params)  
*Create ums thread(scheduler)*
- int [convert\\_to\\_ums\\_thread](#) (unsigned int ums\_thread\_id)  
*Convert thread into ums thread(scheduler)*
- int [convert\\_from\\_ums\\_thread](#) (void)  
*Convert back from ums thread(scheduler)*
- int [dequeue\\_completion\\_list\\_items](#) (int \*read\_wt\_list)  
*Dequeue completion list items.*
- int [switch\\_to\\_worker\\_thread](#) (unsigned int worker\_thread\_id)  
*Switch to worker thread.*
- int [switch\\_back\\_to\\_ums\\_thread](#) ([yield\\_reason\\_t](#) yield\_reason)  
*Convert back from worker thread.*
- [process\\_t](#) \* [get\\_process\\_with\\_pid](#) (pid\_t req\_pid)  
*Get process structure from [process\\_list](#) with specific PID.*
- [completion\\_list\\_t](#) \* [get\\_cl\\_with\\_id](#) ([process\\_t](#) \*process, unsigned int completion\_list\_id)  
*Get completion list from [process::cl\\_list](#) with specific id.*
- [worker\\_thread\\_context\\_t](#) \* [get\\_wt\\_with\\_id](#) ([process\\_t](#) \*process, unsigned int worker\_thread\_id)  
*Get worker thread from [process::worker\\_thread\\_list](#) with specific id.*
- int [get\\_ready\\_wt\\_list](#) ([completion\\_list\\_t](#) \*completion\_list, unsigned int \*ready\_wt\_list)  
*Fill the array of integers with ready worker thread ids from completion list.*
- [worker\\_thread\\_context\\_t](#) \* [get\\_wt\\_run\\_by\\_umst\\_id](#) ([process\\_t](#) \*process, unsigned int ums\_thread\_id)  
*Get worker thread from [process::worker\\_thread\\_list](#) run by specific ums thread(scheduler)*
- [ums\\_thread\\_context\\_t](#) \* [get\\_umst\\_with\\_id](#) ([process\\_t](#) \*process, unsigned int ums\_thread\_id)  
*Get ums thread(scheduler) from [process::ums\\_thread\\_list](#) with specific id.*
- [ums\\_thread\\_context\\_t](#) \* [get\\_umst\\_run\\_by\\_pid](#) ([process\\_t](#) \*process, pid\_t req\_pid)  
*Get ums thread(scheduler) from [process::ums\\_thread\\_list](#) run by specific thread.*
- int [free\\_process\\_ums\\_thread\\_list](#) ([process\\_t](#) \*process)  
*Clean the list of ums threads(schedulers)*
- int [free\\_process\\_cl\\_list](#) ([process\\_t](#) \*process)  
*Clean the list of completion lists.*
- int [free\\_process\\_worker\\_thread\\_list](#) ([process\\_t](#) \*process)  
*Clean the list of worker threads.*
- int [free\\_process](#) ([process\\_t](#) \*process)  
*Delete/free process structure.*
- unsigned long [get\\_wt\\_running\\_time](#) ([worker\\_thread\\_context\\_t](#) \*worker\_thread\_context)  
*Calculate the total running time of the worker thread.*
- unsigned long [get\\_umst\\_switching\\_time](#) ([ums\\_thread\\_context\\_t](#) \*ums\_thread\_context)  
*Calculate the total switching time of the ums thread.*

## Variables

- [process\\_list\\_t](#) [process\\_list](#)

### 4.12.1 Detailed Description

This file contains the implementation of all main functions of the module.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

### 4.12.2 Function Documentation

#### 4.12.2.1 `add_to_completion_list()`

```
int add_to_completion_list (
    add_wt_params_t * params )
```

Add worker thread to completion list.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary addition steps: Check if exists and get completion list with requested `params::completion_list_id` from `process::cl_list`. Check if exists and get worker thread with requested `params::worker_thread_id` from `process::worker_thread_list`. Set `worker_thread_context::cl_id` to `completion_list::id` and add the worker thread to `completion_list::wt_list`.

#### Parameters

<code>params</code>	pointer to data structure shared to user space to pass parameters
---------------------	---

#### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.12.2.2 convert\_from\_ums\_thread()

```
int convert_from_ums_thread (
    void )
```

Convert back from ums thread(scheduler)

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from `process::ums_thread_list`. Assign to structure necessary values:

- `ums_thread_context::run_by` is set to -1
- `ums_thread_context::state` is set to `ums_state_t::IDLE` Then, perform context switch operation:
- switch current `pt_regs` and `fpu` structures to `ums_thread_context::regs` and `ums_thread_context::fpu_regs`

#### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.12.2.3 convert\_to\_ums\_thread()

```
int convert_to_ums_thread (
    unsigned int ums_thread_id )
```

Convert thread into ums thread(scheduler)

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) with requested `ums_thread_id` from `process::ums_thread_list`. Check if this ums thread(scheduler) is IDLE and not currently run by other thread. Save the context of the current thread:

- `ums_thread_context::run_by` is set to `current->pid`
- `ums_thread_context::state` is set to `ums_state_t::RUNNING`
- `ums_thread_context::switch_count` is increased by 1
- `ums_thread_context::last_switch_time` is set to current time by `ktime_get_real_ts64()`
- `ums_thread_context::regs` is set to the values of `task_pt_regs(current)` function
- `ums_thread_context::fpu_regs` is set to the values of `copy_fxregs_to_kernel()` function
- `ums_thread_context::ret_regs` is set to `ums_thread_context::regs`
- `ums_thread_context::regs::ip` is set to `ums_thread_context::entry_point` Then, perform context switch operation:
- switch current `pt_regs` structure to `ums_thread_context::regs`

## Parameters

<code>ums_thread↔ _id</code>	the id of the ums thread(scheduler) to be converted to
----------------------------------	--

## Returns

`int` exit code 0 for success, otherwise a corresponding error code

4.12.2.4 `create_completion_list()`

```
int create_completion_list (
    void )
```

Create completion list.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary creation steps: Create a new completion list and return a corresponding id. Add the completion list to `process::cl_list` and initialize `completion_list::wt_list` for storing worker threads. Set `completion_list::id` to the current number of completion lists in `process::cl_list`.

## Returns

`int` completion list id

4.12.2.5 `create_ums_thread()`

```
int create_ums_thread (
    ums_thread_params_t * params )
```

Create ums thread(scheduler)

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary creation steps: Create a new ums thread(scheduler) and add the ums thread(scheduler) to `process::ums_thread_list`. Assign to structure all initial values and the ones passed by `params` parameter, hence:

- `ums_thread_context::id` is to the current number of ums threads in `process::ums_thread_list`
- `ums_thread_context::entry_point` is set to the starting function passed by `params::function`
- `ums_thread_context::cl_id` is set to `params.completion_list_id`
- `ums_thread_context::wt_id` is set to -1, no worker thread is currently run by ums thread(scheduler)
- `ums_thread_context::created_by` is set to `process::pid`
- `ums_thread_context::run_by` is set to -1, no thread is currently running the ums thread(scheduler)
- `ums_thread_context::state` is set to `ums_state_t::IDLE`
- `ums_thread_context::switch_count` is set to 0. Additionally, we create `/proc/ums/<PID>/schedulers/<↔ID>`, `/proc/ums/<PID>/schedulers/<ID>/workers` and `/proc/ums/<PID>/schedulers/<ID>/info` entries for ums thread(scheduler). As well as `/proc/ums/<PID>/schedulers/<ID>/workers` and `/proc/ums/<↔PID>/schedulers/<ID>/workers/<ID>` entries for each worker thread in the completion list associated with ums thread(scheduler).

## Parameters

<i>params</i>	pointer to data structure shared to user space to pass parameters
---------------	---

## Returns

`int` ums thread(scheduler) id

## 4.12.2.6 create\_worker\_thread()

```
int create_worker_thread (
    worker_thread_params_t * params )
```

Create worker thread.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary creation steps: Create a new worker thread and return a corresponding id. Add the worker thread to `process::worker_thread_list`. Assign to structure all initial values and the ones passed by `params` parameter, hence:

- `worker_thread_context::id` is to the current number of worker threads in `process::worker_thread_list`
- `worker_thread_context::entry_point` is set to the starting function passed by `params::function`
- `worker_thread_context::created_by` is set to `process::pid`
- `worker_thread_context::run_by` is set to -1 because no scheduler is running the worker thread
- `worker_thread_context::state` is set to `worker_state_t::READY`
- `worker_thread_context::running_time` is set to 0
- `worker_thread_context::switch_count` is set to 0
- `worker_thread_context::regs` is set to the values of `task_pt_regs(current)` function
- `worker_thread_context::regs::ip` is set to `params::function`, the starting function of the worker thread
- `worker_thread_context::regs::di` is set to `params::function_args`, the arguments to the function
- `worker_thread_context::regs::sp` is set to `params::stack_address`
- `worker_thread_context::regs::bp` is set to `params::stack_address`
- `worker_thread_context::fpu_regs` is set to the values of `copy_fxregs_to_kernel()` function

## Parameters

<i>params</i>	pointer to data structure shared to user space to pass parameters
---------------	---

## Returns

`int` worker thread id

#### 4.12.2.7 dequeue\_completion\_list\_items()

```
int dequeue_completion_list_items (
    int * read_wt_list )
```

Dequeue completion list items.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary dequeue steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from [process::ums\\_thread\\_list](#). Check if exists and get completion list associated with ums thread(scheduler) by [ums\\_thread\\_context::cl\\_id](#). Allocated temporary array of integers and fill it with runnable worker thread ids with the help of auxiliary function [get\\_ready\\_wt\\_list\(\)](#). Copy data from temporary array into array allocated by user.

##### Parameters

<i>ready_wt_list</i>	the pointer to an allocated by user array of integers which will be filled with ready to run worker thread ids
----------------------	--

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.12.2.8 exit\_ums()

```
void exit_ums (
    void )
```

Clean/free the list of processes [process\\_list](#).

Delete and free each item in the list of processes [process\\_list](#).

#### 4.12.2.9 exit\_ums\_process()

```
int exit_ums_process (
    void )
```

Delete current process that enabled UMS.

First, we check if process that invokes this function is the one that enabled UMS. Clean up memory allocated for the data structures that are associated with the process by [free\\_process\(\)](#) function.

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

**4.12.2.10 free\_process()**

```
int free_process (
    process_t * process )
```

Delete/free process structure.

Delete and free specific process. In particular, delete every element in:

- `process::cl_list`
- `process::worker_thread_list`
- `process::ums_thread_list` And after that delete the process from `process_list`, the list of all processes that enabled UMS.

**Parameters**

<code>process</code>	the pointer to the process structure of the process to be deleted
----------------------	---

**Returns**

`int` exit code 0 for success, otherwise a corresponding error code

**4.12.2.11 free\_process\_cl\_list()**

```
int free_process_cl_list (
    process_t * process )
```

Clean the list of completion lists.

Delete and free each item in the list of completion lists

**Parameters**

<code>process</code>	the pointer to the process structure of the current process
----------------------	---

**Returns**

`int` exit code 0 for success, otherwise a corresponding error code

**4.12.2.12 free\_process\_ums\_thread\_list()**

```
int free_process_ums_thread_list (
    process_t * process )
```

Clean the list of ums threads(schedulers)

Delete and free each item in the list of ums threads(schedulers)

**Parameters**

<i>process</i>	the pointer to the process structure of the current process
----------------	---

**Returns**

`int` exit code 0 for success, otherwise a corresponding error code

**4.12.2.13 free\_process\_worker\_thread\_list()**

```
int free_process_worker_thread_list (
    process_t * process )
```

Clean the list of worker threads.

Delete and free each item in the list of worker threads

**Parameters**

<i>process</i>	the pointer to the process structure of the current process
----------------	---

**Returns**

`int` exit code 0 for success, otherwise a corresponding error code

**4.12.2.14 get\_cl\_with\_id()**

```
completion_list_t * get_cl_with_id (
    process_t * process,
    unsigned int completion_list_id )
```

Get completion list from `process::cl_list` with specific id.

**Parameters**

<i>process</i>	the pointer to the process structure of the current process
<i>completion_list_id</i>	the id of the completion list requested to be retrieved

**Returns**

`completion_list_t` the pointer to completion list



#### 4.12.2.15 get\_process\_with\_pid()

```
process_t * get_process_with_pid (
    pid_t req_pid )
```

Get process structure from [process\\_list](#) with specific PID.

##### Parameters

<i>req_pid</i>	the PID of the current process
----------------	--------------------------------

##### Returns

*process\_t* the pointer to process structure

#### 4.12.2.16 get\_ready\_wt\_list()

```
int get_ready_wt_list (
    completion_list_t * completion_list,
    unsigned int * ready_wt_list )
```

Fill the array of integers with ready worker thread ids from completion list.

##### Parameters

<i>completion_list</i>	the pointer to the completion list from which to get worker thread ids
<i>ready_wt_list</i>	the pointer to the array of integers that will be filled

##### Returns

*int* exit code 0 for success, otherwise a corresponding error code

#### 4.12.2.17 get\_umst\_run\_by\_pid()

```
ums_thread_context_t * get_umst_run_by_pid (
    process_t * process,
    pid_t req_pid )
```

Get ums thread(scheduler) from [process::ums\\_thread\\_list](#) run by specific thread.

##### Parameters

<i>process</i>	the pointer to the process structure of the current process
<i>req_pid</i>	the PID of the thread that runs ums thread(scheduler)

**Returns**

`ums_thread_context_t` the pointer to ums thread(scheduler)

**4.12.2.18 get\_umst\_switching\_time()**

```
unsigned long get_umst_switching_time (
    ums_thread_context_t * ums_thread_context )
```

Calculate the total switching time of the ums thread.

**Parameters**

<code>ums_thread_context</code>	the pointer to the ums thread which swithcing time to be calculated
---------------------------------	---

**Returns**

`unsigned long` calculated total switching time

**4.12.2.19 get\_umst\_with\_id()**

```
ums_thread_context_t * get_umst_with_id (
    process_t * process,
    unsigned int ums_thread_id )
```

Get ums thread(scheduler) from `process::ums_thread_list` with specific id.

**Parameters**

<code>process</code>	the pointer to the process structure of the current process
<code>ums_thread_id</code>	the id of the ums thread(scheduler) requested to be retrieved

**Returns**

`ums_thread_context_t` the pointer to ums thread(scheduler)

**4.12.2.20 get\_wt\_run\_by\_umst\_id()**

```
worker_thread_context_t * get_wt_run_by_umst_id (
    process_t * process,
    unsigned int ums_thread_id )
```

Get worker thread from `process::worker_thread_list` run by specific ums thread(scheduler)

## Parameters

<i>process</i>	the pointer to the process structure of the current process
<i>ums_thread↔ _id</i>	the id of the ums thread(scheduler) that runs worker thread

## Returns

`worker_thread_context_t` the pointer to worker thread

**4.12.2.21 `get_wt_running_time()`**

```
unsigned long get_wt_running_time (
    worker_thread_context_t * worker_thread_context )
```

Calculate the total running time of the worker thread.

## Parameters

<i>worker_thread_context</i>	the pointer to the worker thread which running time to be calculated
------------------------------	--

## Returns

`unsigned long` calculated running time

**4.12.2.22 `get_wt_with_id()`**

```
worker_thread_context_t * get_wt_with_id (
    process_t * process,
    unsigned int worker_thread_id )
```

Get worker thread from `process::worker_thread_list` with specific id.

## Parameters

<i>process</i>	the pointer to the process structure of the current process
<i>worker_thread↔ _id</i>	the id of the worker thread requested to be retrieved

## Returns

`worker_thread_context_t` the pointer to worker thread

#### 4.12.2.23 `init_ums_process()`

```
int init_ums_process (
    void )
```

Initialize/enable UMS in the process.

First, we check if the process has already enabled UMS. In order to start utilizing UMS mechanism, we need to enable UMS for the process. By this we create process element in the `process_list` which contains all processes that enabled UMS mechanism. If not, we create `process_t` and initialize:

- `process::cl_list` - A list of completion list in the process environment
- `process::worker_thread_list` - A list of worker thread in the process environment
- `process::ums_thread_list` - A list of ums thread(schedulers) in the process environment. Additionally, we create `/proc/ums/<PID>` and `/proc/ums/<PID>/schedulers` entries

#### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.12.2.24 `switch_back_to_ums_thread()`

```
int switch_back_to_ums_thread (
    yield_reason_t yield_reason )
```

Convert back from worker thread.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from `process::ums_thread_list`. Check if exists and get worker thread run by `ums_thread_context::id` from `process::worker_thread_list`. Check if the yield reason is not `yield_reason_t::BUSY` or `yield_reason_t::FINISHED`. Then, perform context saving and context switch operations:

- `worker_thread_context::run_by` is set to -1
- `worker_thread_context::state` is set to `worker_state_t::BUSY` or `worker_state_t::FINISHED`
- `worker_thread_context::running_time` is set by auxiliary function `get_wt_running_time()`
- `worker_thread_context::regs` is set to the values of `task_pt_regs(current)` function
- `worker_thread_context::fpu_regs` is set to the values of `copy_fxregs_to_kernel()` function
- `ums_thread_context::wt_id` is set to -1
- switch current `pt_regs` and `fpu` structures to `ums_thread_context::regs` and `ums_thread_context::fpu_regs`

#### Parameters

<code>yield_reason</code>	reason which defines if worker thread should be paused or finished, <code>yield_reason_t</code>
---------------------------	---

## Returns

`int` exit code 0 for success, otherwise a corresponding error code

4.12.2.25 `switch_to_worker_thread()`

```
int switch_to_worker_thread (
    unsigned int worker_thread_id )
```

Switch to worker thread.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from `process::ums_thread_list`. Check if exists and get worker thread with requested params::`worker_thread_id` from `process::worker_thread_list`. Check if this worker thread is not BUSY and/or FINISHED. Save the context of the ums thread(scheduler):

- `ums_thread_context::wt_id` is set to `worker_thread_context::id`
- `ums_thread_context::switch_count` is increased by 1
- `ums_thread_context::last_switch_time` is set to current time by `ktime_get_real_ts64()`
- `ums_thread_context::regs` is set to the values of `task_pt_regs(current)` function
- `ums_thread_context::ret_regs` is set to `ums_thread_context::regs`
- `ums_thread_context::fpu_regs` is set to the values of `copy_fxregs_to_kernel()` function
- `ums_thread_context::switch_count` is increased by 1
- `ums_thread_context::switching_time` is set by an auxiliary function `get_umst_switching_time()`

Then, perform context switch operation:

- `worker_thread_context::run_by` is set to `ums_thread_context::id`
- `worker_thread_context::state` is set to `worker_state_t::BUSY`
- `worker_thread_context::switch_count` is increased by 1
- `worker_thread_context::last_switch_time` is set to current time by `ktime_get_real_ts64()`
- switch current `pt_regs` and `fpu` structures to `worker_thread_context::regs` and `worker_thread_context::fpu_regs`

**NOTE:** If worker thread is BUSY function returns 2, if it is FINISHED it returns 1. These cases are not considered as real ERROR but handled as a special cases in userspace. In case of BUSY thread, scheduler in userspace will try to switch to next READY worker thread. For the case of FINISHED worker thread, scheduler in userspace will update list of ready worker threads.

## Parameters

<code>worker_thread_id</code>	the id of the worker thread to be switched to
-------------------------------	---

**Returns**

`int` exit code 0 for success, otherwise a corresponding error code

**4.12.3 Variable Documentation****4.12.3.1 process\_list**

```
process_list_t process_list
```

**Initial value:**

```
= {
    .list = LIST_HEAD_INIT(process_list.list),
    .process_count = 0
}
```

**4.13 ums.h File Reference**

This file is a header of the main module functionality.

```
#include <asm/current.h>
#include <asm/fpu/internal.h>
#include <asm/fpu/types.h>
#include <linux/list.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/sched/task_stack.h>
#include <linux/uaccess.h>
#include <linux/ktime.h>
#include "device_shared.h"
#include "proc.h"
```

**Data Structures**

- struct [process\\_list](#)  
*The list of processes that initialized/enabled UMS mechanism.*
- struct [cl\\_list](#)  
*The list of completion lists.*
- struct [worker\\_thread\\_list](#)  
*The list of worker threads.*
- struct [ums\\_thread\\_list](#)  
*The list of ums threads(schedulers)*
- struct [process](#)  
*The process that initialized/enabled UMS mechanism.*
- struct [completion\\_list](#)  
*The completion list of worker threads.*
- struct [worker\\_thread\\_context](#)  
*The worker thread.*
- struct [ums\\_thread\\_context](#)  
*The ums thread(scheduler)*

## Macros

- `#define UMS_LOG "UMS: "`
- `#define ERROR_UMS_FAIL 1`
- `#define ERROR_PROCESS_ALREADY_INITIALIZED 500`
- `#define ERROR_PROCESS_NOT_INITIALIZED 501`
- `#define ERROR_COMPLETION_LIST_NOT_FOUND 502`
- `#define ERROR_WORKER_THREAD_NOT_FOUND 503`
- `#define ERROR_UMS_THREAD_NOT_FOUND 504`
- `#define ERROR_UMS_THREAD_ALREADY_RUNNING 505`

## Typedefs

- typedef struct [process\\_list](#) [process\\_list\\_t](#)  
*The list of processes that initialized/enabled UMS mechanism.*
- typedef struct [cl\\_list](#) [cl\\_list\\_t](#)  
*The list of completion lists.*
- typedef struct [worker\\_thread\\_list](#) [worker\\_thread\\_list\\_t](#)  
*The list of worker threads.*
- typedef struct [ums\\_thread\\_list](#) [ums\\_thread\\_list\\_t](#)  
*The list of ums threads(schedulers)*
- typedef struct [process](#) [process\\_t](#)  
*The process that initialized/enabled UMS mechanism.*
- typedef struct [completion\\_list](#) [completion\\_list\\_t](#)  
*The completion list of worker threads.*
- typedef enum [worker\\_state](#) [worker\\_state\\_t](#)  
*The state of the worker thread.*
- typedef struct [worker\\_thread\\_context](#) [worker\\_thread\\_context\\_t](#)  
*The worker thread.*
- typedef enum [ums\\_state](#) [ums\\_state\\_t](#)  
*The state of the ums thread(scheduler)*
- typedef struct [ums\\_thread\\_context](#) [ums\\_thread\\_context\\_t](#)  
*The ums thread(scheduler)*

## Enumerations

- enum [worker\\_state](#) { [BUSY](#) , [READY](#) , [FINISHED](#) }  
*The state of the worker thread.*
- enum [ums\\_state](#) { [RUNNING](#) , [IDLE](#) }  
*The state of the ums thread(scheduler)*

## Functions

- int [init\\_ums\\_process](#) (void)  
*Initialize/enable UMS in the process.*
- int [exit\\_ums\\_process](#) (void)  
*Delete current process that enabled UMS.*
- void [exit\\_ums](#) (void)  
*Clean/free the list of processes [process\\_list](#).*
- int [create\\_completion\\_list](#) (void)  
*Create completion list.*
- int [create\\_worker\\_thread](#) ([worker\\_thread\\_params\\_t](#) \*params)  
*Create worker thread.*
- int [add\\_to\\_completion\\_list](#) ([add\\_wt\\_params\\_t](#) \*params)  
*Add worker thread to completion list.*
- int [create\\_ums\\_thread](#) ([ums\\_thread\\_params\\_t](#) \*params)  
*Create ums thread(scheduler)*
- int [convert\\_to\\_ums\\_thread](#) (unsigned int ums\_thread\_id)  
*Convert thread into ums thread(scheduler)*
- int [convert\\_from\\_ums\\_thread](#) (void)  
*Convert back from ums thread(scheduler)*
- int [dequeue\\_completion\\_list\\_items](#) (int \*runnable\_wt\_ptr)  
*Dequeue completion list items.*
- int [switch\\_to\\_worker\\_thread](#) (unsigned int worker\_thread\_id)  
*Switch to worker thread.*
- int [switch\\_back\\_to\\_ums\\_thread](#) ([yield\\_reason\\_t](#) yield\_reason)  
*Convert back from worker thread.*
- [process\\_t](#) \* [get\\_process\\_with\\_pid](#) (pid\_t req\_pid)  
*Get process structure from [process\\_list](#) with specific PID.*
- [completion\\_list\\_t](#) \* [get\\_cl\\_with\\_id](#) ([process\\_t](#) \*process, unsigned int completion\_list\_id)  
*Get completion list from [process::cl\\_list](#) with specific id.*
- [worker\\_thread\\_context\\_t](#) \* [get\\_wt\\_with\\_id](#) ([process\\_t](#) \*process, unsigned int worker\_thread\_id)  
*Get worker thread from [process::worker\\_thread\\_list](#) with specific id.*
- int [get\\_ready\\_wt\\_list](#) ([completion\\_list\\_t](#) \*completion\_list, unsigned int \*ready\_wt\_list)  
*Fill the array of integers with ready worker thread ids from completion list.*
- [worker\\_thread\\_context\\_t](#) \* [get\\_wt\\_run\\_by\\_umst\\_id](#) ([process\\_t](#) \*process, unsigned int ums\_thread\_id)  
*Get worker thread from [process::worker\\_thread\\_list](#) run by specific ums thread(scheduler)*
- [ums\\_thread\\_context\\_t](#) \* [get\\_umst\\_with\\_id](#) ([process\\_t](#) \*process, unsigned int ums\_thread\_id)  
*Get ums thread(scheduler) from [process::ums\\_thread\\_list](#) with specific id.*
- [ums\\_thread\\_context\\_t](#) \* [get\\_umst\\_run\\_by\\_pid](#) ([process\\_t](#) \*process, pid\_t req\_pid)  
*Get ums thread(scheduler) from [process::ums\\_thread\\_list](#) run by specific thread.*
- int [free\\_process\\_ums\\_thread\\_list](#) ([process\\_t](#) \*process)  
*Clean the list of ums threads(schedulers)*
- int [free\\_process\\_cl\\_list](#) ([process\\_t](#) \*process)  
*Clean the list of completion lists.*
- int [free\\_process\\_worker\\_thread\\_list](#) ([process\\_t](#) \*process)  
*Clean the list of worker threads.*
- int [free\\_process](#) ([process\\_t](#) \*process)  
*Delete/free process structure.*
- unsigned long [get\\_wt\\_running\\_time](#) ([worker\\_thread\\_context\\_t](#) \*worker\_thread\_context)  
*Calculate the total running time of the worker thread.*
- unsigned long [get\\_umst\\_switching\\_time](#) ([ums\\_thread\\_context\\_t](#) \*ums\_thread\_context)  
*Calculate the total switching time of the ums thread.*



### 4.13.1 Detailed Description

This file is a header of the main module functionality.

Copyright (C) 2021 Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

This file is part of UMS implementation (Kernel Module).

UMS implementation (Kernel Module) is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

UMS implementation (Kernel Module) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with UMS implementation (Kernel Module). If not, see <http://www.gnu.org/licenses/>.

This file contains all main data structures, macros, error codes and function declarations of the module

#### Author

Sultan Umarbaev [name.sul27@gmail.com](mailto:name.sul27@gmail.com)

### 4.13.2 Typedef Documentation

#### 4.13.2.1 cl\_list\_t

```
typedef struct cl_list cl_list_t
```

The list of completion lists.

The purpose of this list is to store all completion lists created by the process

#### 4.13.2.2 completion\_list\_t

```
typedef struct completion_list completion_list_t
```

The completion list of worker threads.

This is a node in the `process::cl_list`. This is a description of the completion list.

#### 4.13.2.3 process\_list\_t

```
typedef struct process_list process_list_t
```

The list of processes that initialized/enabled UMS mechanism.

The purpose of this list is to store all processes that initialized/enabled UMS mechanism

#### 4.13.2.4 process\_t

```
typedef struct process process_t
```

The process that initialized/enabled UMS mechanism.

This is a node in the [process\\_list](#). This is a process that initialized/enabled UMS mechanism. Each such process has 3 lists:

- list of completion lists
- list of ums threads(schedulers)
- list of worker threads

#### 4.13.2.5 ums\_thread\_context\_t

```
typedef struct ums_thread_context ums_thread_context_t
```

The ums thread(scheduler)

This is a node in the [process::ums\\_thread\\_list](#). This is a description of ums thread(scheduler).

#### 4.13.2.6 ums\_thread\_list\_t

```
typedef struct ums_thread_list ums_thread_list_t
```

The list of ums threads(schedulers)

The purpose of this list is to store all ums threads(schedulers) created by the process

#### 4.13.2.7 worker\_thread\_context\_t

```
typedef struct worker_thread_context worker_thread_context_t
```

The worker thread.

This is a node in the [process::worker\\_thread\\_list](#). This is a description of worker thread.

#### 4.13.2.8 worker\_thread\_list\_t

```
typedef struct worker_thread_list worker_thread_list_t
```

The list of worker threads.

The purpose of this list is to store all worker threads created by the process

### 4.13.3 Enumeration Type Documentation

#### 4.13.3.1 ums\_state

```
enum ums_state
```

The state of the ums thread(scheduler)

## Enumerator

RUNNING	The ums thread(scheduler) is running when thread is converted to it
IDLE	The ums thread(scheduler) is idle when no thread is running it

## 4.13.3.2 worker\_state

```
enum worker_state
```

The state of the worker thread.

## Enumerator

BUSY	The worker thread is busy because it is being runned
READY	The worker thread is ready and can be switched to run
FINISHED	The worker thread is finished because it has completed its task

## 4.13.4 Function Documentation

## 4.13.4.1 add\_to\_completion\_list()

```
int add_to_completion_list (
    add_wt_params_t * params )
```

Add worker thread to completion list.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary addition steps: Check if exists and get completion list with requested params::completion\_list\_id from [process::cl\\_list](#). Check if exists and get worker thread with requested params::worker\_thread\_id from [process::worker\\_thread\\_list](#). Set [worker\\_thread\\_context::cl\\_id](#) to [completion\\_list::id](#) and add the worker thread to [completion\\_list::wt\\_list](#).

## Parameters

<i>params</i>	pointer to data structure shared to user space to pass parameters
---------------	---

## Returns

```
int
```

 exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.2 `convert_from_ums_thread()`

```
int convert_from_ums_thread (
    void )
```

Convert back from ums thread(scheduler)

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from `process::ums_thread_list`. Assign to structure necessary values:

- `ums_thread_context::run_by` is set to -1
- `ums_thread_context::state` is set to `ums_state_t::IDLE` Then, perform context switch operation:
- switch current `pt_regs` and `fpu` structures to `ums_thread_context::regs` and `ums_thread_context::fpu_regs`

#### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.3 `convert_to_ums_thread()`

```
int convert_to_ums_thread (
    unsigned int ums_thread_id )
```

Convert thread into ums thread(scheduler)

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) with requested `ums_thread_id` from `process::ums_thread_list`. Check if this ums thread(scheduler) is IDLE and not currently run by other thread. Save the context of the current thread:

- `ums_thread_context::run_by` is set to `current->pid`
- `ums_thread_context::state` is set to `ums_state_t::RUNNING`
- `ums_thread_context::switch_count` is increased by 1
- `ums_thread_context::last_switch_time` is set to current time by `ktime_get_real_ts64()`
- `ums_thread_context::regs` is set to the values of `task_pt_regs(current)` function
- `ums_thread_context::fpu_regs` is set to the values of `copy_fxregs_to_kernel()` function
- `ums_thread_context::ret_regs` is set to `ums_thread_context::regs`
- `ums_thread_context::regs::ip` is set to `ums_thread_context::entry_point` Then, perform context switch operation:
- switch current `pt_regs` structure to `ums_thread_context::regs`

## Parameters

<code>ums_thread↔ _id</code>	the id of the ums thread(scheduler) to be converted to
----------------------------------	--

## Returns

`int` exit code 0 for success, otherwise a corresponding error code

4.13.4.4 `create_completion_list()`

```
int create_completion_list (
    void )
```

Create completion list.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary creation steps: Create a new completion list and return a corresponding id. Add the completion list to [process::cl\\_list](#) and initialize [completion\\_list::wt\\_list](#) for storing worker threads. Set [completion\\_list::id](#) to the current number of completion lists in [process::cl\\_list](#).

## Returns

`int` completion list id

4.13.4.5 `create_ums_thread()`

```
int create_ums_thread (
    ums_thread_params_t * params )
```

Create ums thread(scheduler)

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary creation steps: Create a new ums thread(scheduler) and add the ums thread(scheduler) to [process::ums\\_thread\\_list](#). Assign to structure all initial values and the ones passed by `params` parameter, hence:

- [ums\\_thread\\_context::id](#) is to the current number of ums threads in [process::ums\\_thread\\_list](#)
- [ums\\_thread\\_context::entry\\_point](#) is set to the starting function passed by `params::function`
- [ums\\_thread\\_context::cl\\_id](#) is set to `params.completion_list_id`
- [ums\\_thread\\_context::wt\\_id](#) is set to -1, no worker thread is currently run by ums thread(scheduler)
- [ums\\_thread\\_context::created\\_by](#) is set to [process::pid](#)
- [ums\\_thread\\_context::run\\_by](#) is set to -1, no thread is currently running the ums thread(scheduler)
- [ums\\_thread\\_context::state](#) is set to [ums\\_state\\_t::IDLE](#)
- [ums\\_thread\\_context::switch\\_count](#) is set to 0. Additionally, we create `/proc/ums/<PID>/schedulers/<↔ID>`, `/proc/ums/<PID>/schedulers/<ID>/workers` and `/proc/ums/<PID>/schedulers/<ID>/info` entries for ums thread(scheduler). As well as `/proc/ums/<PID>/schedulers/<ID>/workers` and `/proc/ums/<↔PID>/schedulers/<ID>/workers/<ID>` entries for each worker thread in the completion list associated with ums thread(scheduler).

## Parameters

<i>params</i>	pointer to data structure shared to user space to pass parameters
---------------	---

## Returns

```
int ums_thread(scheduler) id
```

## 4.13.4.6 create\_worker\_thread()

```
int create_worker_thread (
    worker_thread_params_t * params )
```

Create worker thread.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary creation steps: Create a new worker thread and return a corresponding id. Add the worker thread to [process::worker\\_thread\\_list](#). Assign to structure all initial values and the ones passed by params parameter, hence:

- [worker\\_thread\\_context::id](#) is to the current number of worker threads in [process::worker\\_thread\\_list](#)
- [worker\\_thread\\_context::entry\\_point](#) is set to the starting function passed by [params::function](#)
- [worker\\_thread\\_context::created\\_by](#) is set to [process::pid](#)
- [worker\\_thread\\_context::run\\_by](#) is set to -1 because no scheduler is running the worker thread
- [worker\\_thread\\_context::state](#) is set to [worker\\_state\\_t::READY](#)
- [worker\\_thread\\_context::running\\_time](#) is set to 0
- [worker\\_thread\\_context::switch\\_count](#) is set to 0
- [worker\\_thread\\_context::regs](#) is set to the values of [task\\_pt\\_regs\(current\)](#) function
- [worker\\_thread\\_context::regs::ip](#) is set to [params::function](#), the starting function of the worker thread
- [worker\\_thread\\_context::regs::di](#) is set to [params::function\\_args](#), the arguments to the function
- [worker\\_thread\\_context::regs::sp](#) is set to [params::stack\\_address](#)
- [worker\\_thread\\_context::regs::bp](#) is set to [params::stack\\_address](#)
- [worker\\_thread\\_context::fpu\\_regs](#) is set to the values of [copy\\_fxregs\\_to\\_kernel\(\)](#) function

## Parameters

<i>params</i>	pointer to data structure shared to user space to pass parameters
---------------	---

## Returns

```
int worker thread id
```

#### 4.13.4.7 dequeue\_completion\_list\_items()

```
int dequeue_completion_list_items (
    int * read_wt_list )
```

Dequeue completion list items.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary dequeue steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from [process::ums\\_thread\\_list](#). Check if exists and get completion list associated with ums thread(scheduler) by [ums\\_thread\\_context::cl\\_id](#). Allocated temporary array of integers and fill it with runnable worker thread ids with the help of auxiliary function [get\\_ready\\_wt\\_list\(\)](#). Copy data from temporary array into array allocated by user.

##### Parameters

<i>ready_wt_list</i>	the pointer to an allocated by user array of integers which will be filled with ready to run worker thread ids
----------------------	--

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.8 exit\_ums()

```
void exit_ums (
    void )
```

Clean/free the list of processes [process\\_list](#).

Delete and free each item in the list of processes [process\\_list](#).

#### 4.13.4.9 exit\_ums\_process()

```
int exit_ums_process (
    void )
```

Delete current process that enabled UMS.

First, we check if process that invokes this function is the one that enabled UMS. Clean up memory allocated for the data structures that are associated with the process by [free\\_process\(\)](#) function.

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.10 free\_process()

```
int free_process (
    process_t * process )
```

Delete/free process structure.

Delete and free specific process. In particular, delete every element in:

- `process::cl_list`
- `process::worker_thread_list`
- `process::ums_thread_list` And after that delete the process from `process_list`, the list of all processes that enabled UMS.

##### Parameters

<code>process</code>	the pointer to the process structure of the process to be deleted
----------------------	---

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.11 free\_process\_cl\_list()

```
int free_process_cl_list (
    process_t * process )
```

Clean the list of completion lists.

Delete and free each item in the list of completion lists

##### Parameters

<code>process</code>	the pointer to the process structure of the current process
----------------------	---

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.12 free\_process\_ums\_thread\_list()

```
int free_process_ums_thread_list (
    process_t * process )
```

Clean the list of ums threads(schedulers)

Delete and free each item in the list of ums threads(schedulers)



## Parameters

<i>process</i>	the pointer to the process structure of the current process
----------------	---

## Returns

`int` exit code 0 for success, otherwise a corresponding error code

**4.13.4.13 free\_process\_worker\_thread\_list()**

```
int free_process_worker_thread_list (  
    process_t * process )
```

Clean the list of worker threads.

Delete and free each item in the list of worker threads

## Parameters

<i>process</i>	the pointer to the process structure of the current process
----------------	---

## Returns

`int` exit code 0 for success, otherwise a corresponding error code

**4.13.4.14 get\_cl\_with\_id()**

```
completion_list_t * get_cl_with_id (  
    process_t * process,  
    unsigned int completion_list_id )
```

Get completion list from `process::cl_list` with specific id.

## Parameters

<i>process</i>	the pointer to the process structure of the current process
<i>completion_list_id</i>	the id of the completion list requested to be retrieved

## Returns

`completion_list_t` the pointer to completion list

#### 4.13.4.15 `get_process_with_pid()`

```
process_t * get_process_with_pid (
    pid_t req_pid )
```

Get process structure from `process_list` with specific PID.

##### Parameters

<code>req_pid</code>	the PID of the current process
----------------------	--------------------------------

##### Returns

`process_t` the pointer to process structure

#### 4.13.4.16 `get_ready_wt_list()`

```
int get_ready_wt_list (
    completion_list_t * completion_list,
    unsigned int * ready_wt_list )
```

Fill the array of integers with ready worker thread ids from completion list.

##### Parameters

<code>completion_list</code>	the pointer to the completion list from which to get worker thread ids
<code>ready_wt_list</code>	the pointer to the array of integers that will be filled

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.17 `get_umst_run_by_pid()`

```
ums_thread_context_t * get_umst_run_by_pid (
    process_t * process,
    pid_t req_pid )
```

Get ums thread(scheduler) from `process::ums_thread_list` run by specific thread.

##### Parameters

<code>process</code>	the pointer to the process structure of the current process
<code>req_pid</code>	the PID of the thread that runs ums thread(scheduler)

**Returns**

`ums_thread_context_t` the pointer to ums thread(scheduler)

**4.13.4.18 get\_umst\_switching\_time()**

```
unsigned long get_umst_switching_time (
    ums_thread_context_t * ums_thread_context )
```

Calculate the total switching time of the ums thread.

**Parameters**

<code>ums_thread_context</code>	the pointer to the ums thread which swithcing time to be calculated
---------------------------------	---

**Returns**

`unsigned long` calculated total switching time

**4.13.4.19 get\_umst\_with\_id()**

```
ums_thread_context_t * get_umst_with_id (
    process_t * process,
    unsigned int ums_thread_id )
```

Get ums thread(scheduler) from `process::ums_thread_list` with specific id.

**Parameters**

<code>process</code>	the pointer to the process structure of the current process
<code>ums_thread_id</code>	the id of the ums thread(scheduler) requested to be retrieved

**Returns**

`ums_thread_context_t` the pointer to ums thread(scheduler)

**4.13.4.20 get\_wt\_run\_by\_umst\_id()**

```
worker_thread_context_t * get_wt_run_by_umst_id (
    process_t * process,
    unsigned int ums_thread_id )
```

Get worker thread from `process::worker_thread_list` run by specific ums thread(scheduler)

## Parameters

<i>process</i>	the pointer to the process structure of the current process
<i>ums_thread↔ _id</i>	the id of the ums thread(scheduler) that runs worker thread

## Returns

`worker_thread_context_t` the pointer to worker thread

**4.13.4.21 `get_wt_running_time()`**

```
unsigned long get_wt_running_time (
    worker_thread_context_t * worker_thread_context )
```

Calculate the total running time of the worker thread.

## Parameters

<i>worker_thread_context</i>	the pointer to the worker thread which running time to be calculated
------------------------------	--

## Returns

`unsigned long` calculated running time

**4.13.4.22 `get_wt_with_id()`**

```
worker_thread_context_t * get_wt_with_id (
    process_t * process,
    unsigned int worker_thread_id )
```

Get worker thread from `process::worker_thread_list` with specific id.

## Parameters

<i>process</i>	the pointer to the process structure of the current process
<i>worker_thread↔ _id</i>	the id of the worker thread requested to be retrieved

## Returns

`worker_thread_context_t` the pointer to worker thread

#### 4.13.4.23 init\_ums\_process()

```
int init_ums_process (
    void )
```

Initialize/enable UMS in the process.

First, we check if the process has already enabled UMS. In order to start utilizing UMS mechanism, we need to enable UMS for the process. By this we create process element in the [process\\_list](#) which contains all processes that enabled UMS mechanism. If not, we create [process\\_t](#) and initialize:

- [process::cl\\_list](#) - A list of completion list in the process environment
- [process::worker\\_thread\\_list](#) - A list of worker thread in the process environment
- [process::ums\\_thread\\_list](#) - A list of ums thread(schedulers) in the process environment. Additionally, we create `/proc/ums/<PID>` and `/proc/ums/<PID>/schedulers` entries

##### Returns

`int` exit code 0 for success, otherwise a corresponding error code

#### 4.13.4.24 switch\_back\_to\_ums\_thread()

```
int switch_back_to_ums_thread (
    yield_reason_t yield_reason )
```

Convert back from worker thread.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from [process::ums\\_thread\\_list](#). Check if exists and get worker thread run by [ums\\_thread\\_context::id](#) from [process::worker\\_thread\\_list](#). Check if the yield reason is not [yield\\_reason\\_t::BUSY](#) or [yield\\_reason\\_t::FINISHED](#). Then, perform context saving and context switch operations:

- [worker\\_thread\\_context::run\\_by](#) is set to -1
- [worker\\_thread\\_context::state](#) is set to [worker\\_state\\_t::BUSY](#) or [worker\\_state\\_t::FINISHED](#)
- [worker\\_thread\\_context::running\\_time](#) is set by auxiliary function [get\\_wt\\_running\\_time\(\)](#)
- [worker\\_thread\\_context::regs](#) is set to the values of `task_pt_regs(current)` function
- [worker\\_thread\\_context::fpu\\_regs](#) is set to the values of `copy_fxregs_to_kernel()` function
- [ums\\_thread\\_context::wt\\_id](#) is set to -1
- switch current `pt_regs` and `fpu` structures to [ums\\_thread\\_context::regs](#) and [ums\\_thread\\_context::fpu\\_regs](#)

##### Parameters

<code>yield_reason</code>	reason which defines if worker thread should be paused or finished, <a href="#">yield_reason_t</a>
---------------------------	--

**Returns**

`int` exit code 0 for success, otherwise a corresponding error code

**4.13.4.25 switch\_to\_worker\_thread()**

```
int switch_to_worker_thread (
    unsigned int worker_thread_id )
```

Switch to worker thread.

First, we check if process that invokes this function is the one that enabled UMS. Then we perform all necessary conversion steps: Check if exists and get ums thread(scheduler) run by `current->pid` thread from `process::ums_thread_list`. Check if exists and get worker thread with requested params::`worker_thread_id` from `process::worker_thread_list`. Check if this worker thread is not BUSY and/or FINISHED. Save the context of the ums thread(scheduler):

- `ums_thread_context::wt_id` is set to `worker_thread_context::id`
- `ums_thread_context::switch_count` is increased by 1
- `ums_thread_context::last_switch_time` is set to current time by `ktime_get_real_ts64()`
- `ums_thread_context::regs` is set to the values of `task_pt_regs(current)` function
- `ums_thread_context::ret_regs` is set to `ums_thread_context::regs`
- `ums_thread_context::fpu_regs` is set to the values of `copy_fxregs_to_kernel()` function
- `ums_thread_context::switch_count` is increased by 1
- `ums_thread_context::switching_time` is set by an auxiliary function `get_umst_switching_time()`

Then, perform context switch operation:

- `worker_thread_context::run_by` is set to `ums_thread_context::id`
- `worker_thread_context::state` is set to `worker_state_t::BUSY`
- `worker_thread_context::switch_count` is increased by 1
- `worker_thread_context::last_switch_time` is set to current time by `ktime_get_real_ts64()`
- switch current `pt_regs` and `fpu` structures to `worker_thread_context::regs` and `worker_thread_context::fpu_regs`

**NOTE:** If worker thread is BUSY function returns 2, if it is FINISHED it returns 1. These cases are not considered as real ERROR but handled as a special cases in userspace. In case of BUSY thread, scheduler in userspace will try to switch to next READY worker thread. For the case of FINISHED worker thread, scheduler in userspace will update list of ready worker threads.

**Parameters**

<code>worker_thread_id</code>	the id of the worker thread to be switched to
-------------------------------	---

## Returns

int exit code 0 for success, otherwise a corresponding error code

## 4.14 ums.h

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

```

1
2
31 #pragma once
32
33 #include <asm/current.h>
34 #include <asm/fpu/internal.h>
35 #include <asm/fpu/types.h>
36 #include <linux/list.h>
37 #include <linux/kernel.h>
38 #include <linux/slab.h>
39 #include <linux/sched.h>
40 #include <linux/sched/task_stack.h>
41 #include <linux/uaccess.h>
42 #include <linux/ktime.h>
43
44 #include "device_shared.h"
45 #include "proc.h"
46
47 #define UMS_LOG "UMS: "
48
49 #define ERROR_UMS_FAIL 1
50 #define ERROR_PROCESS_ALREADY_INITIALIZED 500
51 #define ERROR_PROCESS_NOT_INITIALIZED 501
52 #define ERROR_COMPLETION_LIST_NOT_FOUND 502
53 #define ERROR_WORKER_THREAD_NOT_FOUND 503
54 #define ERROR_UMS_THREAD_NOT_FOUND 504
55 #define ERROR_UMS_THREAD_ALREADY_RUNNING 505
56
57 /*
58  * Structs
59  */
60
61 typedef struct process_list {
62     struct list_head list;
63     unsigned int process_count;
64 } process_list_t;
65
66 typedef struct cl_list {
67     struct list_head list;
68     unsigned int cl_count;
69 } cl_list_t;
70
71 typedef struct worker_thread_list {
72     struct list_head list;
73     unsigned int worker_thread_count;
74 } worker_thread_list_t;
75
76 typedef struct ums_thread_list {
77     struct list_head list;
78     unsigned int ums_thread_count;
79 } ums_thread_list_t;
80
81 typedef struct process {
82     pid_t pid;
83     cl_list_t cl_list;
84     worker_thread_list_t worker_thread_list;
85     ums_thread_list_t ums_thread_list;
86     struct list_head list;
87 } process_t;
88
89 typedef struct completion_list {
90     struct list_head list;
91     struct list_head wt_list;
92     unsigned int id;
93     unsigned int worker_thread_count;
94 } completion_list_t;
95
96 typedef enum worker_state {
97     BUSY,
98     READY,
99     FINISHED
100 } worker_state_t;
101
102 typedef struct worker_thread_context {

```

```

154     unsigned int id;
155     struct list_head list;
156     struct list_head wt_list;
157     unsigned long entry_point;
158     unsigned int cl_id;
159     pid_t created_by;
160     unsigned int run_by;
161     worker_state_t state;
162     unsigned long running_time;
163     unsigned int switch_count;
164     struct timespec64 last_switch_time;
165     struct pt_regs regs;
166     struct fpu fpu_regs;
167 } worker_thread_context_t;
168
169 typedef enum ums_state {
170     RUNNING,
171     IDLE
172 } ums_state_t;
173
174 typedef struct ums_thread_context {
175     unsigned int id;
176     struct list_head list;
177     unsigned long entry_point;
178     unsigned int cl_id;
179     unsigned int wt_id;
180     pid_t created_by;
181     pid_t run_by;
182     ums_state_t state;
183     unsigned int switch_count;
184     struct timespec64 last_switch_time;
185     unsigned long switching_time;
186     unsigned long avg_switching_time;
187     struct pt_regs regs;
188     struct fpu fpu_regs;
189     struct pt_regs ret_regs;
190 } ums_thread_context_t;
191
192 /*
193  * Functions
194  */
195 int init_ums_process(void);
196 int exit_ums_process(void);
197 void exit_ums(void);
198 int create_completion_list(void);
199 int create_worker_thread(worker_thread_params_t *params);
200 int add_to_completion_list(add_wt_params_t *params);
201 int create_ums_thread(ums_thread_params_t *params);
202 int convert_to_ums_thread(unsigned int ums_thread_id);
203 int convert_from_ums_thread(void);
204 int dequeue_completion_list_items(int *runnable_wt_ptr);
205 int switch_to_worker_thread(unsigned int worker_thread_id);
206 int switch_back_to_ums_thread(yield_reason_t yield_reason);
207
208 /*
209  * Auxiliary functions
210  */
211 process_t *get_process_with_pid(pid_t req_pid);
212 completion_list_t *get_cl_with_id(process_t *process, unsigned int completion_list_id);
213 worker_thread_context_t *get_wt_with_id(process_t *process, unsigned int worker_thread_id);
214 int get_ready_wt_list(completion_list_t *completion_list, unsigned int *ready_wt_list);
215 worker_thread_context_t *get_wt_run_by_umst_id(process_t *process, unsigned int ums_thread_id);
216 ums_thread_context_t *get_umst_with_id(process_t *process, unsigned int ums_thread_id);
217 ums_thread_context_t *get_umst_run_by_pid(process_t *process, pid_t req_pid);
218 int free_process_ums_thread_list(process_t *process);
219 int free_process_cl_list(process_t *process);
220 int free_process_worker_thread_list(process_t *process);
221 int free_process(process_t *process);
222 unsigned long get_wt_running_time(worker_thread_context_t *worker_thread_context);
223 unsigned long get_umst_switching_time(ums_thread_context_t *ums_thread_context);

```



# Index

- add\_to\_completion\_list
  - ums.c, [42](#)
  - ums.h, [59](#)
- add\_wt\_params, [5](#)
  - completion\_list\_id, [5](#)
  - worker\_thread\_id, [5](#)
- avg\_switching\_time
  - ums\_thread\_context, [12](#)
- BUSY
  - ums.h, [59](#)
- cl\_count
  - cl\_list, [6](#)
- cl\_id
  - ums\_thread\_context, [12](#)
  - worker\_thread\_context, [18](#)
- cl\_list, [6](#)
  - cl\_count, [6](#)
  - process, [8](#)
- cl\_list\_t
  - ums.h, [57](#)
- completion\_list, [6](#)
  - id, [7](#)
  - list, [7](#)
  - worker\_thread\_count, [7](#)
  - wt\_list, [7](#)
- completion\_list\_id
  - add\_wt\_params, [5](#)
  - ums\_thread\_params, [17](#)
- completion\_list\_t
  - ums.h, [57](#)
- convert\_from\_ums\_thread
  - ums.c, [42](#)
  - ums.h, [59](#)
- convert\_to\_ums\_thread
  - ums.c, [43](#)
  - ums.h, [60](#)
- create\_completion\_list
  - ums.c, [44](#)
  - ums.h, [61](#)
- create\_process\_entry
  - proc.c, [31](#)
  - proc.h, [37](#)
- create\_ums\_thread
  - ums.c, [44](#)
  - ums.h, [61](#)
- create\_umst\_entry
  - proc.c, [31](#)
  - proc.h, [37](#)
- create\_worker\_thread
  - ums.c, [45](#)
  - ums.h, [62](#)
- create\_wt\_entry
  - proc.c, [31](#)
  - proc.h, [37](#)
- created\_by
  - ums\_thread\_context, [12](#)
  - ums\_thread\_entry, [14](#)
  - worker\_thread\_context, [18](#)
- dequeue\_completion\_list\_items
  - ums.c, [45](#)
  - ums.h, [62](#)
- device.c, [23](#)
  - init\_device, [24](#)
- device.h, [24](#)
  - init\_device, [25](#)
- device\_shared.h, [26](#)
  - FINISH, [28](#)
  - PAUSE, [28](#)
  - yield\_reason, [27](#)
- entry
  - process\_entry, [9](#)
  - ums\_thread\_entry, [14](#)
  - worker\_thread\_entry, [20](#)
- entry\_point
  - ums\_thread\_context, [12](#)
  - worker\_thread\_context, [18](#)
- exit\_proc
  - proc.c, [32](#)
  - proc.h, [38](#)
- exit\_ums
  - ums.c, [46](#)
  - ums.h, [63](#)
- exit\_ums\_process
  - ums.c, [46](#)
  - ums.h, [63](#)
- FINISH
  - device\_shared.h, [28](#)
- FINISHED
  - ums.h, [59](#)
- fpu\_regs
  - ums\_thread\_context, [12](#)
  - worker\_thread\_context, [18](#)
- free\_process
  - ums.c, [46](#)
  - ums.h, [63](#)

- free\_process\_cl\_list
  - ums.c, [47](#)
  - ums.h, [64](#)
- free\_process\_ums\_thread\_list
  - ums.c, [47](#)
  - ums.h, [64](#)
- free\_process\_worker\_thread\_list
  - ums.c, [48](#)
  - ums.h, [65](#)
- function
  - ums\_thread\_params, [17](#)
  - worker\_thread\_params, [22](#)
- function\_args
  - worker\_thread\_params, [22](#)
- get\_cl\_with\_id
  - ums.c, [48](#)
  - ums.h, [65](#)
- get\_process\_entry\_with\_pid
  - proc.c, [32](#)
  - proc.h, [38](#)
- get\_process\_with\_pid
  - ums.c, [48](#)
  - ums.h, [65](#)
- get\_ready\_wt\_list
  - ums.c, [49](#)
  - ums.h, [66](#)
- get\_ums\_thread\_entry\_with\_id
  - proc.c, [32](#)
  - proc.h, [38](#)
- get\_umst\_run\_by\_pid
  - ums.c, [49](#)
  - ums.h, [66](#)
- get\_umst\_switching\_time
  - ums.c, [50](#)
  - ums.h, [67](#)
- get\_umst\_with\_id
  - ums.c, [50](#)
  - ums.h, [67](#)
- get\_wt\_run\_by\_umst\_id
  - ums.c, [50](#)
  - ums.h, [67](#)
- get\_wt\_running\_time
  - ums.c, [51](#)
  - ums.h, [68](#)
- get\_wt\_with\_id
  - ums.c, [51](#)
  - ums.h, [68](#)
- id
  - completion\_list, [7](#)
  - ums\_thread\_context, [12](#)
  - ums\_thread\_entry, [14](#)
  - worker\_thread\_context, [18](#)
  - worker\_thread\_entry, [20](#)
- IDLE
  - ums.h, [59](#)
- info\_entry
  - ums\_thread\_entry, [15](#)
- init\_device
  - device.c, [24](#)
  - device.h, [25](#)
- init\_proc
  - proc.c, [33](#)
  - proc.h, [39](#)
- init\_ums\_process
  - ums.c, [51](#)
  - ums.h, [68](#)
- last\_switch\_time
  - ums\_thread\_context, [12](#)
  - worker\_thread\_context, [18](#)
- list
  - completion\_list, [7](#)
  - worker\_thread\_context, [18](#)
- module.c, [28](#)
- module.h, [29](#)
- PAUSE
  - device\_shared.h, [28](#)
- pid
  - process, [8](#)
  - process\_entry, [9](#)
- proc.c, [30](#)
  - create\_process\_entry, [31](#)
  - create\_umst\_entry, [31](#)
  - create\_wt\_entry, [31](#)
  - exit\_proc, [32](#)
  - get\_process\_entry\_with\_pid, [32](#)
  - get\_ums\_thread\_entry\_with\_id, [32](#)
  - init\_proc, [33](#)
  - process\_entry\_list, [33](#)
  - ums\_thread\_entry\_list, [33](#)
  - worker\_thread\_entry\_list, [34](#)
- proc.h, [34](#)
  - create\_process\_entry, [37](#)
  - create\_umst\_entry, [37](#)
  - create\_wt\_entry, [37](#)
  - exit\_proc, [38](#)
  - get\_process\_entry\_with\_pid, [38](#)
  - get\_ums\_thread\_entry\_with\_id, [38](#)
  - init\_proc, [39](#)
  - process\_entry\_list\_t, [36](#)
  - process\_entry\_t, [36](#)
  - ums\_thread\_entry\_list\_t, [36](#)
  - ums\_thread\_entry\_t, [36](#)
  - worker\_thread\_entry\_list\_t, [36](#)
  - worker\_thread\_entry\_t, [36](#)
- process, [7](#)
  - cl\_list, [8](#)
  - pid, [8](#)
  - ums\_thread\_list, [8](#)
  - worker\_thread\_list, [8](#)
- process\_count
  - process\_list, [11](#)
- process\_entry, [9](#)
  - entry, [9](#)

- pid, 9
- schedulers\_entry, 9
- process\_entry\_count
  - process\_entry\_list, 10
- process\_entry\_list, 10
  - proc.c, 33
  - process\_entry\_count, 10
- process\_entry\_list\_t
  - proc.h, 36
- process\_entry\_t
  - proc.h, 36
- process\_list, 10
  - process\_count, 11
  - ums.c, 54
- process\_list\_t
  - ums.h, 57
- process\_t
  - ums.h, 57
- READY
  - ums.h, 59
- regs
  - ums\_thread\_context, 13
  - worker\_thread\_context, 19
- ret\_regs
  - ums\_thread\_context, 13
- run\_by
  - ums\_thread\_context, 13
  - worker\_thread\_context, 19
- RUNNING
  - ums.h, 59
- running\_time
  - worker\_thread\_context, 19
- schedulers\_entry
  - process\_entry, 9
- stack\_address
  - worker\_thread\_params, 22
- stack\_size
  - worker\_thread\_params, 22
- state
  - ums\_thread\_context, 13
  - worker\_thread\_context, 19
- switch\_back\_to\_ums\_thread
  - ums.c, 52
  - ums.h, 69
- switch\_count
  - ums\_thread\_context, 13
  - worker\_thread\_context, 19
- switch\_to\_worker\_thread
  - ums.c, 53
  - ums.h, 70
- switching\_time
  - ums\_thread\_context, 13
- ums.c, 40
  - add\_to\_completion\_list, 42
  - convert\_from\_ums\_thread, 42
  - convert\_to\_ums\_thread, 43
  - create\_completion\_list, 44
  - create\_ums\_thread, 44
  - create\_worker\_thread, 45
  - dequeue\_completion\_list\_items, 45
  - exit\_ums, 46
  - exit\_ums\_process, 46
  - free\_process, 46
  - free\_process\_cl\_list, 47
  - free\_process\_ums\_thread\_list, 47
  - free\_process\_worker\_thread\_list, 48
  - get\_cl\_with\_id, 48
  - get\_process\_with\_pid, 48
  - get\_ready\_wt\_list, 49
  - get\_umst\_run\_by\_pid, 49
  - get\_umst\_switching\_time, 50
  - get\_umst\_with\_id, 50
  - get\_wt\_run\_by\_umst\_id, 50
  - get\_wt\_running\_time, 51
  - get\_wt\_with\_id, 51
  - init\_ums\_process, 51
  - process\_list, 54
  - switch\_back\_to\_ums\_thread, 52
  - switch\_to\_worker\_thread, 53
- ums.h, 54
  - add\_to\_completion\_list, 59
  - BUSY, 59
  - cl\_list\_t, 57
  - completion\_list\_t, 57
  - convert\_from\_ums\_thread, 59
  - convert\_to\_ums\_thread, 60
  - create\_completion\_list, 61
  - create\_ums\_thread, 61
  - create\_worker\_thread, 62
  - dequeue\_completion\_list\_items, 62
  - exit\_ums, 63
  - exit\_ums\_process, 63
  - FINISHED, 59
  - free\_process, 63
  - free\_process\_cl\_list, 64
  - free\_process\_ums\_thread\_list, 64
  - free\_process\_worker\_thread\_list, 65
  - get\_cl\_with\_id, 65
  - get\_process\_with\_pid, 65
  - get\_ready\_wt\_list, 66
  - get\_umst\_run\_by\_pid, 66
  - get\_umst\_switching\_time, 67
  - get\_umst\_with\_id, 67
  - get\_wt\_run\_by\_umst\_id, 67
  - get\_wt\_running\_time, 68
  - get\_wt\_with\_id, 68
  - IDLE, 59
  - init\_ums\_process, 68
  - process\_list\_t, 57
  - process\_t, 57
  - READY, 59
  - RUNNING, 59
  - switch\_back\_to\_ums\_thread, 69
  - switch\_to\_worker\_thread, 70

- ums\_state, 58
- ums\_thread\_context\_t, 58
- ums\_thread\_list\_t, 58
- worker\_state, 59
- worker\_thread\_context\_t, 58
- worker\_thread\_list\_t, 58
- ums\_state
  - ums.h, 58
- ums\_thread\_context, 11
  - avg\_switching\_time, 12
  - cl\_id, 12
  - created\_by, 12
  - entry\_point, 12
  - fpu\_regs, 12
  - id, 12
  - last\_switch\_time, 12
  - regs, 13
  - ret\_regs, 13
  - run\_by, 13
  - state, 13
  - switch\_count, 13
  - switching\_time, 13
  - wt\_id, 13
- ums\_thread\_context\_t
  - ums.h, 58
- ums\_thread\_count
  - ums\_thread\_list, 16
- ums\_thread\_entry, 14
  - created\_by, 14
  - entry, 14
  - id, 14
  - info\_entry, 15
  - workers\_entry, 15
- ums\_thread\_entry\_count
  - ums\_thread\_entry\_list, 15
- ums\_thread\_entry\_list, 15
  - proc.c, 33
  - ums\_thread\_entry\_count, 15
- ums\_thread\_entry\_list\_t
  - proc.h, 36
- ums\_thread\_entry\_t
  - proc.h, 36
- ums\_thread\_list, 16
  - process, 8
  - ums\_thread\_count, 16
- ums\_thread\_list\_t
  - ums.h, 58
- ums\_thread\_params, 16
  - completion\_list\_id, 17
  - function, 17
- worker\_state
  - ums.h, 59
- worker\_thread\_context, 17
  - cl\_id, 18
  - created\_by, 18
  - entry\_point, 18
  - fpu\_regs, 18
  - id, 18
  - last\_switch\_time, 18
  - list, 18
  - regs, 19
  - run\_by, 19
  - running\_time, 19
  - state, 19
  - switch\_count, 19
  - wt\_list, 19
- worker\_thread\_context\_t
  - ums.h, 58
- worker\_thread\_count
  - completion\_list, 7
  - worker\_thread\_list, 21
- worker\_thread\_entry, 20
  - entry, 20
  - id, 20
- worker\_thread\_entry\_count
  - worker\_thread\_entry\_list, 21
- worker\_thread\_entry\_list, 20
  - proc.c, 34
  - worker\_thread\_entry\_count, 21
- worker\_thread\_entry\_list\_t
  - proc.h, 36
- worker\_thread\_entry\_t
  - proc.h, 36
- worker\_thread\_id
  - add\_wt\_params, 5
- worker\_thread\_list, 21
  - process, 8
  - worker\_thread\_count, 21
- worker\_thread\_list\_t
  - ums.h, 58
- worker\_thread\_params, 22
  - function, 22
  - function\_args, 22
  - stack\_address, 22
  - stack\_size, 22
- workers\_entry
  - ums\_thread\_entry, 15
- wt\_id
  - ums\_thread\_context, 13
- wt\_list
  - completion\_list, 7
  - worker\_thread\_context, 19
- yield\_reason
  - device\_shared.h, 27