# UMS module

Generated by Doxygen 1.9.2

# Chapter 1

# Data Structure Index

## 1.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 2

# File Index

## 2.1 File List

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

# 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

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

### 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

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

### 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

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

### 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 FINSIH 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 invokation |

## 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
85 } yield_reason_t;
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

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

## 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

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

## 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

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

### 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**

| | |
|---|---|
| *req_pid* | 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**

| | |
|---|---|
| *req_pid* | the pid that created the scheduler |
| *id* | 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

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

### 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**

| | |
|---|---|
| *req_pid* | 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**

| | |
|---|---|
| *req_pid* | the pid that created the scheduler |
| *id* | 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
55 typedef struct process_entry_list {
56     struct list_head list;
57     unsigned int process_entry_count;
58 } process_entry_list_t;
59
66 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 
77 typedef struct worker_thread_entry_list {
78     struct list_head list;
79     unsigned int worker_thread_entry_count;
80 } worker_thread_entry_list_t;
81 
88 typedef struct process_entry {
89     pid_t pid;
90     struct list_head list;
91     struct proc_dir_entry *entry;
92     struct proc_dir_entry *schedulers_entry;
93 } process_entry_t;
94 
101 typedef struct ums_thread_entry {
102     unsigned int id;
103     pid_t created_by;
104     struct list_head list;
105     struct proc_dir_entry *entry;
106     struct proc_dir_entry *workers_entry;
107     struct proc_dir_entry *info_entry;
108 } ums_thread_entry_t;
109 
116 typedef struct worker_thread_entry {
117     unsigned int id;
118     struct list_head list;
119     struct proc_dir_entry *entry;
120 } worker_thread_entry_t;
121 
122 /*
123  * Functions
124  */
125 int init_proc(void);
126 void exit_proc(void);
127 int create_process_entry(pid_t pid);
128 int create_umst_entry(pid_t pid, unsigned int umst_id);
129 int create_wt_entry(unsigned int umst_id, unsigned int wt_id);
130 
131 /*
132  * Auxiliary functions
133  */
134 process_entry_t *get_process_entry_with_pid(pid_t req_pid);
135 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 runnning 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.



**Author**

Sultan Umarbaev 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**

| | |
|---|---|
| *params* | 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 convertion 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 convertion 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**

| | |
|---|---|
| *ums_thread↩ _id* | 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 Additionaly, 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 competion list associated with ums thread(scheduler).

**Parameters**

| | |
|---|---|
| *params* | 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**

| | |
|---|---|
| *params* | 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 rnnable worker thread ids with the help of auxiliary function get_ready_wt_list(). Copy data from temporary array into array allocated by user.

**Parameters**

| | |
|---|---|
| *ready_wt_list* | 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() funciton.

**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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *completion_list_id* | 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**

| | |
|---|---|
| *req_pid* | 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**

| | |
|---|---|
| *completion_list* | the pointer to the completion list from which to get worker thread ids |
| *ready_wt_list* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *req_pid* | 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**

| | |
|---|---|
| *ums_thread_context* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *ums_thread_id* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *ums_thread↩ _id* | 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 runnning time of the worker thread.

**Parameters**

| | |
|---|---|
| *worker_thread_context* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *worker_thread↩ _id* | 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 enbled 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 Additionaly, 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 convertion 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 auxiiary 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**

| | |
|---|---|
| *yield_reason* | reason which defines if worker thread should be paused or finished, yield_reason_t |

**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 convertion 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 userpace will update list of ready worker threads.

**Parameters**

| | |
|---|---|
| *worker_thread↩ _id* | 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 runnning 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.



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

**Author**

Sultan Umarbaev 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**

| | |
|---|---|
| *params* | 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 convertion 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 convertion 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**

| | |
|---|---|
| *ums_thread_id* | 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 Additionaly, 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 competion list associated with ums thread(scheduler).

**Parameters**

| | |
|---|---|
| *params* | 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**

| | |
|---|---|
| *params* | 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 rnnable worker thread ids with the help of auxiliary function get_ready_wt_list(). Copy data from temporary array into array allocated by user.

**Parameters**

| | |
|---|---|
| *ready_wt_list* | 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() funciton.

**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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *completion_list_id* | 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**

| | |
|---|---|
| *req_pid* | 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**

| | |
|---|---|
| *completion_list* | the pointer to the completion list from which to get worker thread ids |
| *ready_wt_list* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *req_pid* | 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**

| | |
|---|---|
| *ums_thread_context* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *ums_thread_id* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *ums_thread↩ _id* | 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 runnning time of the worker thread.

**Parameters**

| | |
|---|---|
| *worker_thread_context* | 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**

| | |
|---|---|
| *process* | the pointer to the process structure of the current process |
| *worker_thread↩ _id* | 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 enbled 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 Additionaly, 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 convertion 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 auxiiary 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**

| | |
|---|---|
| *yield_reason* | reason which defines if worker thread should be paused or finished, yield_reason_t |

**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 convertion 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 userpace will update list of ready worker threads.

**Parameters**

| | |
|---|---|
| *worker_thread↩ _id* | 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
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
67 typedef struct process_list {
68     struct list_head list;
69     unsigned int process_count;
70 } process_list_t;
71
78 typedef struct cl_list {
79     struct list_head list;
80     unsigned int cl_count;
81 } cl_list_t;
82
89 typedef struct worker_thread_list {
90     struct list_head list;
91     unsigned int worker_thread_count;
92 } worker_thread_list_t;
93
100 typedef struct ums_thread_list {
101     struct list_head list;
102     unsigned int ums_thread_count;
103 } ums_thread_list_t;
104
115 typedef struct process {
116     pid_t pid;
117     cl_list_t cl_list;
118     worker_thread_list_t worker_thread_list;
119     ums_thread_list_t ums_thread_list;
120     struct list_head list;
121 } process_t;
122
129 typedef struct completion_list {
130     struct list_head list;
131     struct list_head wt_list;
132     unsigned int id;
133     unsigned int worker_thread_count;
134 } completion_list_t;
135
136
141 typedef enum worker_state {
142     BUSY,
143     READY,
144     FINISHED
145 } worker_state_t;
146
153 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 
173 typedef enum ums_state {
174     RUNNING,
175     IDLE
176 } ums_state_t;
177 
184 typedef struct ums_thread_context {
185     unsigned int id;
186     struct list_head list;
187     unsigned long entry_point;
188     unsigned int cl_id;
189     unsigned int wt_id;
190     pid_t created_by;
191     pid_t run_by;
192     ums_state_t state;
193     unsigned int switch_count;
194     struct timespec64 last_switch_time;
195     unsigned long switching_time;
196     unsigned long avg_switching_time;
197     struct pt_regs regs;
198     struct fpu fpu_regs;
199     struct pt_regs ret_regs;
201 } ums_thread_context_t;
202 
203 /*
204  * Functions
205  */
206 int init_ums_process(void);
207 int exit_ums_process(void);
208 void exit_ums(void);
209 int create_completion_list(void);
210 int create_worker_thread(worker_thread_params_t *params);
211 int add_to_completion_list(add_wt_params_t *params);
212 int create_ums_thread(ums_thread_params_t *params);
213 int convert_to_ums_thread(unsigned int ums_thread_id);
214 int convert_from_ums_thread(void);
215 int dequeue_completion_list_items(int *runnable_wt_ptr);
216 int switch_to_worker_thread(unsigned int worker_thread_id);
217 int switch_back_to_ums_thread(yield_reason_t yield_reason);
218 
219 /*
220  * Auxiliary functions
221  */
222 process_t *get_process_with_pid(pid_t req_pid);
223 completion_list_t *get_cl_with_id(process_t *process, unsigned int completion_list_id);
224 worker_thread_context_t *get_wt_with_id(process_t *process, unsigned int worker_thread_id);
225 int get_ready_wt_list(completion_list_t *completion_list, unsigned int *ready_wt_list);
226 worker_thread_context_t *get_wt_run_by_umst_id(process_t *process, unsigned int ums_thread_id);
227 ums_thread_context_t *get_umst_with_id(process_t *process, unsigned int ums_thread_id);
228 ums_thread_context_t *get_umst_run_by_pid(process_t *process, pid_t req_pid);
229 int free_process_ums_thread_list(process_t *process);
230 int free_process_cl_list(process_t *process);
231 int free_process_worker_thread_list(process_t *process);
232 int free_process(process_t *process);
233 unsigned long get_wt_running_time(worker_thread_context_t *worker_thread_context);
234 unsigned long get_umst_switching_time(ums_thread_context_t *ums_thread_context);
```

# Index