User Mode thread Scheduling (kernel module)

Generated by Doxygen 1.9.3

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 completion_queue_descriptor Struct Reference

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

`#include <ums.h>`

Data Fields

- struct list_head completion_queue
- int id
- unsigned int used_by_couter
- struct hlist_node hlist

3.1.1 Detailed Description

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

3.1.2 Field Documentation

3.1.2.1 completion_queue

`struct list_head completion_queue_descriptor::completion_queue`

list of all the worker thread in the cq

3.1.2.2 hlist

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

hlist node for the htable

3.1.2.3 id

```
int completion_queue_descriptor::id
```

id of the completion queue

3.1.2.4 used_by_couter

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

conter of the ums that use this cq

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

- ums.h

3.2 cq_proc Struct Reference

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

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

Data Fields

- struct proc_dir_entry * dir_entry
- struct proc_dir_entry * link_to
- int cq_id
- char name_dir [NAME_BUFF]
- char path [NAME_BUFF]
- struct hlist_node hlist

3.2.1 Detailed Description

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

3.2.2 Field Documentation

3.2.2.1 cq_id

```
int cq_proc::cq_id
```

id of the completion queue

3.2.2.2 dir_entry

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

proc_dir_entry of the directory for the completion queue

3.2.2.3 hlist

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

hlist_node used for htable

3.2.2.4 link_to

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

will contain the proc_dir_entry of the link to the UMS scheduler that use this completion queue

3.2.2.5 name_dir

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

name of the completion queue dir (id to string)

3.2.2.6 path

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

path to reach the completion queue dir inside of the CQ_ALL_DIR

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

- proc.h

3.3 owner_proc Struct Reference

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

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

Data Fields

- struct proc_dir_entry * dir_entry
- struct proc_dir_entry * sched_entry
- unsigned pid
- char name_dir [NAME_BUFF]
- struct hlist_node hlist

3.3.1 Detailed Description

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

3.3.2 Field Documentation

3.3.2.1 dir_entry

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

directory of the owner

3.3.2.2 hlist

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

hlist_node used for htable

3.3.2.3 name_dir

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

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

3.3.2.4 pid

```
unsigned owner_proc::pid
```

pid of the owner

3.3.2.5 sched_entry

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

directory that will contain all the ums schedulers

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

- proc.h

3.4 ums_cq_param Struct Reference

Data Fields

- int completion_queue_id
- int pids [COMPLETION_QUEUE_BUFF]

3.4.1 Field Documentation

3.4.1.1 completion_queue_id

```
int ums_cq_param::completion_queue_id
```

id of the completion queue

3.4.1.2 pids

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

array with pid of the worker threads

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

- shared.h

3.5 ums_km_param Struct Reference

Data Fields

- int cq_id
- int owner_pid

3.5.1 Field Documentation

3.5.1.1 cq_id

```
int ums_km_param::cq_id
```

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

3.5.1.2 owner_pid

```
int ums_km_param::owner_pid
```

process that has request the creation of the ums

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

- shared.h

3.6 ums_proc Struct Reference

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

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

Data Fields

- struct proc_dir_entry * dir_entry
- struct proc_dir_entry * info_entry
- struct proc_dir_entry * link_to
- unsigned pid_owner
- unsigned pid
- int cq_id
- char name_dir [NAME_BUFF]
- char path [NAME_BUFF]
- struct hlist_node hlist

3.6.1 Detailed Description

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

3.6.2 Field Documentation

3.6.2.1 cq_id

```
int ums_proc::cq_id
```

id of the completion queue used by the UMS thread

3.6.2.2 dir_entry

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

proc_dir_entry of the directory for the UMS thread

3.6.2.3 hlist

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

hlist_node used for htable

3.6.2.4 info_entry

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

proc_dir_entry of the file containing the info of the UMS thread

3.6.2.5 link_to

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

will contain the proc_dir_entry of the link to the owner dir

3.6.2.6 name_dir

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

name of the UMS thread dir (pid to string)

3.6.2.7 path

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

path to reach the UMS thread dir inside of the UMS_ALL_DIR

3.6.2.8 pid

```
unsigned ums_proc::pid
```

pid of the UMD thread

3.6.2.9 pid_owner

```
unsigned ums_proc::pid_owner
```

pid of the owner of the UMS thread

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

- proc.h

3.7 ums_scheduler Struct Reference

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

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

Data Fields

- int pid
- struct task_struct * task_struct
- int owner_pid
- int ums_cq_id
- struct list_head * cq_list
- int pid_wkt_sched
- worker_thread_t * wkt_sched_struct
- struct pt_regs saved_pt_regs
- struct fpu saved_fpu_regs
- unsigned long total_switch
- unsigned long last_wkt_runtime
- long state
- struct list_head list
- struct hlist_node hlist

3.7.1 Detailed Description

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

3.7.2 Field Documentation

3.7.2.1 cq_list

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

completion_queue of this scheduler

3.7.2.2 hlist

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

for implementing the master_ums_hashlist

3.7.2.3 last_wkt_runtime

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

time needed for the last worker thread switch

3.7.2.4 list

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

all the scheduler are into a global list

3.7.2.5 owner_pid

```
int ums_scheduler::owner_pid
```

pid of the parent process that has generated the ums

3.7.2.6 pid

```
int ums_scheduler::pid
```

pid of the ums thread (is also its id)

3.7.2.7 pid_wkt_sched

```
int ums_scheduler::pid_wkt_sched
```

pid of the worker scheduled or -1

3.7.2.8 saved_fpu_regs

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

saved fpu register

3.7.2.9 saved_pt_regs

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

current context of the cpu registers

3.7.2.10 state

```
long ums_scheduler::state
```

current state of the scheduler IDLE | RUNNING

3.7.2.11 task_struct

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

task_struct of the ums thread

3.7.2.12 total_switch

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

total number of switch done by this scheduler

3.7.2.13 ums_cq_id

```
int ums_scheduler::ums_cq_id
```

id of the completion queue used by this ums

3.7.2.14 wkt_sched_struct

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

datastructire of the wkt scheduled

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

- ums.h

3.8 worker_proc Struct Reference

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

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

Data Fields

- struct proc_dir_entry * dir_entry
- struct proc_dir_entry * info_entry
- struct proc_dir_entry * link_to
- unsigned pid
- char name_dir [NAME_BUFF]
- char path [NAME_BUFF]
- struct hlist_node hlist

3.8.1 Detailed Description

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

3.8.2 Field Documentation

3.8.2.1 dir_entry

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

proc_dir_entry of the directory of the worker thread

3.8.2.2 hlist

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

hlist_node used for htable

3.8.2.3 info_entry

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

proc_dir_entry of the file containing the info of the worker thread

3.8.2.4 link_to

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

will contain the proc_dir_entry of the link to the completion queue dir

3.8.2.5 name_dir

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

name of the worker dir (pid to string)

3.8.2.6 path

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

path to reach the worker dir inside of the WKT_ALL_DIR

3.8.2.7 pid

```
unsigned worker_proc::pid
```

pid of the worker threead

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

- proc.h

3.9 worker_thread Struct Reference

Struct with the representation of the worker thread.

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

Data Fields

- int pid
- struct task_struct * task_struct
- struct pt_regs saved_pt_regs
- struct fpu saved_fpu_regs
- int scheduled_by
- unsigned long time_at_switch
- unsigned long total_switch
- unsigned long total_runtime
- long state
- struct list_head list
- struct hlist_node hlist

3.9.1 Detailed Description

Struct with the representation of the worker thread.

3.9.2 Field Documentation

3.9.2.1 hlist

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

for implementing the master_wt_hashlist

3.9.2.2 list

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

for implementing the ready queue

3.9.2.3 pid

```
int worker_thread::pid
```

pid of the worker thread

3.9.2.4 saved_fpu_regs

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

saved fpu register

3.9.2.5 saved_pt_regs

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

current context of the cpu registers

3.9.2.6 scheduled_by

```
int worker_thread::scheduled_by
```

pid of the scheduler that has scheduled this thread

3.9.2.7 state

```
long worker_thread::state
```

current state of the worker

3.9.2.8 task_struct

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

task_struct of the worker thread

3.9.2.9 time_at_switch

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

3.9.3 of switch that this work thread caused

3.9.3.1 total_runtime

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

total running time of the thread

3.9.3.2 total_switch

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

3.9.4 of switch that this work thread caused

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

- ums.h

Chapter 4

File Documentation

4.1 device.c File Reference

contain the main ioctl switch for the requested ops

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

Functions

- **DEFINE_SPINLOCK** (ioctl_spinlock)
- int try_start_device (void)

  try to register the char device
- void **stop_device** (void)

  deregister the device

Variables

- char * cmds [UMS_IOC_MAXNR+1]

4.1.1 Detailed Description

contain the main ioctl switch for the requested ops

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.1.2 Function Documentation

4.1.2.1 try_start_device()

```
int try_start_device (
            void  )
```

try to register the char device

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

Returns

int

4.1.3 Variable Documentation

4.1.3.1 cmds

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

Initial value:

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

4.2 device.h File Reference

The header of the device section of the module.

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

Macros

- #define **MODULE_DEV_LOG** "UMS_DEV: "

Functions

- int try_start_device (void)

  try to register the char device
- void **stop_device** (void)

  deregister the device

4.2.1 Detailed Description

The header of the device section of the module.

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.2.2 Function Documentation

4.2.2.1 try_start_device()

```
int try_start_device (
            void  )
```

try to register the char device

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

Returns

int

4.3 device.h

Go to the documentation of this file.

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

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

4.4 module.c File Reference

This file contains the implementation of the kernel module.

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

Functions

- **MODULE_LICENSE** ("GPL")
- **MODULE_AUTHOR** ("Tiziano Colagrossi <tiziano.colagrossi@gmail.com>")
- **MODULE_DESCRIPTION** ("User Module Thread Scheduling kernel module")
- **MODULE_VERSION** ("1.0.0")
- **module_init** (ums_module_init)
- **module_exit** (ums_module_exit)

4.4.1 Detailed Description

This file contains the implementation of the kernel module.

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.5 module.h File Reference

The header of the kernel module.

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

Macros

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

Functions

- int **init_core** (void)
- void **destroy_core** (void)

4.5.1 Detailed Description

The header of the kernel module.

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.6 module.h

Go to the documentation of this file.

```
1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10  * WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12  * General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program. If not, see <http://www.gnu.org/licenses/>.
16  */
17
26 #ifndef __MODULE_HEADER
27 #define __MODULE_HEADER
28
29 #include "shared.h"
30 #include "device.h"
31 #include "ums.h"
32
33 #include <linux/init.h>
34
35
36 #define MODULE_LOG "UMS: "
37
38 #define UMS_MOD_DEBUG
39
40
41 int init_core(void);
42 void destroy_core(void);
43
44 #endif
```

4.7 proc.c File Reference

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

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

Functions

- **DEFINE_HASHTABLE** (own_htable, HASH_KEY_SIZE)
- **DEFINE_HASHTABLE** (ums_htable, HASH_KEY_SIZE)
- **DEFINE_HASHTABLE** (wkt_htable, HASH_KEY_SIZE)
- **DEFINE_HASHTABLE** (cq_htable, HASH_KEY_SIZE)
- int Proc_Update_Worker_Created (int wkt_pid)

  Insert the created worker thread into the procfs.
- int Proc_Update_Worker_Ended (int wkt_pid)

  Remove the worker that has ended from the procfs.
- int Proc_Update_Worker_Appended (int wkt_pid, int id)

  link a worker to its completion queue
- int Proc_Update_Ums_Created (int ums_pid, int owner_pid, int id)

  Insert the new UMS into the proc fs and link the completion queue path to the UMS folder.
- int Proc_Update_Ums_Ended (int ums_pid)

  Remove the UMS directory from the proc fs.
- int Proc_Update_Cq_Created (int id)

  create a completion queue folder into the proc fs
- int Proc_Update_Cq_Deleted (int id)

  remove the completion queue folder from the proc fs
- int try_build_ums_proc (void)

  initialize the main data structures for the proc part for the ums
- int clear_ums_proc (void)

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

4.7.1 Detailed Description

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

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.7.2 Function Documentation

4.7.2.1 clear_ums_proc()

```
int clear_ums_proc (
            void  )
```

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

Returns

int

4.7.2.2 Proc_Update_Cq_Created()

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

create a completion queue folder into the proc fs

Parameters

id	completion queue id

Returns

int

4.7.2.3 Proc_Update_Cq_Deleted()

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

remove the completion queue folder from the proc fs

Parameters

id	completion queue id

Returns

int

4.7.2.4 Proc_Update_Ums_Created()

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

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

Parameters

ums_pid	pid of the UMS
owner_pid	pid of the owner process of the UMS
id	completion queue id of the UMS

Returns

int

4.7.2.5 Proc_Update_Ums_Ended()

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

Remove the UMS directory from the proc fs.

Parameters

ums_pid	pid of the ums

Returns

int

4.7.2.6 Proc_Update_Worker_Appended()

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

link a worker to its completion queue

Parameters

wkt_pid	pid of the worker
id	completion queue where the worker need to be appended

Returns

int

4.7.2.7 Proc_Update_Worker_Created()

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

Insert the created worker thread into the procfs.

Parameters

wkt_pid	pid of the worker

Returns

int

4.7.2.8 Proc_Update_Worker_Ended()

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

Remove the worker that has ended from the procfs.

Parameters

wkt_pid	pid of the worker

Returns

int

4.7.2.9 try_build_ums_proc()

```
int try_build_ums_proc (
            void  )
```

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

Returns

int

4.8 proc.h File Reference

This file is the header of the proc.c file.

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

Data Structures

- struct owner_proc

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

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

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

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

Macros

- #define **MODULE_PROC_LOG** "UMS_PROC: "
- #define **UMS_PROC_DEBUG**
- #define **HASH_KEY_SIZE** 10
- #define **NAME_BUFF** 30
- #define **PROC_DIR** "ums"
- #define **SCHED_DIR** "schedulers"
- #define **WKT_DIR** "workers"
- #define **UMS_INFO** "ums_info"
- #define **WKT_INFO** "wkt_info"
- #define **UMS_ALL_DIR** ".all_ums"
- #define **WKT_ALL_DIR** ".all_wkt"
- #define **CQ_ALL_DIR** ".all_cq"

Typedefs

- typedef struct owner_proc **owner_proc_t**

  contains the data of the ums owner directory that contain the scheduler dir with all the UMS thread created by the owner
- typedef struct worker_proc **worker_proc_t**

  contains the data of the worker thread directory that contain the info file of the worker.
- typedef struct ums_proc **ums_proc_t**

  contains the data of the UMS thread directory that will contain the info file of the UMS and the directory of the completion queue used by the UMS.
- typedef struct cq_proc **cq_proc_t**

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

Functions

- int Proc_Update_Worker_Created (int pid)

  Insert the created worker thread into the procfs.
- int Proc_Update_Worker_Ended (int wkt_pid)

  Remove the worker that has ended from the procfs.
- int Proc_Update_Worker_Appended (int wkt_pid, int id)

  link a worker to its completion queue
- int Proc_Update_Ums_Created (int pid, int pid_owner, int id)

  Insert the new UMS into the proc fs and link the completion queue path to the UMS folder.
- int Proc_Update_Ums_Ended (int ums_pid)

  Remove the UMS directory from the proc fs.
- int Proc_Update_Cq_Created (int id)

  create a completion queue folder into the proc fs
- int Proc_Update_Cq_Deleted (int id)

  remove the completion queue folder from the proc fs
- int try_build_ums_proc (void)

  initialize the main data structures for the proc part for the ums
- int clear_ums_proc (void)

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

4.8.1 Detailed Description

This file is the header of the proc.c file.

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.8.2 Function Documentation

4.8.2.1 clear_ums_proc()

```
int clear_ums_proc (
            void  )
```

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

Returns

int

4.8.2.2 Proc_Update_Cq_Created()

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

create a completion queue folder into the proc fs

Parameters

id	completion queue id

Returns

int

4.8.2.3 Proc_Update_Cq_Deleted()

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

remove the completion queue folder from the proc fs

Parameters

id	completion queue id

Returns

int

4.8.2.4 Proc_Update_Ums_Created()

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

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

Parameters

ums_pid	pid of the UMS
owner_pid	pid of the owner process of the UMS
id	completion queue id of the UMS

Returns

int

4.8.2.5 Proc_Update_Ums_Ended()

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

Remove the UMS directory from the proc fs.

Parameters

ums_pid	pid of the ums

Returns

int

4.8.2.6 Proc_Update_Worker_Appended()

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

link a worker to its completion queue

Parameters

wkt_pid	pid of the worker
id	completion queue where the worker need to be appended

Returns

int

4.8.2.7 Proc_Update_Worker_Created()

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

Insert the created worker thread into the procfs.

Parameters

wkt_pid	pid of the worker

Returns

int

4.8.2.8 Proc_Update_Worker_Ended()

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

Remove the worker that has ended from the procfs.

Parameters

wkt_pid	pid of the worker

Returns

int

4.8.2.9 try_build_ums_proc()

```
int try_build_ums_proc (
            void  )
```

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

Returns

int

4.9 proc.h

Go to the documentation of this file.

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

```
28
29 #include "utility.h"
30 #include "ums.h"
31
32 #include <linux/kernel.h>
33 #include <linux/module.h>
34 #include <linux/slab.h>
35 #include <linux/proc_fs.h>
36 #include <linux/seq_file.h>
37 #include <linux/hashtable.h>
38
39 #define MODULE_PROC_LOG "UMS_PROC: "
40
41 #define UMS_PROC_DEBUG
42
43 #define HASH_KEY_SIZE 10
44 #define NAME_BUFF     30
45
46 #define PROC_DIR      "ums"
47 #define SCHED_DIR     "schedulers"
48 #define WKT_DIR       "workers"
49 #define UMS_INFO      "ums_info"
50 #define WKT_INFO      "wkt_info"
51
52 #define UMS_ALL_DIR   ".all_ums"
53 #define WKT_ALL_DIR   ".all_wkt"
54 #define  CQ_ALL_DIR   ".all_cq"
55
56
62 typedef struct owner_proc{
63     struct proc_dir_entry *   dir_entry;
64     struct proc_dir_entry * sched_entry;
65     unsigned                      pid;
66     char            name_dir[NAME_BUFF];
67     struct hlist_node             hlist;
68 }owner_proc_t;
69
74 typedef struct worker_proc{
75     struct proc_dir_entry *   dir_entry;
76     struct proc_dir_entry *  info_entry;
77     struct proc_dir_entry *     link_to;
78     unsigned                      pid;
79     char            name_dir[NAME_BUFF];
80     char                path[NAME_BUFF];
81     struct hlist_node             hlist;
82 }worker_proc_t;
83
89 typedef struct ums_proc{
90     struct proc_dir_entry *   dir_entry;
91     struct proc_dir_entry *  info_entry;
92     struct proc_dir_entry *     link_to;
93     unsigned                  pid_owner;
94     unsigned                      pid;
95     int                          cq_id;
96     char            name_dir[NAME_BUFF];
97     char                path[NAME_BUFF];
98     struct hlist_node             hlist;
99 }ums_proc_t;
100
105 typedef struct cq_proc{
106     struct proc_dir_entry *   dir_entry;
107     struct proc_dir_entry *     link_to;
108     int                          cq_id;
109     char             name_dir[NAME_BUFF];
110     char                path[NAME_BUFF];
111     struct hlist_node             hlist;
112 }cq_proc_t;
113
114 /*
115 * Function exported for create file and directory inside the procfs
116 */
117 int Proc_Update_Worker_Created  (int pid    );
118 int Proc_Update_Worker_Ended    (int wkt_pid);
119 int Proc_Update_Worker_Appended (int wkt_pid , int id      );
120 int Proc_Update_Ums_Created     (int pid     , int pid_owner, int id);
121 int Proc_Update_Ums_Ended       (int ums_pid);
122 int Proc_Update_Cq_Created      (int id     );
123 int Proc_Update_Cq_Deleted      (int id     );
124
125 /*
126 * Function used to initialize this part of the UMS kernel module
127 */
128 int try_build_ums_proc (void);
129 int clear_ums_proc     (void);
130
131 #endif
```

4.10 shared.h File Reference

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

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

Data Structures

- struct ums_km_param
- struct ums_cq_param

Macros

- #define **MODULE_NAME** "ums"
- #define **MODULE_PATH** "/dev/ums"
- #define **EXIT_SUCCESS** 0
- #define **EXIT_FAILURE** -1
- #define **THREAD_RUNNG** -3
- #define **CQ_FULL** -4
- #define **UMS_IOC_MAGIC** 0xF4
- #define **UMS_IOCRESET** _IO (UMS_IOC_MAGIC, 0)
- #define **UMS_IOC_THREAD_YIELD** _IO (UMS_IOC_MAGIC, 1)
- #define **UMS_IOC_THREAD_EXECUTE** _IOW (UMS_IOC_MAGIC, 2, unsigned)
- #define **UMS_IOC_ENTER_UMS_SCHEDULING_MODE** _IOW (UMS_IOC_MAGIC, 3, unsigned long)
- #define **UMS_IOC_END_UMS_SCHEDULER** _IO (UMS_IOC_MAGIC, 4)
- #define **UMS_IOC_INIT_WORKER_THREAD** _IO (UMS_IOC_MAGIC, 5)
- #define **UMS_IOC_END_WORKER_THREAD** _IO (UMS_IOC_MAGIC, 6)
- #define **UMS_IOC_INIT_COMPLETION_QUEUE** _IOR (UMS_IOC_MAGIC, 7, unsigned long)
- #define **UMS_IOC_APPEND_TO_COMPLETION_QUEUE** _IOW (UMS_IOC_MAGIC, 8, int)
- #define **UMS_IOC_DEQUEUE_COMPLETION_LIST** _IOR (UMS_IOC_MAGIC, 9, unsigned long)
- #define **UMS_IOC_MAXNR** 9
- #define **COMPLETION_QUEUE_BUFF** 100

Typedefs

- typedef void ∗(∗ **worker_job_t**) (void ∗)
- typedef struct ums_km_param **ums_km_param_t**
- typedef struct ums_cq_param **ums_cq_param_t**

4.10.1 Detailed Description

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

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.11 shared.h

Go to the documentation of this file.

```
1 /*
2  * This file is part of the User Mode Thread Scheduling (Kernel Module).
3  * Copyright (c) 2021 Tiziano Colagrossi.
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU General Public License as published by
7  * the Free Software Foundation, version 3.
8  *
9  * This program is distributed in the hope that it will be useful, but
10  * WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12  * General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program. If not, see <http://www.gnu.org/licenses/>.
16  */
17
26 #ifndef __SHARED_HEADER
27 #define __SHARED_HEADER
28
29 #include <linux/ioctl.h>
30
31 #define MODULE_NAME "ums"
32 #define MODULE_PATH "/dev/ums"
33
34 #define EXIT_SUCCESS  0
35 #define EXIT_FAILURE -1
36 #define THREAD_RUNNG -3
37 #define CQ_FULL      -4
38
39 // Define ioctl command
40 #define UMS_IOC_MAGIC 0xF4
41
42 #define UMS_IOCRESET                       _IO  (UMS_IOC_MAGIC, 0)
43 #define UMS_IOC_THREAD_YIELD               _IO  (UMS_IOC_MAGIC, 1)
44 #define UMS_IOC_THREAD_EXECUTE             _IOW (UMS_IOC_MAGIC, 2, unsigned)
45 #define UMS_IOC_ENTER_UMS_SCHEDULING_MODE  _IOW (UMS_IOC_MAGIC, 3, unsigned long)
46 #define UMS_IOC_END_UMS_SCHEDULER          _IO  (UMS_IOC_MAGIC, 4)
47 #define UMS_IOC_INIT_WORKER_THREAD         _IO  (UMS_IOC_MAGIC, 5)
48 #define UMS_IOC_END_WORKER_THREAD          _IO  (UMS_IOC_MAGIC, 6)
49 #define UMS_IOC_INIT_COMPLETION_QUEUE      _IOR (UMS_IOC_MAGIC, 7, unsigned long)
50 #define UMS_IOC_APPEND_TO_COMPLETION_QUEUE _IOW (UMS_IOC_MAGIC, 8, int)
51 #define UMS_IOC_DEQUEUE_COMPLETION_LIST    _IOR (UMS_IOC_MAGIC, 9, unsigned long)
52
53
54 #define UMS_IOC_MAXNR 9
55 #define COMPLETION_QUEUE_BUFF 100
56
57 typedef void *(*worker_job_t)(void *);
58
59 typedef struct ums_km_param{
60     int     cq_id;
61     int owner_pid;
62 }ums_km_param_t;
63
64 typedef struct ums_cq_param{
65     int         completion_queue_id;
66     int pids[COMPLETION_QUEUE_BUFF];
67 }ums_cq_param_t;
68
69 #endif
```

4.12 ums.c File Reference

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

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

Functions

- DECLARE_BITMAP (cq_id_bitmap, BITMAP_CQ_SIZE)
- DEFINE_HASHTABLE (master_wkt_hashlist, HASH_KEY_SIZE)
- DEFINE_HASHTABLE (master_cq_hashlist, HASH_KEY_SIZE)
- DEFINE_HASHTABLE (master_ums_hashlist, HASH_KEY_SIZE)
- int yield_to_ums (spinlock_t ioctl_lock)

  perform a context switch from wkt to ums that host it
- int execute_wkt (spinlock_t ioctl_lock, unsigned *u_wkt_pid)

  perform a context switch from ums to wkt selected
- int init_cq (spinlock_t ioctl_lock, void *cq_id_u_ptr)

  Initialize the struct for a new completion queue.
- int append_to_cq (spinlock_t ioctl_lock, ums_cq_param_t *args)

  Append worker threads to a completion queue.
- int dequeue_cq (spinlock_t ioctl_lock, ums_cq_param_t *ret_cq)

  return the first COMPLETION_QUEUE_BUFF size of worker thread id in the completion queue
- int init_ums_scheduler (spinlock_t ioctl_lock, ums_km_param_t *args)

  convert a standard pthread into an UmsSchedulerThread
- int end_ums_scheduler (spinlock_t ioctl_lock)

  clear the data structure used by the ums
- int init_worker_thread (spinlock_t ioctl_lock)

  block at startup the worker_thread to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS
- int end_worker_thread (spinlock_t ioctl_lock)

  if SWITCH_PT_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task_struct. Also remove the worker_thread_struct from the completion queue and the hashtable of the worker thread. if SWITCH_PT_REGS is not defined wake up the UMS
- worker_thread_t * Get_WKT (int wkt_pid)

  retrive a pointer to worker_thread_t struct
- ums_scheduler_t * Get_UMS (int ums_pid)

  retrive a pointer to ums_scheduler_t struct
- ums_scheduler_t * Get_UMS_from_WKT (int wkt_pid)

  retrive a pointer to ums_scheduler_t struct
- char * Get_UMS_Info (int ums_pid)

  *retrive a char * with the info of the UMS in a human readable fashion*
- char * Get_WKT_Info (int wkt_pid)

  *retrive a char * with the info of the worker in a human readable fashion*
- void ums_do_wait (worker_thread_t *from_wkt, ums_scheduler_t *to_ums)

  handler used in the case that a worker thread is put on wait
- void ums_do_unwait (worker_thread_t *from_wkt, struct task_struct *p)

  handler used in the case that a worker thread previously put on wait is awakened
- int try_build_ums_core (void)

  Initialize the data structure needed for the UMS.
- void **clear_ums_core** (void)

  Destroy the ums data structure.

Variables

- char * state [5]

4.12.1 Detailed Description

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

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.12.2 Function Documentation

4.12.2.1 append_to_cq()

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

Append worker threads to a completion queue.

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

Parameters

ioctl_lock	
args	pointer to the ums_cq_param_t struct placed in the user space

Returns

int

4.12.2.2 DECLARE_BITMAP()

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

bitmap to keep track of the completion queue id

4.12.2.3 DEFINE_HASHTABLE() [1/3]

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

master hash table key:id data:ums_completion_queue_list_t

4.12.2.4 DEFINE_HASHTABLE() [2/3]

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

hashtable of all created ums_scheduler

4.12.2.5 DEFINE_HASHTABLE() [3/3]

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

master hash table key:pid data:worker_thread_t

4.12.2.6 dequeue_cq()

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

return the first COMPLETION_QUEUE_BUFF size of worker thread id in the completion queue

Parameters

ioctl_lock	
ret_cq	pointer to the ums_cq_param_t in the user space used to return the worker thread pid in the cq that are ready

Returns

int

4.12.2.7 end_ums_scheduler()

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

clear the data structure used by the ums

Parameters

ioctl_lock	

Returns

int

4.12.2.8 end_worker_thread()

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

if SWITCH_PT_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task_struct. Also remove the worker_thread_struct from the completion queue and the hashtable of the worker thread. if SWITCH_PT_REGS is not defined wake up the UMS

4.12.3 Implementation

Check done by the function:

- The current pid needs to be a UMS pid if SWITCH_PT_REGS is defined else the current pid represent the worker scheduled
- if SWITCH_PT_REGS is defined, try to retrive the worker_thread_t struct else try to retrive the UMS using the worker pid

After this check perform:

- Restore context of the worker into his own task_struct if SWITCH_PT_REGS is defined
- Restore ums context if SWITCH_PT_REGS is defined
- Remove the worker thread from the completion queue and from the hashtable and free the memory of the worker_thread_t struct
- Finally resume the worker thread
- If SWITCH_PT_REGS is not defined wake up also the UMS

Parameters

ioctl_lock	

Returns

int 0 if all went ok, else -1

4.12.3.1 execute_wkt()

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

perform a context switch from ums to wkt selected

4.12.4 Implementation

Initially the current check are done:

- Try to copy from user the value passed to the module by ioctl
- The current need to be an ums pid because only an UMS can schedule a worker thread.
- I try to retrive the worker_thread_t from the cq_list using the wkt_pid_to_switch passed by the user arg
- If the state of the worker selected is W_RUNNING the function end because the thread is scheduled by another UMS

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH_PT_REGS is defined, by saving the current state (pt_↩ regs and fxregs) into the ums_scheduler_t stuct and restoring the previously saved state of the worker thread from the worker_thread_t struct. if SWITCH_PT_REGS is not defined, by stop the UMS and waking up the worker thread.

Parameters

ioctl_lock	
u_wkt_pid	pointer to the user space mem of the id of the worker thread choose to be executed

Returns

int 0 if all went ok, else -1

4.12.4.1 Get_UMS()

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

retrive a pointer to ums_scheduler_t struct

Parameters

ums_pid	pid of the ums thread to find

Returns

ums_scheduler_t*

4.12.4.2 Get_UMS_from_WKT()

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

retrive a pointer to ums_scheduler_t struct

Parameters

wkt_pid	pid of the worker actual scheduled by the ums

Returns

ums_scheduler_t*

4.12.4.3 Get_UMS_Info()

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

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

Parameters

ums_pid	pid of the ums

Returns

char*

4.12.4.4 Get_WKT()

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

retrive a pointer to worker_thread_t struct

Parameters

wkt_pid	pid of the worker thread to find

Returns

worker_thread_t∗

4.12.4.5 Get_WKT_Info()

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

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

Parameters

wkt_pid	oid of the worker thread

Returns

char∗

4.12.4.6 init_cq()

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

Initialize the struct for a new completion queue.

4.12.5 Implementation

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

Parameters

ioctl_lock	
cq_id_u_ptr	pointer to the cq_id in user space

Returns

int 0 if all went ok, else -1

4.12.5.1 init_ums_scheduler()

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

convert a standard pthread into an UmsSchedulerThread

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

4.12.6 Implementation

Initially the current check are done:

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

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

- Create a new ums_scheduler_t struct for the new UMS created
- Initialize the struct with default value
- Link the completion queue to this UMS and update the used_by_couter entry in the completion_queue_↩ descriptor_t descriptor cause the completion queue can be shared among multiple UMS

Parameters

ioctl_lock	
args	pointer to the ums_km_param_t struct in user space

Returns

int 0 if all went ok, else -1

4.12.6.1 init_worker_thread()

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

block at startup the worker_thread to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS

4.12.7 Implementation

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

Parameters

ioctl_lock	

Returns

int 0 if all went ok, else -1

4.12.7.1 try_build_ums_core()

```
int try_build_ums_core (
            void  )
```

Initialize the data structure needed for the UMS.

It initialize the following hashtables:

- master_wkt_hashlist (key:pid , data:worker_thread_t)
- master_cq_hashlist (key:id , data:completion_queue_descriptor_t)
- master_ums_hashlist (key:pid , data:ums_scheduler_t)
- cq_id_bitmap (is used to check the avaiable id for the completion queues)

Returns

int

4.12.7.2 ums_do_unwait()

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

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

Parameters

from_wkt	pointer to worker_thread_t struct of the worker awakened
p	task_struct of the worker thread awakened

4.12.7.3 ums_do_wait()

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

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

Parameters

from_wkt	pointer to worker_thread_t struct of the worker thread that will put in wait
to_ums	pointer to ums_scheduler_t struct if the ums thread that will wakeup

4.12.7.4 yield_to_ums()

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

perform a context switch from wkt to ums that host it

4.12.8 Implementation

Initially the current check are done:

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

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH_PT_REGS is defined, by saving the current state (pt_↩ regs and fxregs) into the worker_thread_t struct and restoring the previously saved state of the ums from the ums_scheduler_t struct. if SWITCH_PT_REGS is not defined, by stop the worker thread and waking up the UMS scheduler that previously has scheduled the worker.

Parameters

ioctl_lock	

Returns

int 0 if all went ok, else -1

4.12.9 Variable Documentation

4.12.9.1 state

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

Initial value:

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

4.13 ums.h File Reference

This file is the header of ums.c.

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

Data Structures

- struct completion_queue_descriptor

  Is the representation of the completion queue contain the id of the completion queue and the completion queue itself.
- struct ums_scheduler

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

  Struct with the representation of the worker thread.

Macros

- #define **MODULE_UMS_LOG** "UMS_MASTER_FUNCS: "
- #define **F_DEQUEUE** "DEQUEUE: "
- #define **F_APPEND** "APPEND: "
- #define **F_DESTROY_CQ** "DESTROY_CQ: "
- #define **F_INIT_CQ** "INIT_CQ: "
- #define **F_INIT_WORKER** "INIT_WORKER: "
- #define **F_END_WORKER** "END_WORKER: "
- #define **F_INIT_UMS** "INIT_UMS: "
- #define **F_SCHED_WRK** "SCHED_WKT: "
- #define **F_SCHED_UMS** "SCHED_UMS: "
- #define **HASH_KEY_SIZE** 10
- #define **BITMAP_CQ_SIZE** 128
- #define **STATE_RUNNING** 0x0000
- #define **STATE_IDLE** 0x0001
- #define **STATE_READY** 0x0002
- #define **STATE_WAITING** 0x0004
- #define **ums_is_idle**(ums) ((ums->state & STATE_IDLE) != 0)
- #define **ums_is_running**(ums) ((ums->state & STATE_RUNNING) != 0)
- #define **wkt_is_running**(wkt) ((wkt->state & STATE_RUNNING) != 0)
- #define **wkt_is_ready**(wkt) ((wkt->state & STATE_READY) != 0)
- #define **wkt_is_waiting**(wkt) ((wkt->state & STATE_WAITING) != 0)
- #define **fxsave**(fpu) copy_fxregs_to_kernel(fpu)
- #define **fxrestore**(fpu) copy_kernel_to_fxregs(&((fpu)->state.fxsave))
- #define **UMS_CORE_DEBUG**
- #define **SWITCH_PT_REGS**

Typedefs

- typedef struct ums_scheduler **ums_scheduler_t**

  This struct contain all the data related to a Ums scheduler.
- typedef struct worker_thread **worker_thread_t**

  Struct with the representation of the worker thread.
- typedef enum direction **direction_t**

  This enum represent the possible direction for update the stats.
- typedef struct completion_queue_descriptor **completion_queue_descriptor_t**

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

Enumerations

- enum direction { **YIELD** , **EXECUTE** , **WAIT** }

  This enum represent the possible direction for update the stats.

Functions

- int yield_to_ums (spinlock_t ioctl_lock)

  perform a context switch from wkt to ums that host it
- int execute_wkt (spinlock_t ioctl_lock, unsigned *u_wkt_pid)

  perform a context switch from ums to wkt selected
- int init_cq (spinlock_t ioctl_lock, void *cq_id_u_ptr)

  Initialize the struct for a new completion queue.
- int append_to_cq (spinlock_t ioctl_lock, ums_cq_param_t *args)

  Append worker threads to a completion queue.
- int dequeue_cq (spinlock_t ioctl_lock, ums_cq_param_t *ret_cq)

  return the first COMPLETION_QUEUE_BUFF size of worker thread id in the completion queue
- int init_ums_scheduler (spinlock_t ioctl_lock, ums_km_param_t *args)

  convert a standard pthread into an UmsSchedulerThread
- int end_ums_scheduler (spinlock_t ioctl_lock)

  clear the data structure used by the ums
- int init_worker_thread (spinlock_t ioctl_lock)

  block at startup the worker_thread to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS
- int end_worker_thread (spinlock_t ioctl_lock)

  if SWITCH_PT_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task_struct. Also remove the worker_thread_struct from the completion queue and the hashtable of the worker thread. if SWITCH_PT_REGS is not defined wake up the UMS
- worker_thread_t * Get_WKT (int wkt_pid)

  retrive a pointer to worker_thread_t struct
- ums_scheduler_t * Get_UMS (int ums_pid)

  retrive a pointer to ums_scheduler_t struct
- ums_scheduler_t * Get_UMS_from_WKT (int wkt_pid)

  retrive a pointer to ums_scheduler_t struct
- char * Get_UMS_Info (int ums_pid)

  *retrive a char * with the info of the UMS in a human readable fashion*
- char * Get_WKT_Info (int wkt_pid)

  *retrive a char * with the info of the worker in a human readable fashion*
- void ums_do_wait (worker_thread_t *from_wkt, ums_scheduler_t *to_ums)

  handler used in the case that a worker thread is put on wait
- void ums_do_unwait (worker_thread_t *from_wkt, struct task_struct *p)

  handler used in the case that a worker thread previously put on wait is awakened
- int try_build_ums_core (void)

  Initialize the data structure needed for the UMS.
- void **clear_ums_core** (void)

  Destroy the ums data structure.

4.13.1 Detailed Description

This file is the header of ums.c.

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.13.2 Function Documentation

4.13.2.1 append_to_cq()

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

Append worker threads to a completion queue.

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

Parameters

ioctl_lock	
args	pointer to the ums_cq_param_t struct placed in the user space

Returns

int

4.13.2.2 dequeue_cq()

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

return the first COMPLETION_QUEUE_BUFF size of worker thread id in the completion queue

Parameters

ioctl_lock	
ret_cq	pointer to the ums_cq_param_t in the user space used to return the worker thread pid in the cq that are ready

Returns

int

4.13.2.3 end_ums_scheduler()

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

clear the data structure used by the ums

Parameters

ioctl_lock	

Returns

int

4.13.2.4 end_worker_thread()

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

if SWITCH_PT_REGS is defined restore the original worker thread not hosted from the ums in order to permit the end of the worker thread and its task_struct. Also remove the worker_thread_struct from the completion queue and the hashtable of the worker thread. if SWITCH_PT_REGS is not defined wake up the UMS

4.13.3 Implementation

Check done by the function:

- The current pid needs to be a UMS pid if SWITCH_PT_REGS is defined else the current pid represent the worker scheduled
- if SWITCH_PT_REGS is defined, try to retrive the worker_thread_t struct else try to retrive the UMS using the worker pid

After this check perform:

- Restore context of the worker into his own task_struct if SWITCH_PT_REGS is defined
- Restore ums context if SWITCH_PT_REGS is defined
- Remove the worker thread from the completion queue and from the hashtable and free the memory of the worker_thread_t struct
- Finally resume the worker thread
- If SWITCH_PT_REGS is not defined wake up also the UMS

Parameters

ioctl_lock	

Returns

int 0 if all went ok, else -1

4.13.3.1 execute_wkt()

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

perform a context switch from ums to wkt selected

4.13.4 Implementation

Initially the current check are done:

- Try to copy from user the value passed to the module by ioctl
- The current need to be an ums pid because only an UMS can schedule a worker thread.
- I try to retrive the worker_thread_t from the cq_list using the wkt_pid_to_switch passed by the user arg
- If the state of the worker selected is W_RUNNING the function end because the thread is scheduled by another UMS

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH_PT_REGS is defined, by saving the current state (pt_↩ regs and fxregs) into the ums_scheduler_t stuct and restoring the previously saved state of the worker thread from the worker_thread_t struct. if SWITCH_PT_REGS is not defined, by stop the UMS and waking up the worker thread.

Parameters

ioctl_lock	
u_wkt_pid	pointer to the user space mem of the id of the worker thread choose to be executed

Returns

int 0 if all went ok, else -1

4.13.4.1 Get_UMS()

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

retrive a pointer to ums_scheduler_t struct

Parameters

ums_pid	pid of the ums thread to find

Returns

ums_scheduler_t*

4.13.4.2 Get_UMS_from_WKT()

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

retrive a pointer to ums_scheduler_t struct

Parameters

wkt_pid	pid of the worker actual scheduled by the ums

Returns

ums_scheduler_t*

4.13.4.3 Get_UMS_Info()

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

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

Parameters

ums_pid	pid of the ums

Returns

char*

4.13.4.4 Get_WKT()

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

retrive a pointer to worker_thread_t struct

Parameters

wkt_pid	pid of the worker thread to find

Returns

worker_thread_t∗

4.13.4.5 Get_WKT_Info()

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

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

Parameters

wkt_pid	oid of the worker thread

Returns

char∗

4.13.4.6 init_cq()

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

Initialize the struct for a new completion queue.

4.13.5 Implementation

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

Parameters

ioctl_lock	
cq_id_u_ptr	pointer to the cq_id in user space

Returns

int 0 if all went ok, else -1

4.13.5.1 init_ums_scheduler()

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

convert a standard pthread into an UmsSchedulerThread

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

4.13.6 Implementation

Initially the current check are done:

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

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

- Create a new ums_scheduler_t struct for the new UMS created
- Initialize the struct with default value
- Link the completion queue to this UMS and update the used_by_couter entry in the completion_queue_↩ descriptor_t descriptor cause the completion queue can be shared among multiple UMS

Parameters

ioctl_lock	
args	pointer to the ums_km_param_t struct in user space

Returns

int 0 if all went ok, else -1

4.13.6.1 init_worker_thread()

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

block at startup the worker_thread to avoid that the linux scheduler shedule this thread, and initialize its kernel struct used for the UMS

4.13.7 Implementation

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

Parameters

ioctl_lock	

Returns

int 0 if all went ok, else -1

4.13.7.1 try_build_ums_core()

```
int try_build_ums_core (
            void  )
```

Initialize the data structure needed for the UMS.

It initialize the following hashtables:

- master_wkt_hashlist (key:pid , data:worker_thread_t)
- master_cq_hashlist (key:id , data:completion_queue_descriptor_t)
- master_ums_hashlist (key:pid , data:ums_scheduler_t)
- cq_id_bitmap (is used to check the avaiable id for the completion queues)

Returns

int

4.13.7.2 ums_do_unwait()

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

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

Parameters

from_wkt	pointer to worker_thread_t struct of the worker awakened
p	task_struct of the worker thread awakened

4.13.7.3 ums_do_wait()

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

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

Parameters

from_wkt	pointer to worker_thread_t struct of the worker thread that will put in wait
to_ums	pointer to ums_scheduler_t struct if the ums thread that will wakeup

4.13.7.4 yield_to_ums()

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

perform a context switch from wkt to ums that host it

4.13.8 Implementation

Initially the current check are done:

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

After this checks:

- Update the stats for the UMS and the worker structs.
- Finally perform the actual context switch: if SWITCH_PT_REGS is defined, by saving the current state (pt_↩ regs and fxregs) into the worker_thread_t struct and restoring the previously saved state of the ums from the ums_scheduler_t struct. if SWITCH_PT_REGS is not defined, by stop the worker thread and waking up the UMS scheduler that previously has scheduled the worker.

Parameters

ioctl_lock	

Returns

int 0 if all went ok, else -1

4.14 ums.h

Go to the documentation of this file.

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

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

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

4.15 utility.h File Reference

This file contains utility macro definition.

Macros

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

  allocate and append to tail a new item to a list
- #define add_new_item_to_hlist(new_item_pointer, hashtable, node, item_type, identifier)

  allocate and append to tail a new item to a hlist
- #define get_hlist_item_by_id(getted_item_pointer, hashtable, node, member_identifier, identifier)

  retrive the item from an hashtable by the key
- #define delete_completion_queue_descriptor(cq_desc_to_delete, node)

  delete the completion queue descriptor from its hashtable and free it
- #define retrive_from_hlist(item_select, hashtable, node, member_identifier, identifier)

  retrive a node in an hlist by its identifier
- #define retrive_from_list(item_select, head, member, member_identifier, identifier)

  retrive a node in an hlist by its identifier
- #define ums_delete_list(entry_cursor, entry_cursor_safe, head, member)

  delete all the item from a list
- #define ums_delete_hlist(cursor, node_ptr, bucket, hashtable, node)

  delete all the item from an hlist

4.15.1 Detailed Description

This file contains utility macro definition.

Author

Tiziano Colagrossi tiziano.colagrossi@gmail.com

4.15.2 Macro Definition Documentation

4.15.2.1 add_new_item_to_hlist

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

Value:

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

allocate and append to tail a new item to a hlist

Parameters

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

4.15.2.2 add_new_item_to_list

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

Value:

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

allocate and append to tail a new item to a list

Parameters

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

4.15.2.3 delete_completion_queue_descriptor

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

Value:

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

delete the completion queue descriptor from its hashtable and free it

Parameters

cq_desc_to_delete	will be filled with the retrived node.
node	the &struct hlist_node of the object to be retrived

4.15.2.4 get_hlist_item_by_id

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

Value:

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

retrive the item from an hashtable by the key

Parameters

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

4.15.2.5 retrive_from_hlist

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

Value:

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

retrive a node in an hlist by its identifier

Parameters

hashtable	hashtable where to search
item_select	will be filled with the retrived node
node	the &struct hlist_node of the object to be retrived
member_identifier	the name of the identifier within the struct
identifier	the key of the object to find

4.15.2.6 retrive_from_list

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

Value:

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

retrive a node in an hlist by its identifier

Parameters

item_select	will be filled with the retrived node
head	the head of the list
member	the &struct list_head of the object to be retrived
member_identifier	the name of the identifier within the struct
identifier	the key of the object to find

4.15.2.7 ums_delete_hlist

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

Value:

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

delete all the item from an hlist

Parameters

cursor	the type * to use as a loop cursor.
bucket	integer to use as bucket loop cursor
hashtable	hashtable where to search
node	the &struct hlist_node of the object to be retrived

4.15.2.8 ums_delete_list

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

Value:

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

delete all the item from a list

Parameters

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

4.16 utility.h

Go to the documentation of this file.

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

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

Index