UMS: Kernel 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 completion_list Struct Reference

The list of the completion lists created by the specific process

`#include <ums_api.h>`

Data Fields

- struct list_head **list**
- unsigned int list_count

3.1.1 Detailed Description

The list of the completion lists created by the specific process

3.1.2 Field Documentation

3.1.2.1 list_count

`unsigned int completion_list::list_count`

Number of completion lists created by the process

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

- ums_api.h

3.2 completion_list_node Struct Reference

Represents a node in the completion_list

`#include <ums_api.h>`

Data Fields

- ums_clid_t clid
- struct list_head **list**
- unsigned int worker_count
- unsigned int finished_count
- state_t state
- worker_list_t * idle_list
- worker_list_t * busy_list

3.2.1 Detailed Description

Represents a node in the completion_list

3.2.2 Field Documentation

3.2.2.1 busy_list

```
worker_list_t* completion_list_node::busy_list
```

List of worker threads that has been completed or currently running

3.2.2.2 clid

```
ums_clid_t completion_list_node::clid
```

Completion list ID

3.2.2.3 finished_count

```
unsigned int completion_list_node::finished_count
```

Number of worker threads that has completed their work

3.2.2.4 idle_list

```
worker_list_t* completion_list_node::idle_list
```

List of worker threads that are ready and waiting to be scheduled

3.2.2.5 state

```
state_t completion_list_node::state
```

State of the completion list

3.2.2.6 worker_count

```
unsigned int completion_list_node::worker_count
```

Number of worker threads assigned to the completion list

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

- ums_api.h

3.3 list_params Struct Reference

Parameters that are created by the scheduler and passed to dequeue the completion list items

```
#include <const.h>
```

Data Fields

- unsigned int size
- unsigned int worker_count
- state_t state
- ums_wid_t workers [ ]

3.3.1 Detailed Description

Parameters that are created by the scheduler and passed to dequeue the completion list items

3.3.2 Field Documentation

3.3.2.1 size

```
unsigned int list_params::size
```

Size of the worker thread array

3.3.2.2 state

```
state_t list_params::state
```

Tracks the state of the completion list which is set by the kernel module after a dequeue call

3.3.2.3 worker_count

```
unsigned int list_params::worker_count
```

Tracks the quantity of the available workers and used as state indicator for scheduler to perform a new dequeue call

3.3.2.4 workers

```
ums_wid_t list_params::workers[]
```

Array of worker threads. Stores ID of worker threads in case they are available to be scheduled (when worker thread is finished, scheduler replaces ID with -1 value)

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

- const.h

3.4 process Struct Reference

Represents a node in the process_list

```
#include <ums_api.h>
```

Data Fields

- pid_t pid
- struct list_head **list**
- state_t state
- completion_list_t * completion_lists
- worker_list_t * worker_list
- scheduler_list_t * scheduler_list
- process_proc_entry_t * proc_entry

3.4.1 Detailed Description

Represents a node in the process_list

3.4.2 Field Documentation

3.4.2.1 completion_lists

```
completion_list_t* process::completion_lists
```

List of completions lists created by the process

3.4.2.2 pid

```
pid_t process::pid
```

pid of the process or tgid of the process threads

3.4.2.3 proc_entry

```
process_proc_entry_t* process::proc_entry
```

Proc entries of the process

3.4.2.4 scheduler_list

```
scheduler_list_t* process::scheduler_list
```

List of schedulers created by the process

3.4.2.5 state

```
state_t process::state
```

State of the process

3.4.2.6 worker_list

```
worker_list_t* process::worker_list
```

List of worker threads created by the process

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

- ums_api.h

3.5 process_list Struct Reference

The list of the processes handled by the UMS kernel module

`#include <ums_api.h>`

Data Fields

- struct list_head **list**
- unsigned int process_count

3.5.1 Detailed Description

The list of the processes handled by the UMS kernel module

3.5.2 Field Documentation

3.5.2.1 process_count

`unsigned int process_list::process_count`

Number of processes handled by the UMS kernel module

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

- ums_api.h

3.6 process_proc_entry Struct Reference

Responsible for tracking proc_dir_entries of the process

`#include <ums_api.h>`

Data Fields

- struct proc_dir_entry * pde
- struct proc_dir_entry * parent
- struct proc_dir_entry * child

3.6.1 Detailed Description

Responsible for tracking proc_dir_entries of the process

3.6.2 Field Documentation

3.6.2.1 child

```
struct proc_dir_entry* process_proc_entry::child
```

Child folder of the process' folder (/proc/ums/<PID>/schedulers/)

3.6.2.2 parent

```
struct proc_dir_entry* process_proc_entry::parent
```

Parent folder of the process' folder (/proc/ums/)

3.6.2.3 pde

```
struct proc_dir_entry* process_proc_entry::pde
```

proc_dir_entry of the process (/proc/ums/<PID>/)

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

- ums_api.h

3.7 scheduler Struct Reference

Represents a node in the scheduler_list

```
#include <ums_api.h>
```

Data Fields

- ums_sid_t sid
- pid_t pid
- pid_t tid
- ums_wid_t wid
- unsigned long entry_point
- unsigned long return_addr
- unsigned long stack_ptr
- unsigned long base_ptr
- state_t state
- struct pt_regs regs
- struct fpu fpu_regs
- completion_list_node_t * comp_list
- struct list_head **list**
- scheduler_proc_entry_t * proc_entry
- unsigned int switch_count
- unsigned long avg_switch_time
- unsigned long time_needed_for_the_last_switch
- unsigned long total_time_needed_for_the_switch
- struct timespec64 time_of_the_last_switch

3.7.1 Detailed Description

Represents a node in the scheduler_list

3.7.2 Field Documentation

3.7.2.1 avg_switch_time

```
unsigned long scheduler::avg_switch_time
```

Average time needed for context switch

3.7.2.2 base_ptr

```
unsigned long scheduler::base_ptr
```

Snapshot of the base pointer that is restored when exiting scheduling mode

3.7.2.3 comp_list

```
completion_list_node_t* scheduler::comp_list
```

Pointer of the completion list that is associated with the scheduler

3.7.2.4 entry_point

```
unsigned long scheduler::entry_point
```

Function pointer and an entry point set by a user, that serves as a starting point of the scheduler. It is a scheduling function that determines the next thread to be scheduled

3.7.2.5 fpu_regs

```
struct fpu scheduler::fpu_regs
```

Snapshot of FPU registers

3.7.2.6 pid

```
pid_t scheduler::pid
```

pid of the process thread that is currently running the scheduler

3.7.2.7 proc_entry

```
scheduler_proc_entry_t* scheduler::proc_entry
```

Proc entry of the scheduler

3.7.2.8 regs

```
struct pt_regs scheduler::regs
```

Snapshot of CPU registers

3.7.2.9 return_addr

```
unsigned long scheduler::return_addr
```

Snapshot of the instruction pointer that is restored when exiting scheduling mode

3.7.2.10 sid

```
ums_sid_t scheduler::sid
```

Scheduler ID

3.7.2.11 stack_ptr

```
unsigned long scheduler::stack_ptr
```

Snapshot of the stack pointer that is restored when exiting scheduling mode

3.7.2.12 state

```
state_t scheduler::state
```

State of the scheduler

3.7.2.13 switch_count

```
unsigned int scheduler::switch_count
```

Number of context switches

3.7.2.14 tid

```
pid_t scheduler::tid
```

pid of the process that created the scheduler

3.7.2.15 time_needed_for_the_last_switch

```
unsigned long scheduler::time_needed_for_the_last_switch
```

Time needed for the last context switch

3.7.2.16 time_of_the_last_switch

```
struct timespec64 scheduler::time_of_the_last_switch
```

Time when the last switch occured

3.7.2.17 total_time_needed_for_the_switch

```
unsigned long scheduler::total_time_needed_for_the_switch
```

Total time needed for the context switches

3.7.2.18 wid

```
ums_wid_t scheduler::wid
```

ID of the worker that is managed by the scheduler

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

- ums_api.h

3.8 scheduler_list Struct Reference

The list of the schedulers created by the specific process

```
#include <ums_api.h>
```

Data Fields

- struct list_head **list**
- unsigned int scheduler_count

3.8.1 Detailed Description

The list of the schedulers created by the specific process

3.8.2 Field Documentation

3.8.2.1 scheduler_count

```
unsigned int scheduler_list::scheduler_count
```

Number of schedulers created by the process

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

- ums_api.h

3.9 scheduler_params Struct Reference

Parameters that are passed in order to create a scheduler

```
#include <const.h>
```

Data Fields

- unsigned long entry_point
- ums_clid_t clid
- ums_sid_t sid
- int core_id

3.9.1 Detailed Description

Parameters that are passed in order to create a scheduler

3.9.2 Field Documentation

3.9.2.1 clid

```
ums_clid_t scheduler_params::clid
```

ID of the completion list that is assigned to the scheduler

3.9.2.2 core_id

```
int scheduler_params::core_id
```

ID of the CPU core that is assigned to the scheduler (It is handled automatically by the library, no user input required)

3.9.2.3 entry_point

```
unsigned long scheduler_params::entry_point
```

Function pointer and an entry point set by a user, that serves as a starting point of the scheduler. It is a scheduling function that determines the next thread to be scheduled

3.9.2.4 sid

```
ums_sid_t scheduler_params::sid
```

ID of the scheduler which is set by the kernel module

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

- const.h

3.10 scheduler_proc_entry Struct Reference

Responsible for tracking proc_dir_entries of the schedulers of the specific process

```
#include <ums_api.h>
```

Data Fields

- struct proc_dir_entry * pde
- struct proc_dir_entry * parent
- struct proc_dir_entry * child
- struct proc_dir_entry * info

3.10.1 Detailed Description

Responsible for tracking proc_dir_entries of the schedulers of the specific process

3.10.2 Field Documentation

3.10.2.1 child

```
struct proc_dir_entry* scheduler_proc_entry::child
```

Child folder of the scheduler's folder (/proc/ums/<PID>/schedulers/<Scheduler ID>/workers/)

3.10.2.2 info

```
struct proc_dir_entry* scheduler_proc_entry::info
```

File that contains information and statistics of the scheduler performance (/proc/ums/<PID>/schedulers/<↩ Scheduler ID>/info)

3.10.2.3 parent

```
struct proc_dir_entry* scheduler_proc_entry::parent
```

Parent folder of the scheduler's folder (/proc/ums/<PID>/schedulers/)

3.10.2.4 pde

```
struct proc_dir_entry* scheduler_proc_entry::pde
```

proc_dir_entry of the scheduler of the specific process (/proc/ums/<PID>/schedulers/<Scheduler ID>/)

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

- ums_api.h

3.11 worker Struct Reference

Represents a node in the worker_list

```
#include <ums_api.h>
```

Data Fields

- ums_wid_t wid
- pid_t pid
- pid_t tid
- ums_sid_t sid
- ums_clid_t clid
- unsigned long entry_point
- unsigned long stack_addr
- struct pt_regs regs
- struct fpu fpu_regs
- struct list_head global_list
- struct list_head local_list
- state_t state
- worker_proc_entry_t * proc_entry
- unsigned int switch_count
- unsigned long total_exec_time
- struct timespec64 time_of_the_last_switch

3.11.1 Detailed Description

Represents a node in the worker_list

3.11.2 Field Documentation

3.11.2.1 clid

```
ums_clid_t worker::clid
```

ID of the completion list where worker thread is assigned to

3.11.2.2 entry_point

```
unsigned long worker::entry_point
```

Function pointer and an entry point set by a user, that serves as a starting point of the worker thread ∗

3.11.2.3 fpu_regs

```
struct fpu worker::fpu_regs
```

Snapshot of FPU registers

3.11.2.4 global_list

```
struct list_head worker::global_list
```

List of the worker threads created by the process

3.11.2.5 local_list

```
struct list_head worker::local_list
```

List of the worker threads of the completion list

3.11.2.6 pid

```
pid_t worker::pid
```

pid of the process thread that is currently running the worker thread

3.11.2.7 proc_entry

```
worker_proc_entry_t* worker::proc_entry
```

Proc entry of the worker thread

3.11.2.8 regs

```
struct pt_regs worker::regs
```

Snapshot of CPU registers

3.11.2.9 sid

```
ums_sid_t worker::sid
```

ID of the scheduler which manages the current worker thread

3.11.2.10 stack_addr

```
unsigned long worker::stack_addr
```

Address of the stack allocated by the UMS library

3.11.2.11 state

```
state_t worker::state
```

State of worker thread's progress

3.11.2.12 switch_count

```
unsigned int worker::switch_count
```

Number of context switches

3.11.2.13 tid

```
pid_t worker::tid
```

pid of the process that created the worker thread

3.11.2.14 time_of_the_last_switch

```
struct timespec64 worker::time_of_the_last_switch
```

Time when the last switch occured

3.11.2.15 total_exec_time

```
unsigned long worker::total_exec_time
```

Total execution time of the worker thread

3.11.2.16 wid

```
ums_wid_t worker::wid
```

Worker thread ID

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

- ums_api.h

3.12 worker_list Struct Reference

The list of the worker threads

```
#include <ums_api.h>
```

Data Fields

- struct list_head **list**
- unsigned int worker_count

3.12.1 Detailed Description

The list of the worker threads

3.12.2 Field Documentation

3.12.2.1 worker_count

```
unsigned int worker_list::worker_count
```

Number of worker threads created by the process

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

- ums_api.h

3.13 worker_params Struct Reference

Parameters that are passed in order to create a worker thread

`#include <const.h>`

Data Fields

- unsigned long entry_point
- unsigned long function_args
- unsigned long stack_size
- unsigned long stack_addr
- ums_clid_t clid

3.13.1 Detailed Description

Parameters that are passed in order to create a worker thread

3.13.2 Field Documentation

3.13.2.1 clid

```
ums_clid_t worker_params::clid
```

ID of the completion list where worker thread is assigned to

3.13.2.2 entry_point

```
unsigned long worker_params::entry_point
```

Function pointer and an entry point set by a user, that serves as a starting point of the worker thread

3.13.2.3 function_args

```
unsigned long worker_params::function_args
```

Pointer of the function arguments that are passed to the entry point/function

3.13.2.4 stack_addr

```
unsigned long worker_params::stack_addr
```

Address of the stack allocated by the UMS library

3.13.2.5 stack_size

```
unsigned long worker_params::stack_size
```

Stack size of the worker thread set by a user

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

- const.h

3.14 worker_proc_entry Struct Reference

Responsible for tracking proc_dir_entries of the worker of the specific process

```
#include <ums_api.h>
```

Data Fields

- struct proc_dir_entry * pde
- struct proc_dir_entry * parent

3.14.1 Detailed Description

Responsible for tracking proc_dir_entries of the worker of the specific process

3.14.2 Field Documentation

3.14.2.1 parent

```
struct proc_dir_entry* worker_proc_entry::parent
```

Parent folder of the worker thread's file (/proc/ums/<PID>/schedulers/<Scheduler ID>/workers/)

3.14.2.2 pde

```
struct proc_dir_entry* worker_proc_entry::pde
```

proc_dir_entry of the worker thread of the completion list that is assigned to a specific scheduler and contains information/statistics regarding worker thread's performance (/proc/ums/<PID>/schedulers/<Scheduler ID>/workers/<Worker ID>)

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

- ums_api.h

Chapter 4

File Documentation

4.1 const.h File Reference

Set of data structures and other constant variables used by UMS kernel module.

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

Data Structures

- struct list_params

  Parameters that are created by the scheduler and passed to dequeue the completion list items
- struct worker_params

  Parameters that are passed in order to create a worker thread
- struct scheduler_params

  Parameters that are passed in order to create a scheduler

Macros

- #define **UMS_NAME** "ums"
- #define **UMS_DEVICE** "/dev/ums"
- #define **UMS_IOC_MAGIC** 'R'
- #define **UMS_IOC_DEVICE_NAME** "ums_sched"
- #define **UMS_MODULE_NAME_LOG** "ums_sched: "
- #define **UMS_PROC_NAME_LOG** "/proc/ums: "
- #define **UMS_MINOR** MISC_DYNAMIC_MINOR
- #define **UMS_BUFFER_LEN** 64
- #define **UMS_ENTER** _IO(UMS_IOC_MAGIC, 1)
- #define **UMS_EXIT** _IO(UMS_IOC_MAGIC, 2)
- #define **UMS_CREATE_LIST** _IO(UMS_IOC_MAGIC, 3)
- #define **UMS_CREATE_WORKER** _IOW(UMS_IOC_MAGIC, 4, unsigned long)
- #define **UMS_ENTER_SCHEDULING_MODE** _IOWR(UMS_IOC_MAGIC, 5, unsigned long)
- #define **UMS_EXIT_SCHEDULING_MODE** _IO(UMS_IOC_MAGIC, 6)
- #define **UMS_EXECUTE_THREAD** _IOW(UMS_IOC_MAGIC, 7, unsigned long)
- #define **UMS_THREAD_YIELD** _IOW(UMS_IOC_MAGIC, 8, unsigned long)
- #define **UMS_DEQUEUE_COMPLETION_LIST_ITEMS** _IOWR(UMS_IOC_MAGIC, 9, unsigned long)

- #define **UMS_SUCCESS** 0

  Succesful execution.
- #define **UMS_ERROR** 1

  Error.
- #define **UMS_ERROR_PROCESS_NOT_FOUND** 1000

  Process is not managed by UMS kernel module.
- #define **UMS_ERROR_PROCESS_ALREADY_EXISTS** 1001

  Process is already managed by UMS kernel module.
- #define **UMS_ERROR_COMPLETION_LIST_NOT_FOUND** 1002

  Completion list cannot be found.
- #define **UMS_ERROR_SCHEDULER_NOT_FOUND** 1003

  Scheduler cannot be found.
- #define **UMS_ERROR_WORKER_NOT_FOUND** 1004

  Worker thread cannot be found.
- #define **UMS_ERROR_STATE_RUNNING** 1005

  The object is still running, thus cannot be modified, updated, deleted.
- #define **UMS_ERROR_CMD_IS_NOT_ISSUED_BY_MAIN_THREAD** 1006

  The command is not issued by the main process thread, e.g. ums_exit()
- #define **UMS_ERROR_WORKER_ALREADY_RUNNING** 1007

  The worker thread is already running.
- #define **UMS_ERROR_WRONG_INPUT** 1008

  Wrong input.
- #define **UMS_ERROR_CMD_IS_NOT_ISSUED_BY_SCHEDULER** 1009

  The command is not issued by the scheduler.
- #define **UMS_ERROR_CMD_IS_NOT_ISSUED_BY_WORKER** 1010

  The command is not issued by the worker.
- #define **UMS_ERROR_WORKER_ALREADY_FINISHED** 1011

  The worker thread has already finished execution.
- #define **UMS_ERROR_NO_AVAILABLE_WORKERS** 1012

  No worker threads are available.
- #define **UMS_ERROR_COMPLETION_LIST_ALREADY_FINISHED** 1013

  All worker threads in the completion list have finished execution.
- #define **UMS_ERROR_FAILED_TO_CREATE_PROC_ENTRY** 1014

  Failed to create proc entry.
- #define **UMS_ERROR_FAILED_TO_PROC_OPEN** 1015

  Failed to open proc entry.
- #define **UMS_ERROR_COMPLETION_LIST_IS_USED_AND_CANNOT_BE_MODIFIED** 1016

  The completion list is being used, thus cannot be modified.

Typedefs

- typedef enum state **state_t**

  States of processes, completion lists and threads (schedulers, worker threads)
- typedef enum worker_status **worker_status_t**

  Status of the worker thread Used as a parameter that is passed for pausing or completing the worker thread.
- typedef unsigned int **ums_sid_t**

  Scheduler ID
- typedef unsigned int **ums_wid_t**

  Worker thread ID
- typedef unsigned int **ums_clid_t**

Completion list ID

- typedef struct list_params **list_params_t**

  Parameters that are created by the scheduler and passed to dequeue the completion list items
- typedef struct worker_params **worker_params_t**

  Parameters that are passed in order to create a worker thread
- typedef struct scheduler_params **scheduler_params_t**

  Parameters that are passed in order to create a scheduler

Enumerations

- enum state { IDLE , RUNNING , FINISHED }

  States of processes, completion lists and threads (schedulers, worker threads)
- enum worker_status { PAUSE , FINISH }

  Status of the worker thread Used as a parameter that is passed for pausing or completing the worker thread.

4.1.1 Detailed Description

Set of data structures and other constant variables used by UMS kernel module.

Author

Bektur Umarbaev hrafnulf13@gmail.com

Date

4.1.2 Enumeration Type Documentation

4.1.2.1 state

```
enum state
```

States of processes, completion lists and threads (schedulers, worker threads)

Enumerator

IDLE	Represents the state when worker thread is waiting to be scheduled; When scheduler waits or searches for available worker threads to run; Completion list has available worker threads to be scheduled
RUNNING	Represents the state when worker thread is scheduled and ran by the scheduler; When scheduler handles worker thread; Completion list is currently used and can't be modified
FINISHED	Represents the state when worker thread has been completed; When scheduler has completed all scheduling work with a completion list; All completion list's worker threads has been completed

4.1.2.2 worker_status

```
enum worker_status
```

Status of the worker thread Used as a parameter that is passed for pausing or completing the worker thread.

Enumerator

PAUSE	Used for pausing a worker thread: ums_thread_pause() == ums_thread_yield(PAUSE)
FINISH	Used for completing a worker thread: ums_thread_exit() == ums_thread_yield(FINISH)

4.2 const.h

Go to the documentation of this file.

```
1
28 #pragma once
29
30 #include <linux/ioctl.h>
31
32 /*
33  * Definitions
34  */
35 #define UMS_NAME            "ums"
36 #define UMS_DEVICE          "/dev/ums"
37 #define UMS_IOC_MAGIC       'R'
38 #define UMS_IOC_DEVICE_NAME "ums_sched"
39 #define UMS_MODULE_NAME_LOG "ums_sched: "
40 #define UMS_PROC_NAME_LOG   "/proc/ums: "
41 #define UMS_MINOR MISC_DYNAMIC_MINOR
42 #define UMS_BUFFER_LEN       64
43
44 /*
45  * IOCTL definitions
46  */
47 #define UMS_ENTER                          _IO(UMS_IOC_MAGIC, 1)
48 #define UMS_EXIT                           _IO(UMS_IOC_MAGIC, 2)
49 #define UMS_CREATE_LIST                    _IO(UMS_IOC_MAGIC, 3)
50 #define UMS_CREATE_WORKER                  _IOW(UMS_IOC_MAGIC, 4, unsigned long)
51 #define UMS_ENTER_SCHEDULING_MODE          _IOWR(UMS_IOC_MAGIC, 5, unsigned long)
52 #define UMS_EXIT_SCHEDULING_MODE           _IO(UMS_IOC_MAGIC, 6)
53 #define UMS_EXECUTE_THREAD                 _IOW(UMS_IOC_MAGIC, 7, unsigned long)
54 #define UMS_THREAD_YIELD                   _IOW(UMS_IOC_MAGIC, 8, unsigned long)
55 #define UMS_DEQUEUE_COMPLETION_LIST_ITEMS  _IOWR(UMS_IOC_MAGIC, 9, unsigned long)
56
57 /*
58  * Errors and return values
59  */
60 #define UMS_SUCCESS                                                0

61 #define UMS_ERROR                                                  1
```

```
62 #define UMS_ERROR_PROCESS_NOT_FOUND                                1000
63 #define UMS_ERROR_PROCESS_ALREADY_EXISTS                           1001
64 #define UMS_ERROR_COMPLETION_LIST_NOT_FOUND                        1002
65 #define UMS_ERROR_SCHEDULER_NOT_FOUND                              1003
66 #define UMS_ERROR_WORKER_NOT_FOUND                                 1004
67 #define UMS_ERROR_STATE_RUNNING                                    1005
68 #define UMS_ERROR_CMD_IS_NOT_ISSUED_BY_MAIN_THREAD                 1006
69 #define UMS_ERROR_WORKER_ALREADY_RUNNING                           1007
70 #define UMS_ERROR_WRONG_INPUT                                      1008
71 #define UMS_ERROR_CMD_IS_NOT_ISSUED_BY_SCHEDULER                   1009
72 #define UMS_ERROR_CMD_IS_NOT_ISSUED_BY_WORKER                      1010
73 #define UMS_ERROR_WORKER_ALREADY_FINISHED                          1011
74 #define UMS_ERROR_NO_AVAILABLE_WORKERS                             1012
75 #define UMS_ERROR_COMPLETION_LIST_ALREADY_FINISHED                 1013
76 #define UMS_ERROR_FAILED_TO_CREATE_PROC_ENTRY                      1014
77 #define UMS_ERROR_FAILED_TO_PROC_OPEN                              1015
78 #define UMS_ERROR_COMPLETION_LIST_IS_USED_AND_CANNOT_BE_MODIFIED   1016
79
84 typedef enum state {
85     IDLE,
86     RUNNING,
87     FINISHED
88 } state_t;
89
94 typedef enum worker_status {
95     PAUSE,
96     FINISH
97 } worker_status_t;
98
103 typedef unsigned int ums_sid_t;
104
109 typedef unsigned int ums_wid_t;
110
115 typedef unsigned int ums_clid_t;
116
121 typedef struct list_params {
122     unsigned int size;
123     unsigned int worker_count;
124     state_t state;
125     ums_wid_t workers[];
126 } list_params_t;
127
132 typedef struct worker_params {
133     unsigned long entry_point;
134     unsigned long function_args;
135     unsigned long stack_size;
136     unsigned long stack_addr;
137     ums_clid_t clid;
138 } worker_params_t;
139
144 typedef struct scheduler_params {
145     unsigned long entry_point;
146     ums_clid_t clid;
147     ums_sid_t sid;
148     int core_id;
149 } scheduler_params_t;
```

4.3 ums_api.c File Reference

Contains implementations of the UMS kernel module API.

```
#include "ums_api.h"
```

Functions

- int enter_ums (void)

  Called by a process to request a scheduling management Checks if the process is already managed or not, if not:
- int exit_ums (void)

  Called by a process to request a completion of the scheduling management Checks if the process is already managed or not, if not:
- process_t * create_process_node (pid_t pid)

  Creates a process data structure to handle the specified process To create a process data structure, UMS kernel module:
- ums_clid_t create_completion_list ()

  Creates a completion_list_node for the process To create a completion_list_node, UMS kernel module:
- ums_wid_t create_worker_thread (worker_params_t *params)

  Creates a worker thread for the process To create a worker, UMS kernel module:
- ums_sid_t enter_scheduling_mode (scheduler_params_t *params)

  Converts a pthread to the scheduler.
- int exit_scheduling_mode (void)

  Converts scheduler back to the pthread.
- int execute_thread (ums_wid_t worker_id)

  Executes a worker thread with a `worker_id` To execute the worker thread:
- int thread_yield (worker_status_t status)

  Pauses or completes the execution of the worker thread To pause or complete the execution of the worker thread:
- int dequeue_completion_list_items (list_params_t *params)

  Provides a list of available worker threads of the completion list that can be scheduled To retrieve the list of available worker threads:
- process_t * check_if_process_exists (pid_t pid)

  Checks if `process` with `pid` is managed by the UMS kernel module
- completion_list_node_t * check_if_completion_list_exists (process_t *process, ums_clid_t clid)

  Checks if completion list with `clid` was created by a `process`
- scheduler_t * check_if_scheduler_exists (process_t *process, ums_sid_t sid)

  Checks if scheduler with `sid` was created by a `process`
- scheduler_t * check_if_scheduler_exists_run_by (process_t *process, pid_t pid)

  Checks if scheduler with `pid` was created by a `process`
- worker_t * check_if_worker_exists (worker_list_t *worker_list, ums_wid_t wid)

  Checks if worker thread with `wid` exists in `worker_list` of the completion list Search is performed using local_list member of worker.
- worker_t * check_if_worker_exists_global (worker_list_t *worker_list, ums_wid_t wid)

  Checks if worker thread with `wid` exists in `worker_list` of the process Search is performed using global_list member of worker.
- state_t check_schedulers_state (process_t *process)

  Checks if schedulers of the process have finished their work
- int delete_process (process_t *process)

  Deletes process from the global process_list The function was used during early phases of development and later was left unused, since proc entries of the process are linked to process (in case user wants to see statistics, the data should be available for read), it was decided that only kernel module can delete data structures allocated for the process Therefore delete_process_safe() is used to delete process Still the function performs a check to see if all schedulers have finished their job and then performs series of deletions of all data structures used by the process.
- int delete_process_safe (process_t *process)

  Called by UMS kernel module when module exits to delete process from the global process_list The function does not perform a check to see if all schedulers have finished their job to delete data structures used by the process It is assumed that all work has been done, therefore user issued a command to UMS kernel module to exit.
- int delete_completion_lists_and_worker_threads (process_t *process)

  Deletes completion lists and worker threads created by the process Calls delete_workers_from_completion_list() and frees allocated memory that was used by the completion lists.

- int delete_workers_from_completion_list (worker_list_t *worker_list)

  Deletes worker threads assigned to the completion list
- int delete_workers_from_process_list (worker_list_t *worker_list)

  Deletes worker threads created by the process Frees allocated memory that was used by the worker threads.
- int delete_schedulers (process_t *process)

  Deletes schedulers created by the process Frees allocated memory that was used by the schedulers.
- int cleanup ()

  Performs a cleanup by deleting all the allocated data structures for all processes that were managed by the UMS kernel module
- unsigned long get_exec_time (struct timespec64 *prev_time)

  Computes time difference between passed `prev_time` and current time, which is used in this case as an indicator of execution time for worker and scheduler
- int init_proc (void)

  Initializes the core proc directory for UMS kernel module
- int delete_proc (void)

  Deletes all proc files of the UMS kernel module
- int create_process_proc_entry (process_t *process)

  Dinamically creates essential proc entries for the process Allocates a memory for process_proc_entry and initializes it:
- int create_scheduler_proc_entry (process_t *process, scheduler_t *scheduler)

  Dinamically creates essential proc entries for the scheduler of the process Allocates a memory for scheduler_proc_entry and initializes it:
- int create_worker_proc_entry (process_t *process, scheduler_t *scheduler, worker_t *worker)

  Dinamically creates essential proc entries for the worker of the process Allocates a memory for worker_proc_entry and initializes it:

Variables

- process_list_t process_list

4.3.1 Detailed Description

Contains implementations of the UMS kernel module API.

Copyright (C) 2021 Bektur Umarbaev `hrafnulf13@gmail.com`

This file is part of the User Mode thread Scheduling (UMS) kernel module.

UMS 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 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 kernel module. If not, see `http://www.gnu.org/licenses/`.

Author

Bektur Umarbaev `hrafnulf13@gmail.com`

Date

4.3.2 Function Documentation

4.3.2.1 check_if_completion_list_exists()

```
completion_list_node_t * check_if_completion_list_exists (
            process_t * process,
            ums_clid_t clid )
```

Checks if completion list with `clid` was created by a `process`

Parameters

process	pointer to process
clid	Completion list ID

Returns

returns pointer to completion_list_node, or `NULL` if no completion list was found

4.3.2.2 check_if_process_exists()

```
process_t * check_if_process_exists (
            pid_t pid )
```

Checks if `process` with `pid` is managed by the UMS kernel module

Parameters

pid	pid of the process

Returns

returns pointer to process, or `NULL` if no process was found

4.3.2.3 check_if_scheduler_exists()

```
scheduler_t * check_if_scheduler_exists (
            process_t * process,
            ums_sid_t sid )
```

Checks if scheduler with `sid` was created by a `process`

Parameters

process	pointer to process
sid	Scheduler ID

Returns

returns pointer to scheduler, or `NULL` if no scheduler was found

4.3.2.4 check_if_scheduler_exists_run_by()

```
scheduler_t * check_if_scheduler_exists_run_by (
            process_t * process,
            pid_t pid )
```

Checks if scheduler with `pid` was created by a `process`

Parameters

process	pointer to process
pid	pid of the pthread which is associated with scheduler

Returns

returns pointer to scheduler, or `NULL` if no scheduler was found

4.3.2.5 check_if_worker_exists()

```
worker_t * check_if_worker_exists (
            worker_list_t * worker_list,
            ums_wid_t wid )
```

Checks if worker thread with `wid` exists in `worker_list` of the completion list Search is performed using local↩ _list member of worker.

Parameters

worker_list	pointer to worker_list
wid	Worker ID

Returns

returns pointer to worker, or `NULL` if no worker was found

4.3.2.6 check_if_worker_exists_global()

```
worker_t * check_if_worker_exists_global (
            worker_list_t * worker_list,
            ums_wid_t wid )
```

Checks if worker thread with `wid` exists in `worker_list` of the process Search is performed using global_list member of worker.

Parameters

worker_list	pointer to worker_list
wid	Worker ID

Returns

returns pointer to worker, or `NULL` if no worker was found

4.3.2.7 check_schedulers_state()

```
state_t check_schedulers_state (
            process_t * process )
```

Checks if schedulers of the process have finished their work

Parameters

process	pointer to process

Returns

returns state

4.3.2.8 cleanup()

```
int cleanup (
            void  )
```

Performs a cleanup by deleting all the allocated data structures for all processes that were managed by the UMS kernel module

Returns

returns `UMS_SUCCESS` if succesful

4.3.2.9 create_completion_list()

```
ums_clid_t create_completion_list (
            void  )
```

Creates a completion_list_node for the process To create a completion_list_node, UMS kernel module:

- Checks if the process is already managed, if not returns `UMS_ERROR_PROCESS_NOT_FOUND`
- Allocates and initializes completion_list_node:
  - Adds the completion list to the list of completion lists created by the process
  - completion_list_node::clid is set to process::completion_lists::list_count value (which is incremented after)
  - completion_list_node::worker_count is set to 0
  - completion_list_node::finished_count is set to 0
  - completion_list_node::state is set to IDLE
  - Allocates and initializes idle_list member of the process to track idle worker threads created by the process
  - Allocates and initializes busy_list member of the process to track finished and running worker threads created by the process

Returns

returns completion list ID

4.3.2.10 create_process_node()

```
process_t * create_process_node (
            pid_t pid )
```

Creates a process data structure to handle the specified process To create a process data structure, UMS kernel module:

- Allocates and initializes process:
  - process::pid is set to `pid`
  - process::state is set to RUNNING
  - Allocates and initializes completion_list member of the process to track completion lists created by the process
  - Allocates and initializes worker_list member of the process to track worker threads created by the process
  - Allocates and initializes scheduler_list member of the process to track schedulers created by the process

Parameters

pid	pid of the process

Returns

returns a pointer to process data structure of the specified process

4.3.2.11 create_process_proc_entry()

```
int create_process_proc_entry (
            process_t * process )
```

Dinamically creates essential proc entries for the process Allocates a memory for process_proc_entry and initializes it:

- Creates a folder to represent the process
- Creates schedulers folder

Parameters

process	pointer to process

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.2.12 create_scheduler_proc_entry()

```
int create_scheduler_proc_entry (
            process_t * process,
            scheduler_t * scheduler )
```

Dinamically creates essential proc entries for the scheduler of the process Allocates a memory for scheduler_proc_entry and initializes it:

- Creates a folder to represent the scheduler
- Creates info file that provides statistics about the scheduler
- Creates workers folder of the completion list that is assigned to the scheduler
- Creates proc entries for each worker thread by calling create_worker_proc_entry()

Parameters

process	pointer to process
scheduler	pointer to scheduler

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.3.2.13 create_worker_proc_entry()

```
int create_worker_proc_entry (
            process_t * process,
            scheduler_t * scheduler,
            worker_t * worker )
```

Dinamically creates essential proc entries for the worker of the process Allocates a memory for worker_proc_entry and initializes it:

- Creates a file that provides statistics about the worker

Parameters

process	pointer to process
scheduler	pointer to scheduler
worker	pointer to worker

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.3.2.14 create_worker_thread()

```
ums_wid_t create_worker_thread (
            worker_params_t * params )
```

Creates a worker thread for the process To create a worker, UMS kernel module:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND
- Checks if completion list exists based on the passed parameters params, if not returns UMS_ERROR_↩ COMPLETION_LIST_NOT_FOUND
- Checks if completion list is used currently, thus cannot be modified and returns UMS_ERROR_↩ COMPLETION_LIST_IS_USED_AND_CANNOT_BE_MODIFIED
- Allocates and initializes worker:
  - Adds the worker to the list of workers created by the process
  - worker::wid is set to process::worker_list::worker_count value (which is incremented after)

- worker::pid is set to -1
- worker::tid is set to `current->tgid`
- worker::clid is set to worker_params::clid
- worker::state is set to IDLE
- worker::entry_point is set to worker_params::entry_point
- worker::stack_addr is set to worker_params::stack_addr
- worker::switch_count is set to 0;
- worker::total_exec_time is set to 0;
- worker::regs is a `pt_regs` data structure and set to a snapshot of current CPU registers of the process
  - regs::ip is set to worker_params::entry_point
  - regs::di is set to worker_params::function_args
  - regs::sp is set to worker_params::stack_addr
  - regs::bp is set to worker_params::stack_addr
- worker::fpu_regs is a `fpu` data structure and set to a snapshot of current FPU registers of the process
- Adds worker to the idle list of the completion list

Parameters

params	pointer to worker_params

Returns

returns worker ID

4.3.2.15 delete_completion_lists_and_worker_threads()

```
int delete_completion_lists_and_worker_threads (
            process_t * process )
```

Deletes completion lists and worker threads created by the process Calls delete_workers_from_completion_list() and frees allocated memory that was used by the completion lists.

Parameters

process	pointer to process

Returns

returns `UMS_SUCCESS` if succesful

4.3.2.16 delete_proc()

```
int delete_proc (
            void  )
```

Deletes all proc files of the UMS kernel module

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.3.2.17 delete_process()

```
int delete_process (
            process_t * process )
```

Deletes process from the global process_list The function was used during early phases of development and later was left unused, since proc entries of the process are linked to process (in case user wants to see statistics, the data should be available for read), it was decided that only kernel module can delete data structures allocated for the process Therefore delete_process_safe() is used to delete process Still the function performs a check to see if all schedulers have finished their job and then performs series of deletions of all data structures used by the process.

Parameters

process	pointer to process

Returns

returns UMS_SUCCESS if succesful

4.3.2.18 delete_process_safe()

```
int delete_process_safe (
            process_t * process )
```

Called by UMS kernel module when module exits to delete process from the global process_list The function does not perform a check to see if all schedulers have finished their job to delete data structures used by the process It is assumed that all work has been done, therefore user issued a command to UMS kernel module to exit.

Parameters

process	pointer to process

Returns

returns UMS_SUCCESS if succesful

4.3.2.19 delete_schedulers()

```
int delete_schedulers (
            process_t * process )
```

Deletes schedulers created by the process Frees allocated memory that was used by the schedulers.

Parameters

process	pointer to process

Returns

returns UMS_SUCCESS if succesful

4.3.2.20 delete_workers_from_completion_list()

```
int delete_workers_from_completion_list (
            worker_list_t * worker_list )
```

Deletes worker threads assigned to the completion list

Parameters

worker_list	pointer to worker_list of the completion_list_node

Returns

returns UMS_SUCCESS if succesful

4.3.2.21 delete_workers_from_process_list()

```
int delete_workers_from_process_list (
            worker_list_t * worker_list )
```

Deletes worker threads created by the process Frees allocated memory that was used by the worker threads.

Parameters

worker_list	pointer to worker_list of the process

Returns

returns UMS_SUCCESS if succesful

4.3.2.22 dequeue_completion_list_items()

```
int dequeue_completion_list_items (
            list_params_t * params )
```

Provides a list of available worker threads of the completion list that can be scheduled To retrieve the list of available worker threads:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND
- Checks if scheduler associated by the pthread already exists, if not returns UMS_ERROR_SCHEDULER_↩ NOT_FOUND
- Checks if there are any available workers, if not modifes params state value to FINISHED to indicate the completion of the work
- Retrieves the list of idle worker threads from the completion list and copies them back to the params

Parameters

params	pointer to list_params

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.3.2.23 enter_scheduling_mode()

```
ums_sid_t enter_scheduling_mode (
            scheduler_params_t * params )
```

Converts a pthread to the scheduler.

To create a scheduler, UMS kernel module:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND
- Checks if completion list exists based on the passed parameters params, if not returns UMS_ERROR_↩ COMPLETION_LIST_NOT_FOUND
- Allocates and initializes scheduler:
  - Adds the scheduler to the list of schedulers created by the process
  - scheduler::sid is set to process::scheduler_list::scheduler_count; value (which is incremented after)
  - scheduler::pid is set to current->pid
  - scheduler::tid is set to current->tgid

- scheduler::wid is set to -1
- scheduler::state is set to IDLE
- scheduler::entry_point is set to scheduler_params::entry_point
- scheduler::avg_switch_time is set to 0;
- scheduler::time_needed_for_the_last_switch is set to 0;
- scheduler::total_time_needed_for_the_switch is set to 0;
- scheduler::comp_list is set to the pointer of the completion list retrieved using check_if_completion_list_exists by passing scheduler_params::clid
- scheduler::regs is a `pt_regs` data structure and set to a snapshot of current CPU registers of the pthread
  - scheduler::return_addr is set to regs::ip
  - scheduler::stack_ptr is set to regs::sp
  - scheduler::base_ptr is set to regs::bp
  - regs::ip is set to scheduler_params::entry_point
- scheduler::fpu_regs is a `fpu` data structure and set to a snapshot of current FPU registers of the pthread
- Sets the state of the completion list assigned to that scheduler to RUNNING, since the scheduling starts after the completion of ioctl call
- Creates scheduler_proc_entry for the scheduler by calling create_scheduler_proc_entry()
- Performs a context switch by copying previosly saved and modified scheduler::regs data structure to `task_pt_regs(current)`

Parameters

params	pointer to scheduler_params

Returns

returns scheduler ID

4.3.2.24 enter_ums()

```
int enter_ums (
            void  )
```

Called by a process to request a scheduling management Checks if the process is already managed or not, if not:

- Creates a process data structure by calling create_process_node()
- Creates the proc entries by calling create_process_proc_entry()

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.2.25 execute_thread()

```
int execute_thread (
            ums_wid_t worker_id )
```

Executes a worker thread with a `worker_id` To execute the worker thread:

- Checks if the process is already managed, if not returns `UMS_ERROR_PROCESS_NOT_FOUND`
- Checks if scheduler associated by the pthread already exists, if not returns `UMS_ERROR_SCHEDULER_↩ NOT_FOUND`
- Checks if the worker thread exists, currently running, completed its' work
- Updates the worker and scheduler data structures
- Records the statistics related to the scheduler and worker, such as number of switches and the time the switch happened
- Saves the register values of the scheduler
- Performs a context switch

Parameters

worker↩ _id	Worker thread ID

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.2.26 exit_scheduling_mode()

```
int exit_scheduling_mode (
            void  )
```

Converts scheduler back to the pthread.

To create a scheduler, UMS kernel module:

- Checks if the process is already managed, if not returns `UMS_ERROR_PROCESS_NOT_FOUND`
- Checks if scheduler associated by the pthread already exists, if not returns `UMS_ERROR_SCHEDULER_↩ NOT_FOUND`
- Modifies scheduler:
  - scheduler::wid is set to -1
  - scheduler::state is set to FINISHED
  - scheduler::regs is modified:

    - regs::ip is set to scheduler::return_addr
    - regs::sp is set to scheduler::stack_ptr
    - regs::bp is set to scheduler::base_ptr
    - regs::ip is set to scheduler_params::entry_point
  - Performs a context switch by copying previosly saved and modified scheduler::regs data structure to `task_pt_regs(current)`
  - Changes current FPU registers to previously saved scheduller::fpu_regs via `copy_kernel_to_fxregs()`

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.2.27 exit_ums()

```
int exit_ums (
            void  )
```

Called by a process to request a completion of the scheduling management Checks if the process is already managed or not, if not:

- Sets the state of the process to `FINISHED`, but does not delete related data structures of the process (which are deleted when UMS kernel module exits)

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.2.28 get_exec_time()

```
unsigned long get_exec_time (
            struct timespec64 * prev_time )
```

Computes time difference between passed `prev_time` and current time, which is used in this case as an indicator of execution time for worker and scheduler

Parameters

prev_time	member of worker and scheduler

Returns

returns unsigned long

4.3.2.29 init_proc()

```
int init_proc (
            void  )
```

Initializes the core proc directory for UMS kernel module

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.2.30 thread_yield()

```
int thread_yield (
            worker_status_t status )
```

Pauses or completes the execution of the worker thread To pause or complete the execution of the worker thread:

- Checks if the process is already managed, if not returns `UMS_ERROR_PROCESS_NOT_FOUND`
- Checks if scheduler associated by the pthread already exists, if not returns `UMS_ERROR_SCHEDULER_NOT_FOUND`
- Checks if the worker thread exists, currently running, completed its' work
- Updates the worker, scheduler and completion list data structures:
  - if `status` is set to PAUSE:
    - worker::state is set to IDLE, so that it can be scheduled later
    - worker is added back to the completion list
  - if `status` is set to FINISH:
    - worker::state is set to FINISHED
    - completion list increments the value of finished workers
- Records the statistics related to the scheduler and worker, such as number of switches and the time the switch happened
- Saves the register values of the worker thread
- Performs a context switch

Parameters

status	value of worker_status, which is the status of the worker thread

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.3.3 Variable Documentation

4.3.3.1 process_list

```
process_list_t process_list
```

Initial value:

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

4.4 ums_api.h File Reference

The header that contains essential functions, data structures and proc filesystem of the UMS kernel module.

```
#include "const.h"
#include <asm/current.h>
#include <asm/fpu/internal.h>
#include <asm/fpu/types.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/list.h>
#include <linux/slab.h>
#include <linux/types.h>
#include <linux/time.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
```

Data Structures

- struct process_list

  The list of the processes handled by the UMS kernel module
- struct process

  Represents a node in the process_list
- struct completion_list

  The list of the completion lists created by the specific process
- struct completion_list_node

  Represents a node in the completion_list
- struct worker_list

  The list of the worker threads
- struct worker

  Represents a node in the worker_list
- struct scheduler_list

The list of the schedulers created by the specific process

- struct scheduler

  Represents a node in the scheduler_list
- struct process_proc_entry

  Responsible for tracking proc_dir_entries of the process
- struct scheduler_proc_entry

  Responsible for tracking proc_dir_entries of the schedulers of the specific process
- struct worker_proc_entry

  Responsible for tracking proc_dir_entries of the worker of the specific process

Typedefs

- typedef struct process_list **process_list_t**

  The list of the processes handled by the UMS kernel module
- typedef struct process **process_t**

  Represents a node in the process_list
- typedef struct completion_list **completion_list_t**

  The list of the completion lists created by the specific process
- typedef struct completion_list_node **completion_list_node_t**

  Represents a node in the completion_list
- typedef struct worker_list **worker_list_t**

  The list of the worker threads
- typedef struct worker **worker_t**

  Represents a node in the worker_list
- typedef struct scheduler_list **scheduler_list_t**

  The list of the schedulers created by the specific process
- typedef struct scheduler **scheduler_t**

  Represents a node in the scheduler_list
- typedef struct process_proc_entry **process_proc_entry_t**

  Responsible for tracking proc_dir_entries of the process
- typedef struct scheduler_proc_entry **scheduler_proc_entry_t**

  Responsible for tracking proc_dir_entries of the schedulers of the specific process
- typedef struct worker_proc_entry **worker_proc_entry_t**

  Responsible for tracking proc_dir_entries of the worker of the specific process

Functions

- int enter_ums (void)

  Called by a process to request a scheduling management Checks if the process is already managed or not, if not:
- int exit_ums (void)

  Called by a process to request a completion of the scheduling management Checks if the process is already managed or not, if not:
- ums_clid_t create_completion_list (void)

  Creates a completion_list_node for the process To create a completion_list_node, UMS kernel module:
- ums_wid_t create_worker_thread (worker_params_t *params)

  Creates a worker thread for the process To create a worker, UMS kernel module:
- ums_sid_t enter_scheduling_mode (scheduler_params_t *params)

  Converts a pthread to the scheduler.
- int exit_scheduling_mode (void)

Converts scheduler back to the pthread.

- int execute_thread (ums_wid_t worker_id)

  Executes a worker thread with a `worker_id` To execute the worker thread:

- int thread_yield (worker_status_t status)

  Pauses or completes the execution of the worker thread To pause or complete the execution of the worker thread:

- int dequeue_completion_list_items (list_params_t *params)

  Provides a list of available worker threads of the completion list that can be scheduled To retrieve the list of available worker threads:

- int delete_process (process_t *process)

  Deletes process from the global process_list The function was used during early phases of development and later was left unused, since proc entries of the process are linked to process (in case user wants to see statistics, the data should be available for read), it was decided that only kernel module can delete data structures allocated for the process Therefore delete_process_safe() is used to delete process Still the function performs a check to see if all schedulers have finished their job and then performs series of deletions of all data structures used by the process.

- int delete_completion_lists_and_worker_threads (process_t *process)

  Deletes completion lists and worker threads created by the process Calls delete_workers_from_completion_list() and frees allocated memory that was used by the completion lists.

- int delete_workers_from_completion_list (worker_list_t *worker_list)

  Deletes worker threads assigned to the completion list

- int delete_workers_from_process_list (worker_list_t *worker_list)

  Deletes worker threads created by the process Frees allocated memory that was used by the worker threads.

- int delete_schedulers (process_t *process)

  Deletes schedulers created by the process Frees allocated memory that was used by the schedulers.

- process_t * create_process_node (pid_t pid)

  Creates a process data structure to handle the specified process To create a process data structure, UMS kernel module:

- process_t * check_if_process_exists (pid_t pid)

  Checks if `process` with `pid` is managed by the UMS kernel module

- completion_list_node_t * check_if_completion_list_exists (process_t *proc, ums_clid_t clid)

  Checks if completion list with `clid` was created by a `process`

- scheduler_t * check_if_scheduler_exists (process_t *proc, ums_sid_t sid)

  Checks if scheduler with `sid` was created by a `process`

- scheduler_t * check_if_scheduler_exists_run_by (process_t *process, pid_t pid)

  Checks if scheduler with `pid` was created by a `process`

- worker_t * check_if_worker_exists (worker_list_t *worker_list, ums_wid_t wid)

  Checks if worker thread with `wid` exists in `worker_list` of the completion list Search is performed using local_list member of worker.

- worker_t * check_if_worker_exists_global (worker_list_t *worker_list, ums_wid_t wid)

  Checks if worker thread with `wid` exists in `worker_list` of the process Search is performed using global_list member of worker.

- state_t **check_if_schedulers_state** (process_t *proc)
- unsigned long get_exec_time (struct timespec64 *prev_time)

  Computes time difference between passed `prev_time` and current time, which is used in this case as an indicator of execution time for worker and scheduler

- int cleanup (void)

  Performs a cleanup by deleting all the allocated data structures for all processes that were managed by the UMS kernel module

- int init_proc (void)

  Initializes the core proc directory for UMS kernel module

- int delete_proc (void)

  Deletes all proc files of the UMS kernel module

- int create_process_proc_entry (process_t *process)

Dinamically creates essential proc entries for the process Allocates a memory for process_proc_entry and initializes it:

- int create_scheduler_proc_entry (process_t *process, scheduler_t *scheduler)

  Dinamically creates essential proc entries for the scheduler of the process Allocates a memory for scheduler_proc_entry and initializes it:
- int create_worker_proc_entry (process_t *process, scheduler_t *scheduler, worker_t *worker)

  Dinamically creates essential proc entries for the worker of the process Allocates a memory for worker_proc_entry and initializes it:
- int **delete_process_proc_entry** (process_t *process)

4.4.1 Detailed Description

The header that contains essential functions, data structures and proc filesystem of the UMS kernel module.

Author

Bektur Umarbaev hrafnulf13@gmail.com

Date

4.4.2 Function Documentation

4.4.2.1 check_if_completion_list_exists()

```
completion_list_node_t * check_if_completion_list_exists (
            process_t * process,
            ums_clid_t clid )
```

Checks if completion list with `clid` was created by a `process`

Parameters

process	pointer to process
clid	Completion list ID

Returns

returns pointer to completion_list_node, or `NULL` if no completion list was found

4.4.2.2 check_if_process_exists()

```
process_t * check_if_process_exists (
            pid_t pid )
```

Checks if `process` with `pid` is managed by the UMS kernel module

Parameters

pid	pid of the process

Returns

returns pointer to process, or `NULL` if no process was found

4.4.2.3 check_if_scheduler_exists()

```
scheduler_t * check_if_scheduler_exists (
            process_t * process,
            ums_sid_t sid )
```

Checks if scheduler with `sid` was created by a `process`

Parameters

process	pointer to process
sid	Scheduler ID

Returns

returns pointer to scheduler, or `NULL` if no scheduler was found

4.4.2.4 check_if_scheduler_exists_run_by()

```
scheduler_t * check_if_scheduler_exists_run_by (
            process_t * process,
            pid_t pid )
```

Checks if scheduler with `pid` was created by a `process`

Parameters

process	pointer to process
pid	pid of the pthread which is associated with scheduler

Returns

returns pointer to scheduler, or `NULL` if no scheduler was found

4.4.2.5 check_if_worker_exists()

```
worker_t * check_if_worker_exists (
            worker_list_t * worker_list,
            ums_wid_t wid )
```

Checks if worker thread with `wid` exists in `worker_list` of the completion list Search is performed using local_list member of worker.

Parameters

worker_list	pointer to worker_list
wid	Worker ID

Returns

returns pointer to worker, or `NULL` if no worker was found

4.4.2.6 check_if_worker_exists_global()

```
worker_t * check_if_worker_exists_global (
            worker_list_t * worker_list,
            ums_wid_t wid )
```

Checks if worker thread with `wid` exists in `worker_list` of the process Search is performed using global_list member of worker.

Parameters

worker_list	pointer to worker_list
wid	Worker ID

Returns

returns pointer to worker, or `NULL` if no worker was found

4.4.2.7 cleanup()

```
int cleanup (
            void  )
```

Performs a cleanup by deleting all the allocated data structures for all processes that were managed by the UMS kernel module

Returns

returns `UMS_SUCCESS` if succesful

4.4.2.8 create_completion_list()

```
ums_clid_t create_completion_list (
            void  )
```

Creates a completion_list_node for the process To create a completion_list_node, UMS kernel module:

- Checks if the process is already managed, if not returns `UMS_ERROR_PROCESS_NOT_FOUND`
- Allocates and initializes completion_list_node:
  - Adds the completion list to the list of completion lists created by the process
  - completion_list_node::clid is set to process::completion_lists::list_count value (which is incremented after)
  - completion_list_node::worker_count is set to 0
  - completion_list_node::finished_count is set to 0
  - completion_list_node::state is set to IDLE
  - Allocates and initializes idle_list member of the process to track idle worker threads created by the process
  - Allocates and initializes busy_list member of the process to track finished and running worker threads created by the process

Returns

returns completion list ID

4.4.2.9 create_process_node()

```
process_t * create_process_node (
            pid_t pid )
```

Creates a process data structure to handle the specified process To create a process data structure, UMS kernel module:

- Allocates and initializes process:
  - process::pid is set to `pid`
  - process::state is set to RUNNING
  - Allocates and initializes completion_list member of the process to track completion lists created by the process
  - Allocates and initializes worker_list member of the process to track worker threads created by the process
  - Allocates and initializes scheduler_list member of the process to track schedulers created by the process

Parameters

pid	pid of the process

Returns

returns a pointer to process data structure of the specified process

4.4.2.10 create_process_proc_entry()

```
int create_process_proc_entry (
            process_t * process )
```

Dinamically creates essential proc entries for the process Allocates a memory for process_proc_entry and initializes it:

- Creates a folder to represent the process
- Creates schedulers folder

Parameters

process	pointer to process

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.4.2.11 create_scheduler_proc_entry()

```
int create_scheduler_proc_entry (
            process_t * process,
            scheduler_t * scheduler )
```

Dinamically creates essential proc entries for the scheduler of the process Allocates a memory for scheduler_proc_entry and initializes it:

- Creates a folder to represent the scheduler
- Creates info file that provides statistics about the scheduler
- Creates workers folder of the completion list that is assigned to the scheduler
- Creates proc entries for each worker thread by calling create_worker_proc_entry()

Parameters

process	pointer to process
scheduler	pointer to scheduler

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.4.2.12 create_worker_proc_entry()

```
int create_worker_proc_entry (
            process_t * process,
            scheduler_t * scheduler,
            worker_t * worker )
```

Dinamically creates essential proc entries for the worker of the process Allocates a memory for worker_proc_entry and initializes it:

- Creates a file that provides statistics about the worker

Parameters

process	pointer to process
scheduler	pointer to scheduler
worker	pointer to worker

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.4.2.13 create_worker_thread()

```
ums_wid_t create_worker_thread (
            worker_params_t * params )
```

Creates a worker thread for the process To create a worker, UMS kernel module:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND
- Checks if completion list exists based on the passed parameters params, if not returns UMS_ERROR_↩ COMPLETION_LIST_NOT_FOUND
- Checks if completion list is used currently, thus cannot be modified and returns UMS_ERROR_↩ COMPLETION_LIST_IS_USED_AND_CANNOT_BE_MODIFIED
- Allocates and initializes worker:
  - Adds the worker to the list of workers created by the process
  - worker::wid is set to process::worker_list::worker_count value (which is incremented after)
  - worker::pid is set to -1
  - worker::tid is set to current->tgid
  - worker::clid is set to worker_params::clid
  - worker::state is set to IDLE
  - worker::entry_point is set to worker_params::entry_point
  - worker::stack_addr is set to worker_params::stack_addr
  - worker::switch_count is set to 0;
  - worker::total_exec_time is set to 0;
  - worker::regs is a pt_regs data structure and set to a snapshot of current CPU registers of the process
    - regs::ip is set to worker_params::entry_point
    - regs::di is set to worker_params::function_args
    - regs::sp is set to worker_params::stack_addr
    - regs::bp is set to worker_params::stack_addr
  - worker::fpu_regs is a fpu data structure and set to a snapshot of current FPU registers of the process
  - Adds worker to the idle list of the completion list

Parameters

params	pointer to worker_params

Returns

returns worker ID

4.4.2.14 delete_completion_lists_and_worker_threads()

```
int delete_completion_lists_and_worker_threads (
            process_t * process )
```

Deletes completion lists and worker threads created by the process Calls delete_workers_from_completion_list() and frees allocated memory that was used by the completion lists.

Parameters

process	pointer to process

Returns

returns UMS_SUCCESS if succesful

4.4.2.15 delete_proc()

```
int delete_proc (
            void  )
```

Deletes all proc files of the UMS kernel module

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.4.2.16 delete_process()

```
int delete_process (
            process_t * process )
```

Deletes process from the global process_list The function was used during early phases of development and later was left unused, since proc entries of the process are linked to process (in case user wants to see statistics, the data should be available for read), it was decided that only kernel module can delete data structures allocated for the process Therefore delete_process_safe() is used to delete process Still the function performs a check to see if all schedulers have finished their job and then performs series of deletions of all data structures used by the process.

Parameters

process	pointer to process

Returns

returns UMS_SUCCESS if succesful

4.4.2.17 delete_schedulers()

```
int delete_schedulers (
            process_t * process )
```

Deletes schedulers created by the process Frees allocated memory that was used by the schedulers.

Parameters

process	pointer to process

Returns

returns UMS_SUCCESS if succesful

4.4.2.18 delete_workers_from_completion_list()

```
int delete_workers_from_completion_list (
            worker_list_t * worker_list )
```

Deletes worker threads assigned to the completion list

Parameters

worker_list	pointer to worker_list of the completion_list_node

Returns

returns UMS_SUCCESS if succesful

4.4.2.19 delete_workers_from_process_list()

```
int delete_workers_from_process_list (
            worker_list_t * worker_list )
```

Deletes worker threads created by the process Frees allocated memory that was used by the worker threads.

Parameters

worker_list	pointer to worker_list of the process

Returns

returns UMS_SUCCESS if succesful

4.4.2.20 dequeue_completion_list_items()

```
int dequeue_completion_list_items (
            list_params_t * params )
```

Provides a list of available worker threads of the completion list that can be scheduled To retrieve the list of available worker threads:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND
- Checks if scheduler associated by the pthread already exists, if not returns UMS_ERROR_SCHEDULER_↩ NOT_FOUND
- Checks if there are any available workers, if not modifes params state value to FINISHED to indicate the completion of the work
- Retrieves the list of idle worker threads from the completion list and copies them back to the params

Parameters

params	pointer to list_params

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.4.2.21 enter_scheduling_mode()

```
ums_sid_t enter_scheduling_mode (
            scheduler_params_t * params )
```

Converts a pthread to the scheduler.

To create a scheduler, UMS kernel module:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND

- Checks if completion list exists based on the passed parameters `params`, if not returns `UMS_ERROR_↩ COMPLETION_LIST_NOT_FOUND`
- Allocates and initializes scheduler:
  - Adds the scheduler to the list of schedulers created by the process
  - scheduler::sid is set to process::scheduler_list::scheduler_count; value (which is incremented after)
  - scheduler::pid is set to `current->pid`
  - scheduler::tid is set to `current->tgid`
  - scheduler::wid is set to -1
  - scheduler::state is set to IDLE
  - scheduler::entry_point is set to scheduler_params::entry_point
  - scheduler::avg_switch_time is set to 0;
  - scheduler::time_needed_for_the_last_switch is set to 0;
  - scheduler::total_time_needed_for_the_switch is set to 0;
  - scheduler::comp_list is set to the pointer of the completion list retrieved using check_if_completion_list_exists by passing scheduler_params::clid
  - scheduler::regs is a `pt_regs` data structure and set to a snapshot of current CPU registers of the pthread
    - scheduler::return_addr is set to regs::ip
    - scheduler::stack_ptr is set to regs::sp
    - scheduler::base_ptr is set to regs::bp
    - regs::ip is set to scheduler_params::entry_point
  - scheduler::fpu_regs is a `fpu` data structure and set to a snapshot of current FPU registers of the pthread
  - Sets the state of the completion list assigned to that scheduler to RUNNING, since the scheduling starts after the completion of ioctl call
  - Creates scheduler_proc_entry for the scheduler by calling create_scheduler_proc_entry()
  - Performs a context switch by copying previosly saved and modified scheduler::regs data structure to `task_pt_regs(current)`

Parameters

params	pointer to scheduler_params

Returns

returns scheduler ID

4.4.2.22 enter_ums()

```
int enter_ums (
            void  )
```

Called by a process to request a scheduling management Checks if the process is already managed or not, if not:

- Creates a process data structure by calling create_process_node()
- Creates the proc entries by calling create_process_proc_entry()

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.4.2.23 execute_thread()

```
int execute_thread (
            ums_wid_t worker_id )
```

Executes a worker thread with a worker_id To execute the worker thread:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND
- Checks if scheduler associated by the pthread already exists, if not returns UMS_ERROR_SCHEDULER_↩ NOT_FOUND
- Checks if the worker thread exists, currently running, completed its' work
- Updates the worker and scheduler data structures
- Records the statistics related to the scheduler and worker, such as number of switches and the time the switch happened
- Saves the register values of the scheduler
- Performs a context switch

Parameters

worker↩ _id	Worker thread ID

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.4.2.24 exit_scheduling_mode()

```
int exit_scheduling_mode (
            void  )
```

Converts scheduler back to the pthread.

To create a scheduler, UMS kernel module:

- Checks if the process is already managed, if not returns UMS_ERROR_PROCESS_NOT_FOUND

- Checks if scheduler associated by the pthread already exists, if not returns `UMS_ERROR_SCHEDULER_↩ NOT_FOUND`
- Modifies scheduler:
  - scheduler::wid is set to -1
  - scheduler::state is set to FINISHED
  - scheduler::regs is modified:
    - regs::ip is set to scheduler::return_addr
    - regs::sp is set to scheduler::stack_ptr
    - regs::bp is set to scheduler::base_ptr
    - regs::ip is set to scheduler_params::entry_point
  - Performs a context switch by copying previosly saved and modified scheduler::regs data structure to `task_pt_regs(current)`
  - Changes current FPU registers to previously saved scheduller::fpu_regs via `copy_kernel_to_↩ fxregs()`

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.4.2.25 exit_ums()

```
int exit_ums (
            void  )
```

Called by a process to request a completion of the scheduling management Checks if the process is already managed or not, if not:

- Sets the state of the process to `FINISHED`, but does not delete related data structures of the process (which are deleted when UMS kernel module exits)

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.4.2.26 get_exec_time()

```
unsigned long get_exec_time (
            struct timespec64 * prev_time )
```

Computes time difference between passed `prev_time` and current time, which is used in this case as an indicator of execution time for worker and scheduler

Parameters

prev_time	member of worker and scheduler

Returns

returns unsigned long

4.4.2.27 init_proc()

```
int init_proc (
            void  )
```

Initializes the core proc directory for UMS kernel module

Returns

returns `UMS_SUCCESS` when succesful or error constant if there are any errors

4.4.2.28 thread_yield()

```
int thread_yield (
            worker_status_t status )
```

Pauses or completes the execution of the worker thread To pause or complete the execution of the worker thread:

- Checks if the process is already managed, if not returns `UMS_ERROR_PROCESS_NOT_FOUND`
- Checks if scheduler associated by the pthread already exists, if not returns `UMS_ERROR_SCHEDULER_NOT_FOUND`
- Checks if the worker thread exists, currently running, completed its' work
- Updates the worker, scheduler and completion list data structures:
  - if `status` is set to PAUSE:
    - worker::state is set to IDLE, so that it can be scheduled later
    - worker is added back to the completion list
  - if `status` is set to FINISH:
    - worker::state is set to FINISHED
    - completion list increments the value of finished workers
- Records the statistics related to the scheduler and worker, such as number of switches and the time the switch happened
- Saves the register values of the worker thread
- Performs a context switch

Parameters

status	value of worker_status, which is the status of the worker thread

Returns

returns UMS_SUCCESS when succesful or error constant if there are any errors

4.5 ums_api.h

Go to the documentation of this file.

```
1
28 #pragma once
29
30 #include "const.h"
31
32 #include <asm/current.h>
33 #include <asm/fpu/internal.h>
34 #include <asm/fpu/types.h>
35 #include <linux/kernel.h>
36 #include <linux/module.h>
37 #include <linux/list.h>
38 #include <linux/slab.h>
39 #include <linux/types.h>
40 #include <linux/time.h>
41 #include <linux/proc_fs.h>
42 #include <linux/seq_file.h>
43
44 typedef struct process_list process_list_t;
45 typedef struct process process_t;
46 typedef struct completion_list completion_list_t;
47 typedef struct completion_list_node completion_list_node_t;
48 typedef struct worker_list worker_list_t;
49 typedef struct worker worker_t;
50 typedef struct scheduler_list scheduler_list_t;
51 typedef struct scheduler scheduler_t;
52
53 typedef struct process_proc_entry  process_proc_entry_t;
54 typedef struct scheduler_proc_entry scheduler_proc_entry_t;
55 typedef struct worker_proc_entry worker_proc_entry_t;
56
57 int enter_ums(void);
58 int exit_ums(void);
59 ums_clid_t create_completion_list(void);
60 ums_wid_t create_worker_thread(worker_params_t *params);
61 ums_sid_t enter_scheduling_mode(scheduler_params_t *params);
62 int exit_scheduling_mode(void);
63 int execute_thread(ums_wid_t worker_id);
64 int thread_yield(worker_status_t status);
65 int dequeue_completion_list_items(list_params_t *params);
66 int delete_process(process_t *process);
67 int delete_completion_lists_and_worker_threads(process_t *process);
68 int delete_workers_from_completion_list(worker_list_t *worker_list);
69 int delete_workers_from_process_list(worker_list_t *worker_list);
70 int delete_schedulers(process_t *process);
71 process_t *create_process_node(pid_t pid);
72 process_t *check_if_process_exists(pid_t pid);
73 completion_list_node_t *check_if_completion_list_exists(process_t *proc, ums_clid_t clid);
74 scheduler_t *check_if_scheduler_exists(process_t *proc, ums_sid_t sid);
75 scheduler_t *check_if_scheduler_exists_run_by(process_t *process, pid_t pid);
76 worker_t *check_if_worker_exists(worker_list_t *worker_list, ums_wid_t wid);
77 worker_t *check_if_worker_exists_global(worker_list_t *worker_list, ums_wid_t wid);
78 state_t check_if_schedulers_state(process_t *proc);
79 unsigned long get_exec_time(struct timespec64 *prev_time);
80 int cleanup(void);
81
82 int init_proc(void);
83 int delete_proc(void);
84 int create_process_proc_entry(process_t *process);
85 int create_scheduler_proc_entry(process_t *process, scheduler_t *scheduler);
86 int create_worker_proc_entry(process_t *process, scheduler_t *scheduler, worker_t *worker);
87 int delete_process_proc_entry(process_t *process);
88
93 typedef struct process_list {
94     struct list_head list;
```

```
95     unsigned int process_count;
96 } process_list_t;
97
102 typedef struct process {
103     pid_t pid;
104     struct list_head list;
105     state_t state;
106     completion_list_t *completion_lists;
107     worker_list_t *worker_list;
108     scheduler_list_t *scheduler_list;
109     process_proc_entry_t *proc_entry;
110 } process_t;
111
116 typedef struct completion_list {
117     struct list_head list;
118     unsigned int list_count;
119 } completion_list_t;
120
125 typedef struct completion_list_node {
126     ums_clid_t clid;
127     struct list_head list;
128     unsigned int worker_count;
129     unsigned int finished_count;
130     state_t state;
131     worker_list_t *idle_list;
132     worker_list_t *busy_list;
133 } completion_list_node_t;
134
139 typedef struct worker_list {
140     struct list_head list;
141     unsigned int worker_count;
142 } worker_list_t;
143
148 typedef struct worker {
149     ums_wid_t wid;
150     pid_t pid;
151     pid_t tid;
152     ums_sid_t sid;
153     ums_clid_t clid;
154     unsigned long entry_point;
155     unsigned long stack_addr;
156     struct pt_regs regs;
157     struct fpu fpu_regs;
158     struct list_head global_list;
159     struct list_head local_list;
160     state_t state;
161     worker_proc_entry_t *proc_entry;
162     unsigned int switch_count;
163     unsigned long total_exec_time;
164     struct timespec64 time_of_the_last_switch;
165 } worker_t;
166
171 typedef struct scheduler_list {
172     struct list_head list;
173     unsigned int scheduler_count;
174 } scheduler_list_t;
175
180 typedef struct scheduler {
181     ums_sid_t sid;
182     pid_t pid;
183     pid_t tid;
184     ums_wid_t wid;
185     unsigned long entry_point;
186     unsigned long return_addr;
187     unsigned long stack_ptr;
188     unsigned long base_ptr;
189     state_t state;
190     struct pt_regs regs;
191     struct fpu fpu_regs;
192     completion_list_node_t *comp_list;
193     struct list_head list;
194     scheduler_proc_entry_t *proc_entry;
195     unsigned int switch_count;
196     unsigned long avg_switch_time;
197     unsigned long time_needed_for_the_last_switch;
198     unsigned long total_time_needed_for_the_switch;
199     struct timespec64 time_of_the_last_switch;
200 } scheduler_t;
201
206 typedef struct process_proc_entry {
207     struct proc_dir_entry *pde;
208     struct proc_dir_entry *parent;
209     struct proc_dir_entry *child;
210 } process_proc_entry_t;
211
216 typedef struct scheduler_proc_entry {
217     struct proc_dir_entry *pde;
```

```
218     struct proc_dir_entry *parent;
219     struct proc_dir_entry *child;
220     struct proc_dir_entry *info;
221 } scheduler_proc_entry_t;
222
227 typedef struct worker_proc_entry {
228     struct proc_dir_entry *pde;
229     struct proc_dir_entry *parent;
230 } worker_proc_entry_t;
```

4.6 ums_dev.c File Reference

Contains implementations of the UMS miscdevice.

```
#include "ums_dev.h"
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/miscdevice.h>
#include <linux/spinlock.h>
#include <linux/fs.h>
```

Functions

- **MODULE_AUTHOR** ("Bektur Umarbaev")
- **MODULE_DESCRIPTION** ("User Mode thread Scheduling (UMS)")
- **MODULE_LICENSE** ("GPL")
- **DEFINE_SPINLOCK** (spinlock_ums)
- **module_init** (init_dev)
- **module_exit** (exit_dev)

Variables

- unsigned long **spinlock_flags_ums**

4.6.1 Detailed Description

Contains implementations of the UMS miscdevice.

Copyright (C) 2021 Bektur Umarbaev hrafnulf13@gmail.com

This file is part of the User Mode thread Scheduling (UMS) kernel module.

UMS 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 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 kernel module. If not, see http://www.gnu.org/licenses/.

Author

Bektur Umarbaev hrafnulf13@gmail.com

Date

4.7 ums_dev.h File Reference

The header that is responsible for UMS miscdevice and includes other headers required by UMS device to function properly.

```
#include "const.h"
#include "ums_api.h"
```

4.7.1 Detailed Description

The header that is responsible for UMS miscdevice and includes other headers required by UMS device to function properly.

Author

Bektur Umarbaev hrafnulf13@gmail.com

Date

4.8 ums_dev.h

Go to the documentation of this file.

```
1
28 #pragma once
29
30 #include "const.h"
31 #include "ums_api.h"
```

Index