UMS - User Mode Scheduling

Generated by Doxygen 1.9.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 completion_list Struct Reference

`#include <UMSList.h>`

Public Attributes

- completion_list_item * **head**
- completion_list_item * **tail**
- int **len**
- sem_t **sem**

3.1.1 Detailed Description

`head` first item of the list
`tail` last item of the list
`len` length of the list
`semaphore` used to access the list

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

- library/UMSList.h

3.2 completion_list_item Struct Reference

`#include <UMSList.h>`

Public Attributes

- struct completion_list_item * **next**
- struct completion_list_item * **prev**
- ums_t **ums_id**
- int **prio**

3.2.1 Detailed Description

`next` next item of the list
`prev` previous item of the list
`ums_id` ID of the thread
`prio` the priority given by the user

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

- library/UMSList.h

3.3 sched_item Struct Reference

`#include <common.h>`

Public Attributes

- unsigned long **id**
- int **worker_num**
- struct task_struct * **task_struct**
- struct proc_dir_entry * **dir**
- struct proc_dir_entry * **workers**
- struct proc_dir_entry * **info**
- unsigned long **counter**
- unsigned long **total_time**
- unsigned long **time**
- unsigned long **last_time**
- int **state**
- unsigned long **running**
- struct list_head **ums_worker_list**
- rwlock_t **worker_list_lock**
- struct list_head **list**

3.3.1 Detailed Description

`ums_id` the id of the scheduler (as given by the threads' implementation)
`worker_num` the number of the workers in the completion list
`task_struct` pointer to the thread's task struct
`dir` pointer to the scheduler/id directory
`workers` pointer to the scheduler/id/workers/ directory
`info` pointer to the scheduler/id/info file
`counter` total number of switches
`total_time` the sum of the time needed to do the switches (used to compute the avg)
`time` the time needed for the last switch
`last_time` auxiliary field used to compute "time"
`state` the state of the scheduler, 1 is running and 0 is idle
`running` the id of the worker which is currently running, -1 if none of them is running
`ums_worker_list` list of workers

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

- module/common.h

3.4 shceduling_wrapper_routine_arg Struct Reference

`#include <UMSLibrary.h>`

Public Attributes

- void ∗(∗ **start_routine**)(completion_list ∗, void ∗)
- void ∗ **arg**
- completion_list ∗ **list**
- int **fd**

3.4.1 Detailed Description

for internal use only

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

- library/UMSLibrary.h

3.5 thread_item Struct Reference

`#include <common.h>`

Public Attributes

- unsigned long **id**
- struct task_struct ∗ **task_struct**
- struct task_struct ∗ **scheduler**
- struct list_head **list**

3.5.1 Detailed Description

`id` the id of the thread
`task_struct` pointer to the thread's task struct
`scheduler` pointer to the (last) scheduler of the thread

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

- module/common.h

3.6 ums_process Struct Reference

`#include <common.h>`

Public Attributes

- int **tgid**
- int **num_sched**
- rwlock_t **counter_lock**
- struct list_head **ums_thread_list**
- rwlock_t **sched_list_lock**
- struct list_head **ums_sched_list**
- rwlock_t **thread_list_lock**
- unsigned long **flags**
- struct proc_dir_entry * **proc_dir**
- struct proc_dir_entry * **sched_dir**
- struct list_head **list**

3.6.1 Detailed Description

`tgid` the tgid of the process
`num_sched` number schedulers this process is managing
`ums_thread_list` list of the workers of this process
`ums_sched_list` list of the schedulers of this process
`proc_dir` pointer to the /proc/pid directory
`sched_dir` pointer to the proc/pid/sched/ directory

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

- module/common.h

3.7 worker_info Struct Reference

`#include <common.h>`

Public Attributes

- int **id**
- unsigned long **ums_id**
- int **state**
- int **counter**
- struct list_head **list**

3.7.1 Detailed Description

`id` the id of the thread (from 0 to n)
`ums_id` the id of the thread (as given by the threads' implementation)
`state` the state of the thread, 1 is running and 0 is idle
`counter` the counter of the times this thread had been switched in

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

- module/common.h

3.8 working_wrapper_routine_arg Struct Reference

`#include <UMSLibrary.h>`

Public Attributes

- void ∗(∗ **start_routine**)(void ∗)
- void ∗ **arg**
- int **fd**

3.8.1 Detailed Description

for internal use only

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

- library/UMSLibrary.h

Chapter 4

File Documentation

4.1 library/examples/UMSHeader.h File Reference

User header.

4.1.1 Detailed Description

User header.

This header is intended to be distributed to the user, it contains all the prototypes and definitions needed to interact with the UMS infrastructres.

4.2 library/UMSLibrary.h File Reference

Header for UMSLibrary.h.

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <semaphore.h>
#include "UMSList.h"
```

Classes

- struct working_wrapper_routine_arg
- struct shceduling_wrapper_routine_arg

Macros

- #define **INIT_UMS_PROCESS** 0
- #define **EXIT_UMS_PROCESS** 1
- #define **INTRODUCE_UMS_TASK** 3
- #define **INTRODUCE_UMS_SCHEDULER** 4
- #define **EXECUTE_UMS_THREAD** 5
- #define **UMS_THREAD_YIELD** 6
- #define **UMS_WORKER_DONE** 7
- #define **UMS_DEQUEUE** 8
- #define **UMS_ERROR_INIT** -1
- #define **UMS_ERROR_IOCTL** -2
- #define **UMS_ERROR_SEM** -3
- #define **UMS_ERROR_FD** -4
- #define **DEVICE_NAME** "ums-dev"
- #define **DEVICE_FOLDER** "/dev/"
- #define **DEVICE_PATH** DEVICE_FOLDER DEVICE_NAME
- #define DO_IOCTL(fd, type, arg)

Typedefs

- typedef pthread_t **ums_t**
- typedef struct working_wrapper_routine_arg working_wrapper_routine_arg
- typedef struct shceduling_wrapper_routine_arg shceduling_wrapper_routine_arg

Functions

- void UMS_init (void)
- void UMS_exit (void)
- ums_t EnterUmsSchedulingMode (void *, void *(*start_routine)(completion_list *, void *), void *)
- ums_t EnterUmsWorkingMode (void *(*start_routine)(void *), void *)
- void ExecuteUmsThread (ums_t)
- void UmsThreadYield (void)
- completion_list * DequeueUmsCompletionListItems (completion_list *)
- int ums_thread_join (ums_t thread, void **retval)
- ums_t ums_get_id (void)
- void * WorkingThreadWrapper (void *)
- void * SchedulerThreadWrapper (void *)

4.2.1 Detailed Description

Header for UMSLibrary.h.

Main definitions and functionalities of the user-side library for UMS scheduling

4.2.2 Macro Definition Documentation

4.2.2.1 DO_IOCTL

```
#define DO_IOCTL(
                fd,
                type,
                arg )
```

Value:

```
do{\
    if(arg != 0){\
        if(ioctl(fd, type, arg) == -1){\
            printf("Could not perform ioctl! Aborting\n");\
            exit(UMS_ERROR_IOCTL);\
        }\
    }\
    else{\
        if(ioctl(fd, type) == -1){\
            printf("Could not perform ioctl! Aborting\n");\
            exit(UMS_ERROR_IOCTL);\
        }\
    }\
}while(0)
```

This macro is used to speed up the call to IOCTL. The communication between the user application and the kernel module are done through the use of this system call.

4.2.3 Typedef Documentation

4.2.3.1 shceduling_wrapper_routine_arg

```
typedef struct shceduling_wrapper_routine_arg shceduling_wrapper_routine_arg
```

for internal use only

4.2.3.2 working_wrapper_routine_arg

```
typedef struct working_wrapper_routine_arg working_wrapper_routine_arg
```

for internal use only

4.2.4 Function Documentation

4.2.4.1 DequeueUmsCompletionListItems()

```
completion_list* DequeueUmsCompletionListItems (
            completion_list * cs )
```

`cs` the complition list of the scheduler

This function returns a completion list of all ready thread to be executed among those which are present in the completion list given in input. The returned list must be deleted by the user using the function completion_list_delete(), otherwise leaks will occur.

4.2.4.2 EnterUmsSchedulingMode()

```
ums_t EnterUmsSchedulingMode (
            void * list,
            void *(*)(completion_list *, void *) start_routine,
            void * arg )
```

`list` the completion list of the scheduler
`start_routine` the function that will execute the scheduler
`arg` the argument of the scheduler's function

Create a scheduler thread that will execute the given function, wit the given arguments. The return value is the ID of the scheduler.

4.2.4.3 EnterUmsWorkingMode()

```
ums_t EnterUmsWorkingMode (
            void *(*)(void *) start_routine,
            void * arg )
```

`start_routine` the function that will execute the worker
`arg` the argument of the worker's function
Create a worker thread that will execute the given function, wit the given arguments. The return value is the ID of the worker, this value is used to point to a worker.

4.2.4.4 ExecuteUmsThread()

```
void ExecuteUmsThread (
            ums_t id )
```

`id` the id of the thread that needs to be executed

Called from a scheduler thread, this function will execute a worker thread. The scheduler thread will remain blocked untill the worker thread will yield, or end.

4.2.4.5 SchedulerThreadWrapper()

```
void* SchedulerThreadWrapper (
            void * arg )
```

`arg` the argument passed from the user, for the scheduler function

Contacts the kernel module to communicate that a new scheduler has spawned (also communicating the completion list) then wait for all the workers to start and then it calls the user-defined scheduler's function.

4.2.4.6 UMS_exit()

```
void UMS_exit (
            void  )
```

Remove the connection with the kernel module. Tells the kernel module that the process is exiting, and closes the file descriptor of the device file. This function is inserted in the section fini_array so that it will be called after the main function ended.

4.2.4.7 ums_get_id()

```
ums_t ums_get_id (
            void  )
```

Returns the id of the caller thread.

4.2.4.8 UMS_init()

```
void UMS_init (
            void  )
```

Initialize the connection with the kernel module; the kernel module need to be already loaded when starting your application. This function is inserted in the section init_array, thus it will be called before the main function; be carefull while modifying that section.

4.2.4.9 ums_thread_join()

```
int ums_thread_join (
            ums_t thread,
            void ** retval )
```

`thread` the thread ID of the thread `retval` a pointer in which the return value will be saved. If null (i.e. 0) will be passed, the value will not be saved

Waits for the completion of the execution of the given thread.

4.2.4.10 UmsThreadYield()

```
void UmsThreadYield (
            void  )
```

Called from a worker thread, pauses the execution and gives back the control to its (last) scheduler.

4.2.4.11 WorkingThreadWrapper()

```
void* WorkingThreadWrapper (
            void * arg )
```

`arg` the argument passed from the user, for the worker function

Contacts the kernel module to communicate that a new worker has spawned then it executes the worker function. When the worker function ends, the kernel module is notified.

4.3 library/UMSList.h File Reference

Manage the completion lists used by the user.

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
```

Classes

- struct completion_list_item
- struct completion_list

Typedefs

- typedef pthread_t **ums_t**
- typedef struct completion_list_item completion_list_item
- typedef struct completion_list completion_list

Functions

- completion_list * completion_list_create ()
- void completion_list_delete (completion_list *)
- void completion_list_add (completion_list *, ums_t, int)
- void completion_list_print (completion_list *)

4.3.1 Detailed Description

Manage the completion lists used by the user.

4.3.2 Typedef Documentation

4.3.2.1 completion_list

```
typedef struct completion_list completion_list
```

`head` first item of the list
`tail` last item of the list
`len` length of the list
`semaphore` used to access the list

4.3.2.2 completion_list_item

```
typedef struct completion_list_item completion_list_item
```

`next` next item of the list
`prev` previous item of the list
`ums_id` ID of the thread
`prio` the priority given by the user

4.3.3 Function Documentation

4.3.3.1 completion_list_add()

```
void completion_list_add (
            completion_list * cs,
            ums_t ums_id,
            int prio )
```

`cs` the completion list on which the add has to be performed `ums_id` the id of the element that needs to be added `prio` the priority of the element that needs to be added

Adds an element to the tail of the given completion list; uses the function completion_list_append which is not exposed.

4.3.3.2 completion_list_create()

```
completion_list* completion_list_create ( )
```

Created an empty completion list. The completion list has to be deleted using the function completion_list_delete() in order to avoid memory leaks. If more than a scheduler is using the same completion list, be carefull not to delete it before every scheduler is done, otherwise it will lead to unpredictable behaviours of the program. Accesses to the list are controlled by semaphores.

4.3.3.3 completion_list_delete()

```
void completion_list_delete (
            completion_list * cs )
```

`cs` the list to be deleted Deletes the list and all its elements. The use of a list after this function was called on it will lead to unpredictable results (in general, use after free problems).

4.3.3.4 completion_list_print()

```
void completion_list_print (
            completion_list * cs )
```

`cs` the completion list that has to be printed

This function prints a completion list; the use of this function is intended only for debugging purposes, it remains exposed in case a user needs to do some debugging checks.

4.4 module/common.h File Reference

Header that defines some struct shared by all files.

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

Classes

- struct thread_item
- struct worker_info
- struct sched_item
- struct ums_process

Typedefs

- typedef struct thread_item thread_item
- typedef struct worker_info worker_info
- typedef struct sched_item sched_item
- typedef struct ums_process ums_process

4.4.1 Detailed Description

Header that defines some struct shared by all files.

This header defines the core definitions used by all the files of the project.

4.4.2 Typedef Documentation

4.4.2.1 sched_item

```
typedef struct sched_item sched_item
```

`ums_id` the id of the scheduler (as given by the threads' implementation)
`worker_num` the number of the workers in the completion list
`task_struct` pointer to the thread's task struct
`dir` pointer to the scheduler/id directory
`workers` pointer to the scheduler/id/workers/ directory
`info` pointer to the scheduler/id/info file
`counter` total number of switches
`total_time` the sum of the time needed to do the switches (used to compute the avg)
`time` the time needed for the last switch
`last_time` auxiliary field used to compute "time"
`state` the state of the scheduler, 1 is running and 0 is idle
`running` the id of the worker which is currently running, -1 if none of them is running
`ums_worker_list` list of workers

4.4.2.2 thread_item

```
typedef struct thread_item thread_item
```

`id` the id of the thread
`task_struct` pointer to the thread's task struct
`scheduler` pointer to the (last) scheduler of the thread

4.4.2.3 ums_process

```
typedef struct ums_process ums_process
```

`tgid` the tgid of the process
`num_sched` number schedulers this process is managing
`ums_thread_list` list of the workers of this process
`ums_sched_list` list of the schedulers of this process
`proc_dir` pointer to the /proc/pid directory
`sched_dir` pointer to the proc/pid/sched/ directory

4.4.2.4 worker_info

```
typedef struct worker_info worker_info
```

`id` the id of the thread (from 0 to n)
`ums_id` the id of the thread (as given by the threads' implementation)
`state` the state of the thread, 1 is running and 0 is idle
`counter` the counter of the times this thread had been switched in

4.5 module/UMSmain.h File Reference

Main kernel header.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/fs.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/ktime.h>
#include <linux/spinlock.h>
#include <linux/timekeeping.h>
#include "UMSProcManager.h"
```

Macros

- #define **DEVICE_NAME** "ums-dev"
- #define **SUCCESS** 0
- #define **UMS_ERROR** -1
- #define **INIT_UMS_PROCESS** 0
- #define **EXIT_UMS_PROCESS** 1
- #define **INTRODUCE_UMS_TASK** 3
- #define **INTRODUCE_UMS_SCHEDULER** 4
- #define **EXECUTE_UMS_THREAD** 5
- #define **UMS_THREAD_YIELD** 6
- #define **UMS_WORKER_DONE** 7
- #define **UMS_DEQUEUE** 8
- #define **MODULE_LOG** "UMSmain: "
- #define UMS_LIST_FIND_BY_ID(id, item, ums_thread_list)
- #define UMS_LIST_FIND_SCHEDULER(sched, ums_thread_list)
- #define UMS_FIND_PROCESS_BY_TGID(pid, item)
- #define UMS_FIND_SCHED_ITEM(p, ts, item)
- #define FIND_WORKER_BY_UMS_ID(s, id, item)

Functions

- int __init ums_init (void)
- void __exit ums_exit (void)
- long device_ioctl (struct file *, unsigned int, unsigned long)
- void **put_task_to_sleep** (void)
- int ums_schedule (unsigned long)
- int ums_thread_yield (void)
- int ums_thread_end (void)
- int **ums_create_worker_list** (sched_item *, unsigned long)
- void **free_sched_list** (ums_process *)
- void **free_work_list** (ums_process *)
- void exit_ums_process (int)
- void init_ums_process (int)
- int new_task_management (unsigned long)
- int new_scheduler_management (unsigned long)
- int ums_dequeue_list (unsigned long)
- void **exit_ums_process_all** (void)
- void **ums_print_list** (void)
- thread_item * **ums_list_find_by_id** (unsigned long)
- struct task_struct * **ums_list_find_scheduler** (void)
- sched_item * **ums_find_sched** (ums_process *, struct task_struct *)
- worker_info * **find_worker_by_ums_id** (sched_item *, unsigned long)

4.5.1 Detailed Description

Main kernel header.

This header defines the core functions for the kernel implementation of this project. Once the module is loaded a device file named /dev/ums-dev is created; this file is used to allow communication via IOCTL between the kernel module and the library. 8 total requests are defined, they are listed as macros in this header. The kernel module has been built and tested on linux kernel version 5.8.

4.5.2 Macro Definition Documentation

4.5.2.1 FIND_WORKER_BY_UMS_ID

```
#define FIND_WORKER_BY_UMS_ID(
                s,
                id,
                item )
```

Value:

```
do{\
    worker_info* current_item;\
    struct list_head* current_item_list;\
    unsigned long flags;\
    read_lock_irqsave(&s->worker_list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, &s->ums_worker_list)\
    {\
        current_item = list_entry(current_item_list, worker_info, list);\
        if(current_item->ums_id == id)\
            item = current_item;\
    }\
    read_unlock_irqrestore(&s->worker_list_lock, flags);\
}while(0)
```

4.5.2.2 UMS_FIND_PROCESS_BY_TGID

```
#define UMS_FIND_PROCESS_BY_TGID(
                pid,
                item )
```

Value:

```
do{\
    ums_process* current_item;\
    struct list_head* current_item_list;\
    unsigned long flags;\
    read_lock_irqsave(&processes_list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, &ums_processes)\
    {\
        current_item = list_entry(current_item_list, ums_process, list);\
        if(current_item->tgid == pid)\
            item = current_item;\
    }\
    read_unlock_irqrestore(&processes_list_lock, flags);\
}while(0)
```

4.5.2.3 UMS_FIND_SCHED_ITEM

```
#define UMS_FIND_SCHED_ITEM(
                p,
                ts,
                item )
```

Value:

```
do{\
    sched_item* current_item;\
    struct list_head* current_item_list;\
    unsigned long flags;\
    read_lock_irqsave(&p->sched_list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, &p->ums_sched_list)\
    {\
        current_item = list_entry(current_item_list, sched_item, list);\
        if(current_item->task_struct == ts)\
            item = current_item;\
    }\
    read_unlock_irqrestore(&p->sched_list_lock, flags);\
}while(0)
```

4.5.2.4 UMS_LIST_FIND_BY_ID

```
#define UMS_LIST_FIND_BY_ID(
                id,
                item,
                ums_thread_list )
```

Value:

```
do{\
    read_lock_irqsave(&p->thread_list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, &ums_thread_list)\
    {\
        current_item = list_entry(current_item_list, thread_item, list);\
        if(current_item->id == id)\
            item = current_item;\
    }\
    read_unlock_irqrestore(&p->thread_list_lock, flags);\
}while(0)
```

4.5.2.5 UMS_LIST_FIND_SCHEDULER

```
#define UMS_LIST_FIND_SCHEDULER(
                sched,
                ums_thread_list )
```

Value:

```
do{\
    read_lock_irqsave(&p->thread_list_lock, flags);\
    list_for_each(current_item_list, &ums_thread_list)\
    {\
        current_item = list_entry(current_item_list, thread_item, list);\
        if(current_item->task_struct == current)\
            sched = current_item->scheduler;\
    }\
    read_unlock_irqrestore(&p->thread_list_lock, flags);\
}while(0)
```

4.5.3 Function Documentation

4.5.3.1 device_ioctl()

```
long device_ioctl (
              struct file * file,
              unsigned int request,
              unsigned long data )
```

`file` the file from which IOCTL was issued
`request` the request issued
`data` the data passed from the user, if any

This function manages the requests done with IOCTL to the kernel module. the value of `request` is analyzed, and for each request the correct handler is called

4.5.3.2 exit_ums_process()

```
void exit_ums_process (
              int pid )
```

`pid` identifier of the process that is exiting UMS. We refer to "pid" in the user-space meaning of the term (i.e. tgid in kernel-space)

This function clears the memory used by a user-space process when using UMS.

4.5.3.3 init_ums_process()

```
void init_ums_process (
              int pid )
```

`pid` identifier of the process that is entring UMS. We refer to "pid" in the user-space meaning of the term (i.e. tgid in kernel-space)

This function initializes all the memory needed by a user space process to use UMS.

4.5.3.4 new_scheduler_management()

```
int new_scheduler_management (
            unsigned long ptr )
```

`ptr` the pointer to the id of the new scheduler thread

Called from a scheduler thread, this function initializes all the data needed to manage a new scheduler thread. To understand how the list is read, refere to the function ums_dequeue_list() since they use the same mechanism.

4.5.3.5 new_task_management()

```
int new_task_management (
            unsigned long data )
```

`data` the pointer to the id of the new thread

Called from a worker thread, this function initializes all the data needed to manage a new worker thread.

4.5.3.6 ums_dequeue_list()

```
int ums_dequeue_list (
            unsigned long ptr )
```

`ptr` pointer to the memory in which the user saved the completion list

This function returns the ready threads that can be executed in the completion list of the calling scheduler thread. This is done by reading first the first 8 byte (sizeof(unsigned long)) of the memory pointed by `ptr` (passed with IOCTL) to read the length of the list, then a new copy_from_user() is issued to read the list (now we know the dimension). This mechanism was used in order to avoid using 2 different IOCTLs (one to read the length, and a second one to read the list). The same mechanism is used in function new_scheduler_management().

4.5.3.7 ums_exit()

```
void __exit ums_exit (
            void  )
```

This function exits the kernel module, it removes the device file and it removes the directory /proc/ums. Moreover, if some memory is left allocated, it is freed.

4.5.3.8 ums_init()

```
int __init ums_init (
            void  )
```

This function initializes the kernel module; it also initializes the device file needed for the communication with user space.

4.5.3.9 ums_schedule()

```
int ums_schedule (
            unsigned long data )
```

`data` containts the pointer to the id of the thread that needs to be scheduled

This function schedules the next thread to be executed by a scheduler. The scheduling is done by putting the scheduler to sleep in TASK_INTERRUPTIBLE and then by waking up the selected thread.

4.5.3.10 ums_thread_end()

```
int ums_thread_end (
            void  )
```

This function is called when a worker thread ends; it cleans that worker's memory and wakes up its (last) scheduler.

4.5.3.11 ums_thread_yield()

```
int ums_thread_yield (
            void  )
```

Called from a worker thread, it gives the control back to its (last) scheduler that will decide the next worker to be run. This is done by putting the thread in TASK_INTERRUPTIBLE state.

4.6 module/UMSProcManager.h File Reference

Header that manages the proc fs.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/proc_fs.h>
#include "common.h"
```

Macros

- #define **MODULE_LOG** "UMSmain: "
- #define **BUFSIZE** 256
- #define **PATH_MAX_LEN** 128
- #define **MAX_ENTRY_LEN** 64
- #define **MAX_STAT_MSG_LEN** 256
- #define **MAX_WORK_INFO_LEN** 128
- #define **MAX_NUM_LEN** 32
- #define **UMS_PREFIX_LEN** 5
- #define **NUMBER_OF_SLASHES_BEFORE_ID** 4
- #define **NUMBER_OF_SLASHES_BEFORE_WORKER** 6
- #define PROC_FIND_PROCESS_BY_TGID(id, item)
- #define PROC_FIND_SCHED(p, sched_id, item)
- #define PROC_FIND_WORKER(s, work_id, item)

Functions

- void ums_create_proc_root (struct list_head *, rwlock_t *)
- void ums_delete_proc_root (void)
- void ums_create_proc_process (ums_process *)
- void ums_delete_proc_process (ums_process *)
- void ums_create_proc_sched (ums_process *, sched_item *)
- void **ums_delete_proc_sched** (sched_item *)
- void ums_create_proc_worker (sched_item *, int)
- ssize_t myproc_read_sched (struct file *file, char __user *ubuf, size_t count, loff_t *offset)
- ssize_t myproc_read_work (struct file *file, char __user *ubuf, size_t count, loff_t *offset)
- ssize_t **myproc_write** (struct file *file, const char __user *ubuf, size_t count, loff_t *offset)
- long **aux_pid_from_path** (char *)
- long **aux_id_from_path** (char *)
- long **aux_worker_from_path** (char *)
- char * **strcat** (char *, const char *)

4.6.1 Detailed Description

Header that manages the proc fs.

This header defines the core functions used to manage the proc fs functionalities.

4.6.2 Macro Definition Documentation

4.6.2.1 PROC_FIND_PROCESS_BY_TGID

```
#define PROC_FIND_PROCESS_BY_TGID(
               id,
               item )
```

Value:

```
do{\
    ums_process* current_item;\
    struct list_head* current_item_list;\
    unsigned long flags;\
    read_lock_irqsave(list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, processes)\
    {\
        current_item = list_entry(current_item_list, ums_process, list);\
        if(current_item->tgid == id)\
            item = current_item;\
    }\
    read_unlock_irqrestore(list_lock, flags);\
}while(0)
```

4.6.2.2 PROC_FIND_SCHED

```
#define PROC_FIND_SCHED(
                p,
                sched_id,
                item )
```

Value:

```
do{\
    sched_item* current_item;\
    struct list_head* current_item_list;\
    unsigned long flags;\
    read_lock_irqsave(&p->sched_list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, &p->ums_sched_list)\
    {\
        current_item = list_entry(current_item_list, sched_item, list);\
        if(current_item->id == sched_id)\
            item = current_item;\
    }\
    read_unlock_irqrestore(&p->sched_list_lock, flags);\
}while(0)
```

4.6.2.3 PROC_FIND_WORKER

```
#define PROC_FIND_WORKER(
                s,
                work_id,
                item )
```

Value:

```
do{\
    worker_info* current_item;\
    struct list_head* current_item_list;\
    unsigned long flags;\
    read_lock_irqsave(&s->worker_list_lock, flags);\
    item = 0;\
    list_for_each(current_item_list, &s->ums_worker_list)\
    {\
        current_item = list_entry(current_item_list, worker_info, list);\
        if(current_item->id == work_id)\
            item = current_item;\
    }\
    read_unlock_irqrestore(&s->worker_list_lock, flags);\
}while(0)
```

4.6.3 Function Documentation

4.6.3.1 myproc_read_sched()

```
ssize_t myproc_read_sched (
              struct file * file,
              char __user * ubuf,
              size_t count,
              loff_t * ppos )
```

`file` the file in which we are trying to read
`ubuf` the user buf in which we have to write the answer

count the length of the read
ppos the offset

This function implements the read functionality for the files in path /proc/ums/<pid>/schedulers/<sched_id>/info. The printed values are:
ID the id of the scheduler (from 0 to n-1, where n is the number of schedulers)
switches the number of switches done during the whole execution
state the state of the scheduler, 0 means that it is running some worker, 1 means that it is not
running the id id of the running thread
last_switch_time how much time (in ns) did it take to do the last switch
avg_switch_time the average time needed to do the switches
completion_list the list of thread with their IDs

4.6.3.2 myproc_read_work()

```
ssize_t myproc_read_work (
            struct file * file,
            char __user * ubuf,
            size_t count,
            loff_t * ppos )
```

file the file in which we are trying to read
ubuf the user buf in which we have to write the answer
count the length of the read
ppos the offset

This function implements the read functionality for the files in path /proc/ums/<pid>/schedulers/<sched_↩ id>/workers/<worker_id>. The printed values are:
ID the id of the worker (from 0 to n-1, where n is the number of workers in that thread)
thread_ID the thread ID of that worker
state the state of the scheduler, 0 means that it is running, 1 means that it is not
number_of_switches the number of switches

4.6.3.3 ums_create_proc_process()

```
void ums_create_proc_process (
            ums_process * p )
```

p process that is issuing the request

Creates the subtree proc/ums/pid for the requesting process.

4.6.3.4 ums_create_proc_root()

```
void ums_create_proc_root (
            struct list_head * ums_processes,
            rwlock_t * lock )
```

ums_processes the list of processes that are currently using UMS

This function initializes the /proc/ums directory.

4.6.3.5 ums_create_proc_sched()

```
void ums_create_proc_sched (
            ums_process * p,
            sched_item * s )
```

p process that is issuing the request
s scheduler that is issuing the request

Creates the entries for the scheduler in the /proc fs.

4.6.3.6 ums_create_proc_worker()

```
void ums_create_proc_worker (
            sched_item * s,
            int id )
```

s scheduler that is issuing the request
id id of the worker thread that is issuing the request

Creates the entries for the worker in the /proc fs.

4.6.3.7 ums_delete_proc_process()

```
void ums_delete_proc_process (
            ums_process * p )
```

p process that is issuing the request

Deletes the subtree /proc/ums/pid

4.6.3.8 ums_delete_proc_root()

```
void ums_delete_proc_root (
            void  )
```

Delete the /proc/ums directory.

Index