IPC-SO_mydoc

Generated by Doxygen 1.9.8

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 files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 changeData Struct Reference

This is the declaration of the type changeData.

`#include <IPC_SO.h>`

Data Fields

- int arrival
  variable used as unique ID
- int delay
- int effective_delay
  represents the delay that the process has to wait

3.1.1 Detailed Description

This is the declaration of the type changeData.

this struct is used to change the data inside sync_pid if necessary

3.1.2 Field Documentation

3.1.2.1 arrival

`int arrival`

variable used as unique ID

3.1.2.2 delay

`int delay`

3.1.2.3 effective_delay

```
int effective_delay
```

represents the delay that the process has to wait

3.2 Message Struct Reference

This is the declaration of the type Message.

```
#include <IPC_SO.h>
```

Data Fields

- int pid_recipient
- int pid_sender
- int priority

  Message's priority ranging from 1 to 10.
- int delay

  Delay with which the message will be sent (expressed in seconds)
- char payload [128]

  Message payload (max 64 alphanumeric symbols)

3.2.1 Detailed Description

This is the declaration of the type Message.

Message structure.

3.2.2 Field Documentation

3.2.2.1 delay

```
int delay
```

Delay with which the message will be sent (expressed in seconds)

3.2.2.2 payload

```
char payload
```

Message payload (max 64 alphanumeric symbols)

3.2.2.3 pid_recipient

```
int pid_recipient
```

3.2.2.4 pid_sender

```
int pid_sender
```

3.2.2.5 priority

```
int priority
```

Message's priority ranging from 1 to 10.

Message priority ranging from 1 to 10.

3.3 Queue_buffer Struct Reference

This is the declaration of the type Queue_buffer.

```
#include <IPC_SO.h>
```

Collaboration diagram for Queue_buffer:

3.4 Queue_mes Struct Reference

This is the declaration of the type Queue_mes.

```
#include <IPC_SO.h>
```

Collaboration diagram for Queue_mes:

Data Fields

- Message mes
  Nested field.
- unsigned long arrival
  Arrival time of the message when enqueue into PQ[i].
- struct Queue_mes * next
  Pointer to the next element.

3.4.1 Detailed Description

This is the declaration of the type Queue_mes.

3.4.2 Field Documentation

3.4.2.1 arrival

```
unsigned long arrival
```

Arrival time of the message when enqueue into PQ[i].

See also

PriorityEnqueue_mes()

3.4.2.2 mes

```
Message mes
```

Nested field.

See also

Message for more info

3.4.2.3 next

```
struct Queue_mes* next
```

Pointer to the next element.

3.5 queue_pid Struct Reference

This is the declaration of the type queue_pid.

```
#include <IPC_SO.h>
```

Collaboration diagram for queue_pid:

Data Fields

- int pid
  This field contains the PID of a subscriber to IPC-SO.
- struct queue_pid * next
  Pointer to the next element.

3.5.1 Detailed Description

This is the declaration of the type queue_pid.

3.5.2 Field Documentation

3.5.2.1 next

```
struct queue_pid* next
```

Pointer to the next element.

3.5.2.2 pid

```
int pid
```

This field contains the PID of a subscriber to IPC-SO.

3.6 sync_pid Struct Reference

This is the declaration of the type sync_pid.

```
#include <IPC_SO.h>
```

Collaboration diagram for sync_pid:

Data Fields

- int pid
- wait_queue_head_t wait_queue
  variable that allows process to sleep
- wait_queue_head_t wait_queue_delay
  variable that allows process to sleep for an amount of time
- bool letto
  flag to check if the wait queue has been woken up
- int delay
- bool change_delay
  flag to check if the delay has been changed
- int first_time
  flag to check if this istance of struct sync_pid has been modified
- int arrival
  variable used as unique ID
- int effective_delay
  represents the delay that the process has to wait
- struct sync_pid * next

3.6.1 Detailed Description

This is the declaration of the type sync_pid.

3.6.2 Field Documentation

3.6.2.1 arrival

```
int arrival
```

variable used as unique ID

3.6.2.2 change_delay

```
bool change_delay
```

flag to check if the delay has been changed

3.6.2.3 delay

```
int delay
```

3.6.2.4 effective_delay

```
int effective_delay
```

represents the delay that the process has to wait

3.6.2.5 first_time

```
int first_time
```

flag to check if this istance of struct sync_pid has been modified

3.6.2.6 letto

```
bool letto
```

flag to check if the wait queue has been woken up

3.6.2.7 next

```
struct sync_pid* next
```

3.6.2.8 pid

```
int pid
```

3.6.2.9 wait_queue

```
wait_queue_head_t wait_queue
```

variable that allows process to sleep

3.6.2.10 wait_queue_delay

```
wait_queue_head_t wait_queue_delay
```

variable that allows process to sleep for an amount of time

Chapter 4

File Documentation

4.1 /home/vboxuser/Desktop/test/kernel/ipc-so.c File Reference

```
#include "IPC_SO.h"
```

Functions

- static int registration_open (struct inode *inode, struct file *file)
- static int registration_release (struct inode *inode, struct file *file)
- static ssize_t registration_read (struct file *file, char __user *user_buf, size_t size, loff_t *offset)
- static ssize_t registration_write (struct file *file, const char __user *user_buf, size_t size, loff_t *offset)
- static int synchronous_open (struct inode *inode, struct file *file)
- static int synchronous_release (struct inode *inode, struct file *file)
- static ssize_t synchronous_read (struct file *file, char __user *user_buf, size_t size, loff_t *offset)
- static ssize_t synchronous_write (struct file *file, const char __user *user_buf, size_t size, loff_t *offset)
- static int shared_memory_open (struct inode *inode, struct file *file)
- static int shared_memory_release (struct inode *inode, struct file *file)
- static ssize_t shared_memory_read (struct file *file, char __user *user_buf, size_t size, loff_t *offset)
- static ssize_t shared_memory_write (struct file *file, const char __user *user_buf, size_t size, loff_t *offset)
- static int __init ipc_os_module_init (void)
- static void __exit ipc_os_module_exit (void)
- queue_pid * reg_init (int id)

  Initialize a new node of type queue_pid.
- void reg_insert (queue_pid **first, queue_pid *item)

  Insert the new item queue_pid into the pid queue.
- void reg_delete (queue_pid **first, int target)

  Delete a target queue_pid item from the pid's queue.
- void reg_deinit (queue_pid **first)

  Deinit the entire pid's queue and free the memory.
- int reg_search (queue_pid **first, int val)

  Search a specific PID into the pid's queue, used to discriminate between registration and unregistration operations.
- sync_pid * sync_init (int sync_pid_val)

  Initialize a new node of type sync_pid.
- void sync_insert (sync_pid **first, sync_pid *item)

  Insert the new item sync_pid into the sync queue.

- void sync_delete (sync_pid **first, int target)

  Delete a target sync_pid item from the sync queue.
- void sync_deinit (sync_pid **first)

  Deinit the entire pid's queue and free the memory.
- sync_pid * sync_search (sync_pid **first, int val)

  Scan the sync queue and determinate if a target PID is in it.
- Message PriorityDequeue_sync (int target_arrival)

  Dequeue a specific message from PQ using it's arrival.
- Queue_mes ** pqinit (void)

  Initialize the vector of Queue_mes, called PQ (Priority Queue)
- void PriorityDequeue_mes (Queue_mes **first, int pid_caller)

  Dequeue a specific Queue_mes from the respective PQ[i].
- int sync_mes_find_check (int var)

  Determine whether there are messages in the priority queues with a specific target PID.
- changeData sync_mes_find (int target_pid)

  Compares different messages with the target pid and return the one with the minimum delay.
- void pq_print (Queue_mes **t)

  Output information about the elements stored in the priority queues for debugging or informational purposes.
- void PriorityEnqueue_mes (Queue_mes **t, Message *m)

  Enqueue a new item Queue_mes in the respective PQ[i] level.
- void pq_deinit (Queue_mes **myPQ)

  Dequeue each PQ[i] and free the memory.
- Queue_buffer * bq_init (Message tmp)

  Initialize a new node of type Queue_buffer.
- void bq_enqueue (Queue_buffer *tmp)

  Enqueue a new item of type Queue_buffer in the buffer queue.
- Message * bq_dequeue (void)

  Dequeue the head item from the buffer queue.
- module_init (ipc_os_module_init)
- module_exit (ipc_os_module_exit)
- MODULE_LICENSE ("GPL")
- MODULE_AUTHOR ("Alex Cattoni, Dennis Cattoni, Manuel Vettori")
- MODULE_DESCRIPTION ("IPC-SO")

Variables

- static struct file_operations fops
- static struct file_operations fops_sync
- static struct file_operations fops_sm

4.1.1 Function Documentation

4.1.1.1 bq_dequeue()

```
Message * bq_dequeue (
            void  )
```

Dequeue the head item from the buffer queue.

Returns

*msg, pointers to the first message in the buffer queue (FIFO)

NULL, if the buffer queue is empty

4.1.1.2 bq_enqueue()

```
void bq_enqueue (
            Queue_buffer * tmp )
```

Enqueue a new item of type Queue_buffer in the buffer queue.

Parameters

tmp	Item of type Queue_buffer, ready to be enqueued

4.1.1.3 bq_init()

```
Queue_buffer * bq_init (
            Message tmp )
```

Initialize a new node of type Queue_buffer.

Parameters

tmp	temporary Message ready to be enqueued

Returns

new_node, memory allocation for the new Queue_buffer node is successful

ERR_PTR(-EFAULT), indicate the allocation failure

4.1.1.4 ipc_os_module_exit()

```
static void __exit ipc_os_module_exit (
            void  )  [static]
```

4.1.1.5 ipc_os_module_init()

```
static int __init ipc_os_module_init (
            void  )  [static]
```

4.1.1.6 MODULE_AUTHOR()

```
MODULE_AUTHOR (
            "Alex Cattoni,
            Dennis Cattoni,
            Manuel Vettori"  )
```

4.1.1.7 MODULE_DESCRIPTION()

```
MODULE_DESCRIPTION (
            "IPC-SO"  )
```

4.1.1.8 module_exit()

```
module_exit (
            ipc_os_module_exit  )
```

4.1.1.9 module_init()

```
module_init (
            ipc_os_module_init  )
```

4.1.1.10 MODULE_LICENSE()

```
MODULE_LICENSE (
            "GPL"  )
```

4.1.1.11 pq_deinit()

```
void pq_deinit (
            Queue_mes ** t )
```

Dequeue each PQ[i] and free the memory.

Parameters

t	Global pointer to the first element

4.1.1.12 pq_print()

```
void pq_print (
            Queue_mes ** t )
```

Output information about the elements stored in the priority queues for debugging or informational purposes.

Parameters

t	Global pointer to the first element

4.1.1.13 pqinit()

```
Queue_mes ** pqinit (
            void  )
```

Initialize the vector of Queue_mes, called PQ (Priority Queue)

Returns

myPQ, valid pointer to an array of Queue_mes pointers

NULL, memory allocation for the array of Queue_mes pointers fails

ERR_PTR(-EFAULT), memory allocation for an individual priority queue within the array fails

4.1.1.14 PriorityDequeue_mes()

```
void PriorityDequeue_mes (
            Queue_mes ** first,
            int pid_caller )
```

Dequeue a specific Queue_mes from the respective PQ[i].

Parameters

first	Global pointer to the first element
pid_caller	target PID that represents the nested Messages of interests

See also

Queue_mes and Message for mor details

4.1.1.15 PriorityDequeue_sync()

```
Message PriorityDequeue_sync (
            int target_arrival )
```

Dequeue a specific message from PQ using it's arrival.

Parameters

target_arrival	Target for the dequeue operation

Returns

target_sync_mes, returns the dequeued target Message

4.1.1.16 PriorityEnqueue_mes()

```
void PriorityEnqueue_mes (
```

```
            Queue_mes ** t,
            Message * m )
```

Enqueue a new item Queue_mes in the respective PQ[i] level.

Parameters

t	Global pointer to the first element
m	Message from the user process

4.1.1.17 reg_deinit()

```
void reg_deinit (
            queue_pid ** first )
```

Deinit the entire pid's queue and free the memory.

Parameters

first	Global pointer to the first element

4.1.1.18 reg_delete()

```
void reg_delete (
            queue_pid ** first,
            int target )
```

Delete a target queue_pid item from the pid's queue.

Parameters

first	Global pointer to the first element
target	target PID of the element that has to be dequeued.

4.1.1.19 reg_init()

```
queue_pid * reg_init (
            int id )
```

Initialize a new node of type queue_pid.

Parameters

id	PID correlated to a new process subscribed to IPC-SO

Returns

ERR_PTR(-EFAULT), If kmalloc returns a NULL pointer,

ttp, Valid pointer to the newly allocated queue_pid structure

4.1.1.20 reg_insert()

```
void reg_insert (
            queue_pid ** first,
            queue_pid * item )
```

Insert the new item queue_pid into the pid queue.

Parameters

first	Global pointer to the first element
item	New item of type queue_pid ready to be enqueued into the pid's queue.

4.1.1.21 reg_search()

```
int reg_search (
            queue_pid ** first,
            int val )
```

Search a specific PID into the pid's queue, used to discriminate between registration and unregistration operations.

Parameters

first	Global pointer to the first element
val	identifier of the target PID

Returns

0, The value is not found into the pid's queue => registration

1, The value is found into the pid's queue => unregistration

4.1.1.22 registration_open()

```
static int registration_open (
            struct inode * inode,
            struct file * file )  [static]
```

4.1.1.23 registration_read()

```
static ssize_t registration_read (
            struct file * file,
            char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

4.1.1.24 registration_release()

```
static int registration_release (
            struct inode * inode,
            struct file * file )  [static]
```

4.1.1.25 registration_write()

```
static ssize_t registration_write (
            struct file * file,
            const char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

4.1.1.26 shared_memory_open()

```
static int shared_memory_open (
            struct inode * inode,
            struct file * file )  [static]
```

4.1.1.27 shared_memory_read()

```
static ssize_t shared_memory_read (
            struct file * file,
            char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

4.1.1.28 shared_memory_release()

```
static int shared_memory_release (
            struct inode * inode,
            struct file * file )  [static]
```

4.1.1.29 shared_memory_write()

```
static ssize_t shared_memory_write (
            struct file * file,
            const char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

4.1.1.30 sync_deinit()

```
void sync_deinit (
            sync_pid ** first_sync )
```

Deinit the entire pid's queue and free the memory.

Parameters

first_sync	Global pointer to the first element

4.1.1.31 sync_delete()

```
void sync_delete (
            sync_pid ** first_sync,
            int target )
```

Delete a target sync_pid item from the sync queue.

Parameters

first_sync	Global pointer to the first element
target	target PID of the element that has to be dequeued.

4.1.1.32 sync_init()

```
sync_pid * sync_init (
            int id )
```

Initialize a new node of type sync_pid.

Parameters

id	It's the PID of the process

Returns

ERR_PTR(-EFAULT), If kmalloc returns a NULL pointer,

ttp, Valid pointer to the newly allocated queue_pid structure

4.1.1.33 sync_insert()

```
void sync_insert (
            sync_pid ** first_sync,
            sync_pid * item )
```

Insert the new item sync_pid into the sync queue.

Parameters

first_sync	Global pointer to the first element
item	New item of type sync_pid ready to be enqueued into the pid's queue.

4.1.1.34 sync_mes_find()

```
changeData sync_mes_find (
            int target_pid )
```

Compares different messages with the target pid and return the one with the minimum delay.

Parameters

target_pid	represents the target PID for which we want to find the delay-related data

Returns

changeData, structure of type changeData

4.1.1.35 sync_mes_find_check()

```
int sync_mes_find_check (
            int var )
```

Determine whether there are messages in the priority queues with a specific target PID.

Parameters

var	Represents the target PID that we want to check for in the messages

Returns

0, if any messages were not found

1, if any messages were found

4.1.1.36 sync_search()

```
sync_pid * sync_search (
            sync_pid ** first_sync,
            int val )
```

Scan the sync queue and determinate if a target PID is in it.

Parameters

first_sync	Global pointer to the first element
val	Target PID value for the research

Returns

found, sync_pid structure with the matching PID if found

NULL, no sync_pid structure with the given PID was found in the queue

4.1.1.37 synchronous_open()

```
static int synchronous_open (
             struct inode * inode,
             struct file * file )  [static]
```

4.1.1.38 synchronous_read()

```
static ssize_t synchronous_read (
             struct file * file,
             char __user * user_buf,
             size_t size,
             loff_t * offset )  [static]
```

4.1.1.39 synchronous_release()

```
static int synchronous_release (
             struct inode * inode,
             struct file * file )  [static]
```

4.1.1.40 synchronous_write()

```
static ssize_t synchronous_write (
             struct file * file,
             const char __user * user_buf,
             size_t size,
             loff_t * offset )  [static]
```

4.1.2 Variable Documentation

4.1.2.1 fops

```
struct file_operations fops  [static]
```

Initial value:

```
= {
    .owner = THIS_MODULE,
    .open = registration_open,
    .release = registration_release,
    .read = registration_read,
    .write = registration_write,
}
```

4.1.2.2 fops_sm

```
struct file_operations fops_sm  [static]
```

Initial value:

```
= {
    .owner = THIS_MODULE,
    .open = shared_memory_open,
    .release = shared_memory_release,
    .read = shared_memory_read,
    .write = shared_memory_write,
}
```

4.1.2.3 fops_sync

```
struct file_operations fops_sync  [static]
```

Initial value:

```
= {
    .owner = THIS_MODULE,
    .open = synchronous_open,
    .release = synchronous_release,
    .read = synchronous_read,
    .write = synchronous_write,
}
```

4.2 /home/vboxuser/Desktop/test/kernel/IPC_SO.h File Reference

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#include <linux/slab.h>
#include <linux/jiffies.h>
#include <uapi/linux/wait.h>
#include <linux/semaphore.h>
```

Data Structures

- struct sync_pid

  This is the declaration of the type sync_pid.
- struct changeData

  This is the declaration of the type changeData.
- struct Message

  This is the declaration of the type Message.
- struct Queue_mes

  This is the declaration of the type Queue_mes.
- struct Queue_buffer

  This is the declaration of the type Queue_buffer.
- struct queue_pid

  This is the declaration of the type queue_pid.

Macros

- #define REGISTRATION_DEVICE "registration"

  Declaration of the registration device.
- #define SYNCHRONOUS_DEVICE "synchronous"

  Declaration of the synchronous device.
- #define SHARED_MEM_DEVICE "shared_memory"

  Declaration of the shared memory device.
- #define REGISTRATION_SIZE 1024

  Declaration of the size of the registration device.
- #define SYNCHRONOUS_SIZE 1024

  Declaration of the size of the synchronous device.
- #define SHARED_MEM_SIZE 4096

  Declaration of the size of the shared memory device.

Typedefs

- typedef struct sync_pid sync_pid

  This is the declaration of the type sync_pid.
- typedef struct changeData changeData

  This is the declaration of the type changeData.
- typedef struct Message Message

  This is the declaration of the type Message.
- typedef struct Queue_mes Queue_mes

  This is the declaration of the type Queue_mes.
- typedef struct Queue_buffer Queue_buffer

  This is the declaration of the type Queue_buffer.
- typedef struct queue_pid queue_pid

  This is the declaration of the type queue_pid.

Functions

- static int __init ipc_os_module_init (void)

  Initialization routine of the module IPC-SO.
- static void __exit ipc_os_module_exit (void)

  Deinitialization routine of the module IPC-SO, destroy the devices and free the memory.
- static int registration_open (struct inode *inode, struct file *file)

  Callback function, part of the file operation structure associated with the device.
- static int registration_release (struct inode *inode, struct file *file)

  Callback function, part of the file operation structure associated with the device.
- static ssize_t registration_read (struct file *file, char __user *user_buf, size_t size, loff_t *offset)

  Callback function, part of the file operation structure associated with the device.
- static ssize_t registration_write (struct file *file, const char __user *user_buf, size_t size, loff_t *offset)

  Callback function, part of the file operation structure associated with the device.
- static int shared_memory_open (struct inode *inode, struct file *file)

  Callback function, part of the file operation structure associated with the device.
- static int shared_memory_release (struct inode *inode, struct file *file)

  Callback function, part of the file operation structure associated with the device.
- static ssize_t shared_memory_read (struct file *file, char __user *user_buf, size_t size, loff_t *offset)

  Callback function, part of the file operation structure associated with the device.
- static ssize_t shared_memory_write (struct file *file, const char __user *user_buf, size_t size, loff_t *offset)

  Callback function, part of the file operation structure associated with the device.
- static int synchronous_open (struct inode *inode, struct file *file)

  Callback function, part of the file operation structure associated with the device.
- static int synchronous_release (struct inode *inode, struct file *file)

  Callback function, part of the file operation structure associated with the device.
- static ssize_t synchronous_read (struct file *file, char __user *user_buf, size_t size, loff_t *offset)

  Callback function, part of the file operation structure associated with the device.
- static ssize_t synchronous_write (struct file *file, const char __user *user_buf, size_t size, loff_t *offset)

  Callback function, part of the file operation structure associated with the device.
- sync_pid * sync_init (int id)

  Initialize a new node of type sync_pid.
- void sync_insert (sync_pid **first_sync, sync_pid *item)

  Insert the new item sync_pid into the sync queue.
- void sync_delete (sync_pid **first_sync, int target)

Delete a target sync_pid item from the sync queue.

- void sync_deinit (sync_pid **first_sync)

  Deinit the entire pid's queue and free the memory.
- sync_pid * sync_search (sync_pid **first_sync, int val)

  Scan the sync queue and determinate if a target PID is in it.
- Message PriorityDequeue_sync (int target_arrival)

  Dequeue a specific message from PQ using it's arrival.
- changeData sync_mes_find (int target_pid)

  Compares different messages with the target pid and return the one with the minimum delay.
- int sync_mes_find_check (int var)

  Determine whether there are messages in the priority queues with a specific target PID.
- queue_pid * reg_init (int id)

  Initialize a new node of type queue_pid.
- void reg_insert (queue_pid **first, queue_pid *item)

  Insert the new item queue_pid into the pid queue.
- void reg_delete (queue_pid **first, int target)

  Delete a target queue_pid item from the pid's queue.
- void reg_deinit (queue_pid **first)

  Deinit the entire pid's queue and free the memory.
- int reg_search (queue_pid **first, int val)

  Search a specific PID into the pid's queue, used to discriminate between registration and unregistration operations.
- Queue_mes ** pqinit (void)

  Initialize the vector of Queue_mes, called PQ (Priority Queue)
- void PriorityDequeue_mes (Queue_mes **first, int pid_caller)

  Dequeue a specific Queue_mes from the respective PQ[i].
- void pq_print (Queue_mes **t)

  Output information about the elements stored in the priority queues for debugging or informational purposes.
- void PriorityEnqueue_mes (Queue_mes **t, Message *m)

  Enqueue a new item Queue_mes in the respective PQ[i] level.
- void pq_deinit (Queue_mes **t)

  Dequeue each PQ[i] and free the memory.
- Queue_buffer * bq_init (Message tmp)

  Initialize a new node of type Queue_buffer.
- void bq_enqueue (Queue_buffer *tmp)

  Enqueue a new item of type Queue_buffer in the buffer queue.
- Message * bq_dequeue (void)

  Dequeue the head item from the buffer queue.

Variables

- static struct semaphore sem

  Declaration of the semaphore sem; This semaphore ensures each Queue_mes got a unique arrival.
- static dev_t dev_num_registration

  Unique device identifier assigned the registration device.
- static struct cdev cdev_registration

  character device structure for the registration device.
- static struct class * cl_registration

  Variable used to create the device node in the /dev directory.
- static dev_t dev_num_synchronous

  Unique device identifier assigned the synchronous device.

- static struct cdev cdev_synchronous
  character device structure for the synchronous device.
- static struct class * cl_synchronous
  Variable used to create the device node in the /dev directory.
- static dev_t dev_num_shared
  Unique device identifier assigned the shared memory device.
- static struct cdev cdev_shared
  character device structure for the shared memory device.
- static struct class * cl_shared
  Variable used to create the device node in the /dev directory.
- static char * registration_buffer
  Pointer to the buffer registration.
- static char * synchronous_buffer
  Pointer to the buffer synchronous.
- static char * shared_buffer
  Pointer to the buffer shared memory.
- sync_pid * first_sync = NULL
  Pointer to the first sync_pid element in the queue.
- sync_pid ** last_sync = NULL
  Pointer to the last sync_pid element in the queue.
- Queue_mes * pq_last [10] = {NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL}
  Array of pointers to the last element of each PQ[i].
- Queue_mes ** PQ
  Array of the 10 priority queues, one for each priority level.
- Queue_buffer * head = NULL
- queue_pid * first = NULL
  Pointer to the first element of the queue queue_pid.
- queue_pid ** last = NULL
  Pointer to the last element of the queue queue_pid.

4.2.1 Macro Definition Documentation

4.2.1.1 REGISTRATION_DEVICE

```
#define REGISTRATION_DEVICE "registration"
```

Declaration of the registration device.

4.2.1.2 REGISTRATION_SIZE

```
#define REGISTRATION_SIZE 1024
```

Declaration of the size of the registration device.

4.2.1.3 SHARED_MEM_DEVICE

```
#define SHARED_MEM_DEVICE "shared_memory"
```

Declaration of the shared memory device.

4.2.1.4 SHARED_MEM_SIZE

```
#define SHARED_MEM_SIZE 4096
```

Declaration of the size of the shared memory device.

4.2.1.5 SYNCHRONOUS_DEVICE

```
#define SYNCHRONOUS_DEVICE "synchronous"
```

Declaration of the synchronous device.

4.2.1.6 SYNCHRONOUS_SIZE

```
#define SYNCHRONOUS_SIZE 1024
```

Declaration of the size of the synchronous device.

4.2.2 Typedef Documentation

4.2.2.1 changeData

```
typedef struct changeData changeData
```

This is the declaration of the type changeData.

this struct is used to change the data inside sync_pid if necessary

4.2.2.2 Message

```
typedef struct Message Message
```

This is the declaration of the type Message.

4.2.2.3 Queue_buffer

```
typedef struct Queue_buffer Queue_buffer
```

This is the declaration of the type Queue_buffer.

4.2.2.4 Queue_mes

```
typedef struct Queue_mes Queue_mes
```

This is the declaration of the type Queue_mes.

4.2.2.5 queue_pid

```
typedef struct queue_pid queue_pid
```

This is the declaration of the type queue_pid.

4.2.2.6 sync_pid

```
typedef struct sync_pid sync_pid
```

This is the declaration of the type sync_pid.

4.2.3 Function Documentation

4.2.3.1 bq_dequeue()

```
Message * bq_dequeue (
            void  )
```

Dequeue the head item from the buffer queue.

Returns

∗msg, pointers to the first message in the buffer queue (FIFO)

NULL, if the buffer queue is empty

4.2.3.2 bq_enqueue()

```
void bq_enqueue (
            Queue_buffer * tmp )
```

Enqueue a new item of type Queue_buffer in the buffer queue.

Parameters

tmp	Item of type Queue_buffer, ready to be enqueued

4.2.3.3 bq_init()

```
Queue_buffer * bq_init (
            Message tmp )
```

Initialize a new node of type Queue_buffer.

Parameters

tmp	temporary Message ready to be enqueued

Returns

new_node, memory allocation for the new Queue_buffer node is successful

ERR_PTR(-EFAULT), indicate the allocation failure

4.2.3.4 ipc_os_module_exit()

```
static void __exit ipc_os_module_exit (
            void  )  [static]
```

Deinitialization routine of the module IPC-SO, destroy the devices and free the memory.

4.2.3.5 ipc_os_module_init()

```
static int __init ipc_os_module_init (
            void  )  [static]
```

Initialization routine of the module IPC-SO.

Returns

0, no errors occurs.

PTR_ERR, Failed to create device class

-ENOMEM, Failed to allocate buffer

ret $<$ 0, Failed to allocate character device region

ret $<$ 0, Failed to add character device

PTR_ERR, PQ is not correctly initialized

4.2.3.6 pq_deinit()

```
void pq_deinit (
            Queue_mes ** t )
```

Dequeue each PQ[i] and free the memory.

Parameters

t	Global pointer to the first element

4.2.3.7 pq_print()

```
void pq_print (
            Queue_mes ** t )
```

Output information about the elements stored in the priority queues for debugging or informational purposes.

Parameters

t	Global pointer to the first element

4.2.3.8 pqinit()

```
Queue_mes ** pqinit (
            void  )
```

Initialize the vector of Queue_mes, called PQ (Priority Queue)

Returns

myPQ, valid pointer to an array of Queue_mes pointers

NULL, memory allocation for the array of Queue_mes pointers fails

ERR_PTR(-EFAULT), memory allocation for an individual priority queue within the array fails

4.2.3.9 PriorityDequeue_mes()

```
void PriorityDequeue_mes (
            Queue_mes ** first,
            int pid_caller )
```

Dequeue a specific Queue_mes from the respective PQ[i].

Parameters

first	Global pointer to the first element
pid_caller	target PID that represents the nested Messages of interests

See also

Queue_mes and Message for mor details

4.2.3.10 PriorityDequeue_sync()

```
Message PriorityDequeue_sync (
            int target_arrival )
```

Dequeue a specific message from PQ using it's arrival.

Parameters

target_arrival	Target for the dequeue operation

Returns

target_sync_mes, returns the dequeued target Message

4.2.3.11 PriorityEnqueue_mes()

```
void PriorityEnqueue_mes (
            Queue_mes ** t,
            Message * m )
```

Enqueue a new item Queue_mes in the respective PQ[i] level.

Parameters

t	Global pointer to the first element
m	Message from the user process

4.2.3.12 reg_deinit()

```
void reg_deinit (
            queue_pid ** first )
```

Deinit the entire pid's queue and free the memory.

Parameters

first	Global pointer to the first element

4.2.3.13 reg_delete()

```
void reg_delete (
            queue_pid ** first,
            int target )
```

Delete a target queue_pid item from the pid's queue.

Parameters

first	Global pointer to the first element
target	target PID of the element that has to be dequeued.

4.2.3.14 reg_init()

```
queue_pid * reg_init (
            int id )
```

Initialize a new node of type queue_pid.

Parameters

id	PID correlated to a new process subscribed to IPC-SO

Returns

ERR_PTR(-EFAULT), If kmalloc returns a NULL pointer,

ttp, Valid pointer to the newly allocated queue_pid structure

4.2.3.15 reg_insert()

```
void reg_insert (
            queue_pid ** first,
            queue_pid * item )
```

Insert the new item queue_pid into the pid queue.

Parameters

first	Global pointer to the first element
item	New item of type queue_pid ready to be enqueued into the pid's queue.

4.2.3.16 reg_search()

```
int reg_search (
            queue_pid ** first,
            int val )
```

Search a specific PID into the pid's queue, used to discriminate between registration and unregistration operations.

Parameters

first	Global pointer to the first element
val	identifier of the target PID

Returns

0, The value is not found into the pid's queue => registration

1, The value is found into the pid's queue => unregistration

4.2.3.17 registration_open()

```
static int registration_open (
            struct inode * inode,
            struct file * file )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

inode	Represents the inode structure of the file being opened
file	Represents the file structure associated with the file being opened

Returns

0, device file has been successfully opened and ready.

4.2.3.18 registration_read()

```
static ssize_t registration_read (
            struct file * file,
            char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

file	Pointer to the 'stuct file' representing the opened file
user_buf	Pointer to the user-space buffer where the data read from the device should be copied
size	Number of bytes that the user-space buffer can hold
offset	Pointer to the current file offset

Returns

$>$ 0, total number of bytes read and copied to the user buffer

0, bytes_to_copy is less than or equal to 0

$<$ 0, an error occurs during the copy_to_user operation

4.2.3.19 registration_release()

```
static int registration_release (
            struct inode * inode,
            struct file * file )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

inode,Represents	the inode structure of the file being opened
file,Represents	the file structure associated with the file being opened

Returns

0, device file has been successfully closed.

4.2.3.20 registration_write()

```
static ssize_t registration_write (
            struct file * file,
            const char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

file	Pointer to the 'stuct file' representing the opened file
user_buf	Pointer to the user-space buffer where the data read from the device should be copied
size	Number of bytes that the user-space buffer can hold
offset	Pointer to the current file offset

Returns

$>$ 0, Number of bytes consumed from the user buffer

0, bytes_to_copy is less than or equal to 0

-EINVAL, Size of the remaining user buffer is less than the size of an integer

-EFAULT, An error occurs during the copy_from_user operation

4.2.3.21 shared_memory_open()

```
static int shared_memory_open (
            struct inode * inode,
            struct file * file )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

inode	Represents the inode structure of the file being opened
file	Represents the file structure associated with the file being opened

Returns

0, device file has been successfully opened and ready.

4.2.3.22 shared_memory_read()

```
static ssize_t shared_memory_read (
            struct file * file,
```

```
            char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

file	Pointer to the 'stuct file' representing the opened file
user_buf	Pointer to the user-space buffer where the data read from the device should be copied
size	Number of bytes that the user-space buffer can hold
offset	Pointer to the current file offset

Returns

> 0, total number of bytes read and copied to the user buffer

0, remaining_size is zero or less, the function returns 0

-EFAULT, an error occurs during the copy_to_user operation

4.2.3.23 shared_memory_release()

```
static int shared_memory_release (
            struct inode * inode,
            struct file * file )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

inode,Represents	the inode structure of the file being opened
file,Represents	the file structure associated with the file being opened

Returns

0, device file has been successfully closed.

4.2.3.24 shared_memory_write()

```
static ssize_t shared_memory_write (
            struct file * file,
            const char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

file	Pointer to the 'stuct file' representing the opened file
user_buf	Pointer to the user-space buffer where the data read from the device should be copied
size	Number of bytes that the user-space buffer can hold
offset	Pointer to the current file offset

Returns

> 0, Number of bytes consumed from the user buffer

0, bytes_to_copy is less than or equal to 0

-ENOMEM, memory allocation for the user_message fails

-EFAULT, An error occurs during the copy_from_user operation

-EACCES, the current process is not running as root (euid is not 0) and it tries to send a message with priority 10

4.2.3.25 sync_deinit()

```
void sync_deinit (
            sync_pid ** first_sync )
```

Deinit the entire pid's queue and free the memory.

Parameters

first_sync	Global pointer to the first element

4.2.3.26 sync_delete()

```
void sync_delete (
            sync_pid ** first_sync,
            int target )
```

Delete a target sync_pid item from the sync queue.

Parameters

first_sync	Global pointer to the first element
target	target PID of the element that has to be dequeued.

4.2.3.27 sync_init()

```
sync_pid * sync_init (
            int id )
```

Initialize a new node of type sync_pid.

Parameters

id	It's the PID of the process

Returns

ERR_PTR(-EFAULT), If kmalloc returns a NULL pointer,

ttp, Valid pointer to the newly allocated queue_pid structure

4.2.3.28 sync_insert()

```
void sync_insert (
            sync_pid ** first_sync,
            sync_pid * item )
```

Insert the new item sync_pid into the sync queue.

Parameters

first_sync	Global pointer to the first element
item	New item of type sync_pid ready to be enqueued into the pid's queue.

4.2.3.29 sync_mes_find()

```
changeData sync_mes_find (
            int target_pid )
```

Compares different messages with the target pid and return the one with the minimum delay.

Parameters

target_pid	represents the target PID for which we want to find the delay-related data

Returns

changeData, structure of type changeData

4.2.3.30 sync_mes_find_check()

```
int sync_mes_find_check (
            int var )
```

Determine whether there are messages in the priority queues with a specific target PID.

Parameters

var	Represents the target PID that we want to check for in the messages

Returns

0, if any messages were not found

1, if any messages were found

4.2.3.31 sync_search()

```
sync_pid * sync_search (
            sync_pid ** first_sync,
            int val )
```

Scan the sync queue and determinate if a target PID is in it.

Parameters

first_sync	Global pointer to the first element
val	Target PID value for the research

Returns

found, sync_pid structure with the matching PID if found

NULL, no sync_pid structure with the given PID was found in the queue

4.2.3.32 synchronous_open()

```
static int synchronous_open (
            struct inode * inode,
            struct file * file )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

inode	Represents the inode structure of the file being opened
file	Represents the file structure associated with the file being opened

Returns

0, device file has been successfully opened and ready.

4.2.3.33 synchronous_read()

```
static ssize_t synchronous_read (
            struct file * file,
            char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

file	Pointer to the 'stuct file' representing the opened file
user_buf	Pointer to the user-space buffer where the data read from the device should be copied
size	Number of bytes that the user-space buffer can hold
offset	Pointer to the current file offset

Returns

$>$ 0, the operation was successful, and the message was successfully read from the synchronous priority queue and copied to the user buffer

0, bytes_this_message is zero or less, the function returns 0

-EFAULT, an error occurs during the copy_to_user operation

4.2.3.34 synchronous_release()

```
static int synchronous_release (
            struct inode * inode,
            struct file * file )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

inode,Represents	the inode structure of the file being opened
file,Represents	the file structure associated with the file being opened

Returns

0, device file has been successfully closed.

4.2.3.35 synchronous_write()

```
static ssize_t synchronous_write (
            struct file * file,
            const char __user * user_buf,
            size_t size,
            loff_t * offset )  [static]
```

Callback function, part of the file operation structure associated with the device.

Parameters

file	Pointer to the 'stuct file' representing the opened file
user_buf	Pointer to the user-space buffer where the data read from the device should be copied
size	Number of bytes that the user-space buffer can hold
offset	Pointer to the current file offset

Returns

> 0, operation was successful, and the process has been successfully registered for synchronous communication

0, bytes_to_copy is zero or less

-EFAULT, an error occurs during the copy_from_user operation

4.2.4 Variable Documentation

4.2.4.1 cdev_registration

```
struct cdev cdev_registration  [static]
```

character device structure for the registration device.

4.2.4.2 cdev_shared

```
struct cdev cdev_shared  [static]
```

character device structure for the shared memory device.

4.2.4.3 cdev_synchronous

```
struct cdev cdev_synchronous  [static]
```

character device structure for the synchronous device.

4.2.4.4 cl_registration

```
struct class* cl_registration  [static]
```

Variable used to create the device node in the /dev directory.

4.2.4.5 cl_shared

```
struct class* cl_shared  [static]
```

Variable used to create the device node in the /dev directory.

4.2.4.6 cl_synchronous

```
struct class* cl_synchronous  [static]
```

Variable used to create the device node in the /dev directory.

4.2.4.7 dev_num_registration

```
dev_t dev_num_registration  [static]
```

Unique device identifier assigned the registration device.

4.2.4.8 dev_num_shared

```
dev_t dev_num_shared  [static]
```

Unique device identifier assigned the shared memory device.

4.2.4.9 dev_num_synchronous

```
dev_t dev_num_synchronous  [static]
```

Unique device identifier assigned the synchronous device.

4.2.4.10 first

```
queue_pid* first = NULL
```

Pointer to the first element of the queue queue_pid.

4.2.4.11 first_sync

```
sync_pid* first_sync = NULL
```

Pointer to the first sync_pid element in the queue.

4.2.4.12 head

```
Queue_buffer* head = NULL
```

4.2.4.13 last

```
queue_pid** last = NULL
```

Pointer to the last element of the queue queue_pid.

4.2.4.14 last_sync

```
sync_pid** last_sync = NULL
```

Pointer to the last sync_pid element in the queue.

4.2.4.15 PQ

```
Queue_mes** PQ
```

Array of the 10 priority queues, one for each priority level.

4.2.4.16 pq_last

```
Queue_mes* pq_last[10] = {NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL}
```

Array of pointers to the last element of each PQ[i].

See also

Queue_mes **PQ

4.2.4.17 registration_buffer

```
char* registration_buffer  [static]
```

Pointer to the buffer registration.

See also

REGISTRATION_SIZE

4.2.4.18 sem

```
struct semaphore sem  [static]
```

Declaration of the semaphore sem; This semaphore ensures each Queue_mes got a unique arrival.

4.2.4.19 shared_buffer

```
char* shared_buffer  [static]
```

Pointer to the buffer shared memory.

See also

SHARED_MEM_SIZE

4.2.4.20 synchronous_buffer

```
char* synchronous_buffer  [static]
```

Pointer to the buffer synchronous.

See also

SYNCHRONOUS_SIZE

4.3 IPC_SO.h

Go to the documentation of this file.

```
00001 #ifndef IPS_SO_H
00002 #define IPS_SO_H
00003
00004 //Libraries
00005 #include <linux/module.h>
00006 #include <linux/init.h>
00007 #include <linux/cdev.h>
00008 #include <linux/fs.h>
00009 #include <linux/device.h>
00010 #include <linux/uaccess.h>
00011 #include <linux/slab.h>
00012 #include <linux/jiffies.h>
00013 #include <uapi/linux/wait.h>
00014 #include <linux/semaphore.h>
00015
00016
00017 //Constant definition
00019 #define REGISTRATION_DEVICE "registration"
00021 #define SYNCHRONOUS_DEVICE "synchronous"
00023 #define SHARED_MEM_DEVICE "shared_memory"
00024
00025
00027 #define REGISTRATION_SIZE 1024
00029 #define SYNCHRONOUS_SIZE 1024
00031 #define SHARED_MEM_SIZE 4096
00032
00035 static struct semaphore sem;
00036
00037
00038 //Devices utilities:
00040 static dev_t dev_num_registration;
00042 static struct cdev cdev_registration;
00044 static struct class *cl_registration;
00045
00047 static dev_t dev_num_synchronous;
00049 static struct cdev cdev_synchronous;
00051 static struct class *cl_synchronous;
00052
00054 static dev_t dev_num_shared;
00056 static struct cdev cdev_shared;
00058 static struct class *cl_shared;
00059
00060
00061 //Devices buffers:
00064 static char *registration_buffer;
00067 static char *synchronous_buffer;
00070 static char *shared_buffer;
00071
00072
00073 //Global: Sync
00075 typedef struct sync_pid {
00076     int pid;                            //pid of the process
00078     wait_queue_head_t wait_queue;       //initial wait queue for the process
00080     wait_queue_head_t wait_queue_delay; //wait queue for the delay
00082     bool letto;                         //flag to check if the wait queue has been woken up
00083     int delay;                          //delay of the process
00085     bool change_delay;                  //flag to check if the delay has been changed
00087     int first_time;                     //flag to check the wait_queue
00089     int arrival;                        //arrival time of the message (ID)
00091     int effective_delay;                //effective delay of the message
00092     struct sync_pid *next;              //pointer to the next element
00093 } sync_pid;
00094
```

```
00097 typedef struct changeData{
00099     int arrival;            //arrival time of the message (ID)
00100     int delay;              //delay of the process
00102     int effective_delay;    //effective delay of the message
00103 } changeData;
00104
00105
00107 sync_pid *first_sync = NULL;
00109 sync_pid **last_sync = NULL;
00110
00111
00112 //Global: Priority Queue
00114 typedef struct Message{
00115     int pid_recipient;    //recipient's PID
00116     int pid_sender;       //sender's PID
00118     int priority;         //priority of the message
00120     int delay;            //delay of the message
00122     char payload[128];    //payload in bytes
00123 } Message;
00124
00126 typedef struct Queue_mes{
00128     Message mes;             //nested field struct Message
00131     unsigned long arrival;   //time of arrival
00133     struct Queue_mes *next;  //pointer to the next element
00134 } Queue_mes;
00135
00137 typedef struct Queue_buffer{
00140     Message mymes;              //buffer queue item
00142     struct Queue_buffer *next;  //pointer to the next element
00143 } Queue_buffer;
00144
00147 Queue_mes *pq_last[10] = {NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL}; //global 'last'
      label to the last pointer of each Queue_mes.
00149 Queue_mes **PQ;
00150 //Pointer to the first element of the queue Queue_buffer
00151 Queue_buffer *head = NULL;
00152
00153
00154 //Global: registration
00156 typedef struct queue_pid{
00158     int pid;
00160     struct queue_pid *next;
00161 } queue_pid;
00162
00164 queue_pid *first = NULL;
00166 queue_pid **last = NULL;
00167
00168
00169
00170 //FUNCTIONS DECLARATION:
00171
00172
00180 static int __init ipc_os_module_init(void);
00182 static void __exit ipc_os_module_exit(void);
00183
00184
00185 //Registration Functions
00190 static int registration_open(struct inode *inode, struct file *file);
00195 static int registration_release(struct inode *inode, struct file *file);
00204 static ssize_t registration_read(struct file *file, char __user *user_buf, size_t size, loff_t
      *offset);
00214 static ssize_t registration_write(struct file *file, const char __user *user_buf, size_t size, loff_t
      *offset);
00215
00216
00217 //Shared memory functions
00222 static int shared_memory_open(struct inode *inode, struct file *file);
00227 static int shared_memory_release(struct inode *inode, struct file *file);
00236 static ssize_t shared_memory_read(struct file *file, char __user *user_buf, size_t size, loff_t
      *offset);
00247 static ssize_t shared_memory_write(struct file *file, const char __user *user_buf, size_t size, loff_t
      *offset);
00248
00249
00250 //Synchronous functions
00255 static int synchronous_open(struct inode *inode, struct file *file);
00260 static int synchronous_release(struct inode *inode, struct file *file);
00269 static ssize_t synchronous_read(struct file *file, char __user *user_buf, size_t size, loff_t
      *offset);
00278 static ssize_t synchronous_write(struct file *file, const char __user *user_buf, size_t size, loff_t
      *offset);
00279
00280
00281 //Sync functions
00286 sync_pid *sync_init(int id);
00290 void sync_insert(sync_pid **first_sync, sync_pid *item);
00294 void sync_delete(sync_pid **first_sync, int target);
```

```
00297 void sync_deinit(sync_pid **first_sync);
00303 sync_pid *sync_search(sync_pid **first_sync, int val);
00307 Message PriorityDequeue_sync(int target_arrival);
00311 changeData sync_mes_find(int target_pid);
00316 int sync_mes_find_check(int var);
00317
00318
00319
00320 //Registration Queue_mes functions:
00325 queue_pid *reg_init(int id);
00329 void reg_insert(queue_pid **first, queue_pid *item);
00333 void reg_delete(queue_pid **first, int target);
00336 void reg_deinit(queue_pid **first);
00342 int reg_search(queue_pid **first, int val);
00343
00344
00345 //PQ functions
00350 Queue_mes **pqinit(void);
00355 void PriorityDequeue_mes(Queue_mes **first, int pid_caller);
00358 void pq_print(Queue_mes **t);                                  //print all the 10 priority queues
00362 void PriorityEnqueue_mes(Queue_mes **t, Message *m);           //enqueue a new message
00365 void pq_deinit(Queue_mes **t);                                //free the memory
00366
00367
00368
00369 //Buffer Queue functions
00374 Queue_buffer *bq_init(Message tmp);
00377 void bq_enqueue(Queue_buffer *tmp);
00381 Message *bq_dequeue(void);
00382
00383
00384 #endif
```

4.4 /home/vboxuser/Desktop/test/user/user_process.c File Reference

```
#include "USER_PROCESS.h"
```

Functions

- int main ()
- Message * newMessage ()

  Function that allows you to create a new message.
- int myRandom (int min, int max)

  Function that chooses a random number between min and max.
- int mymenu (int reg)

  Function that prints the menu to choose the operation to carry out.
- int reg_unreg (int reg)

  Function that allows the registration/delete of the process inside the kernel.
- int process_avaiable ()

  Function that allows you to view all the processes registered within the kernel.
- bool check (int pid)

  Function that checks if the entered pid is present in the list of processes registered in the kernel.
- int mes_write ()

  Function that sends the message to a recipient chosen by the user.
- int mes_read ()

  Function used to read incoming messages.
- int mes_read_sync ()

  Function used to read incoming messages (synchronously)
- void myerr (int er)

  Function that prints the description of the errors within the process.

4.4.1 Function Documentation

4.4.1.1 check()

```
bool check (
            int pid )
```

Function that checks if the entered pid is present in the list of processes registered in the kernel.

Parameters

pid	Indicates the PID to be searched

Returns

true If it finds the PID you are looking for

false If it does not find the PID you are looking for

See also

Use the process_avaiable() function to check process PIDs

4.4.1.2 main()

```
int main ( )
```

4.4.1.3 mes_read()

```
int mes_read ( )
```

Function used to read incoming messages.

Returns

0 -> If there are no errors

7 -> If the process fails to open the shared memory

4.4.1.4 mes_read_sync()

```
int mes_read_sync ( )
```

Function used to read incoming messages (synchronously)

Returns

0 -> If there are no errors

8 -> If the process fails to open the shared memory

9 -> If the process fails the opening of synchronous processes

10 -> If the process does not write the pid in the registration device

4.4.1.5 mes_write()

```
int mes_write ( )
```

Function that sends the message to a recipient chosen by the user.

Returns

0 -> If there are no errors

5 -> If the process fails to open the shared memory

6 -> If the process fails to write to shared memory

4.4.1.6 myerr()

```
void myerr (
            int er )
```

Function that prints the description of the errors within the process.

4.4.1.7 mymenu()

```
int mymenu (
            int reg )
```

Function that prints the menu to choose the operation to carry out.

Parameters

reg	Indicates whether the process is already registered in the kernel

Returns

op Returns the operation that was selected

4.4.1.8 myRandom()

```
int myRandom (
            int min,
            int max )
```

Function that chooses a random number between min and max.

Parameters

min	Lower Limit
max	Upper Limit

4.4.1.9 newMessage()

```
Message * newMessage ( )
```

Function that allows you to create a new message.

Returns

tmp Returns the structure of the message with all fields

4.4.1.10 process_avaiable()

```
int process_avaiable ( )
```

Function that allows you to view all the processes registered within the kernel.

Returns

0 -> If there are no errors

3 -> If the process fails to open the shared memory

4 -> If the process fails read the pid from the kernel

4.4.1.11 reg_unreg()

```
int reg_unreg (
            int reg )
```

Function that allows the registration/delete of the process inside the kernel.

Parameters

reg	Indicates whether the process is already registered in the kernel

Returns

0 -> If there are no errors

1 -> If the process fails the opening of the registration device

2 -> If the process fails write to the registration device

4.5 /home/vboxuser/Desktop/test/user/USER_PROCESS.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
```

```
#include <string.h>
#include <time.h>
#include <stdbool.h>
```

Data Structures

- struct Message

  This is the declaration of the type Message.

Macros

- #define AUTO 0

  Defines a debug variable to auto-initialized struct Message.
- #define REGISTRATION_DEVICE "/dev/registration"
- #define SHARED_MEM_DEVICE "/dev/shared_memory"
- #define SYNCHRONOUS_DEVICE "/dev/synchronous"
- #define MAX_REGISTRATIONS 256

  Defines the maximum number of PIDs that can be written into the registration buffer.

Typedefs

- typedef struct Message Message

  Message structure.

Functions

- Message * newMessage ()

  Function that allows you to create a new message.
- int myRandom (int min, int max)

  Function that chooses a random number between min and max.
- int mymenu (int reg)

  Function that prints the menu to choose the operation to carry out.
- int reg_unreg (int reg)

  Function that allows the registration/delete of the process inside the kernel.
- int process_avaiable ()

  Function that allows you to view all the processes registered within the kernel.
- int mes_write ()

  Function that sends the message to a recipient chosen by the user.
- int mes_read ()

  Function used to read incoming messages.
- int mes_read_sync ()

  Function used to read incoming messages (synchronously)
- void myerr (int er)

  Function that prints the description of the errors within the process.
- bool check (int pid)

  Function that checks if the entered pid is present in the list of processes registered in the kernel.

4.5.1 Macro Definition Documentation

4.5.1.1 AUTO

```
#define AUTO 0
```

Defines a debug variable to auto-initialized struct Message.

4.5.1.2 MAX_REGISTRATIONS

```
#define MAX_REGISTRATIONS 256
```

Defines the maximum number of PIDs that can be written into the registration buffer.

4.5.1.3 REGISTRATION_DEVICE

```
#define REGISTRATION_DEVICE "/dev/registration"
```

4.5.1.4 SHARED_MEM_DEVICE

```
#define SHARED_MEM_DEVICE "/dev/shared_memory"
```

4.5.1.5 SYNCHRONOUS_DEVICE

```
#define SYNCHRONOUS_DEVICE "/dev/synchronous"
```

4.5.2 Typedef Documentation

4.5.2.1 Message

```
typedef struct Message Message
```

Message structure.

4.5.3 Function Documentation

4.5.3.1 check()

```
bool check (
            int pid )
```

Function that checks if the entered pid is present in the list of processes registered in the kernel.

Parameters

pid	Indicates the PID to be searched

Returns

true If it finds the PID you are looking for

false If it does not find the PID you are looking for

See also

Use the process_avaiable() function to check process PIDs

4.5.3.2 mes_read()

```
int mes_read ( )
```

Function used to read incoming messages.

Returns

0 -> If there are no errors

7 -> If the process fails to open the shared memory

4.5.3.3 mes_read_sync()

```
int mes_read_sync ( )
```

Function used to read incoming messages (synchronously)

Returns

0 -> If there are no errors

8 -> If the process fails to open the shared memory

9 -> If the process fails the opening of synchronous processes

10 -> If the process does not write the pid in the registration device

4.5.3.4 mes_write()

```
int mes_write ( )
```

Function that sends the message to a recipient chosen by the user.

Returns

0 -> If there are no errors

5 -> If the process fails to open the shared memory

6 -> If the process fails to write to shared memory

4.5.3.5 myerr()

```
void myerr (
            int er )
```

Function that prints the description of the errors within the process.

4.5.3.6 mymenu()

```
int mymenu (
            int reg )
```

Function that prints the menu to choose the operation to carry out.

Parameters

reg	Indicates whether the process is already registered in the kernel

Returns

op Returns the operation that was selected

4.5.3.7 myRandom()

```
int myRandom (
            int min,
            int max )
```

Function that chooses a random number between min and max.

Parameters

min	Lower Limit
max	Upper Limit

4.5.3.8 newMessage()

```
Message * newMessage ( )
```

Function that allows you to create a new message.

Returns

tmp Returns the structure of the message with all fields

4.5.3.9 process_avaiable()

```
int process_avaiable ( )
```

Function that allows you to view all the processes registered within the kernel.

Returns

0 -> If there are no errors

3 -> If the process fails to open the shared memory

4 -> If the process fails read the pid from the kernel

4.5.3.10 reg_unreg()

```
int reg_unreg (
            int reg )
```

Function that allows the registration/delete of the process inside the kernel.

Parameters

reg	Indicates whether the process is already registered in the kernel

Returns

0 -> If there are no errors

1 -> If the process fails the opening of the registration device

2 -> If the process fails write to the registration device

4.6 USER_PROCESS.h

Go to the documentation of this file.

```
00001 #ifndef USER_PROCESS_H
00002 #define USER_PROCESS_H
00003
00004 //Libraries
00005 #include <stdio.h>
00006 #include <stdlib.h>
00007 #include <fcntl.h>
00008 #include <unistd.h>
00009 #include <stdint.h>
00010 #include <string.h>
00011 #include <time.h>
00012 #include <stdbool.h>
00013
00014
00015 //Type definition
00017 typedef struct Message{
00018     int pid_recipient;    //recipient's PID
00019     int pid_sender;       //sender's PID
00021     int priority;         //priority of the message
00023     int delay;            //delay of the message
00025     char payload[128];    //payload in bytes
00026 } Message;
00027
00028
00029 //Constant definition
00031 #define AUTO 0
```

```
00032 #define REGISTRATION_DEVICE "/dev/registration"
00033 #define SHARED_MEM_DEVICE "/dev/shared_memory"
00034 #define SYNCHRONOUS_DEVICE "/dev/synchronous"
00036 #define MAX_REGISTRATIONS 256
00037
00038
00039 //Function declaration
00042 Message *newMessage();
00043
00047 int myRandom(int min, int max);
00048
00052 int mymenu(int reg);
00053
00059 int reg_unreg(int reg);
00060
00065 int process_avaiable();
00066
00071 int mes_write();
00072
00076 int mes_read();
00077
00083 int mes_read_sync();
00084
00086 void myerr(int er);
00087
00093 bool check(int pid);
00094
00095 #endif
```

Index