OS Project

Generated by Doxygen 1.9.8

Chapter 1

Transito navale di merci - Project report

1.1 Master

The "master.c" module serves as the main controller for the simulation system, managing ports, ships, and weather, through inter-process communication. It utilizes shared memory and semaphores for coordination between different processes.

The simulation involves the movement of ships between ports, cargo generation, trade activities and weather conditions management.

The struct **state** encapsulates the shared data structures and the weather process identifier. It includes pointers to shared data structures for the general configuration, ports, ships, and cargo.

- **shm_general_t**: used for general configuration;
- **shm_port_t**: shared memory segments for ports;
- **shm_ship_t**: shared memory segments for ships;
- **shm_cargo_t**: shared memory segments for cargo.

The `main()` initializes the signal handlers, reads the configuration from a file, and sets up shared memory and semaphores. Then it forks processes for ports, ships, and weather.

The simulation starts after synchronizing the processes using semaphores.

1.1.1 Process initialization

`run_ports()`, `run_ships()`, and `run_weather()` fork processes for ports, ships, and weather, respectively. These functions use the `run_process()` helper function for creating child processes.

1.1.2 Signal handlers

`signal_handler_init()` sets up signal handlers for various signals such as SIGALRM, SIGSEGV, SIGTERM, and SIGINT. The signals are used for daily reporting, error handling, and graceful termination.

1.1.3 Reporting functions

`print_daily_report()` and `print_final_report()` display daily and final simulation reports, respectively. These reports include information about cargo, ships, ports, and weather conditions.

1.1.4 Terminating the simulation

When we reach the end of the simulation (after SO_DAYS seconds), ships and ports processes detach themselves from shared memory and terminate.

The master process waits for child processes termination, then it cleans up the shared memory and semaphores.

- `check_ships_all_dead()` determines whether all ships are dead.
- `close_all()` terminates the simulation, sending signals to all relevant processes, deleting IPC resources, and printing the final report.

1.2 Shared memory

`lib/shm.h` is a helper library that has been used as a facilitation to create/attach/detach/destroy shared memory segment on the aforementioned `shm_*` header files dedicated to the shared structures.

1.3 Semaphore

`lib/semaphore.h` is a helper library that has been used as a facilitation to create/handle/destroy arrays of semaphores.

Semaphores have been used to synchronize the starting of the simulation, for managing port docks and for managing access to the cargo share memory.

1.4 Signal

- **SIGDAY**: defined as SIGUSR1, used by master to signal a new day which triggers new cargo generations and daily reports;
- **SIGSWELL**: defined as SIGUSR2, used by weather to signal if a SWELL occurs to a port;
- **SIGSTORM**: defined as SIGUSR2, used by weather to signal if a STORM occurs to a ship;
- **SIGMAELSTROM**: defined as SIGTERM, used by weather to signal if a MAELSTROM occurs to a ship that terminates after it.

1.5 Message

`src/msg_commerce.h` contains structures and functions to handle messages between ports and ships.

Every message has a status used to decode the request and brings a support structure containing all the possible informations.

From port to ship message status:

- **STATUS_ACCEPTED**: port accepts all the offer proposed from the ship;
- **STATUS_PARTIAL**: port accepts a part of the offer proposed from the ship, depending on the request;
- **STATUS_REFUSED**: port refuses the ship's offer.

From ship to port message status:

- **STATUS_SELL**: ship sends goodies to port
- **STATUS_BUY**: ship take goodies from port

1.6 Port

The port interacts with ships, manages cargo, and participates in commerce through offers and demands.

1.6.1 Functionality

- `main()` initializes the state, attaches to shared memory, generates coordinates, and enters the main loop.
- `loop()` represents the main operational logic of the port, handling daily tasks and processing incoming commerce messages.
- `respond_ship_msg()` manages the response to commerce messages, including buying and selling cargo.
- `signal_handler()` handle various signals, including simulating swell effects and responding to termination signals.

1.7 Ship

The ship moves between ports, trades cargo, and responds to signals such as storms and maelstroms.

1.7.1 Functionality

- `main()` initializes the state, attaches to shared memory, generates initial location, and enters the main loop.
- `loop()` moves to a randomly chosen port and starts trading.
- `trade()` manages the trade process, including buying and selling cargo.
- `sell()` initiates the process of selling cargo to the current port sending a commerce message to the port and processes the response.
- `buy()` initiates the process of buying cargo from the current port sending a commerce message to the port and processes the response.
- `signal_handler()` handles various signals, including simulating storms and maelstroms and responding to termination signals.

1.8 Weather

At the beginning of each simulation's day the weather process receives the SIGDAY signal from the master process. It then proceeds to send the SIGSTORM and SIGSWELL signals to random ships and ports respectively. It also implements an itimer in order to be able to send the SIGMAELTROM signal to random ship every SO_MAELTROM simulated hours.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 node Struct Reference

Node structure representing a cargo item.

Data Fields

- int **quantity**
- int **expire**
- struct node ∗ **next**

4.1.1 Detailed Description

Node structure representing a cargo item.

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

- src/cargo_list.c

4.2 o_list Struct Reference

Data Fields

- struct node ∗ **head**

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

- src/cargo_list.c

4.3 shm_cargo Struct Reference

Data Fields

- int **batch_size**
- int **batch_life**
- int **max_offer**
- int **max_demand**
- int **dump_total_generated**
- int **dump_expired_in_port**
- int **dump_expired_on_ship**
- int **dump_received_in_port**
- int **dump_id_max_offer**
- int **dump_id_max_demand**
- int **dump_available_in_port**
- int **dump_available_on_ship**

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

- src/shm_cargo.c

4.4 shm_demand Struct Reference

Data Fields

- int **data**
- int **dump_tot_demanded**

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

- src/shm_offer_demand.c

4.5 shm_general Struct Reference

Data Fields

- double **so_lato**
- int **so_days**
- int **so_navi**
- int **so_speed**
- int **so_capacity**
- int **so_porti**
- int **so_banchine**
- int **so_fill**
- int **so_loadspeed**
- int **so_merci**
- int **so_size**
- int **so_min_vita**

- int **so_max_vita**
- int **so_storm_duration**
- int **so_swell_duration**
- int **so_maelstrom**
- int **current_day**
- int **general_shm_id**
- int **ship_shm_id**
- int **port_shm_id**
- int **cargo_shm_id**
- int **offer_shm_id**
- int **demand_shm_id**
- int **msg_in_id**
- int **msg_out_id**
- int **sem_start_id**
- int **sem_port_init_id**
- int **sem_cargo_id**

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

- src/shm_general.c

4.6 shm_offer Struct Reference

Data Fields

- int **data**
- int **dump_tot_offered**

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

- src/shm_offer_demand.c

4.7 shm_port Struct Reference

Data Fields

- pid_t **pid**
- struct coord **coord**
- int **num_docks**
- bool_t **is_in_swell**
- bool_t **dump_had_swell**
- int **dump_cargo_available**
- int **dump_cargo_shipped**
- int **dump_cargo_received**
- int **sem_docks_id**

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

- src/shm_port.c

4.8 shm_ship Struct Reference

Data Fields

- pid_t **pid**
- int **capacity**
- bool_t **is_dead**
- bool_t **is_moving**
- bool_t **is_at_dock**
- struct coord **coords**
- bool_t **dump_had_storm**
- bool_t **had_maelstrom**

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

- src/shm_ship.c

4.9 state Struct Reference

Data Fields

- shm_general_t ∗ **general**
- shm_port_t ∗ **ports**
- shm_ship_t ∗ **ships**
- shm_cargo_t ∗ **cargo**
- shm_offer_t ∗ **offer**
- shm_demand_t ∗ **demand**
- pid_t **weather**
- int **id**
- shm_port_t ∗ **port**
- o_list_t ∗∗ **cargo_hold**
- int **current_day**
- shm_ship_t ∗ **ship**
- int **curr_port_id**

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

- src/master.c
- src/port.c
- src/ship.c
- src/weather.c

Chapter 5

File Documentation

5.1 lib/semaphore.h File Reference

Library that provides a more user friendly interface for System V semaphores.

```
#include <sys/sem.h>
```

Functions

- int sem_create (key_t sem_key, int nsems)

  Create a new semaphore array and returns the id. Initializes the permissions to 0660.
- int sem_get_id (key_t key)

  Get the id of an existing semaphore associated with the given key.
- void sem_setval (id_t sem_id, int sem_index, int value)

  Sets the initial value of a semaphore in the array.
- int sem_getval (id_t sem_id, int sem_index)

  Returns the current value of a semaphore.
- void sem_execute_semop (id_t sem_id, int sem_index, int op_val, int flags)

  Executes a semaphore operation using semop system call.
- void sem_delete (id_t sem_id)

  Deletes a semaphore array.

5.1.1 Detailed Description

Library that provides a more user friendly interface for System V semaphores.

5.1.2 Function Documentation

5.1.2.1 sem_create()

```
int sem_create (
             key_t sem_key,
             int nsems )
```

Create a new semaphore array and returns the id. Initializes the permissions to 0660.

Parameters

sem_key	semaphore key, if 0 IPC_PRIVATE flag is used.
nsems	number of semaphores in the array.

Returns

the id of the semaphore array.

5.1.2.2 sem_delete()

```
void sem_delete (
             id_t sem_id )
```

Deletes a semaphore array.

Parameters

sem↩ _id	the id of the array that will be deleted.

5.1.2.3 sem_execute_semop()

```
void sem_execute_semop (
             id_t sem_id,
             int sem_index,
             int op_val,
             int flags )
```

Executes a semaphore operation using semop system call.

Parameters

sem_id	the id of the semaphore array.
sem_index	the index of the semaphore in the array.
op_val	the operation performed on the semaphore.
flags	the flags used for the operation. Use 0 for no flags.

5.1.2.4 sem_get_id()

```
int sem_get_id (
             key_t key )
```

Get the id of an existing semaphore associated with the given key.

Parameters

key	the key of an existing semaphore set.

Returns

the id of the semaphore set.

5.1.2.5 sem_getval()

```
int sem_getval (
            id_t sem_id,
            int sem_index )
```

Returns the current value of a semaphore.

Parameters

sem_id	the id of the semaphore array.
sem_index	the index of the semaphore.

Returns

the value of the semaphore.

5.1.2.6 sem_setval()

```
void sem_setval (
            id_t sem_id,
            int sem_index,
            int value )
```

Sets the initial value of a semaphore in the array.

Parameters

sem_id	the id of the semaphore array.
sem_index	the index of the semaphore in the array.
value	the value to set.

5.2 semaphore.h

Go to the documentation of this file.

```
00001
00006 #ifndef OS_PROJECT_SEMAPHORE_H
00007 #define OS_PROJECT_SEMAPHORE_H
```

```
#include <sys/sem.h>

int sem_create(key_t sem_key, int nsems);

int sem_get_id(key_t key);

void sem_setval(id_t sem_id, int sem_index, int value);

int sem_getval(id_t sem_id, int sem_index);

void sem_execute_semop(id_t sem_id, int sem_index, int op_val, int flags);

void sem_delete(id_t sem_id);


#endif
```

5.3 lib/shm.h File Reference

Library that provides a more user friendly interface for System V shared memory.

```
#include <sys/shm.h>
#include <sys/stat.h>
```

Functions

- int shm_create (key_t key, size_t size)

  Creates a new shared memory segment.
- void shm_delete (int id_shm)

  Deletes the shared memory segment.
- void ∗ shm_attach (int id_shm)

  Attaches a shared memory segment.
- void shm_detach (void ∗shm_ptr)

  Detaches a shared memory segment.

5.3.1 Detailed Description

Library that provides a more user friendly interface for System V shared memory.

5.3.2 Function Documentation

5.3.2.1 shm_attach()

```
void * shm_attach (
             int id_shm )
```

Attaches a shared memory segment.

Parameters

id_shm	the id of the shared memory segment.

Returns

5.3.2.2 shm_create()

```
int shm_create (
             key_t key,
             size_t size )
```

Creates a new shared memory segment.

Parameters

key	the key for the shared memory.
size	the size of the segment.

Returns

the id of the created segment. If segment exists with key the id of the existing segment.

5.3.2.3 shm_delete()

```
void shm_delete (
             int id_shm )
```

Deletes the shared memory segment.

Parameters

id_shm	the id of the shared memory segment.

5.3.2.4 shm_detach()

```
void shm_detach (
             void * shm_ptr )
```

Detaches a shared memory segment.

Parameters

shm_ptr	the pointed to the segment.

5.4 shm.h

Go to the documentation of this file.

```c

#ifndef OS_PROJECT_SHM_H
#define OS_PROJECT_SHM_H

#include <sys/shm.h>
#include <sys/stat.h>

int shm_create(key_t key, size_t size);

void shm_delete(int id_shm);

void *shm_attach(int id_shm);

void shm_detach(void *shm_ptr);

#endif
```

Index