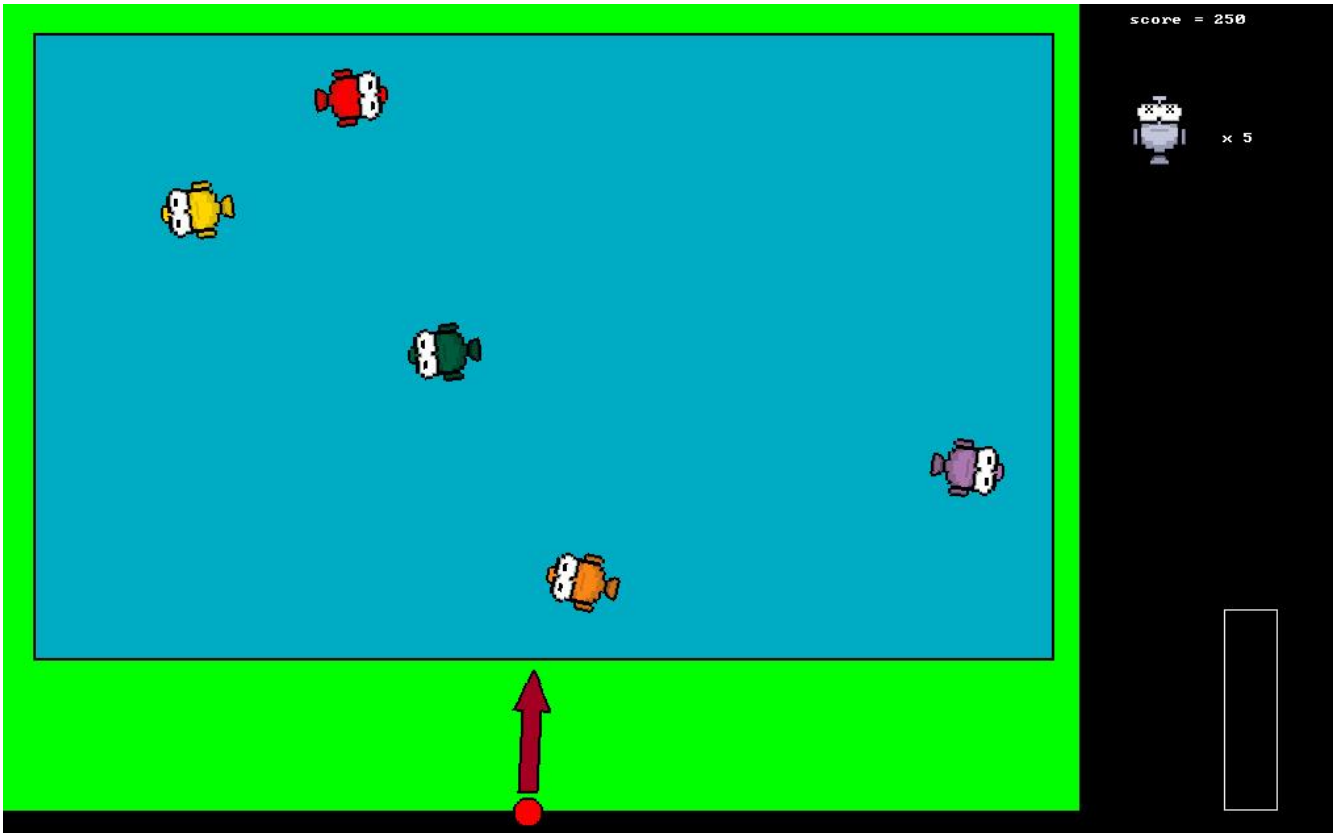
Fishing

Embedded Computing Systems



Elena Lucherini, 482960, eleluche@gmail.com

Sara Falleni, 479374, sarafalleni05@gmail.com

Table of contents

Overview

The project we realized is the Fishing game. The screen shows a lake with fishes swimming in it: the player can fish them, choosing in which direction and how far to cast the line. The fish will notice the bait only if it is of the same color as itself and is within the fish's range of view. After having caught at least one fish, the player can add another fish to the lake. The game ends when all fishes in the lake have been caught.

When in the game, the possible actions that can be taken are:

- Choose the direction of your cast with the right and left arrows.
- Press and hold the space bar to determine how far to cast: an orange bar will appear. When the bar's maximum is reached and the player is still holding the space bar, the bar will go back to its minimum and start growing again – and so on, until the space bar is released. The bait will be shown only if it is cast within the lake's edges. Please note that the bait disappears after 5 seconds from your cast and it is not possible to cast again until then or until it has been eaten.
- Add another fish in the lake by pressing the N key. This is only possible when at least one fish has been caught.

Whenever a fish gets caught, the temporary score is increased by 50 points. At the end of the game, the final score is computed taking into account the time taken to complete the game and the collected points.

The project is structured in five tasks (one that manages the fishes, one for the arrow, one for the bar, one for the bait and the main thread), along with one task for each fish. The inter-task communication is managed through shared data structures.

As far as the code is concerned, it is divided in six files: `resources.h`, `fish.h`, `arrow.h`, `bar.h`, `bait.h` and `fishing.c`.

Resources.h

`Resources.h` is the file where the libraries, the structures, the global variables, the semaphores and the utility functions are defined.

Libraries

The libraries which have been included to the project, in addition to `allegro.h` and `pthread.h`, are, among others, `semaphore.h`, `time.h`, `math.h`.

Structures

Below is a list of the structures that have been defined:

- `fish_struct`: it is the structure that describes the fish and contains the information used by the threads to manage the fish. The structure contains: the pointers to two images representing the fish, one being the flipped version of the other; the coordinates and the inclination of the fish; a few variables to manage its trajectory; two variables for the color; and three variables to manage the fish with respect to the environment.
- `arrow_struct`: it defines the arrow. It is a simple structure, containing only a pointer to an image, a variable that defines its inclination used to move the arrow and compute the arrival point of the bait, and a boolean used to either enable or disable the keys that control the arrow.
- `bait_struct`: it is the structure describing the bait. It contains: two pointers to the images of the bait used for moving and drawing it; two variables for the position of the bait; a variable defining the color of the bait; two timers; and a boolean variable to detect if the bait has been cast out of the lake.
- `task_par`: it is used to manage the period task, as seen on the course slides.

Global variables

Most of the global variables are used for the inter-task communication, while some others are defined as a utility. One of the most important ones is `array_color`, an array of booleans, having as many elements as the number of defined colors, that memorizes which fishes have been caught.

As for the bait, a `bait_struct` structure is globally defined. In addition to it, there are three global boolean variables that are used, which are:

- `thrown_b`: it is initialized to 0 and then set when the space bar is released. It is reset if the bait has been eaten or after 5 seconds from the cast.
- `bait_eaten`: it is initialized to 0 and set when the fish eats the bait. It is used to update the score. It is reset at the next cast.
- `new_calculation`: zero-initialized, it is set when the space bar is released: it is used to notify the tasks that the coordinates have to be updated.

As far as the bar is concerned, a variable `bar_working` is defined. It is initialized to zero and it is used to notify that the space bar is being hold. Another related variable is `force`, an integer that is used to compute the distance to which the bait will be shot.

In order to manage the screen and the in-game error messages, three pointers to as many different bitmaps are defined: `background`, `msg_missed`, `msg_full`.

Finally, the score is managed by a global variable, `score`, and the number of caught fishes is kept in `counter`. An instance of `arrow_struct` is globally defined.

Semaphores

In order to protect the global data structures, the following binary semaphores have been defined: `mux_screen`, `mux_arrow`, `mux_thrown`, `mux_eaten`, `mux_new_calc`, `mux_bait`, `mux_bar`, `mux_array_color`.

Functions

- `check_end()` determinates when the game is finished by checking `array_color`.
- `get_interval()` returns the time elapsed between timer t1 and t2.
- `bait_struct_init()` and `arrow_struct_init()` are used to initialize the respective structure instances.

The other functions defined in the file are used to manage periodic tasks, as seen on the slides.

Fishing.c

`Fishing.c` contains the body of the main thread, as well as the body of the task used to manage the creation of the fish. It also includes all the files that form the project.

The main thread

All the initializations, except for the ones related to the fish structures, are done in the main thread. First of all, the thread IDs, used to create new threads, are defined for the fish manager, the arrow, the bar and the bait threads. Secondly, the mutexes are initialized using the Priority Inheritance Protocol. The chosen scheduler uses the FIFO algorithm. The Allegro library and the screen with all the related static elements, the arrow structure and the bait structure are then initialized.

Then the thread creation and initialization follows. The periods of the threads are defined such that the game can flow smoothly and it is easily playable. The priorities are assigned by the Rate Monotonic algorithm.

Finally, all the threads created are joined. A section defining the behavior of the game at the end follows. The time elapsed from the game start is shown together with the final score.

Fish manager

The manager is a periodic task which creates the fish threads and then manages the addition of new fishes in the lake in its periodic code.

As for the creation of the threads, all the IDs of the threads are created, the bitmaps are loaded and the fish structures are initialized: different colors, positions and trajectory parameters are assigned to each fish structure. Finally, the threads are created and initialized: a different fish structure is passed by reference to each of them.

The periodic code of the manager waits for the N key to be pressed: by checking `array_color`, it determinates whether at least one fish has been caught or if the lake is still full. In the latter case, a message appears on screen and the thread will do nothing. Otherwise, it creates a new fish thread. The chosen implementation for the creation of a new fish is as follows: the manager cycles through `array_color` and the first 'free' color is used to create a new fish, with the same parameters of the caught one.

Fish.h

The file contains the body of the periodic fish tasks along with the functions used by them. Said functions are defined as follows:

- `calc_time()`: checks if the bait has not been eaten for 5 seconds. If so, the bait has to go back to its original position and the arrow has to be enabled again.
- `calc_distance()`: computes the distance between the fish and the bait.
- `fish_to_bait()`: moves the fish toward the bait.
- `fish_struct_init()`: initializes the fish structure.
- `rad_to_aldeg()`: converts radiants into allegro degrees (aldeg).
- `fish_rotate()`: returns the angle used by the fish to rotate in its trajectory to the bait.
- `fish_traj()`: defines the trajectory of the fish when it has not seen any bait.
- `find_bait()`: scans the pixels nearby for a bait of the same color as the fish.
- `fish_draw()`: deletes the bitmap of the fish from the old position and draws it in a new position.

As for the body of the fish thread, the `find_bait()` function is being called in the first part of the periodic code: the function scans the nearby pixels and returns `true` if a bait of the right color has been detected.

The fish's movements are defined as follows: `fish_traj()` generates a random trajectory, the x and y coordinates of the fish are updated with the fish's `speed` and `fish_draw()` is called. In particular, the fish's bitmap is replaced with its flipped version if the edge of the lake has been reached (basically, the fish turns around and keeps swimming); the bitmap is rotated accordingly to the trajectory defined by `fish_traj()`.

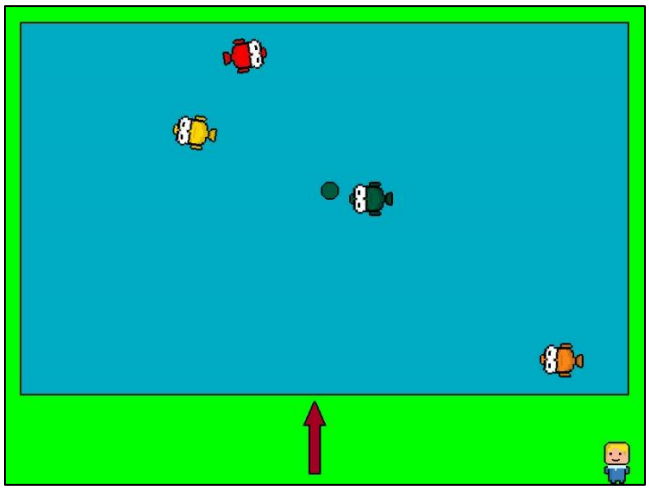
As soon as the bait has been cast, `calc_time()` is called to keep track of the time. The task then checks whether it has been cast in or out of the lake. In the latter case, an in-game error notification will appear and the player will have to cast again. If the cast successfully hit the lake and `find_bait()` returned `true`, the function `fish_to_bait()` is called until the fish is close enough to eat the bait (`calc_distance()` will take care of that), or the `calc_time()` function notifies the time is up: if the fish has been caught, the score is updated and the task ends; if the time is up, it goes back to its default movements, as described above. Either ways, the arrow is once again enabled and it is now possible to cast.

Bait.h

The file contains the body of the bait task, along with the functions used by it. The functions are defined as follows:

- `array_color_update()`: updates `array_color`. When a fish is caught, the color is set to `true`.
- `bar_to_length()`: defines how far to cast, depending on how much the bar has been filled.
- `out()`: checks if the bait has been cast out of the lake.
- `change_color()`: randomly computes the next color for the bait depending on the color of the fishes that are still in the lake. It also changes the bitmap of the bait with the correct one, based on the color.
- `aldeg_to_rad()`: converts a value in allegro degrees to its equivalent value in radians.

As for the body of the bait thread, it works as follows: if the bait has been cast, the thread computes the new position and draws the bait accordingly. Please note that, if the bait has been cast out of the lake, an in-game notification message appears for 2 seconds; afterwards, the bait goes back to its original position.



An important thing to notice is that every bait can be seen only from the fish with the same color. Whenever a fish eats a bait, a new one of a different color immediately appears to be casted again.

Arrow.h

The file contains the body of the periodic task of the arrow, along with an additional utility function:

- `arrow_draw()`: it deletes the arrow bitmap from the old position and it re-draws it rotated by a certain angle.

The thread lets the player move the arrow between the chosen limit angles with the right and left keys and then calls `arrow_draw()`. The angle for the rotation of the arrow is restrained to the [-48 aldeg, 48 aldeg] interval.

When the space key is pressed, the arrow is blocked; as soon as the arrow is enabled again, the thread enables the bar again.

Bar.h

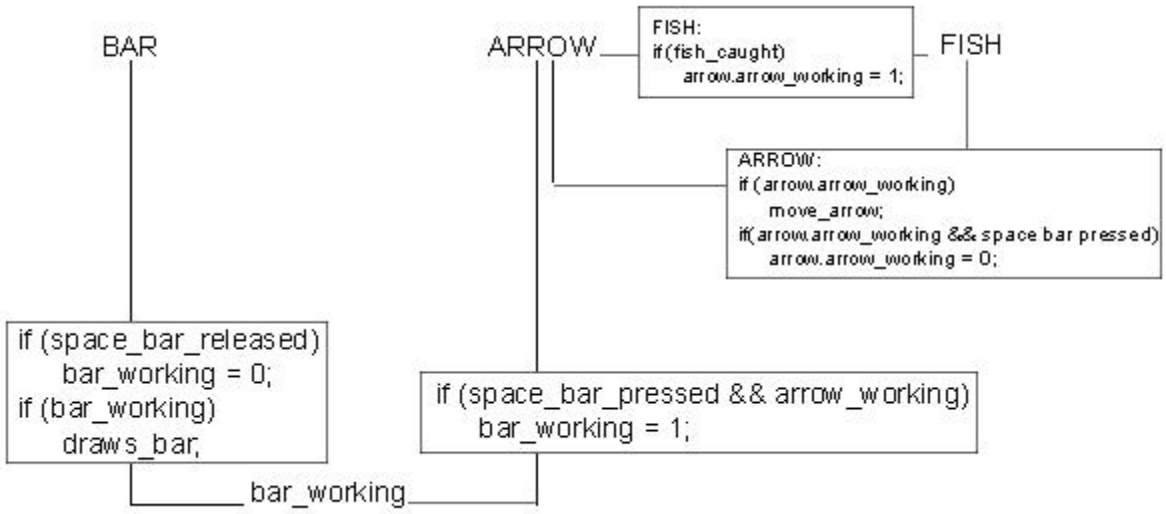
The file contains the periodic task of the bar and a few useful functions. The functions are defined as follows:

- `bar_draw()`: draws the bar; every time the bar is full, the function deletes the bar and make it start again from the beginning.
- `set_force()`: sets the force for the cast when the space key is released and then empties the bar.

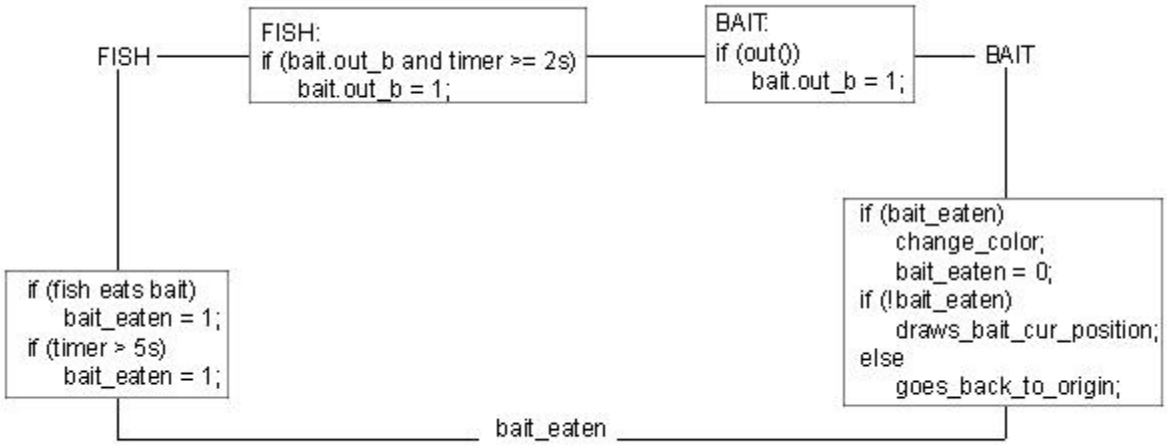
The periodic thread calls the `bar_draw()` function to increment the bar whenever the space key is pressed, or calls the `set_force()` function if the space bar is released.

Task interactions

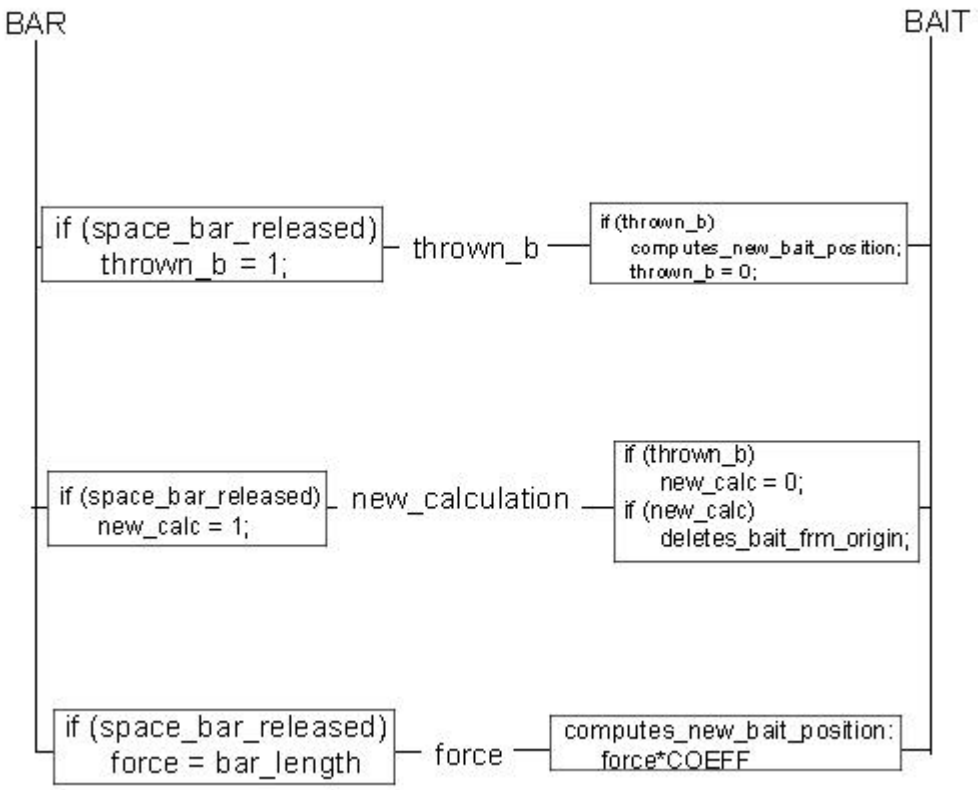
In these section, the most important task interactions are summarized with the help of simple graphs. The uppercase names represent the names of the tasks, the lowercase ones are the names of the global variables involved and the squares contain the actions taken by each task (the name of the task is mentioned when it is not clear).



Graph 1: Bar-Arrow-Fish interactions



Graph 2: Fish-Bait interactions



Graph 3: Bar-Bait interactions

Feasibility test

Worst-case execution time

In order to compute the worst-case execution time of each thread, an additional binary semaphore `mux_cpu` was created: the tasks were allowed to run independently from one another by executing the lock operation on `mux_cpu` at the beginning of the periodic code and unlocking the semaphore at the end of it. Moreover, a timer was set to start as soon as the task locked `mux_cpu` and was set to stop before the unlocking operation was executed. The program was then allowed to run and after a significant amount of tests, the greatest resulting times for each thread were picked. The results are shown in the table below, along with the deadlines of each task.

	Max computation time (ms)	Deadline (ms)
Arrow	11.527374	60
Bait	0.021045	40
Bar	4.642909	30
Fish	13.002396	60
Manager	0.483791	80

Schedulability analysis

Since the tasks are scheduled by the Rate Monotonic algorithm, the priorities are assigned with respect to the periods: the shorter the period, the higher the priority. We applied the Hyperbolic Bound test in order to verify the schedulability. In the table below are specified, for each task, the processor utilization factor U and the maximum blocking time B. The tasks are ordered by decreasing priority.

	U	B
τ_{bar}	0.154763	max (C_{bait}, C_{arrow}, C_{fish}, C_{man}) = 13.002396
τ_{bait}	0.000526	max (C_{arrow}, C_{fish}, C_{man}) = 13.002396
τ_{arrow}	0.192123	max (C_{fish}, C_{man}) = 13.002396
τ_{fish}	0.216707	C_{man} = 0.483791
$\tau_{manager}$	0.006047	0

$\tau_{manager}$

$$(U_{bar} + 1)(U_{bait} + 1)(U_{arrow} + 1)(U_{fish} + 1) = 1.67582 < 2 \qquad \text{OK}$$

τ_{fish}

$$(U_{bar} + 1)(U_{bait} + 1)(U_{arrow} + 1)(\frac{Cfish+Cman}{Tfish} + 1) = 1.686929 < 2 \qquad \text{OK}$$

τ_{arrow}

$$(U_{bar} + 1)(U_{bait} + 1)(\frac{Carrow+Cfish}{Tarrow} + 1) = 1.6277199 < 2 \qquad \text{OK}$$

τ_{bait}

$$(U_{bar} + 1)\ (\frac{Cbait+Cfish}{Tbait} + 1) = 1.530738 < 2 \qquad \text{OK}$$

τ_{bar}

$$(\frac{Cbar+Cfish}{Tbar} + 1) = 1.588177 < 2 \qquad \text{OK}$$

As shown in the tests above, the task set is schedulable by RM.