Processes

Design of Telematic Systems
Second Course. Bachelor in Robotics Engineering
Universidad Carlos III de Madrid
2024–2025

Instructions:

- The assignment must be done in groups of 2 or 3 students.
- Although we strongy recommend that you implement the exercises in Python (because it is easier to program), you can implement them in Python or in C language.
- **Exercise 1** (processes and signals) is the 40% of the final mark of the assignment.
- **Exercise 2** (processes, signals and pipes) is the 60% of the final mark of the assignment.
- Your code must be well documented. You must include comments in your code to explain the different parts of your code.
- There should be clear instructions on how to compile and run your code. Include a README file with the instructions within your submission.
- You must submit the source code of the exercises, a README file and a report in PDF format with the answers to the questions asked in the exercises.
- You must submit your assignment through Aula Global.
- Remember that you will be using your own code on the individual lab exam.

Objective

The objective of this graded lab is to implement a master program that manages N robots moving in a room. The code of the robots will be implemented in the first exercise, while the code of the master that controls all the robots will be implemented in the second exercise.

Understanding additional files

The room

The room is represented by a grid of squares. Each square or cell can contain a treasure or an obstacle that makes impossible to access to that square. Obstacles and treasures do not occupy the same cell.

The information of the room is stored in a file with the following structure:

- First line: number of rows and number of columns
- Second line: number of obstacles and position of each obstacle.
- Third line: number of treasures and position of each treasure.

Remember that we assume that obstacles and treasures do not occupy the same cell.
Example:

```
6 10
3 (0,0) (1,2) (3,4)
2 (1,1) (2,3)
```

This room is 6x10, and has 3 obstacles located at (0,0), (1,2), (3,4) and 2 treasures at (1,1) and (2,3).

Neither the master nor the robots are allowed to read directly this file. This information will be given by the `class Sensor` (or the module `sensor` if working in C language).

You can find two examples of this file at Aula Global, named `room.txt` and `other_room.txt`. Remember that your code should work for any given file name.

Position of the robots within the room

The master (to be implemented in the second exercise) needs to know the number of robots and their position within the room. This information is stored in a file that has one line per robot. The only information in this line is the position of the robot.
Example:

```
(2,3)
(1,4)
(1,5)
```

This file states that there are 3 robots, in the positions (2,3), (1,4) and (1,5).

You can find two examples of this file at Aula Global, named `robots.txt` and `robots_two.txt`. Remember that your code should work for any given file name.

The sensor

Each robot will have a sensor that will use to know if a given position has an obstacle (making impossible to move), or a treasure. **The sensor has no information about the whereabouts of the robots**.

The code of this sensor is already provided and you can find it a Aula Global, under the names `sensor.py` is you are working with Python, and `sensor.c` and `sensor.h` if you are working with C.

Working with Python

You should use the `class Sensor`, by importing the class, and you will interact with it only through its methods. **You are not allowed to access to any internal variable**.

In order to be able to work with the `class Sensor`, you should copy the Aula Global file `sensor.py` in your work directory and include this line at the beginning of your code:

```
import sensor
```

The `Sensor class` exposes the following methods:

- `print_room()`: help function for debugging. It prints the room.
- `dimensions()`: returns the dimensions of the room. It can be used only by the master.
- `n_treasures()`: returns the number of treasures in the the room. It can be used only by the master.
- `with_obstacle(row,column)`: returns `True` if there is an obstable in the given cell. Important: the master can only use it at the master's initialization. The robots can use this method at any given time.
- `with_treasure(row,column)`: returns `True` if there is a treasure in the given cell. Important: the master can only use it at the master's initialization. The robots can use this method at any given time.

You can find at Aula Global `test.py`, an example on how to use the `Sensor class`.

```
$ cat room.txt
6 10
3 (0,0) (1,2) (3,4)
2 (1,1) (2,3)
$ python3 test.py room.txt
X - - - - - - - - -
- T X - - - - - - -
- - - T - - - - - -
- - - - X - - - - -
- - - - - - - - - -
- - - - - - - - - -
dimensions of the room: (6, 10)
number of treasures: 2
(2,3) has no obstacle
(2,3) has a treasure
$ cat other_room.txt
3 15
5 (1,0) (1,2) (2,13) (1,5) (0,14)
3 (0,0) (2,7) (1,9)
$ python3 test.py other_room.txt
T - - - - - - - - - - - - - X
X - X - - X - - - T - - - - -
- - - - - - - T - - - - - X -
dimensions of the room: (3, 15)
number of treasures: 3
(2,3) has no obstacle
(2,3) has no treasure
```

Working with C language

You should use the `sensor` module, by including it in your code, and you will interact with it only through its methods. **You are not allowed to access to any internal variable**.

For including the module, you should copy the Aula Global files `sensor.c` and `sensor.h` in your work directory and include this line at the beginning of your code:

```
#include "sensor.h"
```

The module `sensor` exposes the following methods:

- `int init_sensorize(char *filename)`: initialization function of the sensorization module. This function returns 0 if there was success, and a number different from 0 otherwise.
- `void end_sensorize()`: function that should be called at the end of your program to free all the memory used by the sensorization.
- `void print_room()`: help function for debugging. It prints the room.
- `char with_obstacle(int row, int column)`: returns `1` if there is an obstable in the given cell, `1` otherwise. Important: the master can only use it at the master's initialization. The robots can use this method at any given time.
- `char with_treasure(int row, int column)`: returns `1` if there is a treasure in the given cell, `1` otherwise. Important: the master can only use it at the master's initialization. The robots can use this method at any given time.
- `int n_treasures()`: returns the number of treasures in the the room. It can be used only by the master.
- `void dimensions(int *rows, int *columns)`: returns in the positions pointed by `rows` and `columns` the dimensions of the room. It can be used only by the master.

You can find at Aula Global `test.c`, an example on how to use the module `sensor`.

```
$ gcc -Wall -g -o test test.c sensor.c
$ cat room.txt
6 10
3 (0,0) (1,2) (3,4)
2 (1,1) (2,3)
$ ./test room.txt
Rows: 6, Columns: 10
Number of X: 3
Number of T: 2
```

```
X---------
-TX-------
---T------
----X-----
----------
----------
Dimensions of the Room. Rows: 6, Columns: 10
Number of treasures: 6, Columns: 10
(2,3) has no obstacle
(2,3) has a treasure
$ cat other_room.txt
3 15
5 (1,0) (1,2) (2,13) (1,5) (0,14)
3 (0,0) (2,7) (1,9)
$ ./test other_room.txt
Rows: 3, Columns: 15
Number of X: 5
Number of T: 3
T-------------X
X-X--X---T-----
-------T-----X-
Dimensions of the Room. Rows: 3, Columns: 15
Number of treasures: 3, Columns: 15
(2,3) has no obstacle
(2,3) has no treasure
$
```

Exercise 1. Controlling a robot moving in a room

Implement a program that simulates a robot moving in a room. The room is a grid of squares and the robot moves in the room one square at a time. It can move up, down, left or right. However, it cannot move out of the room, or into a cell with an obstacle.

You can use Python or C language (we strongly recommend you to use Python), and the robot **should use** the interface provided by the provided library sensor (if working in C) or the provided class Sensor (if working in Python). You are not allowed to change the provided code or to access the room file directly.

The robot is controlled by a user through a command-line interface. The program admits the following commands:

- `mv <direction>`: tries to move the robot one cell in the specified direction. (up, down, left, right). If it is possible, the robots changes its position and prints `OK`. If it is not possible to move in the specified direction, its position does not change and prints `KO`.
- `bat`: prints the current battery level.
- `pos`: prints the current position of the robot as a 2-dimension vector.
- `tr`: checks if there is a treasure in the current position. If there is a treasure, it prints `Treasure`. If not, prints `Water`.
- `exit`: the program prints the current position of the robot and its battery level and exits.

The program should handle these signals:

- `SIGINT`: it *suspends* the robot. The robot cannot move until receiving the signal SIGQUIT. In this state, the robot does not consume battery. It can receive signals and commands and it responds to them.
- `SIGQUIT`: it resumes the robot. It can move again.
- `SIGTSTP`: The program prints the identifier of the robot, its position and its battery level.
- `SIGUSR1`: The program sets the battery level to 100.

Additionally, you should *simulate* the battery of the robot. The battery allows the robot to move a certain number of squares that decreases both by time passing and by moving:

- Each time the robot moves after a successful `mv` command, the battery diminishes by 5. If the user introduces a `mv` command and there is no enough battery to allow the robot to move, the robot won't move and will print `KO`.
- The battery level will be decremented by 1 every second. Use `SIGALRM` to implement this functionality.
- Once the battery level reaches 0, the robot cannot move, printing `KO` if it receives a `mv` command.
- Once its battery level is replenished by the reception `SIGUSR1`,the battery level will start again to be decremented by 1 every second.

The robot program receives the following arguments (**in any order**):

- A number that identifies the robot.
- `-f <filename>`:- the name of the file that contains the room information.
- `-pos <row> <column>`: (optional) the initial position of the robot. If not given, the position is `(0, 0)`.
- `-b <battery>`: (optional) the initial battery level. If not given, the level is `100`.

Example:

```
$ python3 robot.py 3 -pos 2 3 -b 50 -f room.txt
```

Here we are invoking a robot with identifier 3, its initial position is in row 2, column 3 and its initial battery level is 50. The room information is in the file room.txt.

At the very beginning the robot should print its PID to standard error and create an instance of the `Sensor` **`class`** with the filename given as input (if in Python) or initialize the sensors (if in C). Then, it should create if the robot's initial position is valid (if not, prints ”Invalid initial position” and exits) and waits for the user commands.
All error messages should be printed to the standard error stream.

Execution examples

We will use as example the output of the Python implementation. However, the output of the C implementation should be exactly the same. All these examples use as room description file the following.

```
6 10
3 (0,0) (1,2) (3,4)
2 (1,1) (2,3)
```

```
$ python3 robot.py 3 -pos 0 0 -f room.txt
PID: 12099
Invalid initial position
```

```
$ python3 robot.py 3 -pos 2 3 -b 50 -f room.txt
PID: 12108
bat
Battery: 48
pos
Position: 2 3
tr
Treasure at 2 3
mv up
OK
mv up
OK
mv up
Robot 3 cannot move up
KO
pos
Position: 0 3
bat
Battery: 9
bat
Battery: 4
bat
Battery: 0
mv down
Robot 3 cannot move  down
KO
mv left
Robot 3 cannot move  left
KO
pos
Position: 0 3
exit
Position: 0 3
Battery: 0
```

In the following example we send the signal SIGINT (using CTRL+C), we try several commands (without success as the robot is *stopped*), we *resume* the execution by sending SIGQUIT (using CTRL+\) and then, we send SIGTSTP (using CTRL+Z) to print the battery and the position. Finally, we send to the process SIGUSR1 from another terminal (using `kill -s USR1 6635`), that replenishes the battery of the robot.

```
$ python3 robot.py 3 -pos 3 3 -b 50 -f room.txt
PID: 6635
```

```
bat
Battery: 48
pos
Position: 3 3
^Cpos
Robot 3 is stopped
bat
Robot 3 is stopped
tr
Robot 3 is stopped
^\pos
Position: 3 3
bat
Battery: 41
^Z                id: 3 P: (3,3) Bat: 38
^Z                id: 3 P: (3,3) Bat: 37
tr
Water at 3 3
^Z                id: 3 P: (3,3) Bat: 96
exit
Position: 3 3
Battery: 92
$
```

Questions to answer in your report

Describe the behaviour of your program. You can use the following questions to guide you:

1. How do you store the position and battery status of the robot?
2. When do you invoke the functions provided by the sensor library/class?
3. How did you process the commands from the user?
4. How did you simulate the decrease of the battery level?
5. How did you handle the signals generated by the user? What did you do when you received them?

Exercise 2. Managing N robots moving in a room

The program admits the following commands:

- `mv <robot_id/all> <direction>`: moves the robot in the specified direction. (up, down, left, right).
  - `robot_id` is the identifier of the robot. If `robot_id` is *all*, the command is sent to all robots.

  - If it is not possible to move in the specified direction (because there is a wall, an obstacle, another robot in the desired new place or the robot has no enough battery), the robot does not move.
  - When `robot_id` is *all*, the movement should be tried in order, e.g. first the robot con identifier 1, then the robot with identifier 2, and so on.
  - If the robot finds a treasure, prints ”Treasure found by robot X!”, being X the identifier of the robot.
  - If the desired movement makes two robots collide, the robot does not move and prints ”Collision between robot x and y”, being *x* and `y` the identifiers of the robots.

- `print`: prints the information the master has so far.
- `bat <robot_id/all>`: prints the current battery level of the robot identified by `robot_id`. If `robot_id` is *all*, the command is sent to all robots.
- `pos <robot_id/all>`: prints the current position of the robot identified by `robot_id`. If `robot_id` is *all*, the command is sent to all robots.
- `suspend <robot_id/all>`: suspends the robot identified by `robot_id`. If `robot_id` is *all*, suspends all robots.
- `resume <robot_id/all>`: resumes the robot identified by `robot_id`. If `robot_id` is *all*, resumes the execution of all robots.
- `exit`: finishes the program. It sends the command to all the robots, printing the information sent by them: the position and the battery level. It also prints the position of all found treasures.

When all the treasures are found, the program must finish. The program should print the error messages to the standard error stream.
The program admits the following signals:

- `SIGINT`: it finishes the program. It prints the current position of the robots, their battery level, waits for the robots to die and exits.
- `SIGQUIT`: replenishes the battery of all robots to 100. In order to accomplish this, the program will send `SIGUSR1` to all the robots. It does not exit.
- `SIGTSTP`: prints the current position and battery level of all the robots.

Additionally:

- If there are n robots, the program will create n children, one for each robot.

- Each robot will be controlled by the program of the previous exercise (`robot.py`, if implemented in Python, `robot.c` if implemented in C).
- The program will use pipes to communicate with the robots.
- When finishing the program must print the exit status of its children.
- It should store the position of the robots and the information gathered so far about the room.
- It should NOT store the battery level of the robots.
- During the execution, if a movement leads to a collision between robots, it should warn the user and not allow the robots to move.
- It should also check if the initial position of the robot is not another robot's position, a wall or an obstacle. If so, the program must print an error message and finish.
- Only at the beginning of its execution the master is allowed to use an instance of the Sensor class/module. It can use its functions to know the number of treasures (to finish the program when all are found), the dimensions of the room, and to check if the position of the robots are valid. Once the setup is finished, the master cannot use the sensor functions again.

The master receives the following arguments (**in any order**):

- `-room <filename>`: the name of the file that contains the room information.
- `-room <filename>`: the name of the file that contains the robots information (their position).

Example:

```
$ python3 master.py -room room.txt -robots robots_2.txt
```

Execution examples

```
$ python3 master.py -room room_2.txt -robots robots_2.txt
Our information about the room so far:
? R ? ? ?
? ? ? ? ?
? R ? ? ?
Robot 1 PID: 7643 Position: (0, 1)
Robot 2 PID: 7644 Position: (2, 1)
Command: PID: 7644
PID: 7643
mv all left
```

```
Treasure found by robot 1!
Robot 2 status: OK
Our information about the room so far:
RT - ? ? ?
? ? ? ? ?
R - ? ? ?
Command: pos all
Robot 1 pid: 7643 status:Position: 0 0

Robot 2 pid: 7644 status:Position: 2 0

Command: mv 1 down
Robot 1 status: OK
Our information about the room so far:
T - ? ? ?
R ? ? ? ?
R - ? ? ?
Command: mv all right
Robot 1 cannot move right
Robot 2 status: OK
Our information about the room so far:
T - ? ? ?
R X ? ? ?
- R ? ? ?
Command: mv 2 right
Robot 2 status: OK
Our information about the room so far:
T - ? ? ?
R X ? ? ?
- - R ? ?
Command: bat
Invalid command
Command: bat all
Robot 1 pid: 7643 status:Battery: 18

Robot 2 pid: 7644 status:Battery: 13

Command: ^Z                id: 1 P: (1,0) Bat: 13
                id: 2 P: (2,2) Bat: 8
^Z                id: 1 P: (1,0) Bat: 7
                id: 2 P: (2,2) Bat: 2
mv 2 right
Robot 2 cannot move  right
Robot 2 status: KO
Our information about the room so far:
```

```
T - ? ? ?
R X ? ? ?
- - R X ?
Command: ^Z                id: 2 P: (2,2) Bat: 0
                id: 1 P: (1,0) Bat: 0
^\Replenishing batteries
^Z                id: 1 P: (1,0) Bat: 98
                id: 2 P: (2,2) Bat: 98
mv 2 right
Robot 2 cannot move right
Robot 2 status: KO
Our information about the room so far:
T - ? ? ?
R X ? ? ?
- - R X ?
Command: mv 2 up
Robot 2 status: OK
Our information about the room so far:
T - ? ? ?
R X R ? ?
- - - X ?
Command: mv 2 right
Robot 2 status: OK
Our information about the room so far:
T - ? ? ?
R X - R ?
- - - X ?
Command: mv 2 right
Robot 2 status: OK
Our information about the room so far:
T - ? ? ?
R X - - R
- - - X ?
Command: mv 2 down
Treasure found by robot 2!
Robot 2 status: OK
All treasures found!
Robot 7643 finished with status 0
Robot 7644 finished with status 0
Robot 1 pid: 7643 last message:
Position: 1 0
Battery: 61

Robot 2 pid: 7644 last message:
Position: 2 4
```

```
Battery: 41

Our information about the room so far:
T - ? ? ?
R X - - -
- - - X RT
$
```

```
$ python3 master.py -room room.txt -robots robots.txt
Our information about the room so far:
? ? ? ? ? ? ? ? ? ?
? ? ? ? R R ? ? ? ?
? ? ? RT ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
Robot 1 PID: 8190 Position: (2, 3)
Robot 2 PID: 8191 Position: (1, 4)
Robot 3 PID: 8192 Position: (1, 5)
Command: PID: 8190
PID: 8191
PID: 8192
mv 1 left
Robot 1 status: OK
Our information about the room so far:
? ? ? ? ? ? ? ? ? ?
? ? ? ? R R ? ? ? ?
? ? R T ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
Command: mv 2 up
Robot 2 status: OK
Our information about the room so far:
? ? ? ? R ? ? ? ? ?
? ? ? ? - R ? ? ? ?
? ? R T ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
Command: exit
Robot 8190 finished with status 0
Robot 8191 finished with status 0
Robot 8192 finished with status 0
Robot 1 pid: 8190 last message:
Position: 2 2
Battery: 71

Robot 2 pid: 8191 last message:
Position: 0 4
Battery: 71

Robot 3 pid: 8192 last message:
Position: 1 5
Battery: 76

Our information about the room so far:
? ? ? ? R ? ? ? ? ?
? ? ? ? - R ? ? ? ?
? ? R T ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
$
```

```
$ python3 master.py -room room_2.txt -robots robots.txt
Invalid initial position for robot at (2, 3)
$
```

Questions to answer in your report

Describe the behaviour of your program. You can use the following questions to guide you:

1. How did you process the commands from the user?
2. How did you create the children? How did you store their pids?
3. When receiving a `pos all` command, how did you communicate with the children?
4. When receiving an `mv all right` command, how did you communicate with the children? And how did you do when receiving an `mv 1 right` command?
5. How did you implement the `exit` command?
6. How did you handle the signals `SIGTSTP` and `SIGQUIT`? What did you do when you received them?
7. How did you handle the signal `SIGINT`? What did you do when you received it?