Middle East Technical University **Ceng336**
Computer Engineering Department **Spring 2018**

Take Home Assignment-4
"Open The Door!"

Revision 1.0

Due: Monday, June 4, 2018, 23:55:00
*Submission: **via ODTUCLASS (NO LATE SUBMISSION))***

The purpose of this assignment is to familiarize you with basic features of PICos18, a small but capable real-time operating system. You will learn about how to develop an embedded application with several different, concurrently executed tasks with support for preemption.

Any clarifications and revisions to the assignment will be posted to the ODTUCLASS.

Hand Out Instructions

- `simulator.zip` : cengRoboSim simulator source files (For Linux and Windows).
- `sample.hex` : sample .hex for testing the connection between simulator and PIC.
- `example.zip` : An example program for serial communication.
- `lcdSample.zip` : Sample LCD routines and an example program to show its usage.

Your Mission

Your mission is to write a PIC program to control a robot to pick a key, solve at least five questions randomly appearing, and open a gate in a simulated environment. The robot is capable of three basic motions:

- move forward
- turn right
- turn left

The robot can sense its environment using a sensor that provides the coordinates of the robot, its orientation, obstacle information near the robot, and the direction information of the key. You will be communicating with the simulator by serial communication, and sending/receiving different types of messages to control the motion of the robot, taking a key, solving at least five questions, finishing the game at a determined point. During the execution, you will be required to use the LCD display to show the location of the key and the number of collected questions.

The Game

There will be a key to pick and nine questions to solve. Initially, the area contains only the robot and some obstacles as explained below. The key appears in a random cell at the very beginning of the game in active mode, and it will stay up to the end of the game. Nine random numbers will be given to you at arbitrary times. You are expected to find the results using given hash function and those numbers as inputs.

You are expected to open the door with the following tasks: you should pick the key, you should solve at least five questions, and you should complete the game at the position of the door.

The Simulator

The robot simulator is written in the Python Language and requires Python 2.6 or newer, together with pygame and pyserial to be installed on your system. You will also need an FTDI universal serial connections to your development board. **You should install the compatible versions of pygame and pyserial with your Python version.** The simulator is provided to you in the `simulator.zip` file. In addition, since the simulator is written to work on the inek machines, you may have to make modifications to it for adjusting port settings, but do not modify anything about the functionality of this code since we will be testing your code on the version that is included with the homework packet. For detailed explanations, please refer to *init* part of the Miniterm class definition inside `cengRoboSim.py`.

For Linux, to install pygame and pyserial you can give the following command:

```
sudo apt-get install python-pygame python-serial
```

For Windows, please refer to following links for installation files:
http://www.pygame.org/download.shtml (.msi installer is available.)
https://pypi.python.org/pypi/pyserial (You should download and extract the tar.gz file and run its `setup.py` with `install` argument.)

After installing the tools or on inek machines you can run the simulator by giving the following command:

```
python cengRoboSim.py
```

Figure 1. A screenshot from simulator.

A screenshot of the simulator is given in Figure 1. The area covers an 800x200 pixel region and is divided into 50x50 pixel, 64 cells. Thus, the region is composed of 16x4 cells. **There are 16 cells in X direction (→) and 4 cells in Y direction (↓).** Coordinates of top left and bottom right corners of the region are **(0, 0)** and **(15, 3)**, respectively. The robot has a 20-pixel diameter and its initial position is always (25, 25). In other words, starting position is the center of top most left cell.

The map area contains fixed 50x50 pixel obstacles (shown as black cells in Figure 1). Figure 1 shows an area constructed with 6 blocks. Note that the positions and the number of obstacles are fixed for an execution, however, obstacle configuration will be generated randomly at grading. Therefore, do not make any assumptions about obstacle positions. You should sense your neighboring cells to navigate (see below SENSE CMD). Other construction rules are:

- Blocks will never be placed in the top leftmost, and rightmost cell.
- Key does not appear on blocked cells.
- There will always be at least one path from the current position to the key and the door.
- The position of the door will not change, in other words, you should complete the game when the robot is inside the coordination of the door which is the right corner of the game area.

The initial orientation of the robot will be fixed and its orientation will be facing the bottom of the area. This corresponds to $\theta = 180°$, with θ increasing in counter-clockwise direction.

Operation Modes of the Simulator

The simulator has three operation modes:

- IDLE: In this mode, the simulator waits without sending commands or responding to the PIC until the user presses the 'g' button on the computer's keyboard. After the user presses 'g', the simulator will send a *GO command* over serial port to the PIC and switch to the ACTIVE mode.
- ACTIVE: In this mode, the simulator expects serial commands from the PIC for controlling the motion of the robot (*motion commands*). In addition, there will be a command to pick a key.

  The simulator stays in this mode until it receives an *END command* or reaches the end of a predefined timeout.

  In this mode, the simulator also shows some status information such as the current coordinates and the orientation of the robot, key flag, remaining time and maximum, minimum, average times between every two motion commands. All time differences between consecutive motion commands are also printed to the command line.

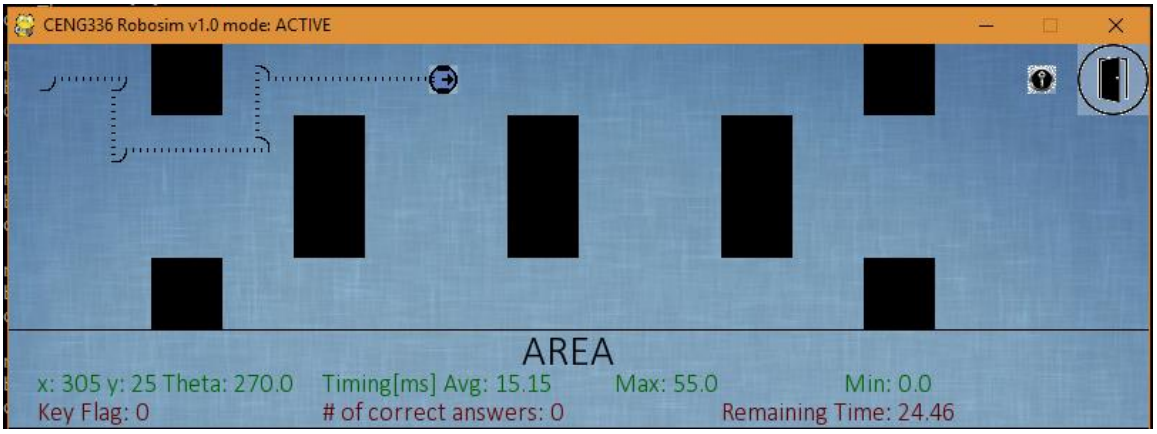

Figure 2. Simulator in ACTIVE mode (a different map from Figure 1).

- END: When the simulator enters this mode, it prints the result of the last ACTIVE mode operation to the command line. The result includes:
  - The key flag which shows whether the key is picked or not and the number of answered questions.
  - Average, maximum and minimum durations between every two motion commands.

  If 'r' is pressed in END mode, the simulator is reset to IDLE mode by clearing all of data from previous ACTIVE mode.

Commands

In general, each command starts and ends with the `'$'` and `':'` characters, respectively. We will refer to `'$'` and `':'` as the delimiter and terminator, respectively.

GO Command:

When the user presses 'g' in the IDLE mode, the simulator sends a *GO command* over the serial port to the PIC and switches to the ACTIVE mode. This command consists of the following 4 bytes (ASCII characters): **`$GO:`**

END Command:

This command should be sent from the PIC to the simulator to terminate the ACTIVE mode. You will use this command after all tasks are completed. After sending this command, the results will be displayed and you will not be able to send any other commands to the simulator (These results will be used in our grading). This command consists of the following 5 bytes (ASCII characters): **`$END:`**

Motion Commands:

In order to control the motion of the robot, you are expected to send commands from the PIC to the simulator while it is in the ACTIVE mode. These commands cause the robot to take forward movement or rotation steps or to sense. After receiving a command, the simulator performs the corresponding motion and possibly sends a *sensor response* back to the PIC. There are four types of motion commands:

- MOVE FORWARD: This command moves the robot forward a certain number of pixels depending on its forward speed. The default forward speed is 5 pixels/command. This command consists of the following 3 bytes (ASCII characters): **`$F:`**

- TURN RIGHT: This command rotates the robot a certain number of degrees in the clockwise direction depending on the configured speed (fixed). The default rotation speed is 9 degrees/command. This command consists of the following 3 bytes (ASCII characters): **`$R:`**

- TURN LEFT: This command rotates the robot a certain number of degrees in the counterclockwise direction depending on the configured speed. The default rotation speed is 9 degrees/command. This command consists of the following 3 bytes (ASCII characters): **`$L:`**

- SENSE: This command stops the robot, causing it to perform a sensor reading. After completing the sensor reading, the simulator sends a *sensor response* back to the PIC. This command consists of the following 3 bytes (ASCII characters): **`$S:`**

Sensor Response:

This transaction is a response to the SENSE command and consists of the following fields: **`$Dx,y,r,abcdp:`**

Each byte should be treated as an ASCII character. ***x*** and ***y*** are the cell coordinates of the robot. ***r*** represents the orientation of the robot. The ranges for ***x*** and ***y*** are [0,15] and [0,3], respectively. In other words they are not pixel coordinates. ***x*** and ***y*** are calculated by simulator using the pixel coordinates of the robot center. The value of the ***r*** is in [0, 39] range. This value will be multiplied with rotational speed, 9 for this homework, to obtain the orientation of the robot on the simulator side. For example, if the computer sends 30, the actual orientation of the robot is calculated as 270 degrees, as depicted in Figure 3. Remember that the degrees are increased in a counter-clockwise manner.

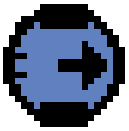

Figure 3. Appearance of the robot when its orientation is 270 degrees.

Note that, the robot can only see its 4 neighboring cells, not the whole map. In order to safely move around you should use a,b,c,d data. ***abcd*** which can take the values of '***0', '1','2'(again ASCII)*** are the cell information of the robot's neighboring cells. If it is 0 then it means that there is no obstacle in that cell, if it is 1 then it means there exist an obstacle in that cell, and if it is 2, then it means that the key is located in that cell. As it can be seen in Figure 4 ***abcd*** are the sensors of the robot which are responsible for providing the cell content respectively.

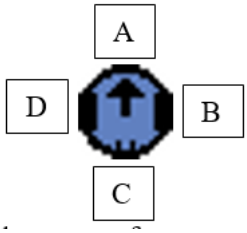

Figure 4. Order of the neighboring cell data.

The robot cannot know the exact location of the key unless its cell is one of the neighboring cells. Therefore you should navigate in the given direction until the robot's vision identifies the key. In this case, ***p*** represents the direction of the key with respect to the robot. Values of the ***p*** can be any one of the following: E (East), W (West), S (South), N (North), SE (Southeast), NE (Northeast), SW (Southwest), NW (Northwest), K (Key). K is sent if the robot and the key are on the same pixel.

Alert Command:

When the simulator generates a question, it sends the information about this question to the PIC. The command sends the following: **$A*ID*,*N*:**

Again, all the bytes are ASCII characters. ***ID*** represents the **order of the question**, and ***N*** represents a **random number** to be solved, which are string values. Any other value may results in an error. When this command is received by the PIC, you should increase the "number of alerts (A:)" value depicting with the given coordinates on the LCD module. Ex: **$A3,53413:** (all ASCII characters).

Pick Command:

This command allows you to pick the desired key. The command sends the simulator the following 3 bytes: **$*P*:**

Here **$**, **P**, **:** are ASCII characters. When the PIC sends this command the robot picks the key if that is in the current cell of the robot.

Compute Command:

When the PIC generates an answer, it immediately sends the information about this answer to the simulator. The command is in the following format: **$C*ID*,*N*:**

Here **$**, **C**: are ASCII characters, however ***ID*** represents the **order of the question**, and ***N*** represents the **answer** for the corresponding question, which are string values. Any other value may results in an error. When this command is received by simulator, you can see the change on the "number of correct answers" value depicting on the simulation environment. Ex: **$C7,435A132C34:** (all ASCII characters).

Detailed Specifications

- **Firstly, and most importantly, your program should make use of Real-Time Operating System principles, creating multiple tasks, setting their priorities and synchronizing them using RTOS primitives such as semaphores. You will be evaluated on your software design as much as the correctness of your implementation.**
- **You MUST properly comment your code, including a descriptive and informative comment at the beginning of your code explaining your design, choice of tasks and their functionality.**
- At the beginning, the LCD Module should show the following screen and your program should wait for a GO command to start the game. You should adjust the LCD module **for no blinking and no cursor** options.

X						O	P	E	N						X
X				T	H	E		D	O	O	R				X

- After receiving the GO command, you should start the game by using the commands of the simulator explained above. During the game, you should send **motion, alert**, **compute** and **pick** commands at **exactly** every **50 ms**, namely at 20 Hz.
- You are allowed to send SENSE command as much as you want. You may need such a case while you are deciding which movement you have to make next and while another task is

running. Note that if you want to stop for more than 50ms, you have to keep sending SENSE commands to the simulator.

- It is recommended that to move the robot with "L shaped" moves. In other words, you are recommended to rotate the robot at multiples of 90 degrees when you need to turn direction. Likewise you are recommended to move the robot between the center of cells when you need to go forward. Otherwise, you can have difficulties to locate the robot at center of the cells and this may cause inaccuracies in sensor responses. You can always check your coordinates using the response from SENSE command.
- You should **pick the key** and **solve at least 5 of 9 questions, and put the robot at the position of the door**. Otherwise, you will be considered as unsuccessful.
- When an **alert command** is received, you should update the LCD module accordingly. When you pick the key, you are expected to simultaneously update the LCD module by providing the cell coordinates of the key. You can make use of the supplied LCD routines. An example area is depicted below:

	K	:	0	3		1	5			A	:	3		
	O	P	E	N		T	H	E		D	O	O	R	!

You are expected to use given hash function for solving compute questions. Implementation is in C language and you will make an RTOS task using this code. **In other words you will use the given compute_hash function in pic_hash_compute.c file as an RTOS task**.

- Your program should be written for **40 MHz** oscillator by using PICos18 operating system.
- **USART** settings should be **115200 bps**, **8N1**, **no parity**.
- **You are expected to write a generic code in order to handle different environments (obstacle positions). You will not know which obstacle map we use for evaluation.**

Resources

- Recitation Documents
- Sample program files provided with homework.
- PIC 18F8722 Datasheet ▪ PICos18 Documents
- PIC Development Tool User Manual
- PIC Development Tool Programming Manual
- Course web page and newsgroup
- C18 Manuals

Hand In Instructions

- You should submit your code as a single file named as `the4_##.zip` through ODTUCLASS (## represents your group number). This file should include all of your project files. Do not forget replace ## with your group number.
- **Only one of the group members should submit the code. Please pay attention to this**, since if both members make submission, speed of grading process will be negatively affected.

Total of the take home exam is 100 points. For grading we will compile and load your program to the development board and run it for the following checks listed below:

1. A brief and detailed explanation of RTOS logic that you implement
   - Duty of each task,
   - The events activate each task,
   - When does their state change and how? (e.g. from suspended to running state),
   - Their priorities (why did you choose such a prioritization)
   - To sum up, running mechanism of your work (by explaining the preemptions)
2. Accuracy of command timings
   - Are you able to send a command at every 50 ms, how much are you good at it?
3. Number of crashes
   - Try to minimize it
4. Proper usage of the LCD module
   - Is it real-time?
     (Are you capable of displaying the motion simultaneously during the progress?)

Note that you should structure your program modularly with different tasks, appropriate priorities and proper use of synchronization primitives. It is unacceptable to implement this system as a single task in the form of a cyclic executive with interrupts. **Include brief but descriptive comments in your code, including a larger comment in the beginning of your file describing the structure of your program** with task descriptions and their relations to one another**. Your grade will greatly depend on** the quality of your design, not just its functional correctness.

Hints

- **CRUCIAL:** Since PICos18 is clearing PIE1, PIE2, RCON, IPR1 and IPR2 registers inside `Kernel/kernel.asm` file (instructions between lines 265-269) which are executed after your `init` function inside `main.c`, your changes on these registers are becoming ineffective. Therefore you have to configure these registers once inside a related task. If you are experiencing an unexpected stop while receiving characters from serial port (due to overrun error making OERR bit 1), this is probably caused from above issue. In that case you have to configure the bits related with receive interrupt (for example a setting like PIE1bits.RCIE = 1;) inside a task. `serialCommunication.zip` inside `recitation7` files is not handling transmit/receive interrupt enable bits properly, therefore you have to correct them using above explanations.
- In order to check the serial communication between simulator and PIC you can use `sample.hex file`. **This is not an example solution and its command timings are not accurate**. Its purpose is only to check the serial communication.

Cheating

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations.

Cheating Policy: Students/Groups may discuss the concepts among themselves or with the instructor or the assistants. However, when it comes to doing the actual work, it must be done by the student/group alone. As soon as you start to write your solution or type it, you should work alone. In other words, if you are copying text directly from someone else - whether copying files or typing from someone else's notes or typing while they dictate - then you are cheating (committing plagiarism, to be more exact). This is true regardless of whether the source is a classmate, a former student, a website, a program listing found in the trash, or whatever. Furthermore, plagiarism even on a small part of the program is cheating. Also, starting out with code that you did not write, and modifying it to look like your own is cheating. Aiding someone else's cheating also constitutes cheating. Leaving your program in plain sight or leaving a computer without logging out, thereby leaving your programs open to copying, may constitute cheating depending upon the circumstances. Consequently, you should always take care to prevent others from copying your programs, as it certainly leaves you open to accusations of cheating. We have automated tools to determine cheating. Both parties involved in cheating will be subject to disciplinary action. [Adapted from http://www.seas.upenn.edu/~cis330/main.html]