Middle East Technical University
Department of Computer Engineering

CENG 336

Spring 2021

Take Home Exam 4

Regulations

Due: Tuesday, July 5th 2022, 23:55.

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 develop several different, concurrently executed tasks with support for preemption.

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

Hand Out Instructions

- `simulator.zip` : cengTamagotchiSim simulator source files.
- `pic_hash_compute.c` : the hash function that you will need for your implementation
- `switchConfiguration.jpeg` : A snapshot of the board with the desired switch configurations of this assignment

Tamagotchi

Tamagotchi is a handheld toy that was created in Japan. The toy hosts a digital pet that has certain needs fulfilled by the player through its interface.

In this assignment, your mission is to write a PIC program that will look after a digital pet similar to Tamagotchi for 90 seconds. Instead of pressing physical buttons, you will send commands through a serial connection. You are capable of three basic actions to handle your pet:

- feed
- give water
- play

You will be communicating with your pet through serial connection, and sending/receiving different types of messages to control the state of the pet and take the necessary actions to keep it alive. You will also display the state on the LCD Module. The actions restore the corresponding need in return for money.

The game consists of 2 difficulty levels; easy and hard. In the easy level, there will be enough money for you to last 90 seconds. In the hard level, you need to earn money by solving hashes. Your goal should be to keep the basic needs of your pet at an acceptable level for 90 seconds.

Simulator

The Tamagotchi simulator is written in Python and requires Python 3, `pygame` and `pyserial` to be installed on your system. The simulator is provided to you in the simulator.zip file.

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

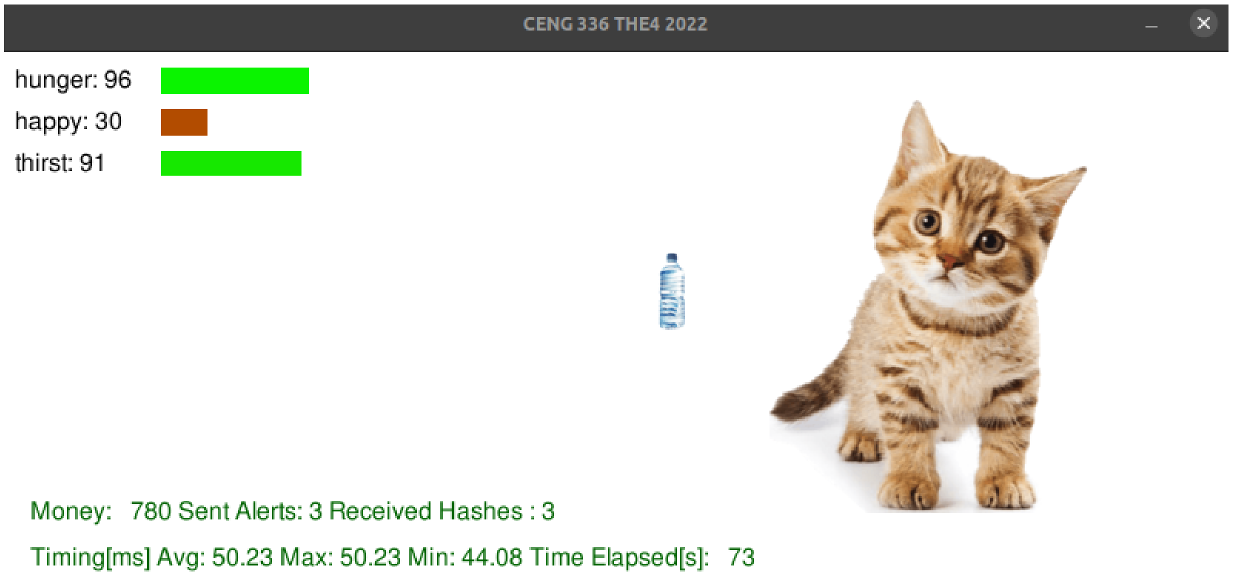
```
sudo apt-get install python3-pip
pip install pygame pyserial
```

After installing the tools or on Inek machines you can run the simulator by typing the following command:

```
python3 cengTamagotchiSim.py
```

Before running the simulator, make sure that you have checked your device file name for the serial port is the same with the `DEFAULT_PORT`, which is `/dev/ttyUSB0` currently, in `cengTamagotchiSim.py`.

If it says 'permission denied', then allow user access to device file with chmod command.



```
hunger: 96
happy: 30
thirst: 91
Money:   780 Sent Alerts: 3 Received Hashes : 3
Timing[ms] Avg: 50.23 Max: 50.23 Min: 44.08 Time Elapsed[s]:   73
```

Figure 1: A screenshot of the simulator.

A screenshot of the simulator is given in Figure 1. The simulator displays some text about the current status. On the top left corner, you will see Hunger meter, Happy meter and Thirst meter. On the bottom left, you will see your current amount of money. On the bottom right, the timing statistics of the commands you send and the time elapsed. The values you read from the simulator should match the displayed values on the LCD.

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 selects one of the difficulty levels, easy or hard, by pressing 'e' or 'h' buttons on the computer's keyboard. After the user selects a difficulty level, 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 digital pet (*check command*) and updating the pet's status (*action commands*). The simulator responds by *sensor responses* or it sends an *alert command.* There are no responses for action commands; the simulator directly updates its current state after receiving such commands.

  The simulator stays in this mode until it has been 90 seconds or one of the meters becomes 0.

  In this mode, the simulator also shows some status information such as the time elapsed, and maximum, minimum and average times between every two commands.
- END: When the simulator enters this mode, it prints the result of the last ACTIVE mode operation to the command line. The result includes:
  - Total operation time.
  - The final values of the hunger, happy, thirst meters and money.
  - Average, maximum and minimum durations between every two commands.

When user presses the 'ESC' button on the computer's keyboard, the simulation program terminates.

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 selects the difficulty level 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 6 bytes (ASCII characters): **{GO<u>XX</u>}**.

  Here **{**, **G**, **O**, **}** are ASCII characters, however **<u>XX</u>** represents the initial money. The first **<u>X</u>** is the most significant byte. You need to calculate your initial money upon starting the game.
- ***END Command :*** This command is sent from the simulator to the PIC to end the ACTIVE mode. After receiving this command from the PIC, you should not send any other command. This command consists of the following 5 bytes (ASCII characters): **{END}**
- ***Action Commands :*** In order to keep your digital pet alive, you are expected to send commands from the PIC to the simulator while it is in the ACTIVE mode. These commands either feed, give water or play games. These actions restore some amount of basic needs and cost money. You can see how much the actions restore the meters and how much you need to spend in Table 1. There are three types of action commands:
  - **Feed:** This command feeds your pet. It affects the Hunger meter. This command consists of the following 3 bytes (ASCII characters): **{F}**

  - **Give Water:** This command gives water to your pet. It affects the Thirst meter. This command consists of the following 3 bytes (ASCII characters): **{W}**
  - **Play:** This command gives toys to your pet to play with. It affects the Happy meter. This command consists of the following 3 bytes (ASCII characters): **{P}**

Action	Restore	Money	Effected Meter
Feed	60	80	Hunger
Give Water	30	30	Thirst
Play	80	150	Happy

Table 1: Actions, how much they restore the meters and their costs

- ***Check Command :*** This command allows you to send a request to check the state of the pet and includes the following 3 bytes: **{C}**

- ***Sensor Response :*** This transaction is a response to the CHECK command and consists of the following 6 bytes: **{S<u>XYZ</u>}**

  Here **{**, **S**, **}** are ASCII characters, however **<u>X</u>**, **<u>Y</u>** and **<u>Z</u>** represent values of 3 sensors. X is for Hunger meter, Y is for Happy meter and Z is for Thirst meter. A sensor value can be in the range of [0,100].

- ***Alert Command :*** In the hard level, the simulator generates a random ascii string of length 8 and it sends it to the PIC. This command consists of the following 11 bytes: **{A<u>STR</u>}**

  Again, all the bytes are ASCII characters. **<u>STR</u>** represents a random string of length 8 (without the null terminator) to produce a hash. This command can be received at any random time in the ACTIVE mode.

- ***Hash Command :*** When the PIC generates a hash using this string, it immediately sends it to the simulator. The 19 byte command is in the following format: **{H<u>CODE</u>}**

  Here **{**, **H**, **}** are ASCII characters, however **<u>CODE</u>** represents the hash for the corresponding string, which is an ascii string itself of length 16 (without the null terminator). Unless the calculated hash is incorrect, when this command is received by simulator, you can see the increase on the current money value depicting on the simulation environment.

- ***Payment Response :*** This transaction is a response to the HASH command. When the PIC consumes the alert string correctly and sends the hash to the simulator, the simulator sends the earned money. This response consists of the following 5 bytes: **{M<u>XX</u>}**

  Again, all the bytes are ASCII characters. **<u>XX</u>** represents the amount of earned money. The first **<u>X</u>** is the most significant byte. After receiving this response, you need to calculate the amount and add it to your current money.

  If the 16 byte string within the HASH command is incorrect, **<u>XX</u>** will have the value 0.

LCD Module and Button Action

The LCD module will also show the information about the status of the pet. At the beginning, the LCD Module should show the following screen and your program should wait for a GO command to start the game.

				C	E	N	G		3	3	6				
			T	A	M	A	G	O	T	C	H	I			

Table 2: Appearance of LCD at the beginning

After receiving the GO command, you should start the game by using the commands of the simulator explained above. During the game, you will use LCD to show your current money and the level of the selected meter. The selection of the meter to be displayed will be done by releasing the **RB1** button. Initially, you will display the Hunger meter (see Table 3). The user will be able to switch between the meters by releasing the **RB1** button. The order of the meters to be displayed is Hunger meter, Happy meter and Thirst meter. If RB1 is released 3 times, you will cycle back to the first one and display the Hunger meter.

M	O	N	E	Y	:							1	0	0	0
H	U	N	G	E	R	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮

Table 3: LCD displaying the Hunger meter

M	O	N	E	Y	:								9	6	0
H	A	P	P	Y	:	▮	▮	▮	▮	▮	▮	▮	▮		

Table 4: LCD displaying the Happy meter

M	O	N	E	Y	:								7	5	0
T	H	I	R	S	T	▮	▮	▮	▮	▮					

Table 5: LCD displaying the Thirst meter

You are expected to check the meters regularly. The values to return from the simulator can be in the range of [0,100]. In the LCD module, you have 10 cells to display how full the meters are. While displaying the level of a meter on the LCD module, you should write the ▮ character (0xFF) for full cells. For the empty cells, you should write space character. You should fill the cells from left to right order. The number of full cells will be calculated by performing an integer division. For an example, if the returned value for Thirst meter is 56, you should fill 5 cells (see Table 5).

During program operation, if the user releases the RB1 button, the LCD will refresh to show the level of the next meter along with the money. **You must use interrupts for this task. We recommend you to use low priority interrupts for compatibility with PICos18.**

You can make use of the supplied LCD routines. You are not expected to refresh the LCD for every update after the check command or upon RB1 release **immediately** to make synchronization issues more easier for you. (You may use double buffers with periodic updates on the LCD). However, the updates on the LCD should be visible within **1 second.**

Detailed Specifications

- **First, and most importantly, your program should make use of Real-Time Operating System principles, creating multiple tasks, setting their priorities and synchronizing them. 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.**
- It is guaranteed that none of the bytes between the delimiters correspond to ascii value of '{' or
- The initial value of the meters are 100 at the beginning of the ACTIVE mode.
- You should adjust the LCD module for **no blinking and no cursor options.**
- You should send your commands at every **50ms**, with an acceptable level of accuracy ± **20 ms shifts** (The time difference between two consecutive commands should not be greater than 70ms, and should not be less than 30ms, while having an average of 50ms).
- You are allowed to send CHECK command as much as you want. You may need such a case while you are deciding which action you want to take next and while another task is running. Note that if you want to meet the specifications for the command frequency, you must continue to send CHECK commands to the simulator.
- For your implementation to be successful, your program should be able to keep the digital pet alive for 90 seconds. The digital pet dies if a meter becomes 0.
- In the easy level, the initial money is just enough for survive the game 90 seconds therefore the timing of your action commands are important. Constantly buying items whenever a need is not fully replenished is a bad tactic that would leave you penniless.
- In the hard level, you must use the given hash function for generating hashes within a seperate RTOS task. **In other words you will use the given compute_hash function in pic_hash_compute.c file as an RTOS task.**
- You will receive exactly 3 alert commands. Each one will be sent at a random time. You are guaranteed to have exactly one hash to generate at any time (i.e. you will not receive the n^{th} alert command before sending the correct hash for the $(n-1)^{th}$ alert).
- Your program should be written for 40 MHz oscillator by using PICos18 operating system.
- USART settings should be 115200 bps, 8N1, no parity.

Resources

- Sample program files provided with homework.
- PIC18F8722 Datasheet
- PICos18 Documents
- PIC Development Tool User and Programming Manual

- Recitation Documents
- ODTUClass Discussions

Hand In Instructions

- You should submit your code as a single file named as the4.zip through ODTUClass. This file should include all of your source and header files.
- By using a text file, you should write ID, name and surname of **all group members and group number**.
- **Only one of the group members should submit the code**. Please pay attention to this, since if more than one member make submission, speed of grading process will be negatively affected.

Grading

Total of the homework is 100 points. For grading we will compile and load your program to the development board. Your program will be considered for grading even if it is incomplete. We advise you to implement your program following the checklist given below.

- 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)
- Accuracy of command timings
  - Are you able to send a command at every 50 ms, how much are you good at it?
- Proper usage of the LCD module
  - Is it real-time?

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.

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.

- 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 commands are not accurate**. Its purpose is only to check the serial communication.
- Microchip's XC series compilers are not compatible with PICos18. For inek machines, MCC18 compiler is already added, so there is nothing you have to do. For your own Linux or Windows machines, you can follow the instructions in 'PicosSettingsInMplabxManual.pdf'.

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]