CompE571 Embedded Operating Systems Term Project: Final Report

***Title: Bot Sensing Simulator with Raspbian OS Task Optimization***

Names: Daniel Foster (817895293) and Vahe Ohanian (818027724)

Contact/Email: [dfoster1130@gmail.com](mailto:dfoster1130@gmail.com) and [vaheohanian@gmail.com](mailto:vaheohanian@gmail.com)

\* ***Source code included in attached text document. \****

Project Description:

We will implement a sensing system for a bot/vehicle designed for scientific exploration of unknown environments. The bot, implemented using a Raspberry Pi 3 Model B microcontroller and tri-wheel bot chassis kit, will move in fixed patterns that will allow our sensing module to pick up measurements and move according to these measurements. For example, according to outside environmental conditions such as temperature or light intensity, the bot will be able to schedule simultaneous tasks/movements from sensory inputs and showcase the management of different priority tasks. Our sensing module will be implemented using the Raspberry Pi 3 microcontroller and a Sense Hat add-on board capable of measuring temperature, humidity, pressure, and movement; all of these sensors would be realistic components of a ground, space, or air vehicle. For the OS component of this project, we plan to modify the Raspbian OS for optimization of our particular tasks. By modifying the kernel for our Raspberry Pi, we hope to increase execution time and provide real-time outputs that are accurate for different priority tasks.

Related Work:

The related work was what first got us particularly interested in this topic. NASA’s Mars Rover, being a scientific robot used for exploration, must be proficient at managing different priority tasks. Obviously, many people have created robots using Raspberry Pi and/or Arduino devices, but we feel ours is different in that we are implementing the Sense Hat into our project. Additionally, our operating system modification, which will be done using kernel modules for Raspbian, would not be a typical aspect of a Raspberry Pi/Arduino start-up project.

Our robot has been designed using aspects from the Mars Rover that we thought important. Certainly this is just a snapshot of the functionality of the Rover, but it can sense temperature and pressure, and has the ability for external I/O in “real time”. Our Raspberry Pi/Sense Hat combination serves as the focal point of our project and differentiates it from other similar, typical projects.

Reasoning/Applicability:

Interested by the concept of the Mars rover that was deadlocked by priority inversion, we thought a bot that could manage low, medium, and high priority tasks with priority inheritance would be unique and would be very applicable to real-world scientific and research scenarios. In a hard real-time system such as this, multitasking must be done without error or risk large financial or information loss. We believe that extending the default functionality of the provided Raspbian OS is not only a difficult enough task to consider this project unique and satisfactory for this class, but we think that this skill will likely help us in the future as the Raspberry Pi is growing in its popularity and useful applications.

Difficult Components:

As we have never worked with a Raspberry Pi or Python before, we hope to be able to pick up Python quickly and implement the Sense Hat add-on board without error. Furthermore, as we have never coded or modified instructions in the kernel of an OS, and we are still not to the point where we feel comfortable doing so, we will need to do extensive research on Raspbian OS, as well as researching task scheduling algorithms and other possibilities for kernel modification. Lastly, our choice of project, which is incredibly applicable in the real world as stated above, has limitless options for further development, some of which we may explore or discuss implementing over the course of project work.

Hardware and Software Implementation:

For this project we plan to implement the following hardware and software –

1. Hardware

* Raspberry Pi 3 Model B
* Raspberry Pi Sense Hat add-on board
* Tri-wheel bot chassis with battery pack and dual motors
* Battery pack/on-chassis power
* L298N motor drive controller board
* Miscellaneous wires, connectors, adaptors

1. Software

* Python (Sensing modules and bot control)
* Raspbian OS (task management/scheduling)

Accomplishments:

The accomplishments met by final project deadline in terms of –

1. Hardware

* Raspberry Pi initial set-up
* Raspberry Pi Sense Hat working with temperature, humidity, pressure, joystick movement, and display
* Bot interfacing with motor driver, power supply, motors, and GPIO
* Working bot system

1. Software

* Raspbian Jessie
* Python for bot control and Sense Hat
* Execution of simultaneous processes (multithreading)
* Installed kernel
* Final project Python source code implementing full priority-based task control using:
* Forking for multithreading of processes
* Creating priority-based scheduler
* Logical task priorities

1. Experimental Design

* Testing of bot movement – completed before progress report
* Sense Hat readings – completed before progress report
* Simultaneous bot movement and sensing – our testing for our completed system includes the following:
* Ensuring smart order of operations for separate tasks
* Sense Hat display and bot movement should be simultaneous
* Bot movements based on priority of ready tasks
* Threshold values for sensing give correct movement

Metrics:

We will measure the success of our project based on:

* Reliability of sensing inputs and outputs
* Optimization of Raspbian OS
* Response-time/execution time
* Task management and scheduling
* Energy efficiency/memory management (*if time permits*) (*not included in final project*)

Experiments/Results:

All deadlocks should be prevented, and as different test environments are ran using the different sensing measurements, scheduling between tasks should be done with execution time and reliability in mind. For example, if the bot is to interact with a light source (ex. flashlight), it should stop the bot or move it in a different direction. If it encounters this task, along with another lower priority task (ex. high heat reading requiring movement in the opposite direction), it should execute the higher priority task and stall the lower priority task. Results include all sensors giving measurements in real time and with a high percentage of correctness, as well as good response time overall for the system; both of these results would be necessities in a real-world, scientific application.

We have tested and researched our project’s functionality extensively, beginning with the submittal of our progress report. Our initial desire was to install the Raspbian OS kernel to the Raspberry Pi before either patching or configuring the kernel to reach the goal of creating priority-based scheduling. After installing the kernel using the guide on the Raspberry Pi website, our first inkling was to patch the kernel with a recent version of RTLinux, which we thought would help us to implement, at least, a soft real-time design for our tasks. After research, we discovered that the functionality of a real-time OS for our design was unnecessary, if not overbearing. In addition, a real-time OS simply wouldn’t help in scheduling a relatively small amount of user defined tasks within a Python program.

Our second idea was to configure the kernel to better optimize our task synchronization. Again, after researching the patching avenue, we already had an idea of the small impact this could have on our user program and severely questioned the configuring option altogether. Nevertheless, we took a look at the Raspbian OS kernel configuration options, with nothing really pertinent to our system other than the “Preemptive Kernel” option. A description of this configuration is listed below, showing the difference between the basic Raspbian voluntary kernel preemption and the configurable preemptible kernel option, respectively. The reason we chose not to turn this on was based on the fact that we could define task priorities themselves in our Python code, reaching our goal of creating a system that would execute based on priority levels of different tasks, while making sure they were scheduled with execution time in mind.

*“CONFIG\_PREEMPT:*

*This option reduces the latency of the kernel by making all kernel code (that is not executing in a critical section) preemptible. This allows reaction to interactive events by permitting a low priority process to be preempted involuntarily even if it is in kernel mode executing a system call and would otherwise not be about to reach a natural preemption point. This allows applications to run more ‘smoothly’ even when the system is under load, at the cost of slightly lower throughput and a slight runtime overhead to kernel code.*

*Select this if you are building a kernel for a desktop or embedded system with latency requirements in the milliseconds range.”*

We chose not to include this, as it does nothing in the realm of actually scheduling our tasks, and our system surely does not depend on execution time changes in the milliseconds range.

Below is a table outlining our results after testing. Two criteria were primarily used to define priority.

1. Logical priority in a real system. For example, would temperature or humidity matter more on Mars? Are movement tasks or display/information tasks more important?
2. Bot functionality. Which priorities allow the bot to function quickest and without issues?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Task Names | | | | | | | | |
| Priority Settings | Forward Movement | Backward Movement | Right Movement | Left Movement | Bot Stop | Get Temp | Get Hum | Display Temp | Display Hum |
| Priority Test 1 | 3 | 4 | 7 | 5 | 6 | 2 | 3 | 1 | 1 |
| Priority Test 2 | 5 | 3 | 6 | 7 | 4 | 2 | 3 | 1 | 1 |
| Priority Test 3 | 4 | 5 | 3 | 6 | 7 | 3 | 2 | 1 | 1 |
| Priority Test 4 | 6 | 7 | 4 | 3 | 5 | 3 | 2 | 1 | 1 |
| Priority Test 5 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 1 | 1 |

Table Explanation:

The table is split into two sections which indicate individual schedulers. Everything to the left of the solid line involves the movement of the bot and is controlled in our first scheduler, while everything to the right of the line involves our sensors and is controlled by our second scheduler. The schedulers run in parallel of each other, but the tasks within each scheduler run in the order of their priority. Like in our code, the lower the number the higher the priority. In our first priority test, the bot should always move in the following sequence as long as the sensing threshold is not reached: Forward, Backward, Left, Stop, Right, and then repeat. For the sensing, we have the bot retrieve data and react if necessary in the order of their priority. Again, for the first test, first the bot will sense the temperature and then humidity. If the threshold is reached in either sensing case, the bot will schedule an additional function “Display Temp” or “Display Hum” with the highest priority depending on what sense passed the threshold. In each one of these functions, they will scheduler a task, either Right Movement (for Display Temp) or a Left Movement (Display Hum) with the highest priority to the movement scheduler. For example, if the temperature threshold was met, at that moment the function will add a Right Movement into the movement scheduler to execute next since it will have a higher priority than the other tasks. Since Display Temp and Display Hum will never be executed at the same time, they are both always given the highest priority, so they always execute before the next sensor reading. All the other priority tests follow this logic, only the order of execution is changed based on the given priority.

Source Code Explanation:

Our code implements two schedulers where one controls all of the bot’s movements, and the other controls sensing and displaying tasks on the Sense Hat. We defined each movement direction as individual tasks with its own function call. Each function is named based on the direction the bot will move in and they simply set the GPIO I/O pins to the correct values to move the bot in that direction. We control how long the bot should be moving in that direction by using the *time.sleep(value)* function.

The functions *getTemp* and *getHum* read and display the sensing values on the Sense Hat add-on. For both functions, if the sensor reading passes a certain threshold, it will schedule a task for the sensing scheduler to execute immediately where is displays a “T” or “H” depending on which threshold was passed. On top of that, the function will scheduler either a Right Movement (in the case of a temperature threshold) or a Left Movement (in the case of a humidity threshold) to be executed immediately to the movement scheduler.

To implement multi-tasking, we used the *pid = os.fork()* function call to create now two running threads running in parallel. These threads will first enter a while loop that runs forever, until a keyboard interrupt is executed (CTRL+C press). To control which thread runs which scheduler, we used conditional statements where based on the pid value, that thread would only run that section of the code. One thread continuously adds the movements of the bot that implements its “pre-defined” route to one scheduler while the other thread continuously adds sensing tasks to the other scheduler. To add tasks to either scheduler, we used the function call of *scheduler’s name.enterabs(time, priority, event, arguments)*. This adds the event into the scheduler’s queue and is ready to execute when the *scheduler’s name.run()* function is called. For the time, we run the function *time.time()* to get the time of the start of the program and assign every task to this time so the execution of the scheduler is based on priority only. Also, when we add a task to a scheduler’s queue, it returns the event to be executed which we store into a dummy variable to keep track of all the tasks and store it in a list. There is a separate list for each scheduler, which we only used when the program is interrupted to delete all the tasks still remaining in the queue as the Raspberry Pi would still run the motors after the program is terminated if we did not clear the queue. To catch the interrupt we used the *signal.signal(signal.SIGINT, signal\_handler)* function and defined the *signal\_handler* function to execute a foreach loop for both lists and cancel every task remaining in each scheduler queue with *scheduler’s name*.cancel(event).

Deliverables:

1. Progress Report

* Bot assembly and movement (no interfacing with sensors at this point)
* Raspberry Pi setup and Sense Hat implementation
* Single sense demo and operation
* Raspbian OS introductory interfacing

1. Project Demo/Final Report

* Working bot with sensory inputs/outputs
* Live demonstration/video of sample test sensory environment (using at least 2 sensing mechanisms)
* Raspbian OS modification for enhanced metrics (ex. task scheduling)
* Pertinent data that supports results
* Problems faced/modification to project design

Possible Extensions to Existing Project:

Extensions could be made to our final project two-fold; the prior being a change that wouldn’t make the project fundamentally different, but would extend its real-world applications:

1. Increase the number of active sensors and add these to the scheduler for processing. Two ideas would be to use the on-board Sense Hat sensor for pressure, which we could have used but decided against because it is hard to change manually, and/or adding functionality for the on-board joystick whenever its state changes.
2. Increase the energy efficiency of our bot. We already made the power-conscious decision of using only two motors instead of four, but we didn’t make any changes to our software with power consumption in mind. We also didn’t look at the efficiency of our software design.

What We Learned:

This project was a great introduction to Python for us, as we came in with no experience with this language. Python is very versatile in that it interfaces easily with hardware and provided us with many options when it came to multiprocessing and tasks management. Our project also gave us a good understanding of scheduling tasks in a system based on priority and operating system basics; not only did we have to apply knowledge of task management, but we also had to consider the real applications of these tasks. Additionally, being able to work with a very high power embedded machine (small computer) gave us good experience with a Raspberry Pi and the many things that can go wrong when working with hardware.

Timeline/Milestones:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Number | Description | Delivery Date | Met? | Completed By | Notes |
| 0 | Submit project proposal | September 25th, 2017 | Yes | Joint | - |
| 1 | Obtain hardware/software components (shipping) | October 1st, 2017 | Yes | Vahe | - |
| 2 | Assemble bot components and research necessary hardware components | October 11th, 2017 | Yes | Dan | - |
| 3 | Set-up Raspberry Pi 3 Model B, configure/install Raspbian OS | October 11th, 2017 | Yes | Dan | - |
| 4 | Raspberry Pi Sense Hat implementation/interfacing | October 16th, 2017 | Yes | Vahe | - |
| 5 | Implement sensing module (ex. light sensitivity) | October 18th, 2017 | Yes | Vahe | - |
| 6 | Start research and modification of Raspbian OS, look at scheduling and task management as it relates to our system, implement simple bot movement | October 25th, 2017 | Yes | Dan | - |
| 7 | Prepare progress report and sample demo | October 30th, 2017 | Yes | Joint | - |
| 8 | Implement remaining sensor modules (ex. gyroscope, temperature, etc.) | November 13th, 2017 | Yes | Dan | - |
| 9 | Modify Raspbian OS for maximum optimization of our project and confirm working bot sensing system | November 29th, 2017 | Yes | Vahe | Priority levels implemented in project code (OS settings not modified) |
| 10 | Test using practical mock environments and collect data | December 6th, 2017 | Yes | Joint | - |
| 11 | Prepare final report and presentation, demo working system with at least 2 unique sensors | December 11th, 2017 | Yes | Joint | - |

Missed Milestones:

No milestones were missed. An explanation of our deviation from our original plan to implement kernel modification is included in the *Experiments/Results* section. Instead of using kernel configuration, we opted for a more straight-forward approach of setting priorities in Python based on our own scheduler design. Final project/results/goals have remained unchanged.

References:

Sense Hat Cheatsheet/Information: <https://www.raspberrypi.org/learning/astro-pi-guide/files/SenseHAT-Cheatsheet.pdf>

sentdex YouTube, Robot Motor Control: <https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ>

Raspberry Pi Kernel Building: <https://www.raspberrypi.org/documentation/linux/kernel/building.md>

Python Coding and Information: <https://www.pythoncentral.io/>