Mars Rover Simulation

GUI Project

Cameron Jones CPRE 4580

CPR E 458//558 Real-Time Systems, Fall 2023

Department of Electrical and Computer Engineering

Iowa State University

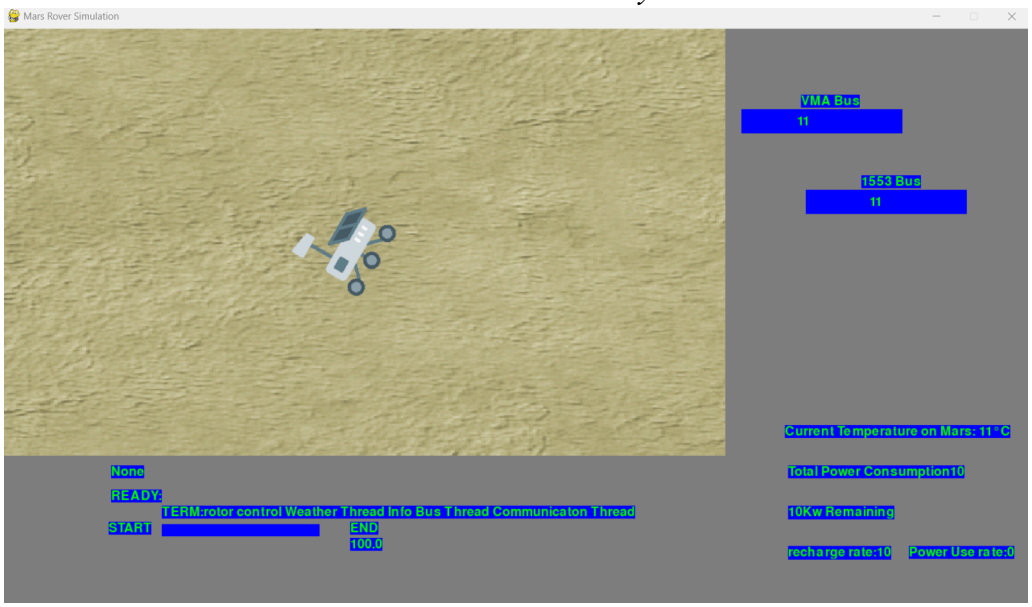

figure 1: A screenshot of the simulation

Abstract

This project seeks to simulate and visualize the Pathfinder Mars rover performing the tasks that caused it to reach deadlock in real time. The project also includes the ability to customize the scheduling algorithm performed by the simulated rover's processor and the resource access protocol used by the rover. Allowing for the simulation to run with RMS EDF and Round robin along with priority ceiling protocol, priority Inheritance protocol and traditional semaphore.

1. Introduction

It is common for beginners in the field of real time systems to start by learning about some of the most common scheduling algorithms RMS EDF and Round Robin. Along with the most common resource access control protocols, traditional semaphore priority ceiling and priority inheritance. However for many beginners these concepts may seem confusing especially in regard to the pitfalls that each method can have, and the particular scenarios that lead up to them. Furthermore there is currently no program that allows users to easily interact with these concepts in a tangible way leaving amateurs in the world of real time systems to be forced to grapple with these concepts on a purely theoretical level. If such a program did exist however it would allow users to work with these concepts and potentially develop an intuitive understanding of the basics of real time systems.

2. Project Objectives & Scope

This project aims to create a simulation which will emulate the Pathfinder Mars rover performing four

separate real time tasks using one of three scheduling algorithms, RMS,EDF or Round Robin. Three of these tasks control and reserve “critical memory areas” which work with three selectable resource access control protocols: traditional Semaphore, priority Ceiling or priority inheritance. Finally there will be a power management system with each task consuming a certain amount of power per second and the rover generating a certain amount of power per second, when all power has been depleted scheduling will cease until power returns. presented on screen will be a visual depiction of the rover moving in real time and the current state of the critical memory areas as well as information of power.

2.1. System Model

The simulation exists on a single machine, in two windows the first opening when the program is run which allows for customization of the simulated real time system. The second window opens when the second window is closed and will depict the simulation.

2.2. Problem Statement

There is no simple way of creating a real time system and playing with it to see what can improve or hinder its performance. By creating this simulation such a tool is created which will allow users to deepen their understanding of the algorithms and protocols implemented in this project.

2.3. Objectives and Scope

The aim of this project is to create a program which simulates and visualizes the Pathfinder Mars rover in action performing the three real time tasks that caused it to reach a state of deadlock along with a fourth non related task. Weather thread, information bus thread, communication thread and rotor control. Each of these tasks will perform a specific action which will visually effect the screen, along with holding and interacting with one or both critical sections the 1553 bus and the VMA bus.

Weather thread: generates a temperature value and places that value in the 1553 Bus. Holds the 1553 Bus during its execution.

Info Bus: places the current value held by the 1553 Bus into the VMA bus. Holds both buses during execution.

Communication thread: Takes the current value held by the VMA bus and places it in the weather report section of the screen displaying the current temperature. Holds the VMA Bus during execution.

Rotor control: When execution has finished rotates the rover on screen and changes the direction it is traveling. Holds no Buses during execution.

When the simulation is begun a window will open with text fields that allow for the user to set the scheduling algorithm critical access protocol used and the period computation time and power consumption of each task.

The selectable scheduling algorithms are RMS, EDF and Round Robin

The selectable priority access control protocols are traditional semaphore, priority inheritance and priority ceiling.

The user will also set the maximum size of the battery aboard the rover and the initial charge rate of the rover.

There will also be a power component each task will consume a specified amount of power each second this will compete with the current rate of power generation which will be randomized. When the battery is empty the current task will hang not progressing but also not getting closer to missing its current deadline.

Finally at the end of each simulation a text file will be generated displaying the schedule performed during the simulation along with a count of the number of missed deadlines that occurred during the simulation.

3. Solution Methodology / Approach

The implementation of the simulation is handled by two separate python files Main.py and scheduling.py. Main handles graphical processing along with controlling the main loop of the program. Scheduling handles the algorithms used in the program along with defining the classes pertaining to the critical sections, power management and any text that is displayed on screen.

3.1. Algorithms / Protocols / Architectures

The key algorithms used in the simulation are RMS EDF and round robin scheduling

The key priority access control protocols used in the simulation are traditional semaphore priority ceiling protocol and priority inheritance protocol.

RMS: Priorities are assigned based on periodicity, the tasks with the highest priority are those with the smallest period and the tasks with the lowest priority are those with the largest period.

EDF: Short for earliest deadline first, the task with the closest deadline will be given highest priority and the task with the furthest deadline will be given the least priority. All tasks in this simulation will have deadlines which correspond to a multiple of their period value.

Round Robin: The version of Round Robin implemented is not priority based. All the tasks are allotted an equal amount of time to complete their task. regardless of if they complete the task or not when their time ends they must wait until every other task finishes its allotted time before it gets a turn to either resume its task or start a new iteration.

Traditional Semaphore: When a critical memory area is held by another task and a task that requires that memory area reaches a part in its code that needs to access this area it will be put in a waiting state. In the context of the simulation when the task begins it will be put in a waiting state.

Priority Inheritance: When a critical memory area is held by a task and another task with higher priority tries to access that area, the task currently holding that critical area will inherit the priority of the other task.

Priority Ceiling: Each critical memory area has a value called a priority ceiling defined by the highest priority task that can enter that critical area. When PCP is in play a task will only enter a critical area if its priority is higher than the priority ceiling of all currently held critical areas.

3.2. Illustrative Example

By being able to set and determine different conditions for a real time system , a user will be able to see how each scheduling algorithm functions by creating a schedule where no tasks will be made to wait due to trying to access a held memory area. Alternatively they could use this simulation to see how the resource access control protocols function by creating a schedule that will guarantee a task will begin when a certain critical section is held. Finally, due to the feature of the program that generates a text file displaying the schedule performed and the number of missed deadlines the user is given a measure of the efficiency of different set ups allowing them to compare and contrast the effectiveness of the different algorithms and protocols.

4. Implementation/ Simulation Architecture

Graphics processing was handled by pygame, which allows for a simulation to be opened and text sprites and shapes to be drawn on screen.

The configuration window is handled by Tkinter, a library which opens a window with the option of adding text fields, checkboxes and drop down menus that link to variables in the program.

Python random library is used for all random generation.

thread class: A super class defining the generic attributes of each task performed in the simulation. Includes a variable for when the task most recently exited the terminated state, and a deadline variable along with period and computation time and priority.

Individual task classes: Each task performed during the simulation is defined as a separate subclass of the thread class. Each with a different definition of the task method which is performed when the task completes a different definition of the CanRun method which checks the availability of the register objects. And corresponding Start and Finish methods which respectively take and release ownership of their respective registers.

Register classes: The register classes have variables defining the text that is passed to them by the thread sub classes and a flag variable that indicates if it is currently held. Finally it includes a priority ceiling variable indicating the highest priority of a task that can hold it.

Scheduling class: The scheduling class defines the text variables that are placed on screen along with instantiating the tasks used in the program. it also defines the methods used for processing each of the scheduling algorithms and any necessary methods for supporting these algorithms.

Basic setup: For most of the programs run time it will be running the same while loop which uses pygames functionality to check if the close window button has been pressed, then runs one of three scheduling methods depending on the selected algorithm. Then finally uses pygames blit() method to draw updated text and sprites on screen.

Priority Queue: A class defined to represent the ready queue. Contains the functions Get and Delete which cycle through a list to find the task with the highest priority, at which point it either deletes it from the list and returns the task that was selected or just returns the task without deleting

RMSPriority Gen: A method defined in the scheduling class. It runs insertion sort on a list of the four tasks instantiated by the scheduling class based on the period of each task, each defining the priority of each task as equal to the iteration of the insertion sort. Then it concludes by placing each task within the ready queue.

RMS: If RMS is the selected scheduling algorithm. The method RMS priority Gen is run. Following this the delete function is used to pop the highest priority task off of the ready queue. That task has its start time set by system time. Each time the main loop passes its start time minus current system time is compared against the task's computation time. If the task has finished its execution time it is placed in the terminated list. As well each iteration of the main loop the priority queue get function is called retrieving the highest priority task in the ready queue without popping it from the queue it is then compared against the priority of the currently selected current task if the priority is greater than the current task the current task is inserted back into the ready queue and the retrieved task is popped from the ready queue. As well every call of the RMS scheduling method comes with a loop through the terminated list to check if system time minus its most recent start time is equal to its deadline variable; if so then that task is added back to the ready queue with its start time and deadline updated. Finally each call of this method includes a parse through the ready queue checking if any deadlines have been missed, if so the deadline counter variable is updated and the tasks starttime and deadline are updated.

EDF: The structure of the EDF method is largely the same as that of RMS however instead of calling a single priority gen function before the main loop starts, instead when the preemption check occurs a separate priority queue function is called getEDF which locates the task with the closest deadline then if that tasks deadline is closer than the current task the current task is preempted.

Round Robin: When each task gets to take its turn current time is taken. Instead of a priority based preemption check each call of the round robin method starts with a check of the turn start time vs the set turn time limit. If a task completes its computation time before its turn finishes the task is added to the terminated queue and a flag is raised that indicates that the current task is finished updating text on screen to reflect this and indicating that the check if the task is nearing completion should not be performed. When its current turn ends the flag is lowered.

Priority Inheritance: If priority inheritance is selected and RMS is selected as the chosen scheduling algorithm, then if the method canRun fails when a new task is selected and the priority of the task attempting to begin is higher than that of the task failing to start is higher than the task currently holding the register then the tasks priority variable is set equal to the priority of the task attempting to begin. When the current task ends both tasks priority is set back to their original values from original priority values set during RMSPriorityGen.

Priority Ceiling: (not entirely implemented) Each register object has a priority ceiling variable set when RMSPriorityGen is called. If Priority Ceiling is selected as the resource access control protocol then when CanRun is run during the preemption check instead of just checking whether the needed register objects are held both registers are be checked and if either bus is held and its priority ceiling is higher than the priority variable of the new task then the task will be kept in the ready queue.

Power system: Max battery power and initial recharge rate are set in on the configuration screen along with power usage rate of each task. each call of

the chosen scheduling method uses this formula to calculate current power

battery power = (current Recharge rate - current power usage) * (current Time - task.startTime)

A random number is generated to determine the number of seconds until a new recharge rate is selected when this number of seconds passes based on system time and the time when the number is generated a new random recharge rate is calculated and a new random number of seconds is calculated.

When the amount of "energy" in the battery reaches zero. The system time when this event is taken and instead of checking whether the task is completed or not a variable is taken that computes the amount of time that the system has not had power for based on the time at which the rover lost power. This variable is then subtracted against the calculation of current time vs the start of the task meaning that the percent completion of the task remains the same while the task does not have power.

player class: defines the Image used for the rover the current angle the rover is facing and the direction the rover is facing. And instantiates the image as a pygame rectangle object allowing for transforms to be performed on the image depicting the rover.

Movement: In order to display the rover continuously while still in the same position on screen the background is made to move rather than the image of the rover itself. Initially there is an image of the background drawn directly over the rover and two in each cardinal and intermediate direction meaning that the "rover" could travers a distance equal to 1.5 the lengths of the background image before reaching a section with no drawn image. To simulate infinite scrolling two background images are kept appended to the current background image at all time in all directions to do this a tally is kept of how many backgrounds of distance the rover has traversed when the rover reaches the end of the first background and moves onto the second one a new background image is appended to the background image the rover has move onto.

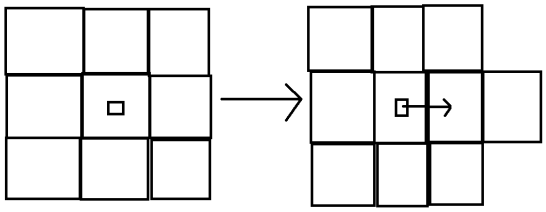

figure 2: illustration of how movement works

a depiction of how movement is handled, the smaller square represents the rover . As the collection of background images shift left a new image is drawn to the right of the rover allowing for infinite scrolling.

5. Evaluation

Test setup: One machine running the simulation

Unit tests:

Running each of the scheduling algorithms with schedules that result in no preemption and no missed dead lines

example[c,p]: [1,40],[1,30],[1,20],[1,15]

Performance metric: The ready queue and the termination list populate in the proper order and with no obvious bugs. The current task remains for the proper amount of time and the task performed at the end of the task updates the screen correctly.

Unit Test: Running each of the scheduling algorithms with schedules that guarantee preemption

example[c,p]: [5,10] [11,20],[15,30],[16,40]

performance metric: The current task updates correctly, missed deadlines process correctly, the missed deadline counter works correctly.

Unit test: For each resource access control protocols run a schedule where priority inheritance, tasks being blocked from beginning was guaranteed to occur.

example[c,p]:
InfoBus[5,10],weather [11,20],[15,30],[16,40]

performance metric: The current task performs the correct action either returning to the ready queue or swapping priorities.

Unit test: Run the program with power consumption rates such that the program was guaranteed to run out of energy but eventually recover.

example: recharge rate 10, power consumption rates: 1000, 1,1,1

performance metric: While unpowered the percent completion of the current task remains consistent and after the rover is powered again begins where it left off.

Full system test: For every combination of scheduling algorithm and resource access control protocol. Run a series of tests with no possible preemption with preemption and with the ability to be preempted due to critical areas being already held.

6. Conclusions

This project was a good opportunity to review the concepts behind the algorithms and protocols addressed within the simulation. Along with this it was good to review and remember the best practices of OOP. For instance, creating god classes should be avoided.

Largely however the project was successful; a working simulation with nearly all planned algorithms and protocols was implemented. However testing could have been more through to help reveal edge cases as well priority ceiling protocol still needs to be completely implemented.

Self-Assessment of Project Completion:

Project learning objectives	**Status (Not/Partially /Mostly/Fully Completed)**	**Pointers in the document**
Create a working accurate simulation of a real time system	Partially Completed	Section 2, page 1
RMS,EDF,RR, PCP,Priority inheritance successfully implemented	All implemented except for PCP with likely many edge case unaccounted for	Section 2.3, page 2
Pygame Tkinter and priority queue successfully implemented	Fully implemented with no issues	Section 2.3, page 2
Testing and evaluation – test cases, metrics, test results, any relevant performance results	Testing largely completed although could have been more through	Section 5, page 5
Overall Project Success assessment	Mostly successful an incomplete yet working prototype has been created	