

Abstract

If you have ever owned a Nokia phone you are sure to recognize this classic game. Snake has been around since the 1970s where it grew into popularity. Even today Snake still remains admired. YouTube has even implemented into its videos while videos buffer. The objective of game is to control a character that resembles a snake to eat apples. With each apple consumed the snake grows one unit. After consuming the apple, another one is randomly generated and displayed elsewhere on the screen. The user must consume as many apples as possible while avoiding running into itself or the wall.

Introduction

The goal of my project is to implement a snake game that uses the TS-7250 board. A problem with the project is allowing the user to control the game with the I/O board for the TS-7250. The use of memory mapping was used to obtain the users button information. This information was stored into a buffer using mailboxes. Another problem arises from the game continuous movement. A thread is used in the background to keep track of the game timer. The information is updated continuously from the inputs and is stored into an array and then printed.

Background

There are various approaches in order capture the users input from the I/O board. For instance creating a module to set up an interrupt service routine (ISR) may have been used. The ISR is created to handle an event for each direction of the snake which corresponds to the lower

pins of port B. The pins are program to cause an interrupt that would trigger the given direction to be initiated during a desired speed that is configured through the interrupt registers of the TS-7250. Another approach would be to use a mailbox. The mailbox allows multiple tasks to pass data or for synchronization. When a task sends data to the mailbox a flag linked with the mailbox indicates that data is available. The data is then read and the flag resets. The method that was chosen to implement to user input was the mailbox structure. The mailbox structure was used to implement the program rather than the ISR created by a module. The reason for avoiding the ISR was due to a chance that the ISR could encounter a block. Real time task were created in the code to keep track of the snake direction determined by the user. This was necessary in order to determine where the snake on the screen was. There weren't any other approaches that were I deemed feasible when completing this task.

Proposed Approach

I created an array called `matrix[]` that is used to store the display data. An array is also created for the body of the snake called `snake[]`. The user interface features the I/O board for the TS-7250 that has been used throughout the semester. The board is connected to PORT B of the TS-7250 where a pointer points to the memory address that is mapped using the `mmap()` function. A real time task is created to check if the button has been pushed and which one. A process is created for the directional input of each button. A message structure that is created contains values that are used to check and debug the data within the mailbox. A mailbox is initialized called `btn_mbox` which is used to for the inter-process communication. Each time a button is pressed by the user it sets a value in the message that corresponds to the button direction which is retrieved from `direct()`. `Rt_mbx_send()` then stores this message. The `main()` function then uses `rt_mbx_receive()` to retrieve these messages which is then used to set the

value of the direction being sent to moveSnake(). moveSnake() then uses the direction value for case statements that moves the snake. It takes the checks the value of the current location of the beginning of the snake to determine the direction. For instance when the direction is 0 it checks case 0 in which the snake moves left by using if statements to determine if the snake has encountered anything by checking if the next position in the array is not the first column of the array, if so the game ends by calling on the endGame() function which resets the wall with a for loop and calls resetSnake() to reinitialize the matrix. resetSnake() then calls clearMatrix() which clears the matrix. When the direction is set to one case 1 is called which checks if the snake has encountered the last column and if it does not the snake is shifted right. After the direction is determined the variable snakehead is then used to check if it has encountered the applepos variable which gives the apple position. When the snake does it calls setApple() and sets a flag that the apple is caught. The score is then incremented by 10 * the level and the level is checked whether it has reached a desired score before being incremented. A for loop is then used to check if the snake has ran into itself by checking if each position in the snake array is equal to the snake head. The next step checks variable applecaught to see if snake array needs to be incremented. If the statement is true the snake matrix is shifted down and the new position of the snake head is stored in the array and the applecaught flag is resets to 0. At the end of the function updateMatrix() is called to set the new values of the matrix using a for loop. In order to generate my apple setApple() is called which uses the rand() function to randomly place the apple. The apple position is then checked if it is being placed on top of another object such as the wall or snake which would cause it to reevaluate its position and try again. For the frame of my game I used drawBorder() to place characters around the game. My final objective was to create a task that enables the snake to move at a certain speed. A thread generated using

pthread_create() is used to as create the timer which allows it to run within the background of the game.

Results

There were a few problems that were encountered throughout the project. One problem was the movement of the snake. The snake seems to not respond quickly when the buttons are pressed.

Another problem encountered was trying to figure out ways to determine if the snake had encountered the walls of the matrix. This was a problem since I chose to use a 1D array rather than 2D which would have lead to a more efficient algorithm for the movement.

Conclusion

The project has taught me how to implement real time tasks and inter-process communication. Due to the time constraints the game results could have been better. The use of a 2D array to implement the display matrix as well as the location of the objects would have been more beneficial. Another task that could have been realized would be color for the display which would allow the user to distinguish between objects better.