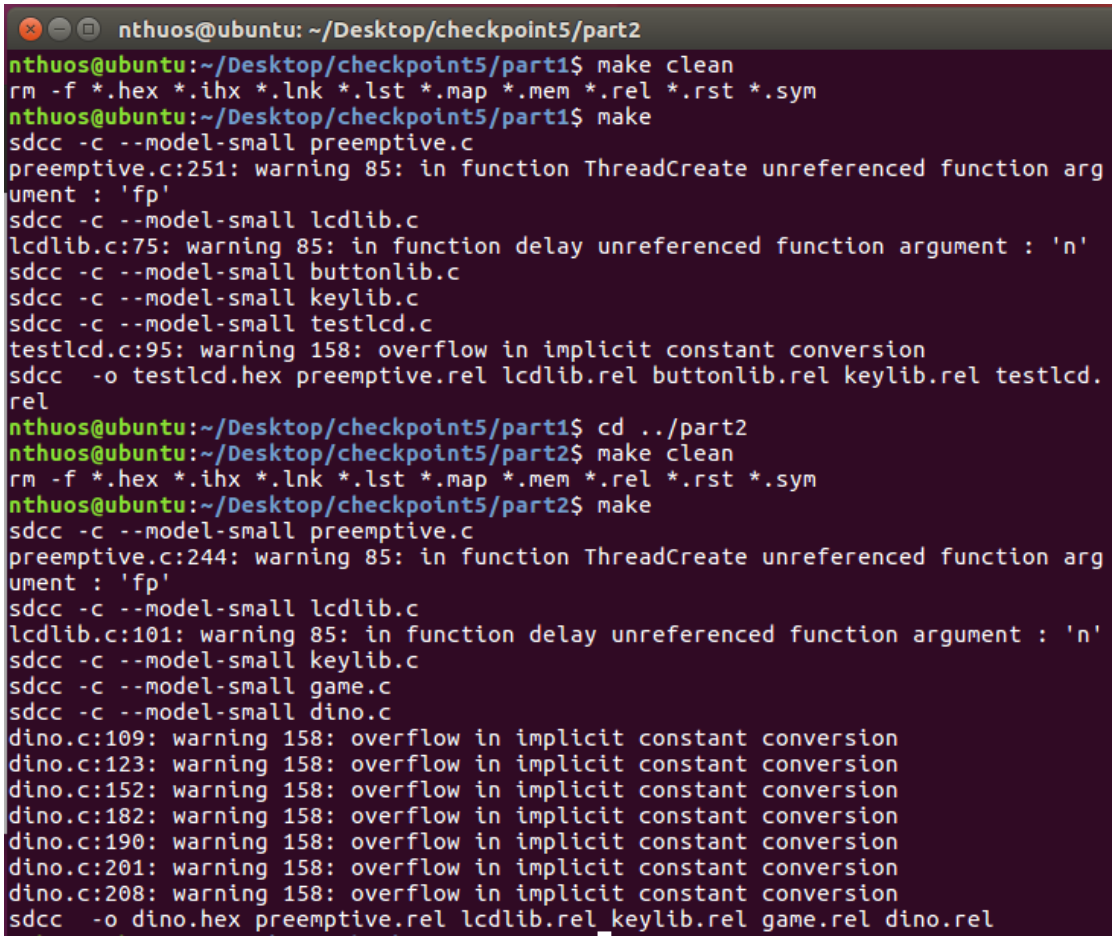
110060035 黃振寧

1.Typescript



2. Explanation

For code explanation, I'm going to explain the functions I implement.

1. Preemptive.c/.h

SAVESTATE Macro:

- This macro is used to save the context of the current thread before switching to another.
- It saves the accumulator (ACC), B register (B), data pointer registers (DPL and DPH), and processor status word (PSW) onto the stack.
- Additionally, it saves the stack pointer (SP) into the saved_sp array indexed by the current thread ID.

RESTORESTATE Macro:

- This macro is used to restore the context of a previously saved thread.
- It assigns the stack pointer (SP) the value stored in the saved_sp array for the current thread.
- Then, it pops the registers PSW, DPH, DPL, B, and ACC from the stack to restore the thread's context.

PUSH_REGS_ON_STACK and POP_REGS_FROM_STACK Macros:

- PUSH_REGS_ON_STACK macro pushes the values of registers R0 to R7 onto the stack.
- POP_REGS_FROM_STACK macro pops the values from the stack and restores them into registers R0 to R7.

void Bootstrap(void) :

- This function is the entry point for the program and initializes the system.
- It initializes data structures for threads, including the thread_bitmap.
- Sets up the timer 0 interrupt for thread scheduling.
- Creates a thread for the main function and sets the current thread ID to this new thread.
- Finally, it restores the context of the newly created thread to begin executing main.

ThreadID ThreadCreate(FunctionPtr) :

- This function creates a new thread and prepares it for execution.
- It checks if the maximum thread count has been reached and returns -1 if so.
- Otherwise, it finds an available thread ID in the thread_bitmap, marks it as in use, and initializes the thread's stack.
- The registers, stack pointer, and processor status word are set up to prepare the thread for execution.

void ThreadExit(void) :

- This function is called by a thread to terminate itself.
- It clears the bit for the current thread in the thread_bitmap, effectively marking it as available for reuse.
- It then selects another valid thread as the current thread, if any, to continue execution.
- If no more valid threads are available, it sets the current thread ID to -1, effectively stopping the program.
- Finally, it restores the context of the selected thread, allowing it to continue execution.

myTimer0Handler():

- It calls BoardRefresh to update the board in the game(if the game has started).
- Next, base on round robin policy to find the next thread to be executed.

2. Testlcd.c (part 1)

   void ProducerButton(void):

   - It enters an infinite loop and never returns.
   - It waits for its turn by acquiring the ProducerButtonTurn semaphore.
   - Inside the loop, it checks if any button is pressed using the AnyButtonPressed() function.
   - If a button is pressed, it retrieves the button character using ButtonToChar() and stores it in tmp.
   - It then acquires the empty and mutex semaphores, ensuring mutual exclusion.
   - It sets SharedBuffer to the value of tmp, signaling that data is ready to be consumed.
   - It releases the mutex semaphore and signals the full semaphore.
   - Finally, it signals the ProducerKeypadTurn semaphore to give the turn to the other producer thread.

void ProducerKeypad(void):

- Similar to ProducerButton(), it enters an infinite loop and never returns.
- It waits for its turn by acquiring the ProducerKeypadTurn semaphore.
- Inside the loop, it checks if any key is pressed using the AnyKeyPressed() function.
- If a key is pressed, it retrieves the key character using KeyToChar() and stores it in tmp.
- It then acquires the empty and mutex semaphores, ensuring mutual exclusion.
- It sets SharedBuffer to the value of tmp, signaling that data is ready to be consumed.
- It releases the mutex semaphore and signals the full semaphore.
- Finally, it signals the ProducerButtonTurn semaphore to give the turn to the other producer thread.

void ConsumerLCD(void):

- It initializes the UART for serial communication and sets up the LCD.
- It enters an infinite loop and never returns.
- Inside the loop, it checks if the LCD is ready using LCD_ready().
- If the LCD is ready, it acquires the full and mutex semaphores, ensuring mutual exclusion.
- It writes the character stored in SharedBuffer to the LCD.
- It releases the mutex semaphore and signals the empty semaphore.

main():

- It initializes the global variables and semaphores.
- It creates two producer threads: ProducerButton and ProducerKeypad.
- It calls the ConsumerLCD function to create the consumer thread.

3. Game.c/.h (part 2)

Void gameInit(void):

- Initializes the game board with obstacles.
- Sets initial game state, interrupt counter, dinosaur row, score, and last input.

Void SetGameLevel(unsigned char level):

- Allows users to set the game level.
- Takes an unsigned character level as input.
- Assigns level to the game state if it's within the valid range (0 to 9).
- Resets the game state to 0 if level is greater than 9.

unsigned char GameStart(void):

- Checks if the game is ready to start.
- Returns 1 if the game state is between 0 and 9 (inclusive), indicating the game is playable.
- Returns 0 if the game state is outside this range, indicating the game is not playable.

void MoveDinosaur(unsigned char direction):

- Allows the player to control the dinosaur's position.
- Takes an unsigned character direction as input.
- Sets the dinosaur's row to 0 for direction 2 (top) and 1 for direction 8 (bottom).

void BoardRefresh(void):

- Updates the game board by shifting it left by one bit and checks for collisions.
- Executes if the game has started (GameStart returns 1).
- Increments the interrupt counter and if the counter equals to 65-4*level, it shifts the input left.
- Shifts bits in the game board arrays to move obstacles.
- Checks for collisions between the dinosaur and obstacles.
- Updates the game state and player's score accordingly.

unsigned char GameOver(void):

- Checks if the game has ended.
- Returns 1 if the game state is 10, indicating the game is over.
- Returns 0 if the game is still in progress.

unsigned char GetScore(void) / unsigned char GetDinosaurRow(void):

- GetScore returns the player's current score as an unsigned character.
- GetDinosaurRow returns the current row of the dinosaur (0 for top, 1 for bottom).

unsigned char GetGameBoard(unsigned char row, unsigned char col):

- Allows querying the status of individual cells on the game board.
- Takes row and col as parameters to specify the cell.
- Returns 1 if the cell contains an obstacle, 0 if it's empty, and -1 if the input is out of bounds.

4. Memory.h(part2)
   This header file allocates the memory to 8051.

5. Dino.c(part2)

   void GameCtrl(void):

   - This function is responsible for managing the game logic in the Dino Game.
   - It continuously checks the game state and acts accordingly.
   - If the game is running, it waits for specific semaphores, then moves the dinosaur on the game board based on user input and board refreshes.
   - If the game is over, it waits for semaphores, checks for any keypress to restart the game.
   - If the game is not started, it waits for semaphores and processes user input to set the game level.

   void KeypadCtrl(void) (part 2):

   - This function handles keypad input.
   - It initializes the keypad and continuously monitors it for key presses.
   - When a key is pressed, it stores the key in the SharedBuffer and signals semaphores to inform other threads.
   - Note that the key needs to be released to complete the input signal.

void RenderTask(void) (part 2):

- This function is responsible for rendering the game on an LCD display.
- It configures the LCD for communication and continuously updates the display based on the game state.
- If the game is running, it displays the game board and the dinosaur's position.
- If the game is over, it displays a game over message and the player's score.
- If the game is not started, it displays the current game level and instructions to start the game.

main()

- The main() function is the entry point of the program.
- It initializes global variables and sets up semaphores.
- It calls GameInit() to initialize the game.
- It creates two threads: GameCtrl and KeypadCtrl for game control and keypad input handling.
- Finally, it calls RenderTask() to manage the LCD display.
- Interrupt Handler: timer0_ISR()
- This function is an interrupt handler for Timer 0. It is invoked periodically by Timer 0 interrupts.
- It redirects control to the _myTimer0Handler function.

void timer0_ISR(void) __interrupt(1):

- The actual interrupt handler for Timer 0.
- It jumps to _myTimerHandler to get its job done..