ISEL
DEETC
Departamento de Engenharia Electrónica e de Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Sistema de Controlo de Acessos (*Access Control System*)

Gonçalo Castro – A49417
Ana Pereira-A49470
Rúben Duarte-A48951

Projeto
de
Laboratório de Informática e Computadores
2022 / 2023 verão

29 de Maio de 2023

1 Introdução

Neste projeto implementa-se um sistema de controlo de acessos (*Access Control System*), que permite controlar o acesso a zonas restritas através de um número de identificação de utilizador (*User Identification Number – UIN*) e um código de acesso (*Personal Identification Number - PIN*). O sistema permite o acesso à zona restrita após a inserção correta de um par *UIN* e *PIN*. Após o acesso válido o sistema permite a entrega de uma mensagem de texto ao utilizador.

O sistema de controlo de acessos é constituído por: um teclado de 12 teclas; um ecrã *Liquid Cristal Display* (LCD) de duas linhas de 16 caracteres; um mecanismo de abertura e fecho da porta (designado por *Door Mechanism*); uma chave de manutenção (designada por M) que define se o sistema de controlo de acessos está em modo de Manutenção; e um PC responsável pelo controlo dos outros componentes e gestão do sistema. O diagrama de blocos do sistema de controlo de acessos é apresentado na Figura 1.

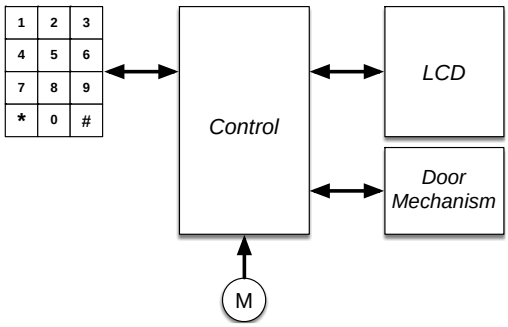


Figura 1 – Sistema de controlo de acessos (*Access Control System*)

Sobre o sistema podem-se realizar as seguintes ações em modo Acesso:

- **Acesso -** Para acesso às instalações, o utilizador deverá inserir os três dígitos correspondentes ao *UIN* seguido da inserção dos quatro dígitos numéricos do *PIN*. Se o par *UIN* e *PIN* estiver correto o sistema apresenta no LCD o nome do utilizador e a mensagem armazenada no sistema se existir, acionando a abertura da porta. A mensagem é removida do sistema caso seja premida a tecla '*' durante a apresentação desta. Todas os acessos deverão ser registados com a informação de data/hora e *UIN* num ficheiro de registos (um registo de entrada por linha), designado por *Log File*.
- **Alteração do PIN –** Esta ação é realizada se após o processo de autenticação for premida a tecla '#'. O sistema solicita ao utilizador o novo *PIN*, este deverá ser novamente introduzido de modo a ser confirmado. O novo *PIN* só é registado no sistema se as duas inserções forem idênticas.

Nota: A inserção de informação através do teclado tem o seguinte critério: se não for premida nenhuma tecla num intervalo de cinco segundos, o comando em curso é abortado; se for premida a tecla '*' e o sistema contiver dígitos, elimina todos os dígitos, se não contiver dígitos, aborta o comando em curso.

Sobre o sistema, podem-se realizar também as seguintes ações em modo Manutenção. Ao contrário das ações em modo Acesso, as ações em modo Manutenção são realizadas através do teclado e ecrã do PC. As ações disponíveis neste modo são:

- **Inserção de utilizador -** Tem como objetivo inserir um novo utilizador no sistema. O sistema atribui o primeiro *UIN* disponível, e espera que seja introduzido pelo gestor do sistema o nome e o *PIN* do utilizador. O nome tem no máximo 16 caracteres.
- **Remoção de utilizador -** Tem como objetivo remover um utilizador do sistema. O sistema espera que o gestor do sistema introduza o *UIN* e pede confirmação depois de apresentar o nome.
- **Inserir mensagem -** Permite associar uma mensagem de informação dirigida a um utilizador específico a ser exibida ao utilizador no processo de autenticação de acesso às instalações.
- **Desligar –** Permite desligar o sistema de controlo de acessos. Este termina após a confirmação do utilizador e reescreve o ficheiro com a informação dos utilizadores. Esta informação deverá ser armazenada num ficheiro de texto (com um utilizador por linha) que é carregado no início do programa e reescrito no final do programa. O sistema armazena até 1000 utilizadores, que são inseridos e suprimidos através do teclado do PC pelo gestor do sistema.

Nota: Durante a execução das ações em modo manutenção, não podem ser realizadas ações no teclado do utilizador e no LCD deve constar a mensagem "*Out of Service*".

2 Arquitetura do sistema

O controlo (designado por *Control*) do sistema de acessos será implementado numa solução híbrida de *hardware* e *software*, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por quatro módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD*, designado por *Serial LCD Controller* (*SLCDC*); *iii*) um módulo de interface com o mecanismo da porta (*Door Mechanism)*, designado por *Serial Door Controller* (*SDC*); e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) deverão ser implementados em *hardware* e o módulo de controlo deverá ser implementado em *software* a executar num PC.

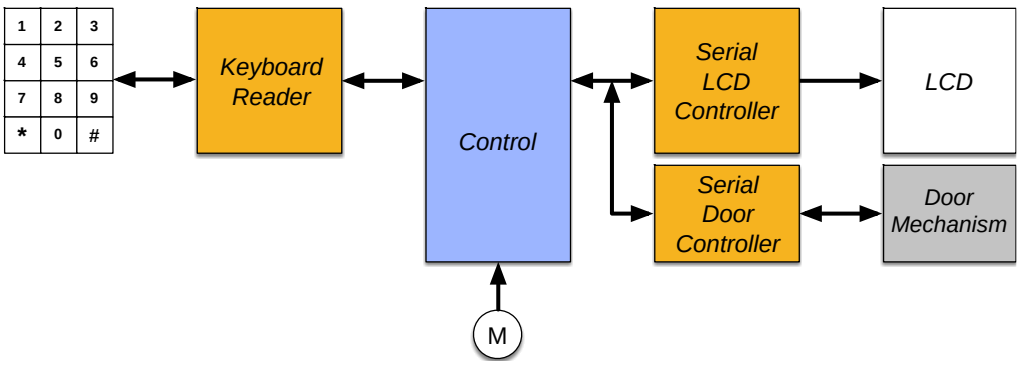


Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (*Access Control System*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando o código desta em quatro bits ao *Control*, caso este esteja disponível para o receber. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de nove códigos. O *Control* processa e envia para o *SLCDC* a informação contendo os dados a apresentar no *LCD*. A informação para o mecanismo da porta é enviada através do *SDC*. Por razões de ordem física, e por forma a minimizar o número de sinais de interligação, a comunicação entre o módulo *Control* e os módulos *SLCDC e SDC* é realizada através de um protocolo série.

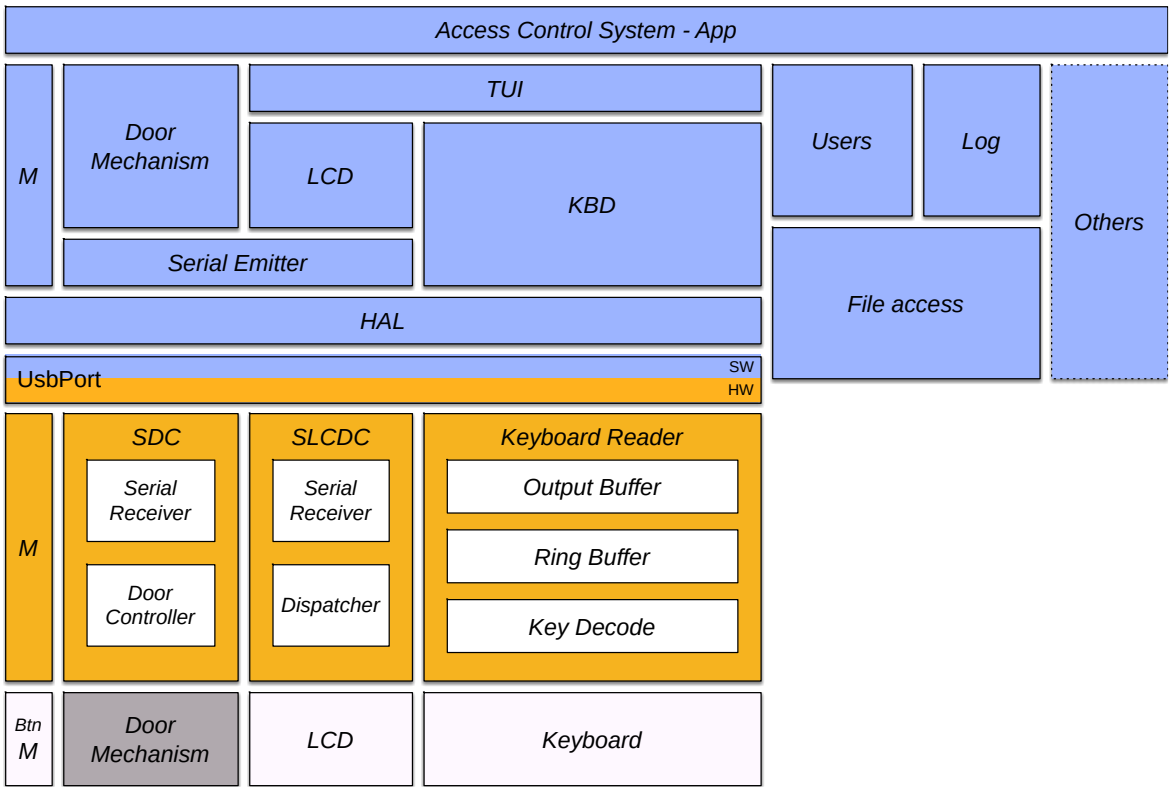
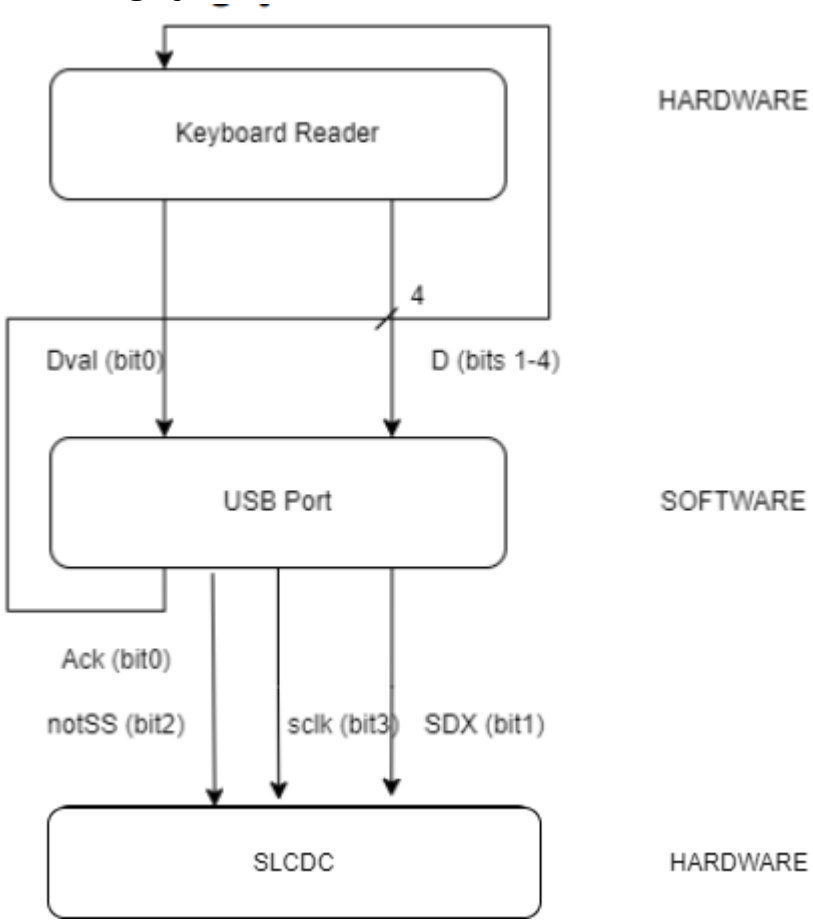


Figura 3 – Diagrama lógico do Sistema de Controlo de Acessos (*Access Control System*)

A. Interligações entre o HW e SW



B. Código *Kotlin - HAL*

```
import isel.leic.UsbPort

object HAL { // Virtualiza o acesso ao sistema UsbPort

    var state = 0

    fun init() {
        UsbPort.write(state)
    }

    // Retorna true se o bit tiver o valor lógico '1'

    fun isBit(mask: Int): Boolean = mask and UsbPort.read()==mask// ver o d mask

    // Retorna os valores dos bits representados por mask presentes no UsbPort
    fun readBits(mask: Int): Int = mask and UsbPort.read()

    // Escreve nos bits representados por mask o valor de value
    fun writeBits(mask: Int, value: Int) {
        val valueUnderMask = mask and value
        val outMaskOn = mask.inv() and state
        val result = valueUnderMask or outMaskOn
        UsbPort.write(result)
        state = result
    }

    // Coloca os bits representados por mask no valor lógico '1'
    fun setBits(mask: Int) {
        state = state or mask
        UsbPort.write(state)
    }

    // Coloca os bits representados por mask no valor lógico '0'
    fun clrBits(mask: Int) {
        state = state and mask.inv()
        UsbPort.write(state)
    }
}

fun main(){


    while(true) {
        val x = HAL.isBit(0x09)
        if (x) println("hey")
    }



}
```

C. Código *Kotlin - KBD*

```
import KBD.NONE
import KBD.waitKey
import isel.leic.utils.Time

object KBD { // Ler teclas. Métodos retornam ‘0’..’9’,’#’,’*’ ou NONE.
    const val DVAL_MASK = 0X01
    const val KACK_MASK=0X01
    const val LEDS_MASK=0x1E// alterar o nome
    const val NONE = 0.toChar()
    private val values= charArrayOf('1','4', '7', '*' ,'2','5','8','0','3','6','9','#', NONE, NONE, NONE, NONE)

    // Inicia a classe
    fun init() {
        HAL.init()
        HAL.clrBits(KACK_MASK)
    }

    // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {// working
        if (HAL.isBit(DVAL_MASK)) {
            val value = HAL.readBits(LEDS_MASK).shr(1)

            HAL.setBits(KACK_MASK)// set bit
            while (HAL.isBit(DVAL_MASK)) {
            }
            HAL.clrBits(KACK_MASK) // o value é sempre 15
            return values[value]
        }
        return NONE
    }

    // Retorna a tecla premida, caso ocorra antes do ‘timeout’ (representado em milissegundos), ou NONE caso contrário.
    fun waitKey(timeout: Long): Char {// se eu emter 100000 por no maximo os 10 000 se encontrar uma tecla retorna
imediatamente
        val StartTime = Time.getTimeInMillis()
        while (Time.getTimeInMillis() - StartTime < timeout) {
            val pressedKey = getKey()
            if (pressedKey != NONE) return pressedKey

        }
        return NONE
    }
}
fun main(){// carregar nas teclas
    HAL.init()
    KBD.init()

    while (true) {
        println(waitKey(123440))
    }
}
```

D. Código *Kotlin – SerialEmitter*

```
import isel.leic.utils.Time

object SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.
    enum class Destination { LCD, DOOR }

    private const val SERIAL_DATA = 0x02
    private const val SERIAL_SS = 0X04
    private const val DOOR_SS = 0X20
    private const val SERIAL_CLK = 0x08
    private const val SERIAL_ACCEPT = 0x20

    // Inicia a classe
    fun init() {
        HAL.init()
        HAL.setBits(SERIAL_SS)
        HAL.clrBits(SERIAL_CLK)

    }

    // Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados em 'data'.
    fun send(addr: Destination, data: Int) {
        var frame = data
        if (addr == Destination.LCD) HAL.clrBits(SERIAL_SS)
        else HAL.clrBits(DOOR_SS)
        repeat(5) {
            if (frame.and(1) == 0) HAL.clrBits(SERIAL_DATA)
            else HAL.setBits(SERIAL_DATA)
            frame = frame.shr(1)
            HAL.setBits(SERIAL_CLK)
            HAL.clrBits(SERIAL_CLK)
        }
        if (addr == Destination.LCD)
            HAL.setBits(SERIAL_SS)
        else HAL.setBits(DOOR_SS)

    }
```

```
    // Retorna true se o canal série estiver ocupado
    fun isBusy(): Boolean = HAL.isBit(SERIAL_ACCEPT)
}
fun main(){
    HAL.init()
    SerialEmitter.init()
    SerialEmitter.send(SerialEmitter.Destination.LCD,0x02   )
}
```

E. Código *Kotlin - LCD*

```
import isel.leic.utils.Time

object LCD { // Escreve no LCD usando a interface a 4 bits.
    private const val LCD_RS = 0x20
    private const val LCD_ENABLE = 0x40
    private const val LCD_DATA = 0x1E
    private const val CLEAR_DISPLAY = 0x01
    private var state = false
    private const val CMD_DISPLAY_LENGHT = 0x28
    private const val CMD_DISPLAY_ENTRY_MODE = 0x06
    private const val CMD_DISPLAY_OFF = 0x08
    private const val CMD_DISPLAY_ON = 0x0F
    private const val CMD_DISPLAY_CLEAR = 0x01


    // Escreve um nibble de comando/dados no LCD em paralelo
    private fun writeNibbleParallel(rs: Boolean, data: Int) {
        HAL.writeBits(LCD_DATA, data shl 1)
        if (rs) HAL.setBits(LCD_RS) else HAL.clrBits(LCD_RS)
        Time.sleep(1)
        HAL.setBits(LCD_ENABLE)
        Time.sleep(1)
        HAL.clrBits(LCD_ENABLE)
        Time.sleep(1)

    }


    // Escreve um nibble de comando/dados no LCD em série
    private fun writeNibbleSerial(rs: Boolean, data: Int) {
        val rsToInt = if (rs) 1 else 0
        val newData = data.shl(1) or rsToInt
        SerialEmitter.send(SerialEmitter.Destination.LCD,newData)
    }

    // Escreve um nibble de comando/dados no LCD
    private fun writeNibble(rs: Boolean, data: Int) {
        if (state) writeNibbleParallel(rs,data) else writeNibbleSerial(rs,data)
    }


    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int) {
        writeNibble(rs,data shr 4) // vai escrever um byte no display// fazer shift
de 4 bits para a direita para ler a parte alta
        Time.sleep(10)
        writeNibble(rs,data and 0x0F) // aqui fazer o and com a  e a mascara para
oter o val
        Time.sleep(20)
    }



    // passa data que é o valor do comando enviado para o display
    fun writeCMD(data: Int) {
        writeByte(false, data)
    }
    // passa data que é o valor dos dados que são enviados para o display
```

```
    private fun writeDATA(data: Int) {
        writeByte(true, data)
    }


    // Envia a sequência de iniciação para comunicação a 4 bits.
    fun init() {
        SerialEmitter.init()
        Time.sleep(15)
        writeNibble(false, 0x03)
        writeNibble(false, 0x03)
        writeNibble(false, 0x03)
        writeNibble(false, 0x02)
        writeCMD(CMD_DISPLAY_LENGHT)
        writeCMD(CMD_DISPLAY_OFF)
        writeCMD(CMD_DISPLAY_CLEAR)
        writeCMD(CMD_DISPLAY_ENTRY_MODE)
        writeCMD(CMD_DISPLAY_ON)
    }

    fun write(c: Char) {
        writeDATA(c.code)
    }

    // a string é escrita no display caracter por caracter
    fun write(text: String) {
        for (c in text) {
            write(c)
        }
    }


    // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
    fun cursor(lin: Int, col: Int) {
        writeCMD( (lin * 0x40 + col) or 0x80) // colocar bit de maior peso a 1
    }
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() {
        writeCMD(CLEAR_DISPLAY)
    }
}
fun main(){// slcdc
    HAL.init()
    LCD.init()
    LCD.cursor(0,0)

    LCD.write("hey word")
    Time.sleep(5000)
    LCD.clear()
    Time.sleep(200)
    while (true){
        LCD.cursor(0,0)
        LCD.write("0123456789ABCDEF")
        LCD.cursor(1,0)
        LCD.write("0123456789ABCDEF")
    }

}
```

F. Door Mechanism.

```
object DoorMechanism { // Controla o estado do mecanismo de abertura da porta.
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init(){
        SerialEmitter.init()

    }
    // Envia comando para abrir a porta, com o parâmetro de velocidade
    fun open(velocity: Int){
        val speed = velocity.shl(1) or (1)
        SerialEmitter.send(SerialEmitter.Destination.DOOR,speed)
    }

    // Envia comando para fechar a porta, com o parâmetro de velocidade
    fun close(velocity: Int){
        val speed=velocity.shl(1)
        SerialEmitter.send(SerialEmitter.Destination.DOOR,speed)
    }

    // Verifica se o comando anterior está concluído
    fun finished() : Boolean = !SerialEmitter.isBusy()
}

fun main(){
        DoorMechanism.init()
        DoorMechanism.open(0x02)
        while (!DoorMechanism.finished());
        DoorMechanism.close(0x07)
        while (!DoorMechanism.finished());

}
```

G. TUI

```
object TUI {
    fun readInt(key: Long): Int {
        val char = KBD.waitKey(key)
        return char.code
    }


    fun useriD(): Int? {
        var id = ""
        writeStr("ID:???")
        LCD.cursor(1,3)
        while (id.length <= 2) {
            println("id: ${id.length}")
            val char = readInt(5000).toChar()
            when{
                char == KBD.NONE -> {
                    break
                }
                id.isNotEmpty() && char == '*' -> {
                    id = ""
                    LCD.cursor(1,0)
                    writeStr("ID:   ")
                    LCD.cursor(1,3)
                }
                char == '*' -> break
                else -> {
                    id += char
                    LCD.write(char)
                }
            }
        }
        return if (id.length == 3) id.toIntOrNull() else null
    }

    fun userPIN(): Int?{
        var pass = ""
```

```
    LCD.cursor(1,4)
    while (pass.length <= 3 ) {
        println("pass: ${pass.length}")
        val char = readInt(5000).toChar()
        when{
            char == KBD.NONE -> {
                break
            }
            pass.isNotEmpty() && char == '*' -> {
                pass = ""
                LCD.cursor(1,0)
                writeStr("ID:   ")
                LCD.cursor(1,3)
            }
            char == '*' -> break
            else -> {
                pass += char
                LCD.write("*")
            }
        }
    }
    return if (pass.length == 4) pass.toIntOrNull() else null
}

fun writeStr (txt: String) {
    if (txt.length >= 16) {
        for (i in 0 until 16) {
            LCD.write(txt[i])
        }
        for (i in 16 until txt.length) {
            LCD.write(txt[i])
        }
    } else
        LCD.write(txt)
}


fun writeLCD(text: String) {
```

```
        LCD.write(text)
    }

    fun init() {
        KBD.init()
        LCD.init()
        LCD.cursor(0, 0)
    }

    fun setCursor(line: Int, column: Int) {
        LCD.cursor(line, column)
    }

    fun clearLCD() {
        LCD.clear()
    }

}



fun main() {
    TUI.init()
}
```

H. Classe M

```
object M {

    val signal = 0x40

    fun manutencao(): Boolean =
            HAL.isBit(signal)

}
```

I. App

```
import isel.leic.utils.Time
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter


object App {

    const val USER_ID: Int = 123
    private const val USER_PASS = 4321
    var flag = false

    fun entry() {

        while (true) {
            TUI.clearLCD()
            TUI.setCursor(0, 0)
            val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")
            val current = LocalDateTime.now().format(formatter)
            TUI.writeLCD(current)
            TUI.setCursor(1, 0)

            val useriD = TUI.useriD()

            if (useriD == null) {
                manutencao()
                if (flag == true) {
                    break
                }
                continue
            }

            if (useriD != USER_ID) {
                TUI.setCursor(1,0)
                TUI.writeLCD("                ")
                TUI.setCursor(1,0)
                TUI.writeLCD("PIN:")
```

```
            Time.sleep(100)
            TUI.userPIN()
            LCD.clear()
            TUI.setCursor(0,2)
            TUI.writeLCD("Login Failed")
            Time.sleep(2500)
            continue
        } else {

            TUI.setCursor(1,0)
            TUI.writeLCD("                ")
            TUI.setCursor(1,0)
            TUI.writeLCD("PIN:")

            if (TUI.userPIN() != USER_PASS) {
                LCD.clear()
                TUI.setCursor(0,2)
                TUI.writeLCD("Login Failed")
                Time.sleep(2500)
                manutencao()
                if (flag == true) {
                    break
                }
                continue
            }

            open()
            close()

        }
    }
}


fun open() {
    LCD.clear()
    TUI.setCursor(0,3)
    TUI.writeLCD("Hello User")
```

```
        Time.sleep(250)
        LCD.clear()
        TUI.setCursor(0,6)
        TUI.writeLCD("User")
        TUI.setCursor(1,1)
        TUI.writeLCD("Opening Door..")
        Time.sleep(500)
        DoorMechanism.open(2)
        while (DoorMechanism.finished() != true) {}
        Time.sleep(500)
        TUI.setCursor(1,0)
        TUI.writeLCD("               ")
        TUI.setCursor(1,3)
        TUI.writeLCD("Door Open")
        Time.sleep(500)
    }


    fun close () {
        TUI.setCursor(1,2)
        TUI.writeLCD("Closing Door")
        Time.sleep(500)
        DoorMechanism.close(2)
        while (DoorMechanism.finished() != true) {}
        LCD.clear()
        TUI.setCursor(1,3)
        TUI.writeLCD("Door Close")
        Time.sleep(250)
    }

    fun manutencao() {
        if (M.manutencao()) {
            flag = true
            LCD.clear()
            TUI.setCursor(0,1)
            TUI.writeLCD("Out Of Service")
            TUI.setCursor(1,5)
            TUI.writeLCD("Wait")
```

```
            }
        }

}


fun main(){
    TUI.init()
    DoorMechanism.init()
    DoorMechanism.close(15)
    App.entry()
}
```