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*)

António Coelho 47236
Gonçalo Ribeiro 48305
Jorge Silva 49504

Projeto
de
Laboratório de Informática e Computadores
2022 / 2023 verão

17 de junho 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.

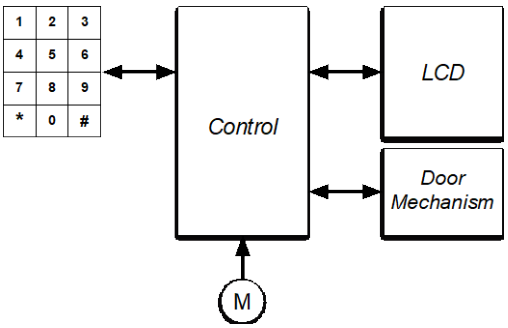

Figura 1 – Sistema de controlo de acessos (*Access Control System*)

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.

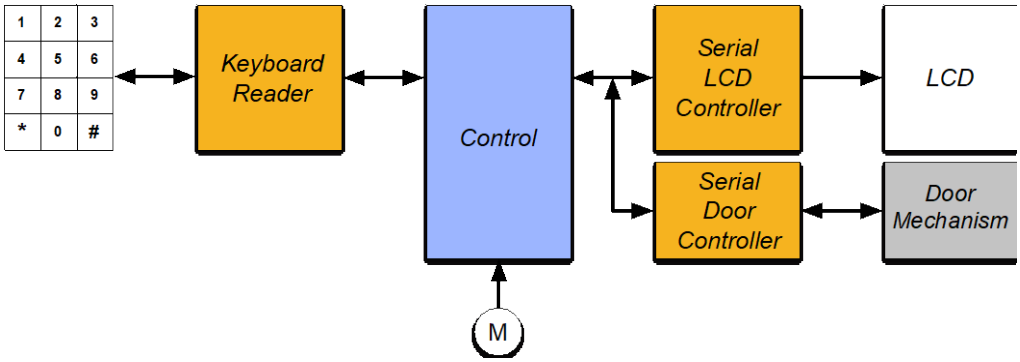

Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (*Access Control System*)

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.

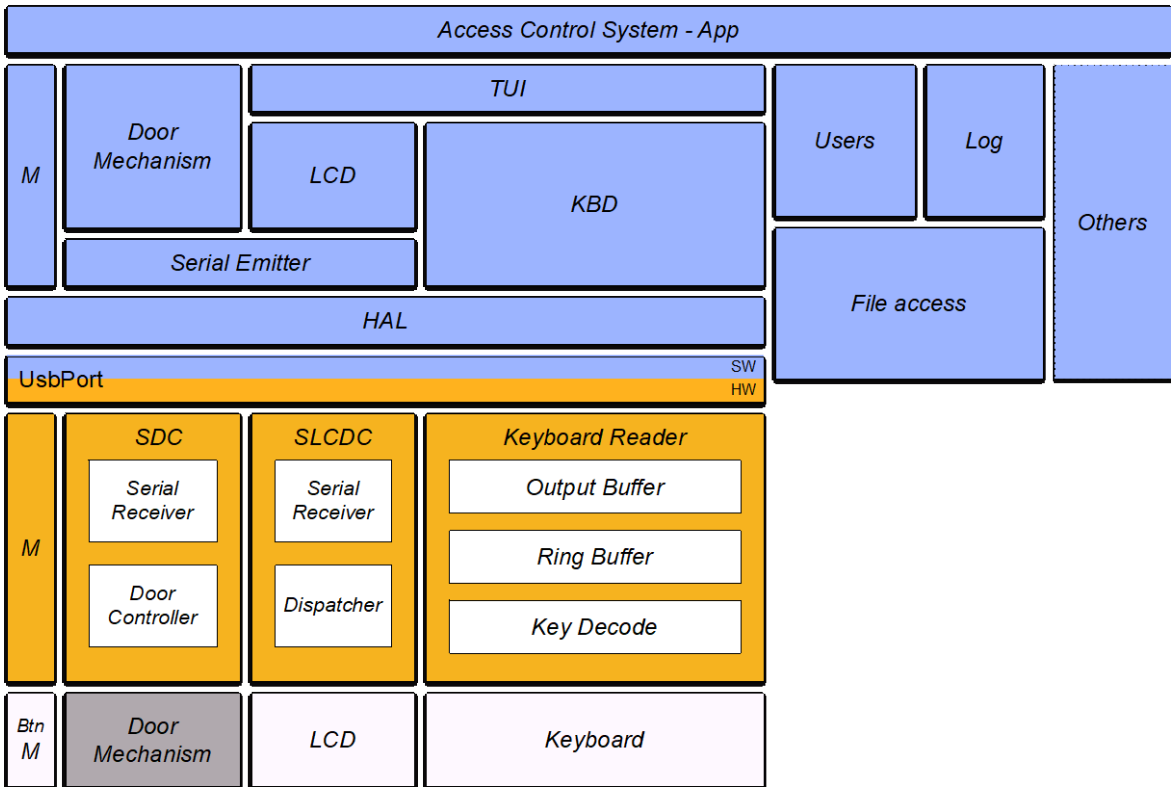

Figura 3 – Diagrama lógico do Sistema de Controlo de Acessos (*Access Control System*)

A. Interligações entre o HW e SW

Saída de dados do módulo Keyboard Reader
D0:3 - UsbPort.I[0-3]
Saída do sinal Dval do módulo Keyboard Reader
Dval - UsbPort.I4
Saída do sinal M
M - UsbPort.I6
Saída do sinal Busy do módulo Serial Door Controller
busy - UsbPort.I7

Entrada do sinal SDX do módulo Serial LCD Controller e do sinal SDX do módulo Serial Door Controller
UsbPort.O0
Entrada do sinal SCLK do módulo Serial LCD Controller e do sinal SCLK do módulo Serial Door Controller
UsbPort.O1
Entrada do sinal SS do módulo Serial Door Controller
UsbPort.O2
Entrada do sinal SS do módulo Serial LCD Controller
SS - UsbPort.O3
Entrada do sinal ACK do Bloco Output Buffer
ACK- UsbPort.O7

Mapeamento								
n	7	6	5	4	3	2	1	0
Usbport.In	busy	M	-	Dval	D(3)	D(2)	D(1)	D(0)
Usbport.On	ACK	-	-	-	LCD	Door	SCLK	SDX

busy - Saída do sinal Busy do módulo Serial Door Controller
M - Saída do sinal M
Dval - Saída do sinal Dval do módulo Keyboard Reader
D0:3 - Saída de dados do módulo Keyboard Reader
LCD - Entrada do sinal SS do módulo Serial LCD Controller
Door - Entrada do sinal SS do módulo Serial Door Controller
SCLK - Entrada do sinal SCLK do módulo Serial LCD Controller e do sinal SCLK do módulo Serial Door Controller
SDX - Entrada do sinal SDX do módulo Serial LCD Controller e do sinal SDX do módulo Serial Door Controller

B. Código *Kotlin - HAL*

```
object HAL {
          var written = 0b0000_0000
        private var ACTIVE = false
          fun init() { // Inicia a classe
              if(!ACTIVE) {
                  UsbPort.write(written)
                  ACTIVE=true
              }
          }

        // Retorna true se o bit tiver o valor lógico ‘1’
        fun isBit(mask: Int): Boolean = (mask and UsbPort.read()) != 0

        // 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
        /**
        * value -> 0000_1001.
        * mask -> 0000_1111.
        * lastWritten -> 1111_0111.
        * new lastWritten -> 1111_1001.
        * 1º: (value and mask) -> 0000_1001 sets the bits in value to be written to the ones in the
        mask.
        * 2º: (lastWritten and mask.inv()) -> 1111_0000 sets the bits in lastWritten that are not
        in the mask,this operation
        *      sets to 0 all the bits in lastWritten that are not in the mask, preparing it to
        receive the updated value.
        * 3º: (value and mask) or (lastWritten and mask.inv()) -> 0000_1001 or 1111_0000 ->
        1111_1001 sets the bits in
        *      lastWritten that are in the mask to the corresponding bits in value and keeps the
        bits that are not in the mask unchanged.
        */

        fun writeBits(mask: Int, value: Int) {
               written = (value and mask) or (written and mask.inv())
                 UsbPort.write(written)
        }

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

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

}
```

C. Código *Kotlin – KBD*

```
//
/**
 * Mapeamento das teclas do teclado matricial 4x4:
 *          Key  Column 1    Key Column 2   Key Column 3
 *Row 1     1    0x00        2   0x04       3   0x08
 *Row 2     4    0x01        5   0x05       6   0x09
 *Row 3     7    0x02        8   0x06       9   0x0A
 *Row 4     *    0x03        0   0x07       #   0x0B
 **/

//Read keys. Methods return '0'..' 9','#','*' or NONE.
object KBD {
    const val NONE = 0.toChar()
    const val DATA = 0x0F      //UsbPort.I0..3
    const val MASK_DVAL = 0x10 //UsbPort.I4
    const val MASK_ACK = 0x80 //UsbPort.O7
    private val KEYS : List<Char> = listOf('1', '4', '7', '*', '2', '5', '8', '0', '3', '6', '9',
'#')
                                        // 0    1    2    3    4    5    6    7    8    9    10
11
    // Starts the class.
    fun init() {
        HAL.init()
    }
     // Returns the pressed or NONE key immediately if there is no key pressed.
    fun getKey(): Char {
        if (HAL.isBit(MASK_DVAL)) {
            val a = HAL.readBits(DATA)
            return if (a in 0..11) {
                HAL.setBits(MASK_ACK)
                HAL.clrBits(MASK_ACK)
                KEYS[a]

            } else NONE
        }
         return NONE
    }
    // Returns when the key is pressed or NONE after millisecond timeout has elapsed.
    fun waitKey(timeout: Long): Char {
        var time = timeout
        while (time > 0) {
            val serial = getKey()
            if (serial != NONE) return serial
            Thread.sleep(1)
            time--
        }
        return NONE
    }

}
```

D. Código *Kotlin – SerialEmitter*

```
/**
 * Mapeamento
 *    n        7   6   5   4   3   2   1   0
 * inputPort :BSY   0   0   0   0   0   0   0      //inputPort(n)
 * outPort   : 0   0   0   0  LCD DOOR SCLK SDX //outputPort(n)
 *                            SS   SS
 * D[0:4] :    -   -   - D(3) D(2) D(1) D(0) RS
 **/

object SerialEmitter {    // Envia tramas para os diferentes módulos Serial Receiver
    enum class Destination {LCD, DOOR}
    private const val MASK_BUSY = 0x80
    private const val MASK_NOT_SS_LCD = 0x08
    private const val MASK_NOT_SS_DOOR = 0x04
    private const val MASK_SCLK = 0x02
    private const val MASK_SDX = 0x01
    private const val DATA_SIZE = 5

    // Inicia a classe
    fun init() {
         HAL.init()
         HAL.setBits(MASK_NOT_SS_LCD)  // SS = 1
         HAL.setBits(MASK_NOT_SS_DOOR)  // SS = 1
         HAL.clrBits(MASK_SCLK)    // SCLK = 0
         HAL.clrBits(MASK_SDX)     // SDX = 0
    }

    // Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados
em‘data’.
    fun send(addr: Destination, value: Int) {
        var data = value
        while (isBusy()) { }
        val address = if (addr == Destination.LCD) MASK_NOT_SS_LCD else MASK_NOT_SS_DOOR
        for (i in 0 until DATA_SIZE) {
            HAL.clrBits(MASK_SCLK)                // SCLK = 0
            HAL.clrBits(address)                  // SS = 0
            HAL.writeBits(0x01, 0x01 and  data)   // SDX = data[0]
            data = data shr 1                     // data = data >> 1 to send bit a bit
            HAL.setBits(MASK_SCLK)                // SCLK = 1
        }
        HAL.clrBits(0x01)                        // SDX = 0
        HAL.clrBits(MASK_SCLK)                   // SCLK = 0
        HAL.setBits(address)                     // SS = 1

    }

    // Retorna true se o canal série estiver ocupado
    fun isBusy(): Boolean = HAL.isBit(MASK_BUSY)

}
```

E. Código *Kotlin - LCD*

```
/**
 * LCD 16*2
 * Display positions
 *
 * 1º line: 0x00 to 0x0F
 * 2º line: 0x40 to 0x4F
 *
 * DDRAM : Display data RAM
 * CGRAM : Character generator RAM
 * ACG   : CGRAM address
 * ADD   : DDRAM address (cursor address)
 * AC    : address counter used for DD and CGRAM addresses
 * DDRAM 0 0 1 ADD ADD ADD ADD ADD ADD
 *
 */
object LCD {
    // Escreve no LCD usando a interface a 4bits
    private const val LINES = 2
    const val COLS = 16  // Dimensão do display.
    private const val DISPLAY_ON = 0xF //Mascara para ligar o display,
    private const val DISPLAY_OFF = 0x8 //Mascara para desligar o display
    private const val DISPLAY_SET = 0x3 //Function set
    private const val MASK_ENTRYMODE = 0x6 //Mascara para entry mode
    private const val DISPLAY_SET_NIBBLE = 0x2 //set 4 bits
    private const val MASK_PARALLEL_RS = 0x10 //Mascara para o bit de RS
    private const val MASK_LOW_DATA = 0x0F //Mascara para os 4 bits menos significativos
    private const val MASK_HIGH_DATA = 0xF0 //Mascara para os 4 bits mais significativos
    private const val DISPLAY_CLEAR : Int = 0x1 //Mascara para instrução de limpar o display
    private const val DISPLAY_CONFIG :Int=0x28//Mascara para configurar as linhas e a fonte do LCD


    // Escreve um nibble de comando/dados no LCD em paralelo
    //data -> d3..d0, recebe os quatro bits de menor peso
    private fun writeNibbleParallel(rs: Boolean, data: Int){
        if (rs) {
            HAL.setBits(MASK_PARALLEL_RS) //RS = 1
        } else {
            HAL.clrBits(MASK_PARALLEL_RS)//RS = 0
        }
        HAL.writeBits(MASK_LOW_DATA, data) //d3..d0
    }


    // Escreve um nibble de comando/dados no LCD em série
    private fun writeNibbleSerial(rs: Boolean, data: Int) {
        var d = data
        if (rs) d = (data shl 1) or 0x01 else d = (d shl 1) or 0x00
        SerialEmitter.send(SerialEmitter.Destination.LCD, d)
    }


    // Escreve um nibble de comando/dados no LCD
    private fun writeNibble(rs: Boolean, data: Int) {
        writeNibbleSerial(rs,data)
```

```
    }

    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int) {
        writeNibble(rs, (MASK_HIGH_DATA and data) shr 4)
        writeNibble(rs, MASK_LOW_DATA and data)
    }

    // Escreve um comando no LCD
    private fun writeCMD(data: Int) {
        writeByte(false, data)

    }
    // Escreve um dado no LCD
    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, DISPLAY_SET)
        Time.sleep(5)
        writeNibble(false, DISPLAY_SET)
        Time.sleep(1)
        writeNibble(false, DISPLAY_SET)
        writeNibble(false, DISPLAY_SET_NIBBLE)
        // Function Set, interface a 4 bits
        writeCMD(DISPLAY_CONFIG) // define N:1, F:0
        writeCMD(DISPLAY_OFF)  // display off
        writeCMD(DISPLAY_CLEAR)    // clear
        writeCMD(MASK_ENTRYMODE)  // define I/D:1, S:0
        writeCMD(DISPLAY_ON) // display on
    }

    // Escreve um caracter na posição corrente.
    fun write(c: Char) =
        writeDATA(c.code)

    // Escreve uma string na posição corrente.
    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(line: Int, column: Int) ...
    fun cursor(line: Int, column: Int): Unit {
         if (line >= LINES || line < 0 || column >= COLS || column < 0) return
        writeCMD((line * 0x40 + column) or 0x80)
    }
```

```
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear() {
        writeCMD(DISPLAY_CLEAR)
        cursor(0,0)
    }

}
```

F. Código *Kotlin* - *DoorMechanism*

```
/*
* Enviar pelo SerialEmmiter tal como no LCD
* Dados:     D4  D3  D2  D1 D0
*            V3  V2  V1  V0 OC
*
* OC -> 0 Fechar | 1 Abrir
* V3..0 -> Velocidade
* */


object DoorMechanism {    // Controla o estado do mecanismo de abertura da porta.

     // Inicia a classe, estabelecendoos valores iniciais.
    fun init() {
        SerialEmitter.init()
    }
    // Envia comando para abrir a porta, com o parâmetro de velocidade
    fun open(velocity: Int) {
        SerialEmitter.send(SerialEmitter.Destination.DOOR, (velocity shl 1) or 1)
//D4..1 -> Velocidade | D0 -> OC -> 1 Abrir
    }
    // Envia comando para fechar a porta, com o parâmetro de velocidade
    fun close(velocity: Int) {
        SerialEmitter.send(SerialEmitter.Destination.DOOR, (velocity shl 1 )or 0)
//D4..1 -> Velocidade | D0 -> OC -> 0 Fechar
    }
    // Verifica se o comando anterior está concluído
    fun finished(): Boolean = !SerialEmitter.isBusy()
}
```

G. Código *Kotlin - TUI*

```
/*
import java.text.SimpleDateFormat
import java.util.*


object TUI {
    private var pos = 4
    fun init() {
        LCD.init()
        KBD.init()
        pos = 4
    }
    //KBD
    fun getKey() = KBD.getKey()

    fun waitKey(timeout: Long) = KBD.waitKey(timeout)

    fun readIntNdigits(digits : Int, timeout: Long): Int {
        var number = ""
        var i = 0
        while (i < digits) {
            val key = waitKey(timeout)
            if (key == '*') {
                if (i == 0) {
                    pos = 4
                    return -1
                }
                for (i in 0 until digits){
                    writeOn(1, i+4, "?")
                }
                cursor(1,4)
                pos = 4
                i = 0
                number = ""
            } else
            if (key == KBD.NONE) {
                pos = 4
                return -1 //to allow users with UIN = 0
            }
            else {
                if (digits == 3) writeWhileWritingUIN(key)
                number += key
                i++
            }
        }
        pos = 4
        return number.toInt()
    }

   private fun writeWhileWritingUIN(number : Char) {
        writeOn(1,pos, number.toString())
        pos++
    }
    fun readInt(timeout: Long): Int {
            val key = waitKey(timeout)
            if (key != KBD.NONE) return key.code - 48
            else return -1
    }
    //LCD
    fun newLine() = LCD.cursor(1,0)
    fun clearScreen() = LCD.clear()
    fun cursor(line: Int, column: Int) = LCD.cursor(line, column)
```

```
    fun write(text: String) =  LCD.write(text)
    fun writeOn(line: Int, column: Int, text: String) {
        LCD.cursor(line, column)
        LCD.write(text)
    }
    fun clearLine(line: Int) {
        if (line == 0) cursor(0,0) else cursor(1,0)
        LCD.write("                ")
        cursor(line,0)
    }

    fun writeCentered(text: String) {
        if (text.length > LCD.COLS) {
            writeOn(0,0,text.substring(0, LCD.COLS))
            val t2 = text.substring(LCD.COLS)
            writeOn(1,(LCD.COLS - t2.length) / 2, t2)
        }
        else {
            writeOn(0,(LCD.COLS - text.length) / 2, text)
        }
    }


    fun writeRight(text: String) {
        val spaces = (LCD.COLS - text.length)
        if (text.length > LCD.COLS) {
            writeOn(0,0,text.substring(0, LCD.COLS))
            val t2 = text.substring(LCD.COLS)
            writeOn(1, (LCD.COLS - t2.length), t2)
        }
        else {
            writeOn(0,spaces, text)
        }
    }

    fun writeLeft(text: String) {
        if (text.length > LCD.COLS) {
            writeOn(0,0, text.substring(0,LCD.COLS))
            writeOn(1,0, text.substring(LCD.COLS))
        }
        else {
            writeOn(0,0, text)
        }
    }

    //Apresenta a data e hora no LCD
    fun showDateTime() {
        val dateFormat = SimpleDateFormat("dd/MM/yyyy HH:mm")
        val date = dateFormat.format(Date())
        writeLeft(date)
    }



}
```

H. Código *Kotlin – FileAccess*

```
import java.io.*

class FileAccess(val file: String) {
    fun init() {
        // Nothing to do
    }


    private fun createFile(filename: String) {
        val file = File(filename)
        file.createNewFile()
    }

    fun clearFile(filename: String) {
        val file = File(filename)
        file.writeText("")
    }

    fun readFile(filename: String): List<String> {
        if (!File(filename).exists()) {
            throw FileNotFoundException("File $filename does not exist")
        }
        val buffer = BufferedReader(FileReader(filename))
        val lines = buildList<String> {
            buffer.forEachLine {
                if (it.isNotEmpty()) {
                    add(it)
                }
            }
        }
        buffer.close()
        return lines
    }

    fun writeFile(fileName: String, lines: List<String>) {
        if (!File(fileName).exists()) {
            createFile(fileName)
        }
        val buffer = BufferedWriter(FileWriter(fileName))
        lines.forEach {
            buffer.write(it)
            buffer.newLine()
        }
        buffer.close()
    }



}
```

I. Código *Kotlin – Users*

```
object Users {
    private const val FILE_NAME = "USERS.txt"
    private const val MAX_USERS = 1000
    private val users : MutableList<User> = mutableListOf()
    private val file = FileAccess(FILE_NAME)

    fun init() {
        val lines : List<String> = file.readFile(FILE_NAME)
        getUsersFromFile(lines)
        users.sortBy { it.uin }
        println("Users: $users")
    }

    private fun getAvailableUIN(): Int {
        var uin = 0
       for (u in users)
            if (u.uin == uin)
            {
                uin++
            } else  break
        return uin
    }

    private fun getUsersFromFile(lines : List<String>) =
        users.addAll(lines.map { User(it) })

    fun writeUsers() {
        users.sortBy { it.uin }
        file.clearFile(FILE_NAME)
        file.writeFile(FILE_NAME, users.map { it.toString() })
    }

    fun findUser(uin: Int): User? = users.firstOrNull { it.uin == uin }
    fun removeUser(uin: Int) = users.removeIf { it.uin == uin }
```

```
    fun addUser(pin: Int, name: String, message: String):Boolean {
        if (users.size >= MAX_USERS) throw Exception("Maximum number of users reached")
        val user = User(getAvailableUIN(), pin, name, message)
        users.add(user)
        users.sortBy { it.uin }
        return true
    }


    override fun toString(): String =
        users.joinToString("\n")
}

class User(var uin: Int = -1,  var pin: Int = -1, var name: String = "", var message: String = "") {

    constructor(line: String) : this() {
        val parts = line.split(";")
        uin = parts[0].toInt()
        pin = parts[1].toInt()
        name = parts[2]
        message = parts[3]
    }


    fun checkPin(pin: Int): Boolean {
        return this.pin == pin
    }

    fun changePin(pin: Int) {
        this.pin = pin
    }
    override fun toString(): String {
        return if (!message.isNullOrEmpty())"$uin;$pin;$name;$message;" else "$uin;$pin;$name;"
    }

    fun deleteMessage() {
        this.message = ""
    }
```

J. Código *Kotlin - Log*

```
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter

object Log {
    private const val LOG_FILE ="Log File.txt"
    private const val FORMAT = "dd/MM/yyyy;HH:mm:ss"
    private val FileAccess = FileAccess(LOG_FILE)
    fun init() {

    }

    fun writeLog(uin: String) {
        val lines:List<String> = FileAccess.readFile(LOG_FILE)
        val line: String = "${LocalDateTime.now().format(DateTimeFormatter.ofPattern(FORMAT))};$uin"
        FileAccess.writeFile(LOG_FILE, lines.plus(line))
    }
}
```

L. Código *Kotlin* da classe *M*

```
import Users.findUser

object M {
    private const val SWITCH = 0x40   //Usbport
    private var Off = false
    fun init () {
        HAL.init()
        LCD.init()
        KBD.init()
    }

    fun isManteinanceMode() = HAL.isBit(SWITCH)

   private fun menu() {
        println("1. Inserir utilizador")
        println("2. Remover utilizador")
        println("3. Inserir mensagem")
        println("4. Desligar")
        println("5. Sair")
    }

    private fun insertUser() {
        var name :String = ""
        do {
            println("Insira o nome do utilizador")
            name = readln()
            if (name.length > 16) println("Insira um nome com um máximo de 16 caracteres")
        } while (name.length > 16)
        println("Insira o pin do utilizador")
        val pin = readln().toInt()
        if(Users.addUser(pin, name, "")){
            println("Utilizador inserido")
        } else {
            println("Utilizador já existe")
        }
    }

   private  fun removeUser() {
        println("Insira o UIN do utilizador")
        val uin = readln().toInt()
        val user = Users.findUser(uin)
       if (user == null) {
           println("Utilizador não encontrado")
           return
       }
       println("Nome do utilizador: ${user.name}")
       println("Confirme a remoção (S/N)")
       val conf = readln().first().uppercaseChar()
       if (conf == 'S'){
        if(Users.removeUser(uin)) {
            println("Utilizador removido")
        } else {
            println("Utilizador não removido")
        }
       }
       else {
           println("Utilizador não removido")
       }
    }

   private fun insertMessage() {
        println("Insira o UIN do utilizador")
```

```
        val uin = readln().toInt()
        println("Insira a mensagem")
        val message = readln()
        val user = findUser(uin)
        if (user != null) {
            user.message = message //TODO()
        } else {
            println("Utilizador não encontrado")
        }
    }

    private fun shutdown() {
        println("Desligar")
        Off = true
    }

   private fun exit() {
        println("Exiting")
        Off = false
    }

    fun action(): Boolean {
        while (isManteinanceMode() && !Off) {
            println("Modo de manutenção")
            menu()

            while (true) {
                val a = readln()
                if (a.isBlank()) break
                when (a.first()) {
                    '1' -> insertUser()
                    '2' -> removeUser()
                    '3' -> insertMessage()
                    '4' -> shutdown()
                    '5' -> exit()
                    else -> println("Opção inválida")
                }
                Thread.sleep(100)
                break
            }
        }
        return Off
    }
}
```

M. Código *Kotlin – Access Control System - App*

```
import isel.leic.utils.Time
import kotlin.system.exitProcess

object App {
    private const val UIN_LEN = 3
    private const val PIN_LEN = 4
    private const val TIMEOUT :Long = 5000
    private const val DOOR_VELOCITY = 0x08

    fun init() {
        TUI.init()
        DoorMechanism.init()
        Users.init()
        Log.init()
    }


    private fun getUIN() = TUI.readIntNdigits(UIN_LEN, TIMEOUT)
    private fun getPIN() = TUI.readIntNdigits(PIN_LEN, TIMEOUT)


   private fun doorControl() {
        TUI.clearScreen()
        TUI.writeCentered("Opening door")
        DoorMechanism.open(DOOR_VELOCITY)
        while (!DoorMechanism.finished()){}
        TUI.clearScreen()
        TUI.writeCentered("Door opened")
        Thread.sleep(5000)
        TUI.clearScreen()
        TUI.writeCentered("Closing door")
        DoorMechanism.close(DOOR_VELOCITY)
        while (!DoorMechanism.finished()){}
       TUI.clearScreen()
       TUI.writeCentered("Door closed")
       Time.sleep(500)
    }

    fun loginErrorUIN() {
        TUI.clearLine(1)
        TUI.writeOn(1, 0, "USER NOT FOUND")
        Thread.sleep(2000)
    }

    fun loginErrorPIN() {
        TUI.clearLine(1)
        TUI.writeOn(1, 0, "WRONG PIN")
        Thread.sleep(2000)
    }

    fun initialScreen() {
        TUI.clearScreen()
        TUI.showDateTime()
        TUI.newLine()
        TUI.write("UIN:???")
        TUI.cursor(1, 4)
    }

    fun pinScreen() {
        TUI.clearLine(1)
        TUI.writeOn(1, 0, "PIN:????")
```

```
    TUI.cursor(1, 4)
}

fun welcomeMessage(user: String) {
    TUI.clearScreen()
    TUI.writeOn(0,(LCD.COLS - "Welcome".length) / 2,"Welcome")
    TUI.writeOn(1, (LCD.COLS - user.length) / 2,user)
    Thread.sleep(1500)
    TUI.clearScreen()

}

fun showMessage(message: String) {
    TUI.clearScreen()
    TUI.writeLeft("Message: $message")
    Thread.sleep(2000)
}


fun changePin(user: User) {
    TUI.clearScreen()
    TUI.writeLeft("Insert new PIN")
    val newPin = TUI.readIntNdigits(PIN_LEN, TIMEOUT)
    TUI.clearScreen()
    TUI.writeLeft("Confirm new PIN")
    val confirmPin = TUI.readIntNdigits(PIN_LEN, TIMEOUT)
    if (newPin != confirmPin) {
        TUI.clearScreen()
        TUI.writeLeft("PINs do not match")
        Thread.sleep(2000)
        return
    }
    TUI.clearScreen()
    TUI.writeLeft("Changing PIN")
    user.changePin(newPin)
    TUI.clearScreen()
    Thread.sleep(2000)
    TUI.writeLeft("PIN changed")
    Thread.sleep(500)
}



fun autentichate(): Boolean {
    val uin = getUIN()
    if (uin == -1) return false
    println(uin)
    Time.sleep(1000)
    val user = Users.findUser(uin)
    println(user)
    if (user == null) {
        loginErrorUIN()
        return false
    }
    pinScreen()
    val pin = getPIN()
    if (pin == -1) return false
    if (!user.checkPin(pin)) {
        loginErrorPIN()
        return false
    }
    Log.writeLog(uin.toString())
    logged(user)
    return false
    }
```

```
    fun isToChangePin() = TUI.waitKey(TIMEOUT) == '#'

    fun isToDeleteMessage() = TUI.waitKey(TIMEOUT) == '*'
    fun logged(user: User) {
        while (true){
             user.name?.let { welcomeMessage(it) }
            if (isToChangePin()){
                changePin(user)
                break
            }
            else break
        }
        Time.sleep(1000)
        val message  = user.message
        println(message)
        if (!message.isNullOrEmpty()) {
            showMessage(message)
            if (isToDeleteMessage()) {
                user.deleteMessage()
                TUI.clearScreen()
                TUI.writeCentered("Message deleted")
            }
                TUI.clearScreen()
                Time.sleep(2000)
        }
        doorControl()
    }

    fun saveUsers() =  Users.writeUsers()

    fun outOfService() {
        TUI.clearScreen()
        TUI.writeCentered("Out of service")
    }

    fun isManteinanceMode() = M.isManteinanceMode()
    fun shutdown() {
        TUI.clearScreen()
        TUI.writeCentered("Shutdown")
    }


}



fun main() {
    App.init()
    var auth = false
    var active = true
    while(active) {
        App.initialScreen()
        while (true) {
            auth = App.autentichate()
            App.saveUsers()
            while (App.isManteinanceMode()) {
                App.outOfService()
                active = !M.action()
                App.saveUsers()
                println("End of manteinance mode")
                break
            }
            if (!auth) break
        }
```

```
    }
    App.shutdown()
    App.saveUsers()
    println("Shutdown")
    exitProcess(0)
}
```