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*)

Projeto
de
Laboratório de Informática e Computadores
2022 / 2023 verão

04 de abril 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.

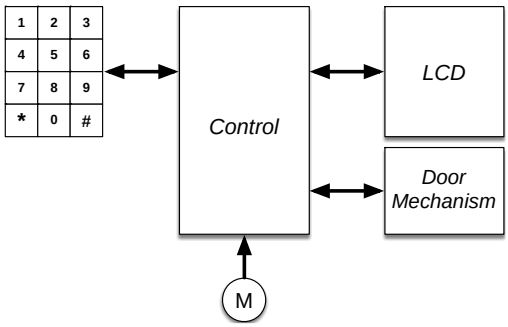

Figura 1 – Sistema de controlo de acessos (*Access Control System*)

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.
- **Informação do modo Manutenção -** Tem como objetivo fornecer ao utilizador todas as funcionalidades do modo Manutenção.
- **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* (S*DC*); 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.

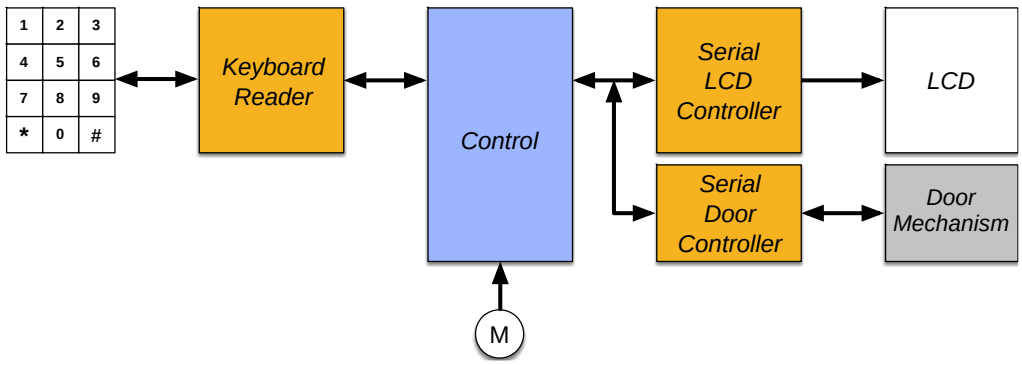

Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (*Access Control System*)

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.

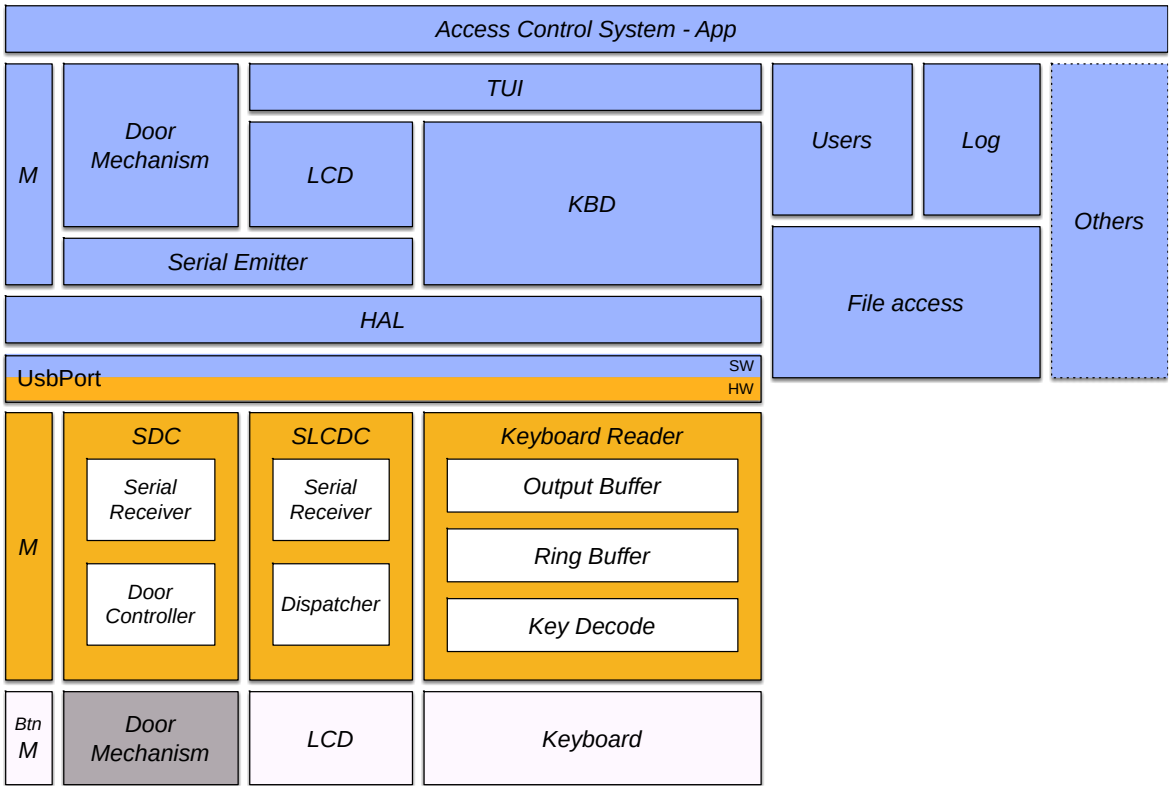

Figura 3 – Diagrama lógico do Sistema de Controlo de Acessos (*Access Control System*)

3 Implementação

A ligação entre o hardware e o software é feita através de uma unidade denominada por USBport que na sua construção inclui 8 bits de entrada e 8 bits de saída, a forma como a comunicação acontece é que o software consulta os bits no input port e é feita uma leitura e interpretação por parte do software, o mesmo acontece quando o software pretende enviar informação para o hardware mas neste caso a informação vem no output port e posteriormente é lida pelo hardware.

O sistema de controlo de acesso – App, utiliza um ficheiro USERS.txt para controlar os utilizadores registados, nesse ficheiro de texto contém todas as informações dos utilizadores (ID de utilizador, nome de utilizador, PIN codificado e uma mensagem de login) e guarda esse conteúdo em memória para que seja facilmente acedido e manipulado. No fim de alguma alteração no conteúdo em memória, o mesmo é escrito de volta no ficheiro de texto.

A. Interligações entre o HW e SW

```
 ## --- Additional packages with Hardware to Simulator
package accessctrl.simul    # for Modules used in Access Control System
UsbPort = UsbPort

# Generic modules to activate from Simulator
kbd = Keyboard("123456789*0#",4,3,0)
lcd = LCD
door = DoorMechanism
m=Switch ; "manut" ; setLabel("M")

# Costume modules from accessctrl package
rb  = RingBuffer
ob  = OutputBuffer
srl = SerialReceiver(5)
dl  = LCD_Dispatcher
srd = SerialReceiver(5)
dc  = DoorController

## --- Project Links ---
# -------------------------------------
# Manut
# -------------------------------------
m.out -> UsbPort.I7

# -------------------------------------
# Keyboard Reader
# -------------------------------------
# Key Decode
1 -> kbd.oe
kbd.val -> rb.DAV
kbd.K[0-3] -> rb.D[0-3]

# Ring Buffer
rb.DAC -> kbd.ack
rb.Q[0-3] -> ob.D[0-3]
rb.Wreg -> ob.Load

# Output Buffer
ob.OBfree -> rb.CTS
ob.Q[0-3] -> UsbPort.I[0-3]
ob.Dval -> UsbPort.I4
UsbPort.O7 -> ob.ACK

# -------------------------------------
# SLCDC
# -------------------------------------
# Serial Receiver
UsbPort.O0 -> srl./SS
UsbPort.O3 -> srl.SDX
UsbPort.O4-> srl.SCLK
srl.DXval -> dl.Dval
srl.D[0-4] -> dl.I[0-4]

# LCD Dispatcher
dl.D[1-4] -> lcd.D[4-7]
dl.D0 -> lcd.rs
dl.WrL -> lcd.e
dl.done -> srl.accept
```

```
# -------------------------------------
# SDC
# -------------------------------------
# Serial Receiver
UsbPort.O1 -> srd./SS
UsbPort.O3 -> srd.SDX
UsbPort.O4-> srd.SCLK
srd.busy -> UsbPort.I6
srd.DXval -> dc.Dval
srd.D[0-4] -> dc.I[0-4]

# Door Controller
dc.V[0-3] -> door.V[0-3]
dc.On -> door.On/Off
dc.Open -> door.Open/Close
dc.done -> srd.accept
door.FCclose -> dc.Sclose
door.FCopen -> dc.Sopen
door.Pdetect -> dc.Psensor
```

B. Código *Kotlin - HAL*

```kotlin
object HAL { // Virtualiza o acesso ao sistema UsbPort
    // Inicia a classe
    var lastoutput=0x00
    fun init(){
        UsbPort.write(lastoutput)
    }
    // Retorna true se o bit tiver o valor lógico '1'
    fun isBit(mask: Int): Boolean {
        return (UsbPort.read() and mask) != 0
    }
    // Retorna os valores dos bits representados por mask presentes no UsbPort
    fun readBits(mask: Int): Int {
        return (UsbPort.read() and mask)
    }
    // Escreve nos bits representados por mask o valor de value
    fun writeBits(mask: Int, value: Int){
        val newvalue=(lastoutput and mask.inv())or(mask and value)
        UsbPort.write(newvalue)
        lastoutput=newvalue
    }
    // Coloca os bits representados por mask no valor lógico '1'
    fun setBits(mask: Int){
        writeBits(mask,mask)
    }
    // Coloca os bits representados por mask no valor lógico '0'
    fun clrBits(mask: Int){
        writeBits(mask,0)
    }
}
```

C. Código *Kotlin - KBD*

```
object KBD { // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
    const val ACK=0x80
    const val DVAL=0x10
    const val CODE=0x0F
    const val NONE = 0.toChar()
    private val DIGITS = arrayOf('1','4','7','*','2','5','8','0','3','6','9','#')

    // Inicia a classe
    fun init() {
        HAL.clrBits(ACK)//Coloca o Kack a '0'

    }
    // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {
        var digit = NONE
        if(HAL.isBit(DVAL)){
            if(HAL.readBits(CODE)<12) {
                digit = DIGITS[HAL.readBits(CODE)]
            }
            HAL.setBits(ACK)
            while(HAL.isBit(DVAL));
            HAL.clrBits(ACK)
        }
        return digit
    }

// Retorna a tecla premida, caso ocorra antes do 'timeout' (representado em
milissegundos), ou NONE caso contrário.

    fun waitKey(timeout: Long): Char {
        val timelimit=Time.getTimeInMillis()+timeout
        var key=NONE
        while(Time.getTimeInMillis()<timelimit){
            if(HAL.isBit(DVAL)) {
                key = getKey()
            }
            if(key!=NONE){
                break
            }
        }
        return key
    }
}
fun main() {
    HAL.init()
    KBD.init()
    while (true) {
        val key = KBD.getKey()
        if(key!=KBD.NONE) println(key)
    }
}
```

D. Código *Kotlin – SerialEmitter*

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

    private const val SDX = 0x08
    private const val SCLK = 0x10
    private const val LCDSELECT = 0x01
    private const val DOORSELECT = 0x02
    private const val BUSY = 0x40

    // Inicia a classe
    fun init() {
        HAL.setBits(LCDSELECT)
        HAL.setBits(DOORSELECT)
        HAL.clrBits(SCLK)
    }

    // 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 sdx=data
        var ss = 0
        if (addr == Destination.LCD) {
            ss = LCDSELECT
        } else if (addr == Destination.DOOR) {
            ss = DOORSELECT
        }
        HAL.clrBits(ss)
        repeat(5) {
            Time.sleep(1)
            val lastBit=sdx%2
            if(lastBit==1) HAL.setBits(SDX)
            else HAL.clrBits(SDX)
            sdx/=2
            HAL.setBits(SCLK)
            Time.sleep(1)
            HAL.clrBits(SCLK)
        }
        HAL.clrBits(SCLK)
        HAL.setBits(ss)
    }

    // Retorna true se o canal série estiver ocupado
    fun isBusy(): Boolean {
        return HAL.isBit(BUSY)
    }
}
```

E. Código *Kotlin - LCD*

```
object LCD { // Escreve no LCD usando a interface a 4 bits.
    enum class WriteType {PARALLEL, SERIAL}
    private var writeType=WriteType.SERIAL
    private var rsInt = 0
    private const val LINE_VALUE = 0x40
    private const val COL_VALUE = 1
    private const val LCD_D = 0x0F
    private const val LCD_RS = 0x10
    private const val LCD_E = 0x20
    private const val LCD_HIGH = 0xF0
    private const val LCD_LOW = 0x0F
    //Instructions
    private const val RETURN_HOME = 0x02
    private const val DISPLAY_OFF = 0x00
    private const val DISPLAY_ON = 0x0F
    private const val CLEAR_DISPLAY = 0x01
    private const val ENTRY_MODE_SET = 0x06
    private const val SET_DDRAM_ADRESS = 0x80
    const val LCD_SIZE=16

    // Escreve um nibble de comando/dados no LCD em paralelo
    private fun writeNibbleParallel(rs: Boolean, data: Int){
        if(!rs) HAL.clrBits(LCD_RS)
        else HAL.setBits(LCD_RS)
        Time.sleep(1)
        HAL.writeBits(LCD_D,data)
        HAL.setBits(LCD_E)
        Time.sleep(1)
        HAL.clrBits(LCD_E)
        Time.sleep(1)
    }
    // Escreve um nibble de comando/dados no LCD em série
    private fun writeNibbleSerial(rs: Boolean, data: Int){
        rsInt = 0
        if(rs) rsInt=1
        var sdata = data shl 1
        sdata += rsInt
        SerialEmitter.send(SerialEmitter.Destination.LCD,sdata)
    }
    // Escreve um nibble de comando/dados no LCD
    private fun writeNibble(rs: Boolean, data: Int){
        if(writeType==WriteType.PARALLEL) writeNibbleParallel(rs,data)
        else if(writeType==WriteType.SERIAL) writeNibbleSerial(rs,data)
    }
    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int){
        writeNibble(rs,data shr 4)
        writeNibble(rs,data and LCD_LOW)
    }
    // 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(){
```

```
        Time.sleep(15)
        writeNibble(false,0x03)
        Time.sleep(5)
        writeNibble(false,0x03)
        Time.sleep(1/10)
        writeNibble(false,0x03)
        writeNibble(false,RETURN_HOME)
        writeCMD(0x02)
        writeCMD(0x08)
        writeCMD(DISPLAY_OFF)
        writeCMD(0x08)
        writeCMD(DISPLAY_OFF)
        writeCMD(CLEAR_DISPLAY)
        writeCMD(DISPLAY_OFF)
        writeCMD(ENTRY_MODE_SET)
    }
    // Escreve um caráter na posição corrente.
    fun write(c: Char){
        writeDATA(c.code)
        writeCMD(DISPLAY_ON)
    }

    // 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){
        val cursorMove = SET_DDRAM_ADRESS + line*LINE_VALUE + column*COL_VALUE
        writeCMD(cursorMove)
    }
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear(){
        writeCMD(CLEAR_DISPLAY)
    }
}
```

F. Código *Kotlin* - *DoorMechanism*

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

    // Envia comando para abrir a porta, com o parâmetro de velocidade
    private fun open(velocity: Int){
        var data = velocity.toString(2)
        data +=1 //adiciona o bit 1
        SerialEmitter.send(SerialEmitter.Destination.DOOR,data.toInt())
    }
    // Envia comando para fechar a porta, com o parâmetro de velocidade
    private fun close(velocity: Int){
        var data = velocity.toString(2)
        data +=0 //adiciona o bit 0
        SerialEmitter.send(SerialEmitter.Destination.DOOR,data.toInt())
    }

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

    //Funcionamento de abertura e fecho da porta e as respetivas mensagens
    fun doorMechanism(){
        LCD.cursor(0,0)
        Users.currentUser?.let { TUI.centerMessageDisplay(it.name) }
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Opening...")
        open(2)
        while(!finished());
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Opened")
        Time.sleep(3000)
        close(2)
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Closing...")
        while(!finished());
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Closed")
        Time.sleep(1000)
    }
}
```

G. Código *Kotlin - TUI*

```
object TUI { //Faz a ligação entre o utilizador e o sistema

    val timeFormat: DateTimeFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss")
    val time: String = LocalDateTime.now().format(timeFormat)

    //Escreve a data no display
    fun init() {
        LCD.clear()
        LCD.write(time)
    }

    //Escreve no LCD a data e as horas, e pergunta o ID do utilizador, e retorna os
dígitos introduzidos.
    fun idTUI(id: MutableList<Char>): String {
        LCD.cursor(1, 0)
        LCD.write("UIN:???")
        var cursor = 4
        LCD.cursor(1, cursor)
        while (id.size < 3) {
            if (HAL.isBit(M.MANUT)) {
                M.maintenance()
                App.accessControlSystem()
            }
            var key: Char
            while (true) {
                key = KBD.getKey()
                if (key != KBD.NONE && key!= '#') break
            }
            if (key == '*') {
                if (id.isNotEmpty()) {
                    id.removeAt(id.size - 1)
                    cursor--
                }
                LCD.cursor(1, cursor)
                LCD.write('?')
            } else {
                id.add(key)
                LCD.cursor(1, cursor)
                LCD.write(key)
                cursor++
            }
        }
        return id.joinToString("")
    }

    //Pergunta ao utilizador o PIN e retorna o que for escrito. se não for escrito
nada em 6 segundos, escreve no LCD "Login Timeout".
    fun pinTUI(id: MutableList<Char>, pin: MutableList<Char>): String {
        LCD.cursor(1, 0)
        LCD.write("PIN:????")
        var cursor = 4
        LCD.cursor(1,cursor)
        while (pin.size < 4) {
            if (HAL.isBit(M.MANUT)) {
                M.maintenance()
                App.accessControlSystem()
            }
            val key = KBD.waitKey(6000)
```

```
            if (key == KBD.NONE && key!= '#') {
                LCD.cursor(1, 0)
                centerMessageDisplay("Login Timeout")
                Time.sleep(2000)
                App.accessControlSystem()
            }

            if (key == '*') {
                if (pin.isNotEmpty()) {
                    pin.removeAt(pin.size - 1)
                    cursor--
                    LCD.cursor(1,cursor)
                    LCD.write("?")
                }
                else App.accessControlSystem()
            }
            else {
                pin.add(key)
                LCD.cursor(1, cursor)
                LCD.write('*')
                cursor++
            }
        }
        return pin.joinToString("")
    }

    //Centra as mensagens no display
    fun centerMessageDisplay(message: String) {
        if (message.length > 16) {
            val listMessage = message.split(" ").toMutableList()
            val firstPart = listMessage.subList(0, listMessage.size /
2).joinToString(" ")
            val secondPart = listMessage.subList(listMessage.size / 2,
listMessage.size).joinToString(" ")
            val marginSize1 = (LCD.LCD_SIZE - firstPart.length) / 2
            val margin1 = " ".repeat(marginSize1)
            LCD.write(margin1 + firstPart + margin1)
            LCD.cursor(1, 0)
            val marginSize2 = (LCD.LCD_SIZE - secondPart.length) / 2
            val margin2 = " ".repeat(marginSize2)
            LCD.write(margin2 + secondPart + margin2)
        } else {
            val marginSize = (LCD.LCD_SIZE - message.length) / 2
            val margin = " ".repeat(marginSize)
            LCD.write(margin + message + margin)
        }
    }

    //Confirma com o utilizador se quer trocar o pin, e pede ao utilizador para
inserir o novo pin 2 vezes, se não forem iguais, cancela a troca de pin
    private fun newPinInterface(): String? {
        val pinFirst = mutableListOf<Char>()
        val pinSecond = mutableListOf<Char>()
        LCD.clear()
        LCD.cursor(0, 0)
        centerMessageDisplay("Change PIN?")
        LCD.cursor(1, 0)
        centerMessageDisplay("Yes=1  No=2")
        while (true) {
            if (KBD.getKey() == '2') return null
```

```
            if (KBD.getKey() == '1') break
        }
        LCD.clear()
        LCD.cursor(0, 0)
        centerMessageDisplay("Insert new:")
        LCD.cursor(1, 0)
        LCD.write("PIN:????")
        var cursor = 4
        LCD.cursor(1,cursor)
        while (pinFirst.size < 4) {
            var key: Char
            while (true) {
                key = KBD.getKey()
                if (key != KBD.NONE) break
            }
            if (key == '*') {
                if (pinFirst.isNotEmpty()) {
                    pinFirst.removeAt(pinFirst.size - 1)
                    cursor--
                    LCD.write("?")
                    LCD.cursor(1,cursor)
                }
                LCD.cursor(1, cursor)
                LCD.write('*')
            } else {
                pinFirst.add(key)
                LCD.cursor(1, cursor)
                cursor++
            }
        }
        LCD.clear()
        LCD.cursor(0, 0)
        centerMessageDisplay("Confirm PIN:")
        LCD.cursor(1, 0)
        LCD.write("PIN:****")
        cursor = 4
        while (pinSecond.size < 4) {
            var key: Char
            while (true) {
                key = KBD.getKey()
                if (key != KBD.NONE) break
            }
            if (key == '*') {
                if (pinSecond.isNotEmpty()) {
                    pinSecond.removeAt(pinSecond.size - 1)
                    cursor--
                }
                LCD.cursor(1, cursor)
                LCD.write('*')
            } else {
                pinSecond.add(key)
                LCD.cursor(1, cursor)
                cursor++
            }
        }
        LCD.clear()
        centerMessageDisplay("PIN")
        if (pinFirst == pinSecond) {
            LCD.cursor(1, 0)
            centerMessageDisplay("has been changed")
```

```
            return pinFirst.joinToString("")
        } else {
            LCD.cursor(1, 0)
            centerMessageDisplay("change failed")
            return null
        }
    }

    //Mostra as mensagens após o login e verifica se o utilizador quer trocar o pin
ou apagar a mensagem de login
    fun loginSuccessInterface() {
        LCD.clear()
        LCD.cursor(0, 0)
        centerMessageDisplay("Hello")
        LCD.cursor(1, 0)
        centerMessageDisplay(Users.currentUser!!.name)
        Time.sleep(1500)
        LCD.clear()
        LCD.cursor(0, 0)
        if (KBD.waitKey(2000) == '#') {
            val newPin = newPinInterface()
            if (newPin != null)
Users.changePin(Users.currentUser!!.id.toInt().toString(), newPin)
        }
        if (Users.currentUser!!.message.isNotEmpty()) {
            centerMessageDisplay(Users.currentUser!!.message)
            if (KBD.waitKey(2000) == '*') deleteMessage()
        }
            Log.addToLog(Users.currentUser!!)
            DoorMechanism.doorMechanism()
    }

    //Confirma e apaga a mensagem de login do utilizador
    private fun deleteMessage(){
        LCD.clear()
        centerMessageDisplay("Remove MSG?")
        LCD.cursor(1, 0)
        centerMessageDisplay("Yes=*")
        if (KBD.waitKey(2000) == '*') {
            Users.changeMessage(Users.currentUser!!.id, Users.currentUser!!.message,
Users.Operation.REMOVE)
            LCD.clear()
            centerMessageDisplay("Message has been changed")
        } else {
            LCD.clear()
            centerMessageDisplay("Message has been helded")
        }
    }
}
```

H. Código *Kotlin* - *FileAccess*

```kotlin
object FileAccess { //Acede aos ficheiros externos à aplicação

    private val br = BufferedReader(FileReader("USERS.txt"))
    private val fileWriterLog = FileWriter("LOG.txt", true)
    private val pwLog = PrintWriter(fileWriterLog)

    //Lê o ficheiro USERS.txt e guarda numa MutableList de Users.User as informações de cada utilizador
    fun readUsersFile(): MutableList<Users.User> {
        var line=br.readLine()
        val users = mutableListOf<Users.User>()
        while(line!=null) {
            val data=(line.split(Regex(";")))
            users.add(Users.User(data[0],data[1],data[2],data[3]))
            line=br.readLine()
        }
        return users
    }

    //Escreve no ficheiro USERS.txt as informações dos utilizadores da MutableList users
    fun writeUsersFile(users:MutableList<Users.User>){
        val fileWriterUsers = FileWriter("USERS.txt", false)
        val pwUsers= PrintWriter(fileWriterUsers)
        users.forEach {
            pwUsers.print("${it.id};${it.pin};${it.name};${it.message};")
            pwUsers.println()
        }
        pwUsers.close()
    }

    //Quando um utilizador faz login, escreve no ficheiro LOG.txt o utilizador que realizou o login e a data
    fun writeLogFile(user:Users.User){
        pwLog.print("Time: ${LocalDateTime.now().format(TUI.timeFormat)}  ")
        pwLog.print("ID: ${user.id} -> ")
        pwLog.print("Name: ${user.name}")
        pwLog.println()
        pwLog.close()
    }
}
```

I. Código *Kotlin - Users*

```
object Users {
    var currentUser: User? = null

    data class User(val id: String, var pin: String, val name: String, val message:
String)

    var users = FileAccess.readUsersFile()

    enum class Operation {ADD, REMOVE}

    //Procura na lista de Users se algum utilizador corresponde ao id e pin do
utilizador, se encontrar retorna a informaçao do utilizador, se não, retornal null
    fun findUser(id: String, password: String, users: List<User>): User? {
        val encodedPassword = passwordEncoder(password)
        for (user in users) {
            if (id.toInt().toString() == user.id && encodedPassword == user.pin) {
                return user
            }
        }
        return null
    }

    //Adiciona utilizadores ao sistema
    fun addUser(id: String, name: String, pin: String){
        User("","","","") //inicialização da variavel
        var i=0
        for(u in users){
            if(u.id==id) break
            i++
        }
        users.add(User(id, passwordEncoder(pin), name, ""))
    }

    //Remove utilizadores do sistema
    fun removeUser(id:String){
        var i=0
        for(u in users){
            if(u.id==id) break
            i++
        }
        users.removeAt(i)
    }

    //Adiciona ou remove mensagens dos utilizador
    fun changeMessage(id: String, message: String, operation: Operation) {
        var user=User("","","","") //inicialização da variavel
        var i=0
        for(u in users){
            if(u.id==id){
                user=u
                break
            }
            i++
        }
        user = if (operation == Operation.ADD) User(user.id, user.pin, user.name,
message = message)
        else User(user.id, user.pin, user.name, message = "")
        users[i]=user
    }
```

```
    //Troca o pin anterior pelo novo pin
    fun changePin(id: String,newPin:String){
        var i=0
        for(u in users){
            if(u.id==id)break
            i++
        }
        users[i].pin=newPin
    }

    //Codifica o pin para registar no ficheiro USERS.txt
    private fun passwordEncoder(password: String): String = password.reversed()

    //Atribui um novo id a um novo utilizador
    fun getNewId(users: List<User>): String {
        var newId = 0
        for (user in users) {
            if (newId == user.id.toInt()) newId++
        }
        return newId.toString()
    }

    //Verifica se esse utilizador existe, se nao existir retorna null
    fun checkExistingUser(id:String):User?{
        for(user in users){
            if(user.id==id) return user
        }
        return null
    }
}
```

J. Código *Kotlin - Log*

```kotlin
object Log{
    fun addToLog(user:Users.User){
        FileAccess.writeLogFile(user)
    }
}
```

L. Código *Kotlin* da classe *M*

```
object M{ //Modo de manutenção da aplicação
    const val MANUT=0x80

    //Pergunta ao utilizador o username e pin para o novo user
    private fun addUserM(){
        println("Username must be under 16 characters.")
        println("PIN must be 4 digits.")
        var name:String
        while(true){
            print("Username= ")
            name= readln()
            if(name=="EXIT"){
                println("Command aborted.")
                return
            }
            if(name.length<=16) break
            println("Username length exceeded.")
        }
        var pin:String
        while(true){
            print("PIN= ")
            pin= readln()
            println(pin)
            if(pin=="EXIT"){
                println("Command aborted.")
                return
            }
            if(pin.length==4) break
            println("Invalid PIN.")
        }
        val newId=Users.getNewId(Users.users)
        Users.addUser(newId,name,pin)
        println("Adding User=$name with UID=$newId")
        println()
    }

    //Pergunta ao utilizador o id do user a remover do sistema
    private fun removeUser(){
        print("UIN= ")
        val id= readln()
        if(id=="EXIT"){
            println("Command aborted.")
            return
        }
        else id.toInt().toString()
        while(true){
            if (Users.checkExistingUser(id)!=null) break
                println("User does not exist.")
        }
        println("Remove user=${Users.checkExistingUser(id)!!.name}")
        print("Y/N?")
        if(readln()=="Y"){
            println("User=${Users.checkExistingUser(id)!!.name} removed.")
            Users.removeUser(id)
            println()
        }
        else{
            println("Command aborted.")
            println()
```

```
        }
    }

    //Pergunta ao utilizador o id do user e a mensagem desejada a ser adicionada
    private fun addMsg(){
        print("UIN= ")
        val id= readln().toInt().toString()
        while(true){
            if (Users.checkExistingUser(id)!=null) break
            println("User does not exist.")
        }
        print("Message= ")
        val message= readln()
        Users.changeMessage(id,message,Users.Operation.ADD)
        println("The message $message has been associated to user=$id ")
        println()
    }

    //Encerra a aplicação
    private fun shutDown(){
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Shutting down...")
        Time.sleep(4000)
        FileAccess.writeUsersFile(Users.users)
        exitProcess(0)
    }

    //Fornece ao utilizador um guião de funcionalidades do modo de manutenção
    private fun helpText(){
        println("NEW: Adds new user;")
        println("DEL: Deletes an existing user;")
        println("MSG: Adds or changes the login message to an existing user;")
        println("EXIT: Aborts the current command;")
        println("OFF: Shutdown the system.")
        println()
    }

    //Avisa no LCD que a aplicação esta no modo de manutenção e lê as ações
desejadas do terminal
    fun maintenance(){
        LCD.clear()
        LCD.cursor(0,0)
        TUI.centerMessageDisplay("Out of Service")
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Please Wait")
        println("Turn M key to off, to terminate maintenance mode.")
        println("Commands: NEW, DEL, MSG, OFF, EXIT, or HELP")
        while (true) {
            if(!HAL.isBit(MANUT)) break
            print("Maintenance> ")
            when (readln()){
                "NEW" -> addUserM()
                "DEL" -> removeUser()
                "MSG" -> addMsg()
                "OFF" -> shutDown()
                "HELP" -> helpText()
                "EXIT" -> println("not currently in any command.")
                else  -> println("Invalid Command.")
            }
        }
```

```
            println("Exiting maintenance mode...")
        }
}
```

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

```
object App { //Sistema de controlo de acessos

    //Verifica se o login foi bem sucedido
    private fun loginSuccess(): Boolean = Users.currentUser != null

    //Sistema de controlo de acessos
    fun accessControlSystem() {
        while(true){
            val id= mutableListOf<Char>()
            val pin= mutableListOf<Char>()
            TUI.init()
            TUI.idTUI(id)
            TUI.pinTUI(id,pin)

            Users.currentUser = Users.findUser(id.joinToString(""),
pin.joinToString(""), Users.users)
            if(loginSuccess())TUI.loginSuccessInterface()
            else {
                LCD.cursor(1, 0)
                TUI.centerMessageDisplay("Login Failed")
                Time.sleep(2000)
            }
            LCD.clear()
            FileAccess.writeUsersFile(Users.users)
        }
    }
}
```