Licenciatura em Engenharia Informática e de Computadores

Sistema de Controlo de Acessos (Access Control System)

N. 50471 Brian Melhorado

N. 50543 Arthur Oliveira

N.50468 João Maria Gonçalves

Projeto de Laboratório de Informática e Computadores

Semestre de Verão 2022/2023

19/06/2023

ÍNDICE

1. Introdução do Projeto

Foi implementado 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.

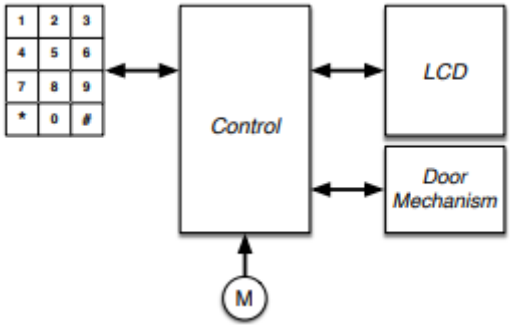

Figura 1 – Sistema de controlo de acessos (Access Control System)

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. Todos os acessos são 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 .

2. Arquitetura do Sistema

O controlo (designado por Control) do sistema de acessos foi 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) foram implementados em hardware e o módulo de controlo foi implementado em software executado num PC.

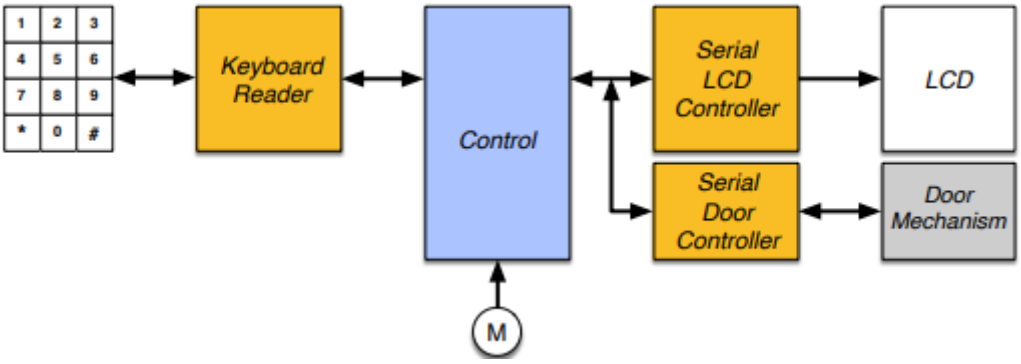

Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (Access Control System)

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.

2.1 Interligações entre o Hardware e o Software

USB Port	7	6	5	4	3	2	1	0
IN	M	busy		Dval	Q (Output Buffer)			
OUT	ACK			SCLK	SDX		SDC_E	LCD_E

3. Control

A implementação do módulo Control foi realizada em software, usando a linguagem Kotlin e seguindo a arquitetura lógica apresentada na Figura 3.

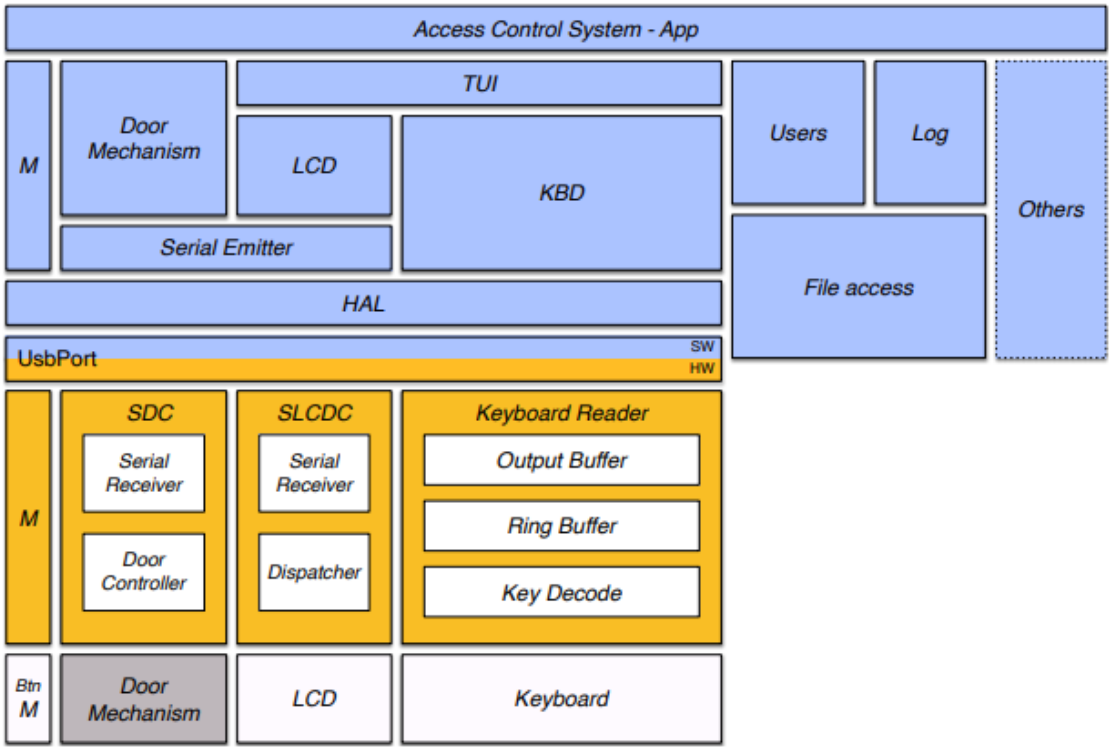

Figura 3 – Diagrama lógico do sistema de controlo de acessos (Access Control System)

3.1 Access Control System App

Faz a leitura do Log (ID e Ping do usuário) e a leitura do modo de manutenção.
Constantes:

- *DO_NOT_SPLIT: é utilizada quando não é pretendido escrever nas duas linhas do LCD.*
- *WAIT_TIME: é o valor do tempo de espera em milissegundos*
- *NONE: representa um valor inválido.*
- *PIN_SIZE: representa o tamanho do PIN.*
- *UIN_SIZE: representa o tamanho do UIN.*

Enum Class Code:

- *SUCCESS: instruçao realizada com sucesso*
- *FAILED: falha ao executar a instrução*
- *INVALID_PIN:*

Funcões:

- `private fun waitTime(wait: Int)` = Dado o parametro wait a função espera esse tempo ou avança caso seja lido uma tecla do teclado.
- `private fun changePassword(uId: Int, key: Char): Code` = A função realiza a leitura de uma senha caso o parâmetro key seja igual a '*' e entra realiza mais uma leitura para confirmação do pin digitado, caso passe nessa verificação o usuário com o uID passado no parâmetro vai ter sua senha alterada.
- `fun maintenance()` = Realiza a leitura do bit Maintenance e caso ele esteja a 1 o sistema é interrompido e então um terminal é aberto para a execução de diferentes instruções.
- `fun login(){`

3.2 Users

É o banco de dados do Access Control System que contem os usuários.

Enum Class Code:

- *SUCCESS:* [explicado no Access Control System App]
- *FAILED:* [explicado no Access Control System App]
- *INVALID_NAME*: Quando o nome do usuario tem mais de 16 caracteres.
- *VALID_NAME*: Quando o nome do usuario tem menos de 17 caracteres.
- *MAX_NUMBER_OFF_USERS_ACHIEVED*: Quando o Sistema possui 1000 usuarios registados.
- *INVALID_PIN:* [explicado no Access Control System App]

Constantes:

- *MAX_USERS: Número máximo de usuários permitidos pelo sistema*

Data Classes:

- `Log` = data class que retorna o Code resultante da instrução e a msg do user (caso a instrução tenha sucesso)
- `User` = data class que contem o UIN do user, nome, senha e mensagem.

Funcões:

- `private fun getUsers()` = Recebe a lista de usuários, senhas, uIn e mensagens.

- `private fun validate(user: String): Code` = Verifica se o nome do usuário tem menos de 17 letras.
- `fun remove(uID: Int): Code` = remove o usuario do sistema.
- `fun changePassword(uID: Int, newPass: Int)` = Dado um uId, é atualizado os dados desse user, trocando a senha.
- `fun add(user: String, pin: Int): Code` = Adiciona uma entrada na lista (Hashmap) com o novo utilizador e password.
- `fun login(uID: Int?, pin: Int?): Log` = verifica se o UID e PIN são validos, ou seja, se o user existe e se assim for, se a password é igual
- `fun removeMsg(char: Char, uID: Int?): Code` = Caso o char seja igual a '*' a mensagem associada ao uId é removida.
- `fun getUser(uID: Int)` = Retorna um usuário.
- `fun addMsg(uID: Int, msg: String): Code` = adiciona uma mensagem ao user de uId = uID
- `fun closure()` = Reescreve os users no ficheiro Users.

3.3 Logs

É o modulo responsável por registar quando é efetuado um login com sucesso.

Funcões:

- `fun addLog(uId: Int)` = escreva uma nova linha no ficheiro logs com o horário, data e UIN do usuário que realizou o login.
- `fun closure()` = encerra a escrita no ficheiro logs.

3.4 File Utils

Contém as funções responsáveis por criar os ficheiros de leitura e escrita.

Funcões:

- `fun createReader(fileName: String): BufferedReader` = cria um ficheiro de leitura.
- `fun createFileWriter(fileName: String): BufferedWriter` = cria um ficheiro que permite a escrita de elementos no formato apend.
- `fun createWriter(fileName: String): PrintWriter` = cria um ficheiro de escrita.

3.5 File Access

É o modulo responsável da gestão dos ficheiros de leitura e escrita.

Constantes:

- *USERS = Endereço do ficheiro Users*
- *LOGS = Endereço do ficheiro Logs*
- *DEL = Delimitador dos dados nos ficheiros*

Funcões:

- `fun init() =` Reescreve os logs já existentes no ficheiro logs.
- `fun add(data: List<String>) =` Rescreve os users no ficheiro Users.
- `fun getUsers(): List<String> =` Lê os users do ficheiro USERS e os coloca em uma lista.
- `fun logRegister(data: String) =` escreve data no ficheiro LOGS.
- `fun close() =` Encerra o ficheiro LOGS.

3.6 Manutenção

Verifica se a chave de manutenção se encontra ativa.

Constantes:

- *MAINTENANCE = Bit de Manutenção*

Funcões:

- `fun maintenanceMode(): Boolean =` Verifica se o MAINTENANCE se encontra diferente de 0.

4. Conclusão

Em conclusão, o sistema de controle de acessos implementado oferece uma solução eficiente para controlar o acesso a áreas restritas. A utilização de um número de identificação de usuário (UIN) e um código de acesso pessoal (PIN) garante a autenticação adequada dos usuários. Com um teclado, um display LCD, um mecanismo de abertura/fechamento da porta, uma chave de manutenção e um PC para controle, o sistema opera de forma integrada e confiável.

A. Código Kotlin da Access Control System App

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

const val DO_NOT_SPLIT = false
const val WAIT_TIME = 3000
object App{
    fun init(){
        HAL.init()
        KBD.init()
        SerialEmitter.init()
        LCD.init()
        DoorMechanism.init()
        TUI.init()
        FileAccess.init()
        Users.init()
        Maintenance.init()
        DoorMechanism.close(15)
    }
    private fun waitTime(wait: Int){
        val begTime = Time.getTimeInMillis()+wait
        while (begTime >= Time.getTimeInMillis()) {
            val key = TUI.getKey()
            if (key != NONE.toChar())
                break
        }
    }
    private fun changePassword(uId: Int, key: Char): Code{
        if (key == '#'){
            TUI.showMsg("Change PIN? (Yes=*)")
            val yesNo = TUI.waitKey()
            if (yesNo == '*'){
                val newPassword = TUI.waitInput(PIN_SIZE,
"New Password").toIntOrNull() ?: return Code.INVALID_PIN
                val confirmPassword =
TUI.waitInput(PIN_SIZE, "Confirm the PIN").toIntOrNull() ?:
return Code.INVALID_PIN
                if (newPassword == confirmPassword){
                    Users.changePassword(uId, newPassword)
                    TUI.showMsg("New PIN saved
successfully")
                    waitTime(WAIT_TIME/2)
                    return Code.SUCCESS
                }
            }
        }
```

```
        return Code.FAILED
    }
    fun maintenance(){
        var maintenanceMade = false
        if (Maintenance.maintenanceMode()) TUI.showMsg("Out
of Service")
        while (Maintenance.maintenanceMode()){
            maintenanceMade = true
            println("Select the operation mode:")
            val modeList = listOf("Add user", "Remove
user", "Add message", "Turn off")
            modeList.forEachIndexed { idx, s ->
                println("$idx -> $s")
            }
            when(readln().toIntOrNull()){
                0 -> {
                    print("Digit the user name: ")
                    val name = readln().trim()
                    print("Digit the pin: ")
                    var pin = readln().toIntOrNull()
                    while (pin == null){
                        print("Invalid pin. Try again: ")
                        pin = readln().toIntOrNull()
                    }
                    val code = Users.add(name, pin)
                    println(code)
                }
                1 -> {
                    print("Digit the DID: ")
                    var uId = readln().toIntOrNull()
                    while (uId == null){
                        print("Invalid UID. Try again: ")
                        uId = readln().toIntOrNull()
                    }
                    val user = Users.getUser(uId)
                    if (user == null)
                        println("UID not found")
                    else{
                        println("Do you want tho proceed?
")
                        var choice =
readln().trim().lowercase()
                        while (choice != "yes" && choice !=
"no") {
                            println("Invalid answer. Try
again")
                            choice =
readln().trim().lowercase()
                        }
```

```
                        if (choice == "yes")
                            Users.remove(uId)
                    }
                }
                2 -> {
                    print("Digit the UID: ")
                    var uId = readln().toIntOrNull()
                    while (uId == null) {
                        print("Invalid UID. Try again: ")
                        uId = readln().toIntOrNull()
                    }
                    print("Digit the message: ")
                    val msg = readln().trim()
                    val code = Users.addMsg(uId, msg)
                    println(code)
                }
                3 -> {
                    print("Do you want to close the
program? ")
                    var choice =
readln().trim().lowercase()
                    while (choice != "yes" && choice !=
"no") {
                        print("Invalid answer. Try again:
")
                        choice =
readln().trim().lowercase()
                    }
                    if (choice == "yes"){
                       Users.closure()
                       Logs.closure()
                       exitProcess(0)
                    }
                }
                else -> continue
            }
        }
        if (maintenanceMade) println("Exiting maintenance
mode...")
    }
    fun login(){
        val uID = TUI.waitInput(UIN_SIZE).toIntOrNull() ?:
return
        val pin = TUI.waitInput(PIN_SIZE).toIntOrNull()
        val log = Users.login(uID, pin)
        if (log.code == Code.SUCCESS){
            if (log.msg != null) TUI.showMsg(log.msg) else
TUI.emptyScreen()
            val begTime = Time.getTimeInMillis()+WAIT_TIME
```

```
            while (begTime >= Time.getTimeInMillis()) {
                val key = TUI.getKey()
                val passwordCode = changePassword(uID, key)
                val code = Users.removeMsg(key, uID)
                if (code == Code.SUCCESS) {
                    TUI.showMsg("Message removed",
DO_NOT_SPLIT)
                    waitTime(WAIT_TIME/2)
                }
                if (code == Code.SUCCESS || key !=
NONE.toChar() || passwordCode == Code.SUCCESS ||
passwordCode == Code.INVALID_PIN)
                    break
            }
            Logs.addLog(uID)
            DoorMechanism.open(8)
            TUI.showMsg("Opening Door...", DO_NOT_SPLIT)
            while (!DoorMechanism.finished());
            TUI.showMsg("Door opened", DO_NOT_SPLIT)
            waitTime(WAIT_TIME/2)
            DoorMechanism.close(4)
            TUI.showMsg("Closing Door...", DO_NOT_SPLIT)
            while (!DoorMechanism.finished());
            TUI.showMsg("Door closed", DO_NOT_SPLIT)
            waitTime(WAIT_TIME/2)
        }
        else {
            TUI.showMsg("Login Failed", DO_NOT_SPLIT)
            waitTime(WAIT_TIME)
        }
    }
}
fun main(){
    App.init()
    while (true){
        App.maintenance()
        App.login()
    }
}
```

B. Código Kotlin dos Users

```
// Error Codes
enum class Code {
                SUCCESS,
                FAILED,
                INVALID_NAME,
                VALID_NAME,
                MAX_NUMBER_OFF_USERS_ACHIEVED,
                INVALID_PIN
}

private const val MAX_USERS = 1000

object Users {

    private var list = hashMapOf<Int, User>()

    fun init() {
        getUsers()
    }

    data class Log(
        val code: Code,
        val msg: String?
    )
    data class User(
        var uID: Int,
        val uName: String,
        var pwd: Int,
        var desc: String?
    )

    private fun getUsers() {
        val data = FileAccess.getUsers()
        for (i in data) {
            val x = i.split(DEL)
            val msg = if (x[3] == "null") null else x[3]
            val user = User(x[0].toInt(), x[1],
x[2].toInt(), msg)
            list[user.uID] = user
        }
    }

    private fun validate(user: String): Code = if
(user.length <= 16) Code.VALID_NAME else Code.INVALID_NAME
```

```
    fun remove(uID: Int): Code{
        list[uID] ?: return Code.FAILED
        list.remove(uID)
        return Code.SUCCESS
    }
    fun changePassword(uID: Int, newPass: Int) {
        list[uID]?.pwd = newPass
    }
    fun add(user: String, pin: Int): Code {
        if (validate(user) == Code.INVALID_NAME) return
Code.INVALID_NAME
        if (list.size < MAX_USERS) {
            val newUser = User(0, user, pin, "Hello $user")
            for (uID in 0 until MAX_USERS) {
                newUser.uID = uID
                val validation = list.putIfAbsent(uID,
newUser)
                val uid = when {
                    uID < 10 -> "00$uID"
                    uID < 100 -> "0$uID"
                    else -> uID
                }
                if (validation == null) {
                    println("New user added successfully,
with the UID = $uid.")
                    return Code.SUCCESS
                }
            }
            return Code.MAX_NUMBER_OFF_USERS_ACHIEVED
        } else
            return Code.FAILED
    }
    fun login(uID: Int?, pin: Int?): Log {
        if (uID == null || pin == null) return
Log(Code.FAILED, null)
        val user = list[uID] ?: return Log(Code.FAILED,
null)
        return if (user.pwd != pin)
            Log(Code.FAILED, null)
        else
            Log(Code.SUCCESS, user.desc)
    }
    fun removeMsg(char: Char, uID: Int?): Code {
        if (uID == null) return Code.FAILED
        val user = list[uID] ?: return Code.FAILED
        return if (char == '*') {
            user.desc = null
            Code.SUCCESS
        } else
```

```
            Code.FAILED
    }
    fun getUser(uID: Int) = list[uID]
    fun addMsg(uID: Int, msg: String): Code{
        val user = list[uID] ?: return Code.FAILED
        user.desc = msg
        return Code.SUCCESS
    }
    fun closure(){
        val users = list.map {
            val uid = when {
                it.value.uID < 10 -> "00${it.value.uID}"
                it.value.uID < 100 -> "0${it.value.uID}"
                else -> it.value.uID
            }

"${uid}$DEL${it.value.uName}$DEL${it.value.pwd}$DEL${it.val
ue.desc}"
        }
        FileAccess.add(users)
    }
}
```

C. Código Kotlin dos Logs

```
import java.time.LocalDate
import java.time.LocalTime

object Logs {
    fun addLog(uId: Int) {
        val showUId = when {
            uId < 10 -> "00$uId"
            uId < 100 -> "0$uId"
            else -> uId
        }

FileAccess.logRegister("${LocalDate.now()}$DEL${LocalTime.n
ow()}$DEL$showUId")
    }
    fun closure() = FileAccess.close()
}
```

D.Código Kotlin do File Utils

```
import java.io.BufferedReader
import java.io.FileReader
import java.io.PrintWriter
import java.io.FileWriter
import java.io.BufferedWriter

fun createReader(fileName: String): BufferedReader =
BufferedReader(FileReader(fileName))
fun createFileWriter(fileName: String): BufferedWriter =
BufferedWriter(FileWriter(fileName))
fun createWriter(fileName: String): PrintWriter =
PrintWriter(fileName)
```

E. Código Kotlin do File Access

```kotlin
import java.io.BufferedWriter

const val USERS = "data/users.txt"
const val LOGS = "data/log.txt"
const val DEL = '$' // delimiter

object FileAccess {
    private var logs = mutableListOf<String>()
    private var logWriter: BufferedWriter? = null
    fun init(){
        val save = createReader(LOGS)
        var line = save.readLine()
        while (line != null){
            logs.add(line)
            line = save.readLine()
        }
        save.close()
        logWriter = createFileWriter(LOGS)
        logs.forEach {
            logWriter?.write(it)
            logWriter?.newLine()
        }
    }
    fun add(data: List<String>){
        val wr = createWriter(USERS)
        data.forEach { wr.println(it) }
        wr.close()
    }
    fun getUsers(): List<String>{
        val rd = createReader(USERS)
        val list = mutableListOf<String>()
        while (true) {
            val line = rd.readLine() ?: break
            list.add(line)
        }
        return list
    }
    fun logRegister(data: String){
        logWriter?.write(data)
        logWriter?.newLine()
    }
    fun close(){
        logWriter?.close()
    }
}
```

F. Código Kotlin da Manutenção

```
private const val MAINTENANCE = 0x80
object Maintenance {
    fun init(){}
    fun maintenanceMode(): Boolean =
        HAL.readBits(MAINTENANCE) != 0
}
```