

# ***Ponto de Controle 4 - Sistemas Operacionais Embarcados***

## **Guarda-volumes eletrônico**

Gabriel B. Pinheiro  
Engenharia Eletrônica  
Universidade de Brasília - Faculdade do Gama  
Brasília-DF, Brasil  
gabrielbopi@gmail.com

Elisa Costa Lima  
Engenharia Eletrônica  
Universidade de Brasília - Faculdade do Gama  
Brasília-DF, Brasil  
eliss.liima@gmail.com

**Abstract**—Este documento descreve o desenvolvimento parcial do projeto no segundo ponto de controle. **Palavras-Chave**—guarda-volumes; praticidade; segurança; Raspberry Pi

### **I. INTRODUÇÃO**

Para o terceiro ponto de controle, era esperado o desenvolvimento de um protótipo com mais recursos, utilizando as ferramentas mais básicas da placa de desenvolvimento, com bibliotecas prontas e também o refinamento do protótipo anterior, acrescentando recursos básicos de sistema. Orientando-se por esse foco, tentou-se fazer a lista de requisitos e a aquisição dos materiais (hardware) necessários para todo o projeto, além desenvolvimento e integração básica dos componentes em que os membros do grupo já tinham acesso/posse.



Fig. 1. Modelo de quadro de energia a ser utilizado no protótipo.

### **II. OBJETIVOS E DESENVOLVIMENTO**

Para este ponto de controle foi pedido para que os alunos testassem implementar cada requisito do sistema em blocos funcionais, assim temos:

- Aperfeiçoamento do software de login de acesso do usuário ao guarda volumes;
- Implantação de comunicação via servidor pelo sistema dos guarda-volumes com um servidor central.
- Aperfeiçoamento do leitor de impressão digital;
- Teste de câmera para leitura do QRCode;
- Implementação e teste de tela;

Foi possível mostrar o funcionamento de todos os componentes exigidos para o projeto separadamente, em implementação básica, como o leitor de digitais, tela e câmera.

Assim para esse primeiros passos do desenvolvimento do projeto foi necessário algumas configurações no Raspberry Pi.

O Raspberry Pi é um minicomputador de fácil interação. Somente é necessário ligá-lo a um monitor, ter um mouse e um teclado para usá-lo como se fosse um outro computador qualquer. Porém para o desenvolvimento da matéria onde iremos tentar construir e implementar um sistema embarcado, não é viável carregar tais periféricos para a configuração do raspberry. É muito mais prático

utilizar outro computador para fazer as alterações a “distância” deste minicomputador. Assim foram os passos seguidos:

1. Inicialmente o cartão SD foi formatado para que depois fosse instalado o SO Raspbian a partir do site oficial do Raspberry Pi[2]. A versão do sistema operacional escolhido foi o “Raspbian Buster with desktop”, uma versão adequada para utilizar um ambiente gráfico. Para a instalação foi utilizado o programa ETCHER[1].
2. Com o cartão inserido na Raspberry foi conectado no mesmo os periféricos: Tv(monitor) via HDMI, mouse e teclado via USB;
3. Ao ligar o sistema foi possível verificar no monitor a correta inicialização do sistema. Duas configurações foram modificadas, para que fosse possível usar o Raspberry de outro computador: as opções SSH e VNC são ligadas na aba de interfaces. O SSH é a opção que ativa o servidor (Secure Shell), que permite entrar no raspberry remotamente pela rede, o acesso é feito via linha de código, já o VNC o acesso é feito graficamente abrindo-se uma janela que simula a tela de visualização do sistema através do programa “Remmina”;
4. Pelo terminal do computador é feito os seguintes comandos:

```
$ if config -- onde toda as configurações dos dispositivos conectados a rede aparecem
```

```
$ sudo apt-get install nmap
```

```
$ nmap -sV -p 22 192.168.0.11-255 -- faz um scan dos dispositivos conectados na rede que tem um número de IP similar ao número em negrito;
```

```
$ ssh pi@192.168.0.30 -- comando usado para ter o acesso remoto por linha de código à raspberry. O número em negrito é modificado pelo número de IP da raspberry encontrado no passo anterior;
```

Executando no computador pessoal o último comando, o cabeçalho do terminal muda para pi@raspberrypi:~\$, o que significa que tudo digitado no terminal terá ação no sistema da Raspberry.

5- Para configurar o VNC é necessário os seguintes comandos, procedimento que só pode ser feito a partir dos passos anteriores:

```
$ sudo apt-get install tightvncserver --- instalando o servidor VNC
```

```
$ vncserver :1 - geometry 1024x600 -depth 16 -pixelformat rgb565 -- exemplo de configuração do tamanho da tela a ser aberta
```

6 -Voltando o terminal para o sistema Linux do PC, instalou o VNC viewer para cliente

```
$ sudo apt-get install xtightvncviewer
```

```
$ vncviewer xxx.xxx.x.xxx:5901 --- Conectando o servidor VNC. O número em negrito é o número do IP da raspberry.
```

7 -Outro problema foi encontrado ao tentar acessar remotamente o raspberry na UnB. A Raspberry não reconhece as redes Wi-fi disponíveis. Como proceder se o acesso remoto só é feito quando o computador e a raspberry são conectados à mesma rede de internet. A solução é conectar com um cabo de rede o computador com o minicomputador. Configurar o computador para o acesso “somente via cabo” nas configurações de rede e digitar no terminal:

```
$ ssh pi@raspberrypi.local
```

Através do comando anterior o IP da raspberry é encontrado e o acesso ao sistema da mesma poderá ser feita pelo mesmo comando ssh pi@<IP\_Rasp>

### III. DESENVOLVIMENTO CORRENTE

Os equipamentos e materiais que foram adquiridos para esse ponto de controle foram:

- Quadro de energia para servir de armário.
- Placa Raspberry Pi 3 modelo B +.
- Cartão SD de 8 gb.
- Teclado matricial 4x4 de contato (barramento UART)
- Tranca elétrica solenoide (acionamento em 12V).
- Módulo relê (acionamento em 5V) [4]
- Módulo óptico de impressão digital JM-101B [6]
- Fonte de 12V;
- Tela LCD 3.5 polegadas;
- Módulo Câmera 5MP.

O teclado matricial foi integrado, parcialmente, ao sistema. Este ainda apresenta bugs no funcionamento, mas que pode ser corrigido com decorrer do tempo.

A tranca foi implementada, o acionamento se dá por um sinal (via software que será explanado na próxima seção) para um relê, armando-o. Ele faz a solenoide (tranca) ser alimentada por uma fonte de 12V, ligando-a.

O software do sistema foi implementado no Raspberry Pi, seu funcionamento será explanado na próxima seção. O acesso ao software da placa se dá por protocolo SSH. Já é possível fazer o cadastro de usuários em arquivo de dados e acesso por login para uso do ‘guarda-volumes’ pelo usuário remotamente.

O software para o sistema central foi desenvolvido para cadastro e gerenciamento de usuário em arquivo de dados

em servidor, seu funcionamento será explanado na próxima seção.

O software do guarda-volumes foi implementado no Raspberry Pi, seu funcionamento será explanado na próxima seção. O acesso ao software da placa se dá por protocolo SSH. Nele é possível fazer o acesso básico do usuário pelo guarda-volumes, a verificação remota de usuário pelo servidor central, com validação posterior do leitor de impressão digital e acionamento da tranca. O módulo óptico de impressão digital se comunica via UART ao *Raspberry pi*.



Fig. 2. Raspberry Pi 3, modelo B.

#### IV. SOFTWARE DESENVOLVIDO

##### A. Teclado Matricial

O teclado matricial 4x4 foi escolhido como um item de projeto por ser acessível financeiramente e por ter muitas referências de uso e programação. Para o ponto de controle 2 o teclado foi testado a partir de um código em python. Como o teclado não possui um circuito interno, tudo o que o programa precisa fazer é buscar um jeito de realizar a leitura de maneira correta e precisa.

Simplificando o que o código faz é: setar os pinos correspondentes às colunas como outputs e com o valor "1" (*high*). Os pinos correspondentes às linhas são configurados como inputs com pull-up resistors. (Todos como "1")

Os GPIOs configurados como outputs são setados como "0" (*low*) um de cada vez. Assim quando uma tecla é pressionada uma linha fica no estado *low*, assim saberemos qual coluna foi selecionada para também estar no estado *low*, já que apenas uma coluna pode estar em *low* em um certo instante de tempo.

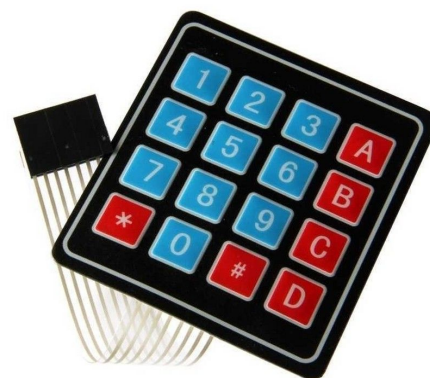


Fig. 3. Teclado matricial, 4x4.

##### B. Login Acionamento de tranca

Para o sistema do guarda-volumes, visando o uso do usuário, foi feito o programa 'cliente\_terminal.c' que possibilita através de login na conta cadastrada previamente no servidor central, com comunicação via *socket* pela rede local e posteriormente cadastro da impressão digital e em seguida, caso validado, o acesso a abertura da porta do guarda-volumes (acionamento da tranca).

Para o correto funcionamento em sua execução, como os dispositivos conectados em rede local, são necessárias as seguintes entradas:

```
$/cliente_terminal <IP do Servidor> <Porta do servidor> <nome do usuario> <senha do usuario>
```

Se o usuário e senha inseridos forem válidos em seguida é solicitada a impressão digital do usuário. O mesmo dedo deve ser lido duas vezes. Caso procedimento tenha sucesso a tranca é liberada por 5 segundos.

Devendo estar no diretório do programa, via terminal no sistema Raspbian. O código do programa encontra-se no apêndice A.

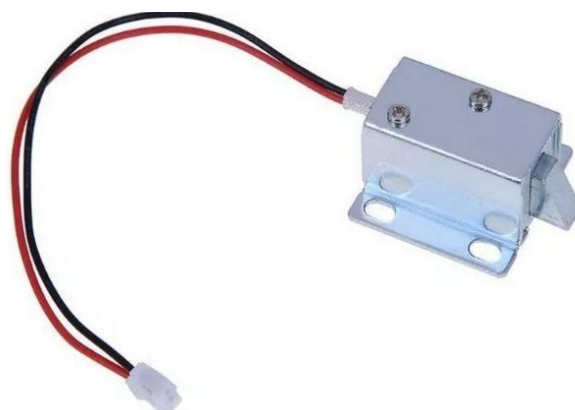


Fig. 4. Tranca solenoide..

O gerenciamento do módulo óptico de impressão digital foi feito pelo código do Apêndice D.



Fig. 5. Leitor óptico de impressão digital.

### C. Software para gerenciamento dos dados no servidor

Visando implementação futura em servidor web, foi feito o programa 'sistema\_contas.c' que possibilita o cadastro de novos usuários, e leitura de usuários já previamente cadastrados no sistema.

Para o correto funcionamento em sua execução, são necessárias as seguintes entradas para ler algum usuário já cadastrado:

```
$/sistema_contas -l usuarios.txt <nome do usuario>
```

Para cadastrar o usuario usa-se as entradas:

```
$/sistema_contas -m usuarios.txt <nome do usuario> <senha> <numero de credits> <senha do usuario>
```

devendo estar no diretório do programa, via terminal no sistema Raspbian. O código do programa encontra-se no apêndice B.

Para implementar as funções "adicionaUsuario(...)" e "leUsuario(...)" necessárias para os dois programas anteriores, foi feita a biblioteca 'gerencia\_contas.h'.

O código do arquivo 'gerencia\_contas.c' encontra-se no apêndice C.

Para a autenticação do login do usuário no terminal do guarda-volumes, foi implementado o programa 'servidor\_sistema.c', que roda no servidor, onde há o arquivo com os dados dos usuários, 'usuarios.txt'. O programa do terminal 'cliente\_terminal' envia os dados digitados pelo usuário a este, via *socket*, que faz a verificação dos dados e envia uma resposta ao terminal.

O código do arquivo 'servidor\_sistema.c' encontra-se no apêndice F.

Os arquivos supracitados encontram-se em [3]

### D. Tela LCD

Uma pequena tela LCD foi implementada usando os seguintes passos:

- Na pasta Download do sistema operacional instalado no Raspberry Pi foi baixado o driver presente no endereço :

```
wget
http://en.keddei.net/raspberry/v6\_1/LCD\_show\_v6\_1\_3.tar.gz
```

- Instalação do arquivo dentro da pasta baixada :  
`sudo ./LCD35_v`
- Após a instalação o Raspberry fez um reboot automaticamente e foi necessário digitar o seguinte comando no terminal para alterar mais uma configuração interna: `raspi-config`
- A configuração interna modificada foi dentro da aba "Advanced Options", onde a opção "A6 GL DRIVER(FULL KMS)" foi habilitada;
- A última modificação feita foi dentro do arquivo de texto "config.txt" dentro da pasta boot. Este de arquivo de texto é lido durante a inicialização do sistema da raspberry pi, as linhas 28, 29 e 48 foram modificadas respectivamente pelas linhas:

```
hdmi_group=2
hdmi_mode=35
dtoverlay=spi=off
```

E ao final do arquivo foram adicionadas as seguintes linhas de instrução:

```
hdmi_force_hotplug=1

gpu_mem=32

start_x=0

enable_uart=1

dtoverlay=w1-gpio
```

Após o sistema ser reiniciado a tela já ligava normalmente, aparecendo a imagem inicial do desktop do sistema operacional raspbian.

### E. Câmera

A instalação da câmera foi feita de maneira bem mais rápida e convencional que a tela. A câmera tem espaço reservado na raspberry B+, sem ocupar nenhum pino ou porta USB. A única modificação a ser feita é habilitar a câmera dentro das configurações do raspberry dentro da opção 5- INTERFACING OPTIONS.

Comandos básicos foram testados para averiguar o funcionamento da câmera como:

raspistill -o minha\_foto.png - Para gerar uma foto através da câmera

raspivid -o nome\_do\_video.h264 -t 10000 -  
gravar um vídeo de 5s de duração

#### REVISÃO BIBLIOGRÁFICA

- [1] Balena ETCHER, <<https://www.balena.io/etcher/>>. Acesso em 1 de outubro de 2019.
- [2] Raspberry Pi <<https://www.raspberrypi.org/>>. Acesso em 1 de outubro de 2019.
- [3] Repositório <[https://github.com/gabrielbopi/SO\\_Embarcados/tree/master/2\\_PC/P3/Codigo](https://github.com/gabrielbopi/SO_Embarcados/tree/master/2_PC/P3/Codigo)>. Acesso em 1 de outubro de 2019.
- [4] Módulo Relê, <<https://www.huinfinito.com.br/modulos/988-modulo-rele-5v1canal.html>>. Acesso em 1 de novembro de 2019.
- [5] Python library for ZFM fingerprint sensors. <<https://github.com/bastianraschke/pyfingerprint>>. Acesso em 1 de novembro de 2019.
- [6] JM-101 Optical Fingerprint Module User Manual. <[http://21st-century-spring.kr/wordpress/?page\\_id=794](http://21st-century-spring.kr/wordpress/?page_id=794)>. Acesso em 1 de novembro de 2019.

## APÊNDICE

Seguem os códigos citados na seção IV.

- A. Programa 'cliente\_terminal.c'. O código pode ser compilado pelo GCC e necessita da biblioteca 'gerencia\_contas.h' e 'gpio\_sysfs.h' no mesmo diretório. Também é necessário o arquivo '/home/pi/Desktop/Codigo/fingerprint\_enroll.py', e consequentemente o Python instalado no sistema. É necessária conexão de rede local com o servidor principal rodando '...'.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <unistd.h>
#include "gerencia_contas.h"
#include "gpio_sysfs.h"
#define entrada_usuario argv[3]
#define entrada_senha argv[4]

int main(int argc, char* const argv[]){
    int socket_id;
    struct sockaddr_in servidorAddr;
    int length;
    unsigned short servidorPorta;
    char *IP_Servidor;
    char *mensagem;
    int resposta;

    const int creditos_minimos = 5;
    struct usuario usuario_corrente;
    int polegarAutenticado;

    if (argc < 3){
        printf("Escolha uma opcao valida.\n");
        printf("Para utilizar o programa escreva no terminal:\n./cliente_terminal\n");
        printf("<IP do Servidor> <Porta do servidor> <nome do usuario> <senha do usuario>\n");
        return -1;
    }

    strcpy(usuario_corrente.nome, entrada_usuario);
    strcpy(usuario_corrente.senha, entrada_senha);

    IP_Servidor = argv[1];
    servidorPorta = atoi(argv[2]);
    //mensagem = argv[3];

    // Abrindo o socket para o cliente
    socket_id = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(socket_id < 0){
        fprintf(stderr, "Erro na criacao do Soquete!\n");
        exit(0);
    }

    // Conectando o socket ao IP "IP_Servidor" pela porta "servidorPorta"
    memset(&servidorAddr, 0, sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr = inet_addr(IP_Servidor);
    servidorAddr.sin_port = htons(servidorPorta);
    if(connect(socket_id, (struct sockaddr *) &servidorAddr, sizeof(servidorAddr)) <
```

```

0){
    fprintf(stderr, "Erro na conexao!\n");
    exit(0);
}

// Mandando mensagem ao servidor
length = sizeof(struct usuario);
//write(socket_id, &length, sizeof(length));
write(socket_id, &usuario_corrente, length);
read(socket_id, &resposta, sizeof(resposta));
//fprintf(stderr, "%d bytes.", length);
read(socket_id, &usuario_corrente, length);
// Fechando o socket local
close(socket_id);

if(resposta == 1){
    printf("ID: %d, usuario: %s, senha:%s, creditos:%d\n",
        usuario_corrente.id, usuario_corrente.nome,
        usuario_corrente.senha, usuario_corrente.creditos);
    printf("Abre-te Sesamo!\n");
    polegarAutenticado = recolheDigital()/256;
    if(polegarAutenticado == 10) {abreTranca();}
}
else if(resposta == 2)    {printf("Senha incorreta!\n");}
else if(resposta == 0)    {printf("Usuario escrito escrito errado ou
inexistente.\n");}
else {printf("Erro no servidor.\n");}

return 0;
}

int recolheDigital(void){
    printf("Resposta: %d\n",system("python
/home/pi/Desktop/Codigo/fingerprint_enroll.py"));
    return 0;
}

int abreTranca(void){
    int pin=21;
    if(setGPIO_Out(pin)){
        printf("Erro ao tentar abrir a porta...(1)\n");
        return -1;
    }
    if (GPIO_Write(pin,1)){
        printf("Erro ao tentar abrir a porta...(2)\n");
        return 1;
    }
    printf("Tranca abre...\n");
    sleep(6);
    if (GPIO_Write(pin,0)){
        printf("Erro ao tentar desligar a tranca...(1)\n");
        return 3;
    }
    if(unsetGPIO(pin)){
        printf("Erro ao tentar desligar a tranca...(2)\n");
        return 2;
    }
    return 0;
}

```

- B. Programa 'sistema\_contas.c'. O código pode ser compilado pelo GCC e necessita da biblioteca gerencia\_contas.h no mesmo diretório.

```
#include <stdio.h>
#include <stdlib.h>
//#include <string.h>
#define entrada_arquivo argv[2]
#define entrada_usuario argv[3]
#define entrada_senha argv[4]
#define entrada_creditos argv[5]

#include "gerencia_contas.h"

int main(int argc, char * argv[]){
    struct usuario usuario_corrente;
    int sucesso;

    if (argc < 3) {
        printf("Escolha uma opcao valida.\n");
        printf("Para utilizar o programa escreva no terminal:\n- Para ler algum
usuario cadastrado: ./sistema_contas -l usuarios.txt <nome do usuario>\n- Para cadastrar
o usuario: ./sistema_contas -m usuarios.txt <nome do usuario> <senha> <numero de
creditos>\n");
        return -1;
    }
    if(argv[1][0]=='-'){
        switch(argv[1][1]){
            case 'l':
                usuario_corrente =
leUsuario(entrada_arquivo,entrada_usuario);
                printf("ID: %d, usuario: %s, senha:%s, creditos:%d\n",
usuario_corrente.id, usuario_corrente.nome, usuario_corrente.senha,
usuario_corrente.creditos);
                break;
            case 'm':
                if (argc < 5) {
                    printf("Escolha uma opcao valida.\n");
                    printf("Para utilizar o programa escreva:\nPara ler algum usuario
cadastrado: ./sistema_contas -l usuarios.txt <nome do usuario>\nPara cadastrar o
usuario: ./sistema_contas -m usuarios.txt <nome do usuario> <senha> <numero de
creditos>\n");
                    return -1;
                }
                sucesso =
adicionaUsuario(entrada_arquivo,entrada_usuario,entrada_senha, atoi(entrada_creditos));
                if (sucesso != 0) {printf("Erro no cadastro!\n");}
                //printf("ID: %d\n", usuario_corrente.id);
                break;
            default:
                printf("Escolha uma opcao valida.\n");
                printf("Para utilizar o programa escreva:\nPara ler algum usuario
cadastrado: ./sistema_contas -l usuarios.txt <nome do usuario>\nPara cadastrar o
usuario: ./sistema_contas -m usuarios.txt <nome do usuario> <senha> <numero de
creditos>\n");
        }
    }

    else {
        printf("Para utilizar o programa escreva:\nPara ler algum usuario
cadastrado: ./sistema_contas -l usuarios.txt <nome do usuario>\nPara cadastrar o
```



```
usuario: ./sistema_contas -m usuarios.txt <nome do usuario> <senha> <numero de
creditos>\n");
    }
}
```

- c. Biblioteca 'gerencia\_contas.c'. O código pode ser compilado como objeto pelo GCC e necessita do arquivo gerencia\_contas.h no mesmo diretório.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include "gerencia_contas.h"

/*struct usuario
{
    int id;
    char nome[30], senha[30];
    int credits;
};*/

struct usuario leUsuario(char *nome_arquivo, char *nome_usuario){
    int fp;
    char i;
    struct usuario usuario_corrente;

    int offset1, offset2;
    usuario_corrente.id = 0;

    fp = open(nome_arquivo, O_RDONLY | O_CREAT, S_IRWXU);
    if(fp == -1){
        printf("Erro ao abrir o arquivo!");
        exit(EXIT_FAILURE);
    }

    offset2 = sizeof(usuario_corrente.senha) + sizeof(usuario_corrente.credits);

    while(read(fp, &i, sizeof(char)) != 0) {
        offset1 = usuario_corrente.id*sizeof(struct usuario) + sizeof(int);
        lseek(fp, offset1, SEEK_SET);
        //printf("usuario_corrente.nome:%s nome_usuario:%s\n", usuario_corrente.nome,
nome_usuario);
        read(fp, usuario_corrente.nome, 30 * sizeof(char));
        if (strcmp(usuario_corrente.nome, nome_usuario) == 0){
            read(fp, usuario_corrente.senha, 30 * sizeof(char));
            read(fp, &usuario_corrente.credits, sizeof(int));

            return usuario_corrente;
        }
        else{
            lseek(fp, offset2, SEEK_CUR);
            usuario_corrente.id++;
        }
    }
    usuario_corrente.id = -1;
    strcpy(usuario_corrente.nome, "");
    strcpy(usuario_corrente.senha, "");
    usuario_corrente.credits = 0;
}
```

```

        printf("Usuario nao existente anteriormente no sistema.\n");
        return usuario_corrente;
    }

int adicionaUsuario(char *nome_arquivo, char *nome_usuario, char *senha_usuario, int
creditos_usuario){
    int fp;
    struct usuario usuario_corrente;

    usuario_corrente = leUsuario(nome_arquivo, nome_usuario);
    if(strcmp(usuario_corrente.nome, nome_usuario) == 0){
        printf("Erro. Esse usuario ja existe.\n" );
        return -1;
    }
    else{
        strcpy(usuario_corrente.nome, nome_usuario);
        strcpy(usuario_corrente.senha, senha_usuario);
        usuario_corrente.creditos = credits_usuario;

        fp = open(nome_arquivo, O_WRONLY | O_APPEND | O_CREAT, S_IRWXU);
        if(fp == -1){
            printf("Erro ao abrir o arquivo!");
            exit(EXIT_FAILURE);
        }

        if(write(fp, &usuario_corrente, sizeof(struct usuario)) != -1){
            printf("Usuario cadastrado com sucesso!\n");
        }
        else{
            printf("Erro no cadastro de usuario!\n");
        }
        return 0;
    }
}

```

#### D. Código do teclado Matricial 4X4 em C

```

#include <wiringPi.h>
#include <stdio.h>

#define ROWS 4

#define COLS 4

char pressedKey = '\0';

int rowPins[ROWS] = {1, 4, 5, 6};

int colPins[COLS] = {12, 3, 2, 0};

```

```

char keys[ROWS][COLS] = {
{'1', '2', '3', 'A'},
{'4', '5', '6', 'B'},
{'7', '8', '9', 'C'},
{'*', '0', '#', 'D'} };

void init_keypad()
{
    for (int c = 0; c < COLS; c++)
    {
        pinMode(colPins[c], OUTPUT);
        digitalWrite(colPins[c], HIGH);
    }

    for (int r = 0; r < ROWS; r++)
    {
        pinMode(rowPins[0], INPUT);
        digitalWrite(rowPins[r], PUD_UP);
    }
}

int findLowRow()
{
    for (int r = 0; r < ROWS; r++)
    {
        if (digitalRead(rowPins[r]) == LOW)
        return r;
    }    return -1;
}

char get_key()

```

```

{   int rowIndex;

    for (int c = 0; c < COLS; c++) {

        digitalWrite(colPins[c], LOW);

        rowIndex = findLowRow();

        if (rowIndex > -1)

            {               if (!pressedKey)

                            pressedKey = keys[rowIndex][c];

                            return pressedKey;

                        }

        digitalWrite(colPins[c], HIGH);
    }   pressedKey = '\0';

    return pressedKey;
}

int main(void)
{   wiringPiSetup();

    init_keypad();

    while(1)

    {       char x = get_key();

            if (x)

                printf("pressed: %c\n", x);

            else

                printf("no key pressed\n");

            delay(250);

        }   return 0;
}

```

E. Código do ‘/home/pi/Desktop/Codigo/fingerprint\_enroll.py’ em Python. Ele é proveniente do repositório em [5]

```

"""
PyFingerprint
Copyright (C) 2015 Bastian Raschke <bastian.raschke@posteo.de>
All rights reserved.

"""

import time
from pyfingerprint.pyfingerprint import PyFingerprint

## Enrolls new finger
##

## Tries to initialize the sensor
try:
    f = PyFingerprint('/dev/ttyS0', 57600, 0xFFFFFFFF, 0x00000000)

    if ( f.verifyPassword() == False ):
        raise ValueError('The given fingerprint sensor password is wrong!')

except Exception as e:
    print('The fingerprint sensor could not be initialized!')
    print('Exception message: ' + str(e))
    exit(1)

## Gets some sensor information
print('Currently used templates: ' + str(f.getTemplateCount()) + '/' +
      str(f.getStorageCapacity()))

## Tries to enroll new finger
try:
    print('Waiting for finger...')

    ## Wait that finger is read
    while ( f.readImage() == False ):
        pass

    ## Converts read image to characteristics and stores it in charbuffer 1
    f.convertImage(0x01)

    ## Checks if finger is already enrolled
    result = f.searchTemplate()
    positionNumber = result[0]

    if ( positionNumber >= 0 ):
        print('Template already exists at position #' + str(positionNumber))
        exit(0)

    print('Remove finger...')
    time.sleep(2)

    print('Waiting for same finger again...')

    ## Wait that finger is read again
    while ( f.readImage() == False ):
        pass

    ## Converts read image to characteristics and stores it in charbuffer 2
    f.convertImage(0x02)

```

```

    ## Compares the charbuffers
    if ( f.compareCharacteristics() == 0 ):
        raise Exception('Fingers do not match')

    ## Creates a template
    f.createTemplate()

    ## Saves template at new position number
    positionNumber = f.storeTemplate()
    print('Finger enrolled successfully!')
    print('New template position #' + str(positionNumber))
    exit(10)

except Exception as e:
    print('Operation failed!')
    print('Exception message: ' + str(e))
    exit(1)

```

- F. Programa 'servidor\_sistema.c'. O código pode ser compilado pelo GCC e necessita da biblioteca gerencia\_contas.h no mesmo diretório.

```

//Servidor central para gerenciamento dos usuarios
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <string.h>
#include <signal.h>
#include <sys/socket.h>
#include <sys/un.h>
#include "gerencia_contas.h"
#define ARQUIVO_CONTAS      "usuarios.txt"

int socket_id;
void sigint_handler(int signum);
void procuraUsuario(int client_socket);
void end_server(void);

int main (int argc, char* const argv[]){
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;

    if (argc < 2) {
        printf("Escolha uma opcao valida.\n");
        printf("Para utilizar o programa escreva no terminal:\n$
./servidor_sistema <Porta do servidor>\n");
        return -1;
    }
    // Definindo o tratamento de SIGINT
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, sigint_handler);

    // Abrindo o socket local
    socket_id = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(socket_id < 0)
    {
        fprintf(stderr, "Erro na criacao do Soquete!\n");
        exit(-1);
    }
}

```

```

// Ligando o socket a porta "servidorPorta"
memset(&servidorAddr, 0, sizeof(servidorAddr));
servidorAddr.sin_family = AF_INET;
servidorAddr.sin_addr.s_addr = htonl(INADDR_ANY);
servidorAddr.sin_port = htons(servidorPorta);
if(bind(socket_id, (struct sockaddr *) &servidorAddr, sizeof(servidorAddr)) < 0){
    fprintf(stderr, "Erro na ligacao!\n");
    exit(-1);
}
// Tornando o socket passivo (para virar um servidor)
if(listen(socket_id,10) < 0){
    fprintf(stderr, "Erro!\n");
    exit(0);
}

while(1){
    int socketCliente;
    struct sockaddr_in clienteAddr;
    unsigned int clienteLength;
    // Aguardando a conexao de um cliente
    clienteLength = sizeof(clienteAddr);
    if((socketCliente = accept(socket_id, (struct sockaddr *) &clienteAddr,
&clienteLength)) < 0)
    {
        fprintf(stderr, "Falha no accept().\n");
    }

    fprintf(stderr, "Conexao do Cliente %s\n",
inet_ntoa(clienteAddr.sin_addr));
    // Tratando comunicacao com o cliente
    procuraUsuario(socketCliente);
    // Fechando a conexao com o cliente
    close(socketCliente);
}

return 0;
}

void sigint_handler(int signum){
    fprintf(stderr, "\nRecebido o sinal CTRL+C... vamos desligar o servidor!\n");
    end_server();
}

void procuraUsuario(int client_socket){
    int length, nome_valido, senha_valida;
    char* text;
    struct usuario usuario_solicitado, usuario_corrente;
    int resposta = 0;

    length = sizeof(struct usuario);
    read(client_socket, &usuario_solicitado, length);
    usuario_corrente = leUsuario(ARQUIVO_CONTAS,usuario_solicitado.nome);

    nome_valido = strcmp(usuario_solicitado.nome, usuario_corrente.nome);
    senha_valida = strcmp(usuario_solicitado.senha, usuario_corrente.senha);

    if(nome_valido == 0){
        if(senha_valida == 0){
            resposta = 1;
            printf("ID: %d, usuario: %s, senha:%s, creditos:%d\n",
                usuario_corrente.id, usuario_corrente.nome,
usuario_corrente.senha, usuario_corrente.creditos);
            write(client_socket, &resposta, sizeof(resposta));
            write(client_socket, &usuario_corrente, length);

```

```
        }else{
            resposta = 2;
            printf("ID: %d, usuario: %s\nSenha incorreta!\n",
                usuario_corrente.id, usuario_corrente.nome);
        }
    }else{
        resposta = 0;
        printf("Usuario nao encontrado...\n");
    }
}

void end_server(void){
    //Fechando o socket local
    close(socket_id);
    exit(0);
}
```