PONTO DE CONTROLE 2
CONTROLE DE ACESSO VIA RECONHECIMENTO DE FACE HUMANA

Antônio Aldísio - 14/0130811 —- Vitor Carvalho de Almeida - 14/0165380

Programa de Graduação em Engenharia Eletrônica, Faculdade Gama
Universidade de Brasília
Gama, DF, Brasil

email: aldisiofilho@gmail.com —- vitorcarvalhoamd@gmail.com

RESUMO

O projeto consiste em construir um sistema de controle de acesso ativado por reconhecimento facial. Será possível enviar os dados de acesso via rede para um banco de dados. Neste ponto de controle é apresentada a integração dos sistemas. São utilizadas threads para paralelizar ações de controle e verificação dos periféricos.

Palavras-chave: Controle de acesso, Raspberry Pi, OpenCV, reconhecimento facial, segurança, threads.

1. INTRODUÇÃO

O mundo encontra-se em uma grande evolução, nos dias atuais a automação utilizada para controle de acesso é a biometria por impressão digital. Porém o usuário tem quer ter uma interação direta e tatil com o sistema para a sua liberação. O controle de acesso via reconhecimento facial elimina a necessidade de interação direta do usuário e pode ser implementado juntamente ao de monitoramento por câmeras, utilizando o mesmo dispositivo para a aquisição das imagens.

Além da facilidade do uso e a eliminação da possibilidade de esquecer a chave de acesso, é possivel armazenar asinformações para utilizar como controle de ponto, ou adaptar para um sistema de controle/monitoramento de produtividade em uma empresa.

Com base nessa tendência e buscando uma facilidadepara o usuário, esse artigo propõe a construção de um sistema de reconhecimento facial para abertura de portas.

O objetivo desse projeto é a construção de um sistema de abertura de porta através do reconhecimento do rosto de usuários cadastrados e enviar dados de acessos pela rede.

Um sistema de reconhecimento facial traz alguns benefícios como: praticidade, segurança. No caso desenvolvimento o enfoque é: a segurança, visto que a porta só se abrirá após o sistema reconhecer um usuário autorizado; e a possibilidade de utilizar essa validação de entrada como um ponto eletrônico para contagem de horas trabalhadas e geração de outros dados estat??sticos

Para este ponto de controle, é necessário comunicar a Raspbberry Pi com os elementos que serão utilizados no projeto.

O projeto em questão utiliza uma trava solenoide, que trabalha com tensão e corrente maiores do que a placa consegue fornecer. Logo, é necessário usar um sistema de chaveamento.

Nas próximas seções são apresentadas as soluções para o problema.

2. DESENVOLVIMENTO

2.1. Descrição do Hardware

Foi montado um sistema de ativação da trava eletrônica. Utilizando os seguintes materiais:

- Trava solenoide 12V (figura 1);
- Fonte DC 12V;
- Resistor de 1 KOhm;
- Transistor NPN (TIP41);
- Jumpers
- Protoboard
- Push-buttom
- Chave 3 pinos

Na protoboard foi montado o circuito da figura 2.

O pino de entrada foi conectado à GPIO4 da Raspberry Pi 3 para que fossem enviados os comandos para abrir a porta.

A trava solenoide mantem a porta fechada até que seja inserida uma tensão de 12V em seus terminais. Neste momento, o solenoide faz com que o ”dente” da trava seja retraído, liberando a abertura da porta. Ao retirar a tensão dos

Fig. 1. Trava eletrônica solenoide 12V

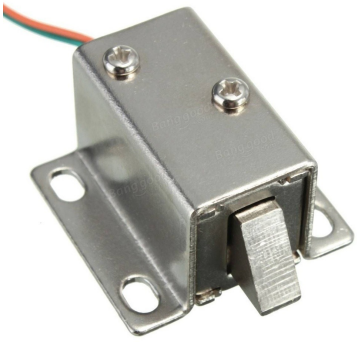
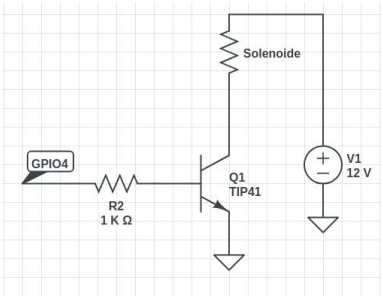


Fig. 2. Ativação da trava eletrônica solenoide 12V



terminais, uma mola retorna a trava para a posição original, travando a porta novamente. [1]

Foi utilizada uma fonte DC de 12V - 2A com conexão Jack P4, ligada na protoboard com um conector Jack P4 fêmea.

Foi conectada uma caixa de som à saída P2 da Raspberry Pi para reproduzir sons de confirmação ou negação de acesso.

Para receber a requisição de acesso, foi montado um circuito com botão em modo Pull-Up, como mostra o esquematico da figura 3

Foi utilizada a câmera NoIR da Raspberry Pi, conectada por meio do cabo flat (figura 4).

Prevendo que um malfeitor poderia arrombar a porta, notou-se a necessidade de instalar uma chave de fim de curso nesta, para identificar se ela encontra-se aberta ou fechada.

Obs: Para este ponto de controle, a chave de fim de curso foi simulada por uma chave comum, e será substituida quando a porta for construida.

2.2. Descrição do Software

Foi criado um sistema cliente-servidor utilizando o protocolo TCP para efeturar a comunicação com o administrador de forma remota. O servidor foi instalado na Raspberry Pi presente na central de comando da porta, e o cliente será executado na máquina do administrador. O cliente envia

Fig. 3. Botão em modo Pull-Up

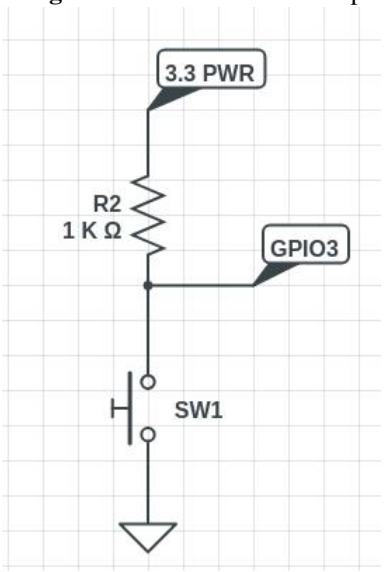
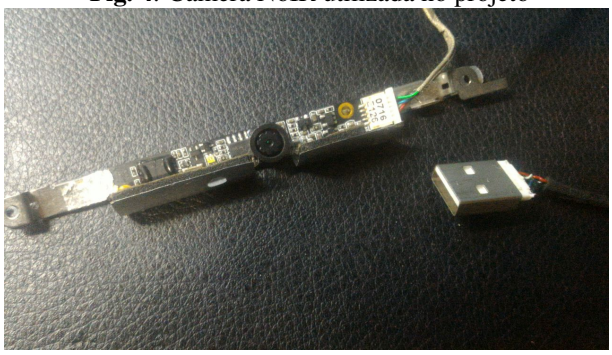


Fig. 4. Câmera NoIR utilizada no projeto



os comandos pela rede, e o servidor os escreve no arquivo *msgs_admin.txt*, assim, o programa principal pode ler os comandos.

Foi criada uma função principal contendo todas as chamadas necessárias para a execução do sistema. No programa, são criados dois processos filhos, mostrados na figura 5.

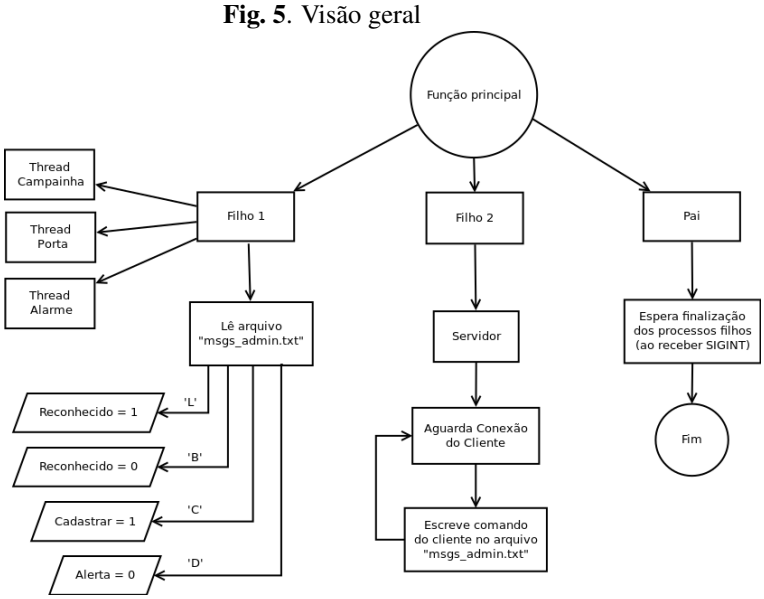
Filho 1: Executa as rotinas de verificação e controle da porta, tais como: verificação da campainha, verificação do estado da porta, ativação do alarme (caso a porta seja aberta sem permissão)

Filho 2: Executa o servidor

No filho 1, são criadas threads para cada elemento, pois todos precisam ser verificados simultaneamente.

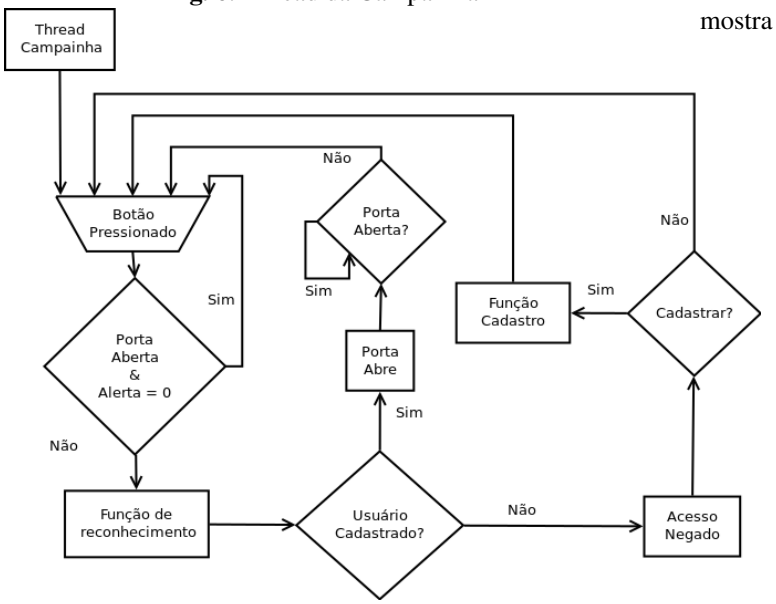
A thread da campainha, cujo funcionamento é mostrado na figura 6, é responsável por verificar mudanças no estado do botão (através da função poll), iniciar a rotina de verificação, e decidir se a porta será aberta ou não. Caso o acesso seja negado, é dada a opção de cadastro. A rotina de

Fig. 5. Visão geral



reconhecimento só é acionada com a porta fechada e quando o alerta de invasão está desativado.

Fig. 6. Thread da Campainha



A thread da porta, explicada pelo diagrama da figura 7 é responsável por verificar se a porta encontra-se aberta ou fechada. Como sistema de segurança, se a porta estiver aberta com a flag *reconhecido* = 0, é emitido um alerta, indicando uma invasão.

A thread do alarme é responsavel apenas por manter o alarme sonoro ligado, caso a flag *alerta* esteja setada, como mostra a figura 8.

Fig. 7. Thread da Porta

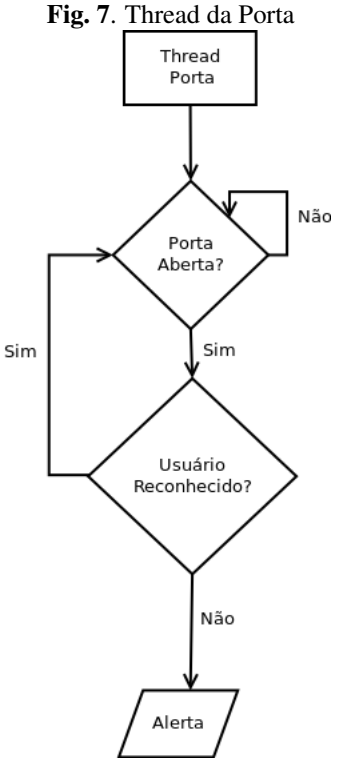
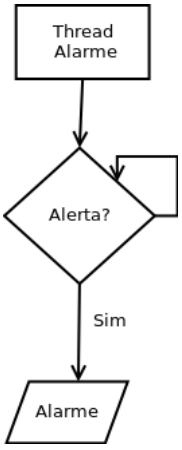


Fig. 8. Thread do Alarme



3. RESULTADOS

O conjunto montado ficou como mostrado na figura 9:

A ativação da trava eletrônica foi realizada com sucesso, sem sobreaquecimento do transistor, nem falha na comunicação.

Foi possível comunicar o sistema com o cliente com sucesso, assim como executar os comandos recebidos, como mostram as figuras 10 e 11.

Fig. 9. Montagem do circuito

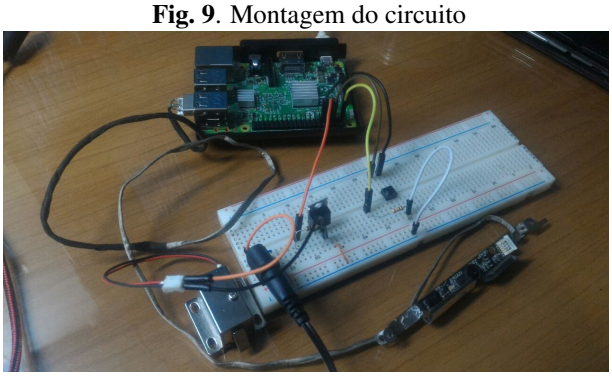
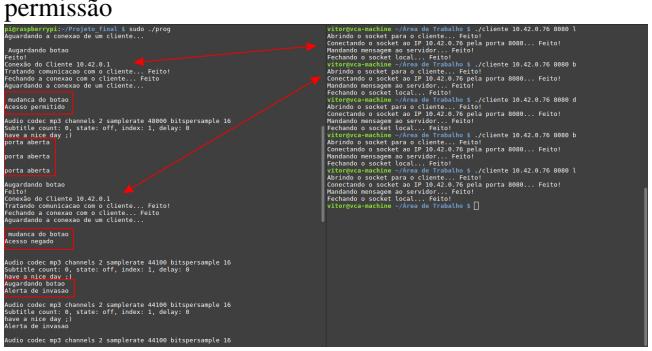
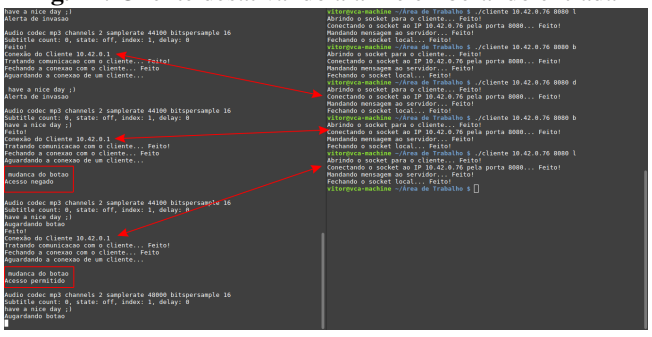


Fig. 10. Cliente liberando porta, porta abrindo com e sem permissão



Aqui, o primeiro comando do cliente é para liberar a entrada. Pode-se observar que o programa só passa a obedecer novamente quando a porta é fechada. NNa segunda instrução, o cliente pede que o acesso seja bloqueado. E então ao pressionar o botão, o sistema informa "Acesso negado". Após isso foi simulada a abertura da porta por arrombamento, sem que o usuário fosse reconhecido. Imediatamente o alerta de invasão foi ativado.

Fig. 11. Cliente desativando alarme e liberando entrada



Aqui, a primeira instrução do cliente é para desativar o alarme. Ocorrem atrasos até que o alarme seja completamente desligado. Na segunda instrução, o cliente bloqueia o acesso e com isso também ativa o alarme. Ao pressionar o botão, o acesso é negado. Então o cliente envia o comando para liberar a porta, e então ao pressionar o botão o acesso é liberado.

4. DISCUSSÃO E CONCLUSÕES

A arquitetura multi-thread permitiu que os periféricos fossem controlados simultaneamente. Isso é fundamental para o projeto, tanto do ponto de vista de experiência do usuário, que não precisa esperar o término de alguma rotina para que a função de interesse seja executada, quanto para a segurança do sistema, visto que há uma vigilancia permanente do estado da porta.

Um dos problemas que a dupla teve durante o desenvolvimento do software, foi o compartilhamento de variáveis entre as threads e processos pai e filho. Inicialmente a alteração das variáveis utilizadas como flags foi feita no processo pai, e as threads criadas no processo filho realizava as leituras. Porém isso não funcionou, porquê os processos não compartilham valores de variáveis, apenas suas declarações. Esse problema foi resolvido realizando todas as operações com variáveis flags dentro do mesmo processo.

Para o encerramento do programa via comando CTRL+C, foi necessário utilizar a captura do sinal SIGINT e encaminhar para uma função de encerramento. Esta realiza o cancelamento das threads e o

O servidor é um programa separado, e não uma função e nem um processo filho. Logo, para o código principal receber comandos através do servidor, foi necessário utilizar métodos de escrita em arquivo para comunicar os dois processos.

Uma limitação das bibliotecas é que elas não diferenciam rostos reais de rostos em fotos mostradas para a câmera. Isso é um grande problema de segurança para o projeto, porém a dupla já está estudando técnicas de diferenciação destes casos.

5. REFERENCIAS

[1] https://www.filipeflop.com/blog/acionando-trava-eletrica-com-rfid/

[2] https://pypi.org/project/pyTelegramBotAPI/0.2.9/

[3] https://core.telegram.org/bots/api

[4] https://www.raspberrypi.org/documentation/usage/webcams/README.md

[5] https://www.raspberrypi.org/documentation/usage/audio/README.md

[6] https://medium.com/@rosbots/ready-to-use-image-raspbian-stretch-ros-opencv-324d6f8dcd96

[7] https://github.com/opencv/opencv

[8] https://github.com/ageitgey/face_recognition

6. APENDICE

Códigos utilizados

Função principal: *main.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/poll.h>
5  #include <unistd.h>
6  #include <pthread.h>
7  #include <signal.h>
8  #include <wiringPi.h>
9  #include <sys/wait.h>
10 #include <string.h>
11
12 #include"funcoes.h"
13
14 int fim_curso = 0;
15 int campainha = 2;
16 int alarme = 3;
17
18 int porta_aberta = 0; //sensor fim de curso
       na porta
19 int reconhecido = 0; //sinal
20 int alerta = 0; //alarme de invasão
21 int child_pid;
22
23 int encerrar=0;
24
25
26 pthread_t id_campainha;
27 pthread_t id_alarme;
28 pthread_t id_porta;
29
30 void encerra_prog(int sig);
31 void encerra_threads(int sig);
32 void* thread_campainha(void*arg);
33 void* thread_alarme(void*arg);
34 void* thread_porta(void*arg);
35
36 int main(int argc, char const *argv[]) {
37
38   wiringPiSetup();
39   pinMode(fim_curso, INPUT);
40   pinMode(alarme, OUTPUT);
41
42   if (fork() == 0){
43       child_pid = getpid();
44       signal(SIGINT,encerra_threads); //
             direcionando sinal de interrupção
             (CTRL+C)
45
46       pthread_create(&id_campainha,NULL,&
             thread_campainha,NULL); //criando
             thread para Campainha
47       pthread_create(&id_alarme,NULL,&
             thread_alarme,NULL); //criando
             thread para Campainha
48       pthread_create(&id_porta,NULL,&
             thread_porta,NULL); //criando
             thread para Campainha
49         int a;
50
51       char comando;
52       while (!encerrar) {
53         a = open("msgs_admin.txt", O_RDONLY)
               ;
54         read(a,&comando,1);
55         close(a);
56         sleep(1);
57         if (comando=='d'){
58           alerta = 0;
59         }
60         if (comando=='1'){
61           reconhecido = 1;
62         }
63         if (comando=='b'){
64           reconhecido = 0;
65         }
66       }
67   }
68
69   if (fork()==0){//filho 2
70       system("./servidor 8080"); // Executa
             o servidor
71   }
72
73   signal(SIGINT,encerra_prog); //
         direcionando sinal de interrupção (
         CTRL+C)
74
75   while (!encerrar);
76
77   wait(NULL);
78   wait(NULL);
79
80   return 0;
81 }
82 void encerra_prog(int sig){
83   encerrar = 1;
84
85 }
86 void encerra_threads(int sig){
87   alerta = 0;
88
89   puts("Encerrando...");
90   encerrar = 1;
91   if (pthread_cancel(id_campainha) ==-1){
92     puts("tread da campainha nao foi
           cancelada");
93   }
94   if (pthread_cancel(id_alarme) ==-1){
95     puts("tread do alarme nao foi cancelada"
           );
96   }
```

```
97
98    if (pthread_cancel(id_porta) ==-1){
99      puts("tread da port nao foi cancelada");
100   }
101 printf("threads canceladas\n" );
102   pthread_join(id_campainha,NULL);
103   pthread_join(id_alarme,NULL);
104   pthread_join(id_porta,NULL);
105
106   system("gpio unexportall");
107  puts("Programa encerrado pelo administrador
       !");
108
109   exit(1);
110
111 }
112
113 void* thread_campainha(void*arg){
114 pthread_setcancelstate(PTHREAD_CANCEL_ENABLE
       , NULL);
115
116   while (!encerrar) {
117
118     poll_bot();
119     if(porta_aberta == 0 && alerta== 0) { //
          só inicia reconhecimento se a porta
          estiver fechada
120       if (reconhecido == 1){ //se o usuário
            for cadastrado
121         printf("Acesso permitido\n\n");
122         system("sudo ./abre.sh");
123
124
125         while(porta_aberta==1 && !encerrar){
126             printf("porta aberta\n\n");
127             sleep(1);
128         } //espera porta fechar
129         reconhecido = 0;
130       }
131       else{ //usuário nao cadastrado
132         puts("Acesso negado\n\n");
133         system("sudo ./negado.sh");
134
135       }
136
137     }
138   }
139   pthread_exit(0);
140
141 }
142
143 void* thread_alarme(void*arg){
144   while(!encerrar){
145       if(alerta == 1 && !encerrar){
146           printf("Alerta de invasao\n\n");
147           system("sudo ./alarme.sh");
148           if (encerrar ==1){
149             pthread_exit(0);
150           }
151       } //espera administrador desativar
            alarme;
152     }
153   pthread_exit(0);
154 }
155
156 void* thread_porta(void*arg){
157   pthread_setcancelstate(
        PTHREAD_CANCEL_ENABLE, NULL);
158   while(!encerrar){
159     porta_aberta = digitalRead(fim_curso);
160     if(porta_aberta == 1 && reconhecido ==
          0){
161       alerta = 1;
162       while (alerta == 1 && !encerrar);
163     }
164   }
165   pthread_exit(0);
166
167 }
```

servidor.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <arpa/inet.h>
5  #include <string.h>
6  #include <signal.h>
7  #include <sys/socket.h>
8  #include <sys/un.h>
9
10 int socket_id;
11 void sigint_handler(int signum);
12 void print_client_message(int client_socket)
       ;
13 void end_server(void);
14
15 int main(int argc, char* const argv[]){
16
17         unsigned short servidorPorta;
18         struct sockaddr_in servidorAddr;
19
20         servidorPorta = atoi(argv[1]);
21
22 //Definindo o tratamento de SIGINT
23         signal(SIGINT, sigint_handler);
24
25
26 //       Abrindo o socket local
27         socket_id = socket(PF_INET,
              SOCK_STREAM, IPPROTO_TCP);
28         if(socket_id < 0)
29         {
30                 fprintf(stderr, "Erro na
                      criacao do socket!\n");
31                 exit(0);
32         }
```

```
33
34
35  //Ligando o socket a porta
36          memset(&servidorAddr, 0, sizeof(
                servidorAddr)); // Zerando a
                estrutura de dados
37          servidorAddr.sin_family = AF_INET;
38          servidorAddr.sin_addr.s_addr = htonl
                (INADDR_ANY);
39          servidorAddr.sin_port = htons(
                servidorPorta);
40          if(bind(socket_id, (struct sockaddr
                *) &servidorAddr, sizeof(
                servidorAddr)) < 0)
41          {
42                  fprintf(stderr, "Erro na
                        ligacao!\n");
43                  exit(0);
44          }
45
46  //Tornando o socket passivo para virar um
        servidor
47          if(listen(socket_id, 10) < 0)
48          {
49                  fprintf(stderr, "Erro!\n");
50                  exit(0);
51          }
52
53          while(1)
54          {
55                  int socketCliente;
56                  struct sockaddr_in
                        clienteAddr;
57                  unsigned int clienteLength;
58
59                  fprintf(stderr, "Aguardando
                        a conexao de um cliente
                        ...\n\n");
60                  clienteLength = sizeof(
                        clienteAddr);
61                  if((socketCliente = accept(
                        socket_id, (struct
                        sockaddr *) &clienteAddr
                        , &clienteLength)) < 0)
62                          fprintf(stderr, "
                                Falha no accept
                                ().\n");
63                  fprintf(stderr, "Feito!\n");
64
65                  fprintf(stderr, "Conexão do
                        Cliente %s\n", inet_ntoa
                        (clienteAddr.sin_addr));
66
67                  fprintf(stderr, "Tratando
                        comunicacao com o
                        cliente... ");
68                  print_client_message(
                        socketCliente);
69                  fprintf(stderr, "Feito!\n");
70
71                  fprintf(stderr, "Fechando a
                        conexao com o cliente...
                        ");
72                  close(socketCliente);
73                  fprintf(stderr, "Feito\n");
74          }
75          return 0;
76  }
77
78  void sigint_handler(int signum)
79  {
80          fprintf(stderr, "\nRecebido o sinal
                CTRL+C... vamos desligar o
                servidor!\n");
81          end_server();
82  }
83
84  void print_client_message(int client_socket)
85  {
86    FILE *arq;
87    arq = fopen("msgs_admin.txt", "wb");
88
89          int length;
90          char text;
91          read(client_socket, &length, sizeof
                (length));
92          read(client_socket, &text, 1);
93    putc(text,arq); //Escreve no arquivo de
          transição;
94    fclose(arq);
95          if (text=='s')
96          {       fprintf(stderr, "Cliente
                pediu para o servidor fechar.\n"
                );
97                  end_server();
98          }
99  }
100
101 void end_server(void)
102 {
103         fprintf(stderr, "Fechando o socket
                local... ");
104         close(socket_id);
105         fprintf(stderr, "Feito!\n");
106         exit(0);
107 }
```

---

abre.sh:

```
1  #!/bin/bash
2
3  GPIO_PATH=/sys/class/gpio
4
5  omxplayer -o local /home/pi/embarcados/
       projeto_final/sons/sim.mp3
6  echo 4 >> $GPIO_PATH/export
7  sudo echo out > $GPIO_PATH/gpio4/direction
```

```
sudo echo 1 > $GPIO_PATH/gpio4/value
sleep 3
echo 0 > $GPIO_PATH/gpio4/value
echo 4 >> $GPIO_PATH/unexport
```

negado.sh:

```
#!/bin/bash
omxplayer -o local ./sons/nao.mp3
```

alarme.sh:

```
#!/bin/bash
omxplayer -o local ./sons/alarme.mp3
```

poll_bot.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/poll.h>
#include <unistd.h>

#include "funcoes.h"


int poll_bot()
{
        struct pollfd pfd;
        char buffer;
        system("echo 27 > /sys/class/gpio/export");
        system("echo falling > /sys/class/gpio/gpio27/edge");
        system("echo in > /sys/class/gpio/gpio27/direction");
        pfd.fd = open("/sys/class/gpio/gpio27/value", O_RDONLY);
        if(pfd.fd < 0)
        {
                puts("Erro abrindo /sys/class/gpio/gpio27/value");
                puts("Execute este programa como root");
                return -1;
        }
        read(pfd.fd, &buffer, 1);
        pfd.events = POLLPRI | POLLERR;
        pfd.revents = 0;
        puts("Augardando botao");
        poll(&pfd, 1, -1);
        if(pfd.revents) puts("mudanca do botao");
        usleep(500000);
        close(pfd.fd);
        system("echo 27 > /sys/class/gpio/unexport");
        return 0;
}
```

Obs: *poll_fim_curso.c* é identico ao *poll_bot.c*, apenas trocando a GPIO 27 para a 17

negado.sh:

```
#!/bin/bash
omxplayer -o local ./sons/nao.mp3
```

alarme.sh:

```
#!/bin/bash
omxplayer -o local ./sons/alarme.mp3
```

face_reco.py

```
import face_recognition

known_image = face_recognition.load_image_file("imagens/vitinho.jpg")
unknown_image = face_recognition.load_image_file("imagens/unknown.jpg")

biden_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([biden_encoding], unknown_encoding)
arq = open('lista_verificacao.txt','w')
arq.write(str(results))
arq.close()
```