RECONHECIMENTO DE COMANDOS DE VOZ PARA A PLATAFORMA EPOSMOTE3 PARA A INTERNET DAS COISAS

VOICE COMMAND RECOGNITION FOR EPOSMOTE3 PLATFORM FOR THE INTERNET OF THINGS

CAMPELLI, Luca Fachini [1]
Graduando em Ciências da Computação – UFSC

BONACOSSA, Rodrigo Brandão [2]
Graduando em Ciências da Computação – UFSC

RESUMO

O objetivo deste trabalho, é desenvolver um componente de interpretação de voz para o Sistema Operacional EPOS2, para que em um ambiente de internet das coisas, possa-se cadastrar e utilizar comandos por voz com os sistemas embarcados, além de ser o trabalho final da disciplina INE5424 - Sistemas Operacionais 2, do curso de Ciências da Computação da UFSC.

Palavras-chave: Reconhecimento de voz, Internet das coisas, EPOS

1.Curso de Ciências da Computação - Universidade Federal de Santa Catarina - http://cco.inf.ufsc.br/ - Matricula: 13200659

2.Curso de Ciências da Computação - Universidade Federal de Santa Catarina - http://cco.inf.ufsc.br/ Matrícula: 13202975

ABSTRACT

The Objective of this work is to develop a voice interpretation component for the EPOS2 Operating System, so that in an Internet of Things context, it be possible to set and use voice commands with the embedded systems, and also serve the purpose of final project for the class INE5424 - Operating Systems 2, from the Computer Science course at UFSC.

***Keywords**: Voice Recognition, Internet of Things, EPOS*

1. INTRODUÇÃO –

A Internet das Coisas (do inglês, Internet of Things) é uma revolução tecnológica a fim de conectar dispositivos eletrônicos utilizados no dia-a-dia (como aparelhos eletrodomésticos, eletroportáteis, máquinas industriais, meios de transporte, etc) à Internet. [3]

Sendo assim, durante as aulas de INE5424 - Sistemas Operacionais 2, o professor Dr. Eng. Rafael Luiz Cancian (a partir de agora simplesmente referido como Prof. Cancian) incentiva os alunos da matéria a aprender e se interessarem nesta área que é o futuro da computação dentro de residências, os dando trabalhos para a matéria dentro deste escopo.

Dentro deste contexto, nos foi atribuída a tarefa de desenvolver um componente para o sistema operacional EPOS2, em que fosse possível efetuar o controle do sistema via comandos de voz, em que fosse utilizada a menor banda possível, para que houvesse a maior eficiência possível.

3.https://pt.wikipedia.org/wiki/Internet_das_coisas

2. PLANEJAMENTO

Durante as semanas iniciais do projeto, nos foram passados 5 (cinco) requisitos funcionais (RF) e 2 (dois) não funcionais (RNF) para que o projeto fosse considerado completo pelo cliente (Prof. Cancian):

RF01	Desenvolver um componente do SO que receba áudio de uma fonte analógica e extraia informações sobre a amplitude em cada faixa de frequência audível
RF02	Com base no processamento do áudio (domínio da frequência), descobrir se há algum som ambiente (além do ruído de fundo) ou não
RF03	Com base no processamento do áudio (domínio da frequência), descobrir se o som ambiente corresponde à voz humana
RF04	Se houve voz humana sendo emitida por cerca de 3~4 segundos e depois parou por pelo menos 1 segundo, identificar isso como um possível comando de voz e enviar o mínimo de informação possível pela rede sem fio que permita a posterior identificação das palavras ditas
RF05	Receber uma mensagem pela rede sem fio e, se for um comando reconhecido, executá-lo
RNF01	O sistema deve executar sobre a plataforma EposMote III (desenvolvimento e testes podem ser feitos inicialmente sobre Epos para PC)
RNF02	A banda utilizada para transmissão do áudio processado deve ser mínima.

Tabela 1: Requisitos do projeto

Dessa forma o planejamento resultou em 11 (onze) partes que se efetuadas corresponderiam às expectativas:

Obs: Em primeira instância, foi-se pensado em utilizar o Google Speech API [4], que se utiliza de um arquivo no formato flac, como ferramenta de reconhecimento de voz, portanto o plano foi-se baseado nessa idéia.

4.https://cloud.google.com/speech/?utm_source=google&utm_medium=cpc&utm_campaign=2015-q2-cloud-latam-solutions-bkws-freetrial-en

1	Checagem periódica lenta, a procura de um possível comando. (StandBy)
2	Caso reconheça uma alteração no sinal começar a gravar, se após 3-4 seg a alteração não cessar, descartar.
3	Se cessar é um possível comando.
4	Dividir dados em sessões de 20ns, e aplicar DFT (Discrete Fourier Transform) para transformar no domínio da frequência
5	Remover frequências indevidas.
6	Se possível comprimir os dados.
7	Enviar para o computador
8	No computador descomprimir e decodificar de volta ao domínio de tempo
9	Gerar um arquivo .flac
10	Enviar para a API
11	Receber resposta e tratar comando.

Tabela 2: Planejamento inicial de projeto.

3 - REALIZAÇÃO COMPONENTE SO

Desde o começo foi pensada na necessidade de duas partes para o componente, uma parte que seja o objeto inteligente, ou seja que amostre o conteúdo e proceda com os comandos recebidos, e outra que servisse de hub, ou base de controle que ficaria conectada ao computador, além de um aplicativo dentro do computador base, que efetuaria o reconhecimento de voz, e enviaria a resposta.

3.1 - Código do Componente

```
 3
 4 #ifndef RECORDER_H
 5 #define RECORDER_H
 6
 7 #include "adc.h"
 8 #include "nic.h"
 9 #include "mutex.h"
10 #include "thread.h"
11 #include "usb.h"
12 #include "alarm.h"
13 #include <utility/ostream.h>
14 //#include "utility/string.h"
15
16 #define NOISE 20
17 #define SIZE 160
18 #define F_PERIOD 500
19 #define S_PERIOD 1000
20
21
22 using namespace EPOS;
23 __BEGIN_SYS
24
25 OStream cout;
26 ADC adc;
27 NIC * nic;
28 Mutex mutex;
29 Mutex Smutex;
30 Mutex Rmutex;
31 static const IEEE802_15_4::Protocol PROTOCOL = IEEE802_15_4::ELP;
32 bool action = false;
33
34 typedef void(Function)();
35
36 Function * _function[2];
37 char* _command[2];
38
39
40 //prints-----------------------------------------------
41 template <typename T>
42 void print(T array){
43     for(int j = 0; j < SIZE; j++){
44         cout<<(array[j]);
45     }
46     //cout<<"TICK"<<endl;
47 }
48
49 //-----------------------------------------------------
50
```

```
51 template <typename T>
52 int absMean(T array[], int size, int zero){
53     int sum = 0;
54     for(int i = 0; i < size; i++){
55         int temp = array[i];
56         temp -= zero;
57         if(temp < 0){
58             temp = -temp;
59         }
60         temp += zero;
61         sum += temp;
62     }
63     return sum/size;
64 }
65
66 /*/-------------------------------------------------
67
68 char* contains(char * s1, char* t2){
69     return 0;
70 }
71
72 *///-------------------------------------------------
73 //NIC-----------------------------------------------
74 template<typename T>
75 int send(T array[]){
76     NIC::Address dst = nic->broadcast();
77
78     Smutex.lock();
79     if(nic->send(dst,PROTOCOL,array,80)){
80         //cout<<"Sent"<<endl;
81     }
82     nic->send(dst,PROTOCOL,&array[80],80);
83     Smutex.unlock();
84     return 0;
85 }
86
87 int send(char cmd){
88     NIC::Address dst = nic->broadcast();
89
90     Smutex.lock();
91     if(nic->send(dst,PROTOCOL,&cmd,sizeof(unsigned char))){
92         //cout<<"Sent"<<endl;
93     }
94     Smutex.unlock();
95     return 0;
96 }
97
98 //-------------------------------------------------
```

```
 99
100 bool contains(char* frase, char* token){
101     int i = 0;
102     int k = 0;
103     while(frase[i] != '\0'){
104         if(frase[i] == token[k]){
105             int j = i;
106             while(token[k] != '\0'){
107                 if(frase[i] != token[k]){
108                     i = j;
109                     k = 0;
110                     break;
111                 } else {
112                     i++;
113                     k++;
114                 }
115             }
116             if(i != j){
117                 return 1;
118             }
119         }
120     i++;
121     }
122     return 0;
123 }
124
125 //Classes----------------------------------------
126
127 class Sender : public IEEE802_15_4::Observer{
128         typedef char data_type;
129
130     public:
131         typedef IEEE802_15_4::Protocol Protocol;
132         typedef IEEE802_15_4::Buffer Buffer;
133         typedef IEEE802_15_4::Frame Frame;
134         typedef IEEE802_15_4::Observed Observed;
135
136     public:
137         Sender(){
138             if(action){
139                 Thread thread1(&sample);
140                 Thread thread(&StandBy);
141                 thread.join();
142                 thread1.join();
143             } else {
144                 act();
145             }
146
```

```
146
147         }
148
149         static int sample(){
150             int i = 0;
151             char samples[SIZE];
152
153             while(true){
154                 mutex.lock();
155                 int times = 0;
156                 cout<<',';
157                 send(',');
158                 while(times < 3000){
159                     samples[i] = adc.read();
160                     i++;
161                     samples[i] = adc.read();
162                     i++;
163                     samples[i] = adc.read();
164                     i++;
165                     samples[i] = adc.read();
166                     i++;
167
168                     if(i > SIZE){
169                         i= 0;
170                         send(samples);
171                         //print(samples);
172                     }
173                     times++;
174                     Alarm::delay(F_PERIOD);
175                 }
176                 cout<<'.';
177                 send('.');
178                 mutex.unlock();
179                 Alarm::delay(F_PERIOD);
180             }
181
182         }
183
184
185         static int StandBy(){
186             USB * usb = new USB();
187
188             int i = 0;
189             char samples[SIZE/4];
190             int media = 0;
191             int mediaTotal = 0;
192             bool locked = false;
193             while(true){
```

```
194                 if(!locked){
195                     mutex.lock();
196                     locked = true;
197                 }
198                 samples[i] = adc.read();
199                 media = absMean(samples, SIZE/4,0);
200                 cout<<"media: " << media << endl;
201                 if(usb->get() == 'o'){//media > NOISE + 64){
202                     cout<<"ok"<<endl;
203                     mutex.unlock();
204                     locked = false;
205                     Alarm::delay(F_PERIOD);
206                 }
207                 i++;
208                 if(i > SIZE/4){
209                     i = 0;
210                 }
211                 Alarm::delay(F_PERIOD);
212             }
213
214
215         }
216
217         //receives a command from PC and acts
218
219         void update(Observed * o, Protocol p, Buffer * b){
220             cout << "Received buffer" << reinterpret_cast<void *>(b) << endl;
221             if(p == PROTOCOL) {
222                 Frame * f = reinterpret_cast<IEEE802_15_4::Frame *>(b->frame());
223                 data_type * d = f->data<data_type>();
224                 cout << endl << "====================" << endl;
225                 cout << "Received " << b->size() << " bytes of payload from " << f->src() << " :" << endl;
226                 //for(int i=0; i<b->size()/sizeof(data_type); i++)
227                   //  cout << d[i] << " ";
228                 bool started = false;
229
230                 for(int j = 0; j < b->size()/sizeof(data_type); j++){
231                     if(!started){
232                         if(d[j] == 'C' && d[j+1] == 'T' && d[j+2] == 'F'){
233                             started = true;
234                             j += 2;
235                         }
236                     } else {
237                         if(contains(d,_command[0])){
238                             cout<<"Lights on"<<endl;
239                             if(_function[0]){
240                                 _function[0]();
```

```
241                             } else {
242                                 cout<<"Function not set"<<endl;
243                             }
244                         }
245                         if(contains(d,_command[1])){
246                             cout<<"Lights off"<<endl;
247                             if(_function[1]){
248                                 _function[1]();
249                             } else {
250                                 cout<<"Function not set"<<endl;
251                             }
252                         }
253                         break;
254                     }
255                 }
256                 cout << endl << "====================" << endl;
257                 nic->free(b);
258             }
259         }
260 
261         int act(){          //done nic needs testing
262             nic->attach(this,PROTOCOL);
263             while(1);
264         }
265 
266 };
267 
268 class Receiver : public IEEE802_15_4::Observer{
269 
270     typedef char data_type;
271 
272 public:
273     typedef IEEE802_15_4::Protocol Protocol;
274     typedef IEEE802_15_4::Buffer Buffer;
275     typedef IEEE802_15_4::Frame Frame;
276     typedef IEEE802_15_4::Observed Observed;
277 
278     public:
279         Receiver(){
280             if(action){
281                 toPC();
282             } else {
283                 Thread thread1(&toEPOS);
284                 thread1.join();
285             }
286         }
287 
```

```
335     void addFunc(Function * f,char* command, int place){
336         cout<<"changed"<<endl;
337         _function[place] = f;
338         _command[place] = command;
339     }
340
341     void start(){
342         USB* usb = new USB();
343         if(usb->get() == 'T'){
344             action = true;
345         } else {
346             action = false;
347         }
348         delete usb;
349         nic = new NIC();
350         IF_CLASS<Traits<REC>::send,Sender,Receiver>::Result op;
351     }
352 };
353
```

Imagem 1: Código do Componente em 8 partes

Estas duas partes seriam as classes Sender e Receiver, que respectivamente, amostra do microfone enquanto analisa para enviar, e roteia mensagens de e para o computador.
Utilizando-se de metaprogramação, foi-se feita uma classe central que em tempo de compilação escolhe qual das duas deve ser a classe ativa, esta é a classe REC.
Ela permite ao usuário cadastrar as funções que desejar com os respectivos comandos customizados, utilizando-se da função addFunc(3), e então iniciar a utilização do componente via a função start();

```
struct Traits
{
    static const bool enabled = true;
    static const bool debugged = true;
    static const bool hysterically_debugged = false;
    typedef TLIST<> ASPECTS;

    static const bool send = true;
};
```

Imagem 2: Excerto da classe Traits, mostrando o atributo que define qual classe será utilizada

3.2 - Sender

A classe Sender, tem como objetivo fazer a amostragem do microfone e o envio ao computador, além do recebimento e tratamento dos comandos reconhecidos pelo computador. Ela possui três Threads, onde a primeira se encarrega da amostragem mais lenta, o modo standby, a segunda do modo amostragem, e a terceira de receber e efetuar comandos.
A função StandBy(), cuida de fazer amostragens mais lentas, 2000 por segundo, onde ela checa a média de cada segundo, para que caso as médias de dois segundos consecutivos variem mais que um certo ruído (NOISE) ela destrave um mutex que impedia a Thread de

amostragem de rodar e tente travá-lo outra vez, assim, dando a vez para a Thread de amostragem. Quando a amostragem terminar, o mutex é liberado, e a Thread StanBy retoma a execução.

A função sample() cuida da amostragem em si. Ela faz amostragens a uma frequência de 8000Hz, em modo PCM 8bits, para depois enviar ao EPOSMote base, pacotes com 20 ns de duração, retirado do protocolo G721, o mesmo usado por telefones fixos.
Ela espera sua execução graças ao mutex que a suspende enquanto o programa está em modo standby.

A função act(), configura um observador para que o EPOS possa receber mensagens via NIC, onde na Função update, ele faz o reconhecimento dos comandos recebidos.

3.3 - Receiver

A classe Receiver funciona como um roteador, recebendo e encaminhando as mensagens entre o computador e os EPOSMote.

A função toPC() liga um observador ao objeto, e assim na função update, ele encaminha as mensagens recebidas para o computador.

A função toEPOS(), encaminha as mensagens recebidas do computador para os EPOSMote,

4. PROBLEMAS ENCONTRADOS

Ao longo do projeto vários problemas foram encontrados, vários dos quais são provenientes do fato de o Sistema Operacional EPOS 2 ainda estar em desenvolvimento e aperfeiçoamento.

1 - Para se amostrar a 8000Hz, é necessário que se faça uma amostra a cada 125us. Em primeira instância tentamos utilizar Threads periódicas, que manteriam o período constante, porém aparentemente, o EPOSMOTE não possui a capacidade de marcar períodos menores que 500us, com qualquer tipo de ferramenta de marcação de tempo, resultando ou em Kernel Panic, ou em o programa ignorar completamente a pausa. Além de que aparentemente o sincronismo entre Threads Periódicas não funciona muito bem [5]. Assim foi-se escolhido marcar os tempos com um delay do alarm, que possui uma resposta melhor aos escalonamentos necessários, porém mesmo assim, o menor período possível, é 500us, portanto por sugestão do professor, são feitas 4 amostras por período de amostragem, o que embora não garanta qualidade garante a taxa de amostragem.
Devido ao escalonador não escalonar corretamente as threads, por motivos de demonstração foi feito uma divisão do programa em duas partes, o caminho de ida para o PC e o de volta para o EPOS, controlados por um comando USB já que o EPOSMOTE não foi apto a escutar a NIC e a USB ao mesmo tempo.[5]

5. Arquivos de teste minimo: ADCTest(_ ,1, 2, 3).cc e ThreadsTest(1, 3, 4, 8, 9).cc

2 - Para que os requisitos fossem cumpridos, deveriam ser amostrados de 3 a 4 segundos de áudio, e depois 1 segundo de silêncio, então feita uma transformada de fourier, retiradas as frequências indesejadas e comprimida a informação para depois ser enviada, porém houveram problemas para o fato de que a memória do EPOSMote não é grande o suficiente para armazenar nem 1 segundo de áudio. Entrando em panic antes mesmo de iniciar a rodar. [6]
A situação exigiu uma reformulação do programa para a versão atual, onde são gravados e enviados os 3 segundos ao mesmo tempo. Embora não acate com o requisito foi o único jeito encontrado para que o programa se comportasse como deveria, mediante a aparente falta de memória.

3 - A transformada de fourier discreta, juntamente com senos e cossenos, necessitam de números em pontos flutuantes, e o EPOS não possui FPU, portanto não consegue trabalhar com pontos flutuantes. Além de que uma “desflutuação” das funções de seno, cosseno e dft são muito complicadas e trabalhosas, além de que sofreriam de uma perda de qualidade muito grande para o escopo do trabalho.

4 - A escolha do mecanismo de reconhecimento de voz foi mudada, já que o google speech api é um serviço online que necessitaria de uma conexão estável com a internet, além de exigir uma taxa de manutenção para que se possa usá-lo. Então foi substituído pelo CMU-Sphinx. Uma API de reconhecimento de voz offline, de código aberto e escrita em JAVA. [7]

5. Houveram problemas com o arquivo de som gerado pelas amostras do microfone, que não são inteligíveis, embora por lógica possa-se discernir onde estão as palavras ficaria impossível discernir o que se foi dito.[8]

6. A restrição do aplicativo de reconhecimento de voz, na questão de como deve ser gravado o arquivo. Especificamente ele deve ser um arquivo WAV gravado a 16000Hz em 16bits PCM,
Impossível de fazer no momento já que a velocidade máxima de amostragem do ADC não supera o necessário para isso.

7. O componente NIC do EPOSMote consegue enviar apenas 116 bytes por vez.[9]

6.Arquivo de exemplo minimo: EXMINMEM.cc
7.http://cmusphinx.sourceforge.net/
8.Arquivo TesteDeSom.wav (Analise usando Audacity ou qualquer editor de som de sua preferencia)
9.Arquivo NIC.cc

5. REALIZAÇÃO APLICATIVO COMPUTADOR

A aplicação para o computador foi realizada em duas partes: uma que recebia as informações do EPOSMote base e enviaria o comando de volta, e uma que faria o reconhecimento de voz.
Desde o começo foi decidido que a aplicação seria desenvolvida em C++, mesmo o mecanismo de reconhecimento de voz escolhido sendo em JAVA.
Assim foi feito uma aplicação em C++ que recolhe as amostras enviadas pelos EPOSMote e envia os comandos recebidos pelo aplicativo CMU-Sphinx. [8]

Suas funcionalidades são divididas nas funções TakeFromUSB e InsertToUSB, onde o primeiro se encarrega de recolher as amostras e criar um arquivo .wav com elas, e a última envia o comando recebido de volta do reconhecimento de voz para o EPOSMote para que ele seja transmitido para os objetos inteligentes.
A classe WAV.cpp cuida da criação e edição do arquivo .wav, onde o construtor confecciona o cabeçalho do arquivo e a função addSample() insere uma amostra no arquivo, que por fim é finalizado com endWav(), que completa o cabeçalho com o tamanho final do arquivo e o fecha.

```
/*
 * File:   main.cpp
 * Author: rodrigo
 *
 * Created on October 26, 2016, 9:50 PM
 */

#include <cstdlib>
#include "Usb_Extractor.h"
#include <pthread.h>
#include <stdio.h>
//#include <iostream>

void USB_Ext() {
    Usb_Extractor* extractor = new Usb_Extractor();
    extractor->takeFromUSB();
}

void USB_Send() {
    Usb_Extractor* extractor = new Usb_Extractor();
    extractor->USB_Sender();
}

int main(int argc, char *argv[]) {
    int status;
    if (fork() == 0) { //Filho
        USB_Ext();
        status = system("my_app");
    } else { //Father
        if (fork() == 0) { //Filho
            USB_Send();
            status = system("my_app");
        } else { //Father

        }
    }
    while(1) sleep (100);
}
```

Imagem 3: main.h

```
/*
 * File:   Usb_Extractor.h
 * Author: rodrigo
 *
 * Created on October 26, 2016, 9:53 PM
 */
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>

#ifndef USB_EXTRACTOR_H
#define USB_EXTRACTOR_H

class Usb_Extractor {

private:

    int set_interface_attribs(int fd, int speed);
    void set_mincount(int fd, int mcount);

public:

    Usb_Extractor();
    int takeFromUSB();
    void InsertInUSB(char* string);
    void USB_Sender();
};


#endif  /* USB_EXTRACTOR_H */

```

Imagem 4: Usb_Extractor.h

```cpp
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sstream>
#include <termios.h>
#include <unistd.h>
#include <fstream>

#include "Usb_Extractor.h"
#include "Wav.h"

Usb_Extractor::Usb_Extractor() {

}

int Usb_Extractor::set_interface_attribs(int fd, int speed) {
    struct termios tty;

    if (tcgetattr(fd, &tty) < 0) {
        printf("Error from tcgetattr: %s\n", strerror(errno));
        return -1;
    }

    cfsetospeed(&tty, (speed_t) speed);
    cfsetispeed(&tty, (speed_t) speed);

    tty.c_cflag |= (CLOCAL | CREAD); /* ignore modem controls */
    tty.c_cflag &= ~CSIZE;
    tty.c_cflag |= CS8; /* 8-bit characters */
    tty.c_cflag &= ~PARENB; /* no parity bit */
    tty.c_cflag &= ~CSTOPB; /* only need 1 stop bit */
    tty.c_cflag &= ~CRTSCTS; /* no hardware flowcontrol */

    /* setup for non-canonical mode */
    tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR | ICRNL | IXON);
    tty.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
    tty.c_oflag &= ~OPOST;

    /* fetch bytes as they become available */
    tty.c_cc[VMIN] = 1;
    tty.c_cc[VTIME] = 1;

    if (tcsetattr(fd, TCSANOW, &tty) != 0) {
        printf("Error from tcsetattr: %s\n", strerror(errno));
        return -1;
    }
    return 0;
}

void Usb_Extractor::set_mincount(int fd, int mcount) {
    struct termios tty;

    if (tcgetattr(fd, &tty) < 0) {
        printf("Error tcgetattr: %s\n", strerror(errno));
        return;
    }

    tty.c_cc[VMIN] = mcount ? 1 : 0;
    tty.c_cc[VTIME] = 5; /* half second timer */

    if (tcsetattr(fd, TCSANOW, &tty) < 0)
        printf("Error tcsetattr: %s\n", strerror(errno));
}

int Usb_Extractor::takeFromUSB() {
    char *portname = "/dev/ttyACM0"; //Port for taking information
    int fd;
    int wlen;
    printf("Trying to Open Archive\n");

```

```
    while (1) {
        while (access("/dev/ttyACM0", F_OK) == -1) sleep(1); //Until file do not exists

        fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
        if (fd < 0) {
            cout << "Error while opening archive" << errno << endl;
            return -1;
        }

        printf("Opened\n");
        /*baudrate 115200, 8 bits, no parity, 1 stop bit */
        set_interface_attribs(fd, B115200);

        /* simple noncanonical input */
        char buf;
        int rdlen;

        while (buf != ',') rdlen = read(fd, &buf, 1); //Caracter início

        printf("Criando Wav:\n");

        Wav* wave = new Wav("teste");

        while (buf != '.') {//Caracter Fim
            rdlen = read(fd, &buf, 1);
            if (rdlen > 0) {

                printf("Sample: %c", buf);
                wave->addSample(buf);

            } else if (rdlen < 0) {
                printf("Error from read: %d: %s\n", rdlen, strerror(errno));
            }
        }
        wave->endWav();
        close(fd);
        printf("Wav Criado\n");
    }
    .
}

void Usb_Extractor::InsertInUSB(char* string) {

    char *portname = "/dev/ttyACM0"; //Port for taking information
    int fd;
    int wlen;

    while (access("/dev/ttyACM0", F_OK) == -1) sleep(1); //Until file do not exists

    fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
    if (fd < 0) {
        cout << "Error while opening archive" << errno << endl;
        return;
    }

    wlen = write(fd, string, 16);
    if (wlen != 7) {
        printf("numero de caracteres: %d\n", wlen);
    }

}

void Usb_Extractor::USB_Sender() {

    while (1) {
        string inicio = "CTF ";
        string fim = "@";
        string output;
        stringstream ss;
        string s;

        while (access("/home/lucampelli/comand.txt", F_OK) == -1) sleep(10); //Until file do not exists
        ifstream comandFile;
        comandFile.open("/home/lucampelli/comand.txt");
```

```
147
148         if (comandFile.is_open()) {
149             getline(comandFile, output);
150             int i = 0;
151             while (output[i]) {
152                 output[i] = toupper(output[i]);
153                 i++;
154             }
155             ss << inicio << output << fim;
156             s = ss.str();
157         }
158
159         char* saida = &s[0];
160         printf("%s\n", saida);
161
162         comandFile.close(); //Fechando arquivo
163         //Apagando comand.txt
164         if (remove("/home/lucampelli/comand.txt") != 0) {
165             perror("Error deleting file");
166         } else {
167             puts("File successfully deleted");
168
169             InsertInUSB(saida);
170         }
171
172     }
173 }
```

Imagem 5: Usb_Extractor.cc em 5 partes

```
1  /*
2   * File:   Wav.h
3   * Author: rodrigo
4   *
5   * Created on October 26, 2016, 9:51 PM
6   */
7  #include <cmath>
8  #include <fstream>
9  #include "string.h"
10 #include <iostream>
11
12 #ifndef WAV_H
13 #define WAV_H
14 /*
15 Cria um arqivo no construtor com cabeçálho
16 pré definido e adiciona samples neste arquivo até receber um término.
17  */
18 using namespace std;
19
20 class Wav {
21 private:
22     size_t data_chunk_pos;
23
24 public:
25
26     Wav(string nome);
27
28     void addSample(int sample);
29
30     void endWav();
31
32 };
33 #endif  /* WAV_H */
```

Imagem 6: Wav.h

```cpp
#include "Wav.h"


namespace little_endian_io {

    template <typename Word>
    std::ostream& write_word(std::ostream& outs, Word value, unsigned size = sizeof ( Word)) {
        for (; size; --size, value >>= 8)
            outs.put(static_cast<char> (value & 0xFF));
        return outs;
    }
}

using namespace little_endian_io;

    //Starting the header:
    ofstream f("/home/lucampelli/Test1.wav", ios::binary);


    Wav::Wav(string nome) {

        //char* t= strcat(strdup(nome.c_str()),".wav");



        // Write the file headers
        f << "RIFF----WAVEfmt "; // (chunk size to be filled in later)
        write_word(f, 16, 4); // no extension data
        write_word(f, 1, 2); // PCM - integer samples
        write_word(f, 1, 2); // two channels (stereo file)
        write_word(f, 8000, 4); // samples per second (Hz)
        write_word(f, 16000, 4); // (Sample Rate * BitsPerSample * Channels) / 8
        write_word(f, 2, 2); // data block size (size of two integer samples, one for each channel, in bytes)
        write_word(f, 8, 2); // number of bits per sample (use a multiple of 8)

        // Write the data chunk header
        data_chunk_pos = f.tellp();
        f << "data----"; // (chunk size to be filled in later)
    }

    void Wav::addSample(int sample) {
        cout<<"Sample"<<sample<<endl;
        write_word(f, sample, 2);
    }

    void Wav::endWav() {
        // (We'll need the final file size to fix the chunk sizes above)
        size_t file_length = f.tellp();
        cout<<"File Length "<<file_length;
        // Fix the data chunk header to contain the data size
        f.seekp(data_chunk_pos + 4);
        write_word(f, file_length - data_chunk_pos + 8);

        // Fix the file header to contain the proper RIFF chunk size, which is (file size - 8) bytes
        f.seekp(0 + 4);
        write_word(f, file_length - 8, 4);

    }


```

Imagem 7: Wav.cc em 2 partes

6.LINKS DO TRABALHO

Git:
https://github.com/lucampelli/20162-INE5424-Agrupamento1-Tema4.5-reconhecimento-de-voz-para-EPOS--grupoOP.git

DropBox:
https://www.dropbox.com/sh/atd7qkdq83o31f3/AACee95pf_G-23OyJnmtQEsIa?dl=0