UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

IF747 – REDES AUTOMOTIVAS

PROJETO – REDE CAN

PROFESSOR: Divanilson Campelo
MONITOR: Paulo Freitas

Rubem Moura

Thaysa Barros

Recife – 2016

PROPOSTA

Como de projeto de Redes Automotivas foi lançada a proposta de implementar um sistema de captura de dados para um veículo em tempo real, cujos dados deveriam constituir um banco de dados. Utilizando esse banco de dados, o objetivo foi simular uma rede CAN de dois nós, dos quais um seria o transmissor da mensagem e o outro o receptor, ou seja, aquele que está escutando o barramento. Além disso, antes ou depois da simulação, os dados deveriam ser convertidos e apresentados através de uma interface gráfica.

TESTES EM CAMPO

Baseado nas solicitações do projeto, o primeiro passo foi adquirir um Scanner Automotivo OBD2, o ELM 327, que envia dados aos aplicativos via Bluetooth. Com o ELM 327 foram realizadas três coletas de dados em um Ford Fiesta 2014, cada uma com um aplicativo diferente, o Torque Free, Torque Pro e OBD Doctor, sendo os dois primeiros usados em celular e o último em um notebook. No entanto, as duas versões do Torque não forneciam dados no formato de frames, que foi estabelecido como objetivo do projeto, e quanto ao OBD Doctor, a aquisição de dados não foi iniciada, apesar de estarem conectados.

Diante dos resultados negativos, outro Scanner foi utilizado, Vgate Bluetooth Wireless, e o OBD Doctor no notebook comunicando-se via Wifi. Nessa configuração, foram obtidos os frames de quatro sensores de um Ford Ka 2015, os quais informam velocidade, rotação, temperatura de resfriamento do motor e temperatura do ar que entra para participar da combustão, essa quantidade de sensores não confere com o solicitado, 5 sensores, mas dentre os muitos sensores disponíveis, esses foram os únicos que geraram informação através desse aplicativo. Os scanners podem ser visualizados na Figura 1 e as interfaces de comunicação, na Figura 2 do Apêndice.

INTERPRETAÇÃO DOS DADOS

Assim, com um arquivo txt gerado pelo OBD Doctor, Figura 3, os dados foram transferidos para o Excel, Figura 4, onde as informações são facilmente separáveis por colunas, a fim de facilitar a visualização e manipulação do frame e a interpretação dele. Nessa figuras com trechos dos dados coletados, está indicada a coluna de direção, que significa o sentido de envio da mensagem: >> indica o envio de uma mensagem de requisição da ECU para o sensor e << indica a mensagem de resposta que parte do sensor e segue para a ECU.

Abaixo seguem exemplos de mensagem requisitada, Tabela 1, e mensagem de reposta, Tabela 2, separados de acordo com o significado de cada caractere.

MODO	PID
01	05

Tabela 1 – Mensagem-requisição

	BYTE						
PID TYPE	0	1	2	3	4	5	6
PID	ADDITIONAL DATA	SHOW CURRENT DATA	DATA	DATA	DATA	DATA	DATA
7E8	03	41	05	7F			
7E8	04	41	0C	0E	EF		
7E8	03	41	0D	00			
7E8	03	41	0F	54			

Tabela 2 – Frame de mensagem-resposta.

Acompanhando a tabela mensagem-requisição, o modo 01, o único que aparece neste teste, pede aos sensores que mostrem seus valores em tempo real, enquanto o PID revela o sensor ao qual a mensagem é destinada.

Seguindo para a tabela de mensagem-resposta, o PID TYPE é um identificador de uma ECU, o 7E8 é o identificador de resposta para a ECU #1 que corresponde ao módulo de controle do motor. As demais colunas formam o *payload*, são ao todo 8 bytes, mas os 6 da tabela serão suficientes para explicar os dados coletados. O byte 0 informa o byte máximo de dado que será enviado, nos exemplos tivemos dois casos, 03 e 04, observe que onde tem 03 o dado está preenchido até o byte 3 e o 04, até o byte 4. O byte 2 é um modo de consulta e 41 indica modo em tempo real, o byte 3 é a identificação do sensor e os seguintes são os que correspondem ao valor assumido pelos sensores. A identificação do sensor ou PID para os sensores utilizados no teste é: 05 para temperatura de resfriamento do motor, 0C para rotação, 0D para velocidade e 0F para temperatura do ar de admissão.

SIMULAÇÃO DA REDE CAN

Seguindo as solicitações, para criar a rede CAN utilizando a SocketCan é necessário ter a versão do Python que tem suporte a esse API, por isso, o primeiro passo foi instalar o Python 3.5, a partir do qual é possível usar funções da biblioteca Phyton CAN para controlar a rede. Ademais, a biblioteca CanUtils também foi necessária para a virtualização da rede através dos comandos da Figura 8.

Com as ferramentas em mãos e usando dois terminais que se comportam como um nó cada, foi possível escrever uma mensagem no barramento dentro de um terminal enquanto o outro escuta, imprime no console a mensagem recebida e salva em outro arquivo. No entanto, ao expandir a simulação

para o banco de dados, os comandos utilizados não foram suficientes e outros comandos foram acrescentados, a CanPlayer, que entende os dados em uma formatação específica e, por isso, o banco de dados foi editado segundo a formatação exigida pelo CanPlayer, sem que os valores fossem alterados; e o CanDump da Canutils para escutar o tráfego na rede CAN e, ao mesmo tempo, armazenar os dados em um arquivo. Essas modificações resultaram na necessidade de um programa em C para retornar os dados para a formatação anterior, a fim de poder usar o programa em C que converte hexadecimal para decimal, feito antes dessa alteração na simulação.

PREPARAÇÃO DOS DADOS

Durante a execução da aquisição de dados, foi determinado que a interpretação do *payload* seria feita antes da simulação para que os próximos passos do projeto fossem feitos com dado tratado, no entanto, tendo em vista a limitação apresentada na simulação usando a função *canplayer*, os dados passaram a ser tratados ao fim da simulação e antes de ser usado na interface gráfica.

Dessa forma, foi desenvolvido um programa em linguagem C que converte de hexadecimal para decimal os itens "DATA" do *payload*, Figura 6. Além disso, é necessário mais do que a simples conversão de base numérica, é preciso aplicar as fórmulas abaixo, segundo a referência [1]. As fórmulas 2 e 3 são, respectivamente, dos sensores 0C, 0D e a fórmula 1 é a mesma para os sensores 05, e 0F.

$$T = A - 40 \qquad (1)$$

$$R = \frac{256A+B}{4} \qquad (2)$$

$$V = A \qquad (3)$$

Observe que o valor A corresponde ao decimal do primeiro byte de dado e B é o decimal do segundo byte de dado.

No código, os dados de mensagem-resposta foram importados de um arquivo txt, iniciando a sequência de eventos que varre todo o banco de dados até que a conversão completa seja feita. Os eventos do programa consistem principalmente em importar um frame – Figura 5, a partir da linha 163, *main* –, converter de hexadecimal para decimal – Figura 5, funções da linha 6 a 70 – e salvar em um novo arquivo – Figura 5, linha 72, dentro da função *processPayload* –, repetindo sequencialmente esses passos até o fim do banco de dados. Esses dados são salvos em um novo arquivo, Figura 7.

INTERFACE GRÁFICA

Para a visualização das informações do banco de dados foi desenvolvida uma página web destinada ao diagnóstico de veículos em tempo real. Essa aplicação foi a motivação para a escolha desse

tipo de interface, mas atualmente está operando como um aplicativo web, no qual o usuário pode inserir seu banco de dados diretamente no diretório do aplicativo e visualizar o comportamento de um carro a partir de dados adquiridos anteriormente.

A interface foi implementada utilizando HTML, CSS E JAVASCRIPT, os quais proporcionaram a criação de um layout com quatro compartimentos, visto que foram coletadas mensagens de quatro sensores, e o uso de animação para indicar as mudanças nos valores lidos dos sensores. Além disso, para a página foi criado um nome fantasia, Frottor, que remete a ideia de Frota e ela pode ser encontrada no repositório GitHub (https://github.com/rubemmoura/if747-RedesAutomotivas).

APÊNDICE

Figura 1 – ELM 327 e Vgate.

Figura 2 – Interfaces para aquisição de dados.

TIMESTAMP	DIRECTION	DATA
2016-11-16 19:37:12	>>	0101
2016-11-16 19:37:13	<<	7E80641010007E100
2016-11-16 19:37:17	>>	0105
2016-11-16 19:37:17	<<	7E80341057F
2016-11-16 19:37:17	>>	010C
2016-11-16 19:37:17	<<	7E804410C0EFE
2016-11-16 19:37:17	>>	010D
2016-11-16 19:37:18	<<	7E803410D00
2016-11-16 19:37:18	>>	010F
2016-11-16 19:37:18	<<	7E803410F54
2016-11-16 19:37:18	>>	0105
2016-11-16 19:37:18	<<	7E80341057F

Figura 3 – Arquivo txt gerado pelo OBD Doctor.

TIMESTAMP		DIRECTION	DATA
16/11/2016	19:37:12	>>	0101
16/11/2016	19:37:13	<<	7E80641010007E100
16/11/2016	19:37:17	>>	0105
16/11/2016	19:37:17	<<	7E80341057F
16/11/2016	19:37:17	>>	010C
16/11/2016	19:37:17	<<	7E804410C0EFE
16/11/2016	19:37:17	>>	010D
16/11/2016	19:37:18	<<	7E803410D00
16/11/2016	19:37:18	>>	010F
16/11/2016	19:37:18	<<	7E803410F54
16/11/2016	19:37:18	>>	0105
16/11/2016	19:37:18	<<	7E80341057F
16/11/2016	19:37:18	>>	010C
16/11/2016	19:37:19	<<	7E804410C0F65
16/11/2016	19:37:19	>>	010D
16/11/2016	19:37:19	<<	7E803410D00
16/11/2016	19:37:19	>>	010F
16/11/2016	19:37:19	<<	7E803410F54

Figura 4 – Banco de dados no Excel.

```
6   int convertHexToInt(int hex_value, int indice_base){
7       int result = 0;
8       int base_hex = 16;
9
10      if(indice_base == 0){
11          base_hex = 1;
12      }
13      result = hex_value * base_hex;
14
15      return result;
16  }
18  int convertLetterToInt(char letter){
19      int result = 0;
20
21      switch(letter){
22          case 'A':
23              result = 10;
24              break;
25          case 'B':
26              result = 11;
27              break;
28          case 'C':
29              result = 12;
30              break;
31          case 'D':
32              result = 13;
33              break;
34          case 'E':
35              result = 14;
36              break;
37          case 'F':
38              result = 15;
39              break;
40      }
41      return result;
42  }
```

Figura 5 – Código de conversão do banco de dados.

```
45  int letterOrNotLetter(char letter){
46      int result;
47      if(letter == 'A' || letter == 'B' || letter == 'C' || letter == 'D' || letter == 'E' || letter == 'F'){
48          result = convertLetterToInt(letter);
49      }else{
50          result =  letter - '0';
51      }
52
53      return result;
54  }
56  int captureOneByte(char byte0, char byte1){
57      //convertendo char para int
58      int value       = 0;
59      int value2      = 0;
60      int valueInt    = 0;
61
62      value   = letterOrNotLetter(byte0);
63      value2  = letterOrNotLetter(byte1);
64
65      value       = convertHexToInt(value, 1);
66      value2      = convertHexToInt(value2, 0);
67      valueInt    = value + value2;
68
69      return valueInt;
70  }
72  int processPayload(char* byte, char* payload){
73
74      char sensorHb = payload[8];
75      char byte0[2];
76      byte0[0] = byte[0];
77      byte0[1] = byte[1];
78
79      char ch;
80      FILE *out;
81      out = fopen("result.txt", "a");
82
83      if(out == NULL){
84              printf("Erro, nao foi possivel abrir o arquivo\n");
85      }else{
86  //          fseek(out, 7, SEEK_END);
87          int result = 0;
88          switch(sensorHb){
89              case '5':
90              //Engine coolant temperature!
91              //A-40
92                  printf(" Engine coolant temperature!\n");
93
94                  int coolantTemperature = 0;
95                  coolantTemperature = captureOneByte(byte0[0], byte0[1]) - 40;
96                  result = coolantTemperature;
97                  fprintf(out,"Coolant Temperature: %d\n", result);
98                  printf("\n Coolant Temperature: %d graus Celsius\n", coolantTemperature);
99                  break;
```

Figura 5 – Código de conversão do banco de dados.

```
            case 'F':
            //Intake air temperature
            //A-40
                printf(" Intake air temperature!\n");

                int intakeAirTemperature = 0;
                intakeAirTemperature = captureOneByte(byte0[0], byte0[1]) - 40;
                result = intakeAirTemperature;
                fprintf(out,"Intake Air Temperature: %d\n", result);
                printf("\n Intake Air Temperature: %d graus Celsius\n", intakeAirTemperature);
                break;
            case 'C':
            //Engine RPM
            //((A*256)+B)/4
                printf(" Engine RPM!\n");

                char byte1[2];
                byte1[0] = payload[11];
                byte1[1] = payload[12];

                int RPM = 0;
                int A   = 0;
                int B   = 0;

                A = captureOneByte(byte0[0], byte0[1]);
                B = captureOneByte(byte1[0], byte1[1]);

                RPM = ((A*256)+B)/4;
                result = RPM;
                fprintf(out,"RPM: %d\n", result);
                printf("\n RPM %d \n", RPM);


                break;
            case 'D':
            //Vehicle speed
            //A
                printf(" Vehicle speed!\n");

                int vehicleSpeed = 0;
                vehicleSpeed = captureOneByte(byte0[0], byte0[1]);
                result = vehicleSpeed;
                fprintf(out,"Vehicle speed: %d\n", result);
                printf("\n Vehicle speed: %d km/h\n", vehicleSpeed);
                break;
        }
    }

    fclose(out);
    printf("\n\n-------------------------------------------------------\n\n");
    return 0;
}
```

Figura 5 – Código de conversão do banco de dados.

```c
int main()
{
    char payload[] = "7E804410C1CEB";

    char *ch;
    int count = 0;
    char string_frame[13];
    FILE *frame;
    frame = fopen("frames.txt", "r");

    if(frame == NULL){
        printf("Erro, nao foi possivel abrir o arquivo\n");
    }
    else{
        while((ch=fgetc(frame))!= EOF){
            while(ch != '\n'){
                string_frame[count] = ch;
                count++;
                ch=fgetc(frame);
            }
            string_frame[count] = '\0';
            count = 0;
            printf(" String_frame = %s \n\n", string_frame);
            char byte0[2];
            byte0[0] = string_frame[9];
            byte0[1] = string_frame[10];

            processPayload(byte0, string_frame);

            *string_frame = NULL;
        }
    }

    fclose(frame);
    return 0;
}
```

Figura 5 – Código de conversão do banco de dados.

```
--------------------------------------------------------

String_frame = 7E803410D1E
Vehicle speed!
Vehicle speed: 30 km/h

--------------------------------------------------------

String_frame = 7E803410F57
Intake air temperature!
Intake Air Temperature: 47 graus Celsius

--------------------------------------------------------

String_frame = 7E803410580
Engine coolant temperature!
Coolant Temperature: 88 graus Celsius

--------------------------------------------------------

String_frame = 7E804410C1BA7
Engine RPM!
RPM 1769

--------------------------------------------------------

Process returned 0 (0x0)   execution time : 27.089 s
Press any key to continue.
```

Figura 6 – Console com a conversão de dados.

```
Coolant Temperature: 87
RPM: 959
Vehicle speed: 0
Intake Air Temperature: 44
Coolant Temperature: 87
RPM: 985
Vehicle speed: 0
Intake Air Temperature: 44
Coolant Temperature: 87
RPM: 1225
Vehicle speed: 2
Intake Air Temperature: 44
Coolant Temperature: 87
```

Figura 7 – Novo arquivo com dados convertidos.

```
sudo modprobe vcan
sudo ip link add dev vcan0 type vcan
sudo ip link set vcan0 up
```

Figura 8 - Virtualização da Rede CAN.

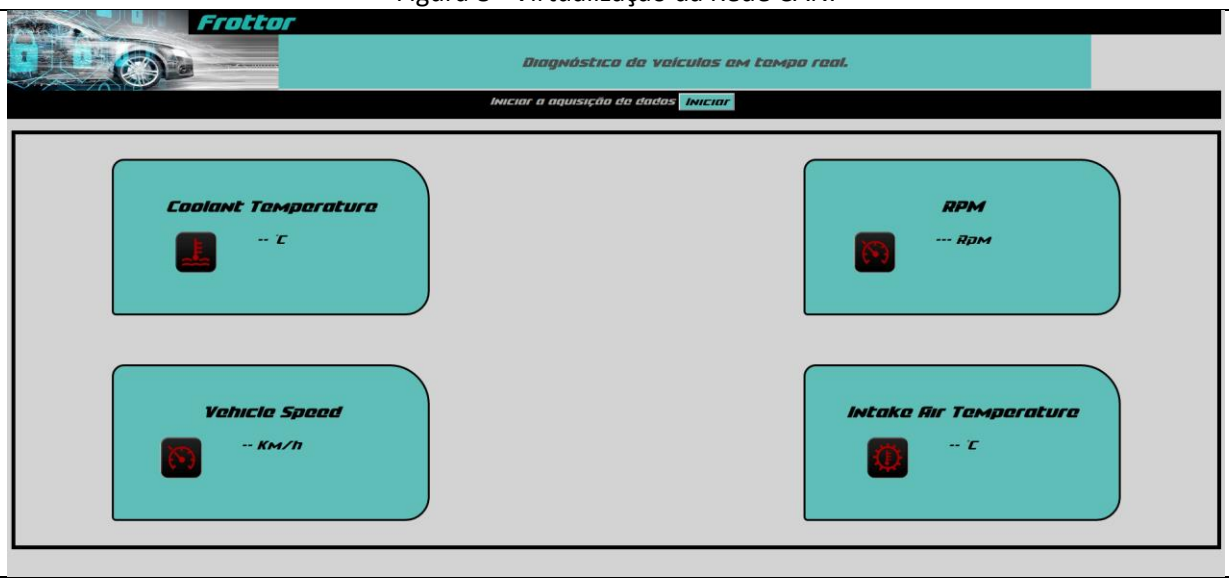

Figura 9 – Página Web.

REFERÊNCIAS

[1] https://en.wikipedia.org/wiki/OBD-II_PIDs#Mode_01

[2] Material de Aula – IF747