Explorando a Eficiência da SocketCAN: Uma Abordagem de Código Aberto para Comunicação CAN

Gabriel Soares[1]

[1]Centro de informática (CIN) – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brazil

gss12@cin.ufpe.br

Abstract. *This paper highlights the convenience and advantages of* `SocketCAN` *in conjunction with* `VCAN`*, offering a broad view of their capabilities and educational benefits.* `SocketCAN` *is a native socket API for CAN controllers in Linux systems, while* `vcan` *is a virtual interface that simulates a CAN bus in the kernel space. By exploring these technologies, this report demonstrates how they streamline the development and debugging of CAN protocol-related applications, as well as providing an accessible platform for learning and experimentation in vehicle communication. The report also provides detailed instructions on installing and configuring* `SocketCAN` *and* `VCAN`*, enabling readers to explore their functionalities. It is expected that, by the end, readers will grasp the practical and educational value of these tools and be able to effectively utilize them in their own applications.*

Resumo. *Este artigo destaca a conveniência e vantagens do* `SocketCAN` *em conjunto com o* `VCAN`*, oferecendo uma visão ampla de suas capacidades e benefícios educacionais.* `SocketCAN` *é uma API nativa de sockets para controladores CAN em sistemas Linux, enquanto o* `VCAN` *é uma interface virtual que simula um barramento CAN no espaço do kernel. Ao explorar essas tecnologias, este relatório mostra como elas simplificam o desenvolvimento e depuração de aplicativos relacionados ao protocolo CAN, além de oferecerem uma plataforma acessível para aprendizado e experimentação em comunicação veicular. O relatório também fornece instruções detalhadas sobre a instalação e configuração do* `SocketCAN` *e* `VCAN`*, permitindo que os leitores explorem suas funcionalidades. Espera-se que, ao final, os leitores compreendam o valor prático e educacional dessas ferramentas e possam utilizá-las efetivamente em suas próprias aplicações.*

1. Introdução

Nos últimos anos, houve avanços notáveis nos meios de comunicação dentro dos veículos, impulsionados pelo progresso tecnológico e pela complexidade crescente dos sistemas automotivos. Desde a adoção do protocolo Controller Area Network (CAN) até as redes flexíveis e Ethernet veiculares, os carros modernos estão mais conectados e dependem de comunicações confiáveis entre suas unidades de controle eletrônico (ECUs) para operar eficientemente.

No entanto, essa maior interconexão também aumenta a vulnerabilidade das comunicações veiculares. Com mais interfaces digitais e sistemas integrados, os veículos

estão mais suscetíveis a ameaças cibernéticas, que vão desde ataques de negação de serviço até manipulações maliciosas de sistemas de segurança críticos. Diante disso, a necessidade de profissionais qualificados nessa área é crucial. A demanda por especialistas em segurança veicular e comunicação CAN está crescendo, já que montadoras e empresas de tecnologia buscam proteger seus veículos contra possíveis ataques e garantir a integridade dos sistemas embarcados.

Portanto, este relatório examina as vantagens e conveniências do `SocketCAN` em conjunto com o `VCAN` como ferramenta de aprendizagem. Ele oferece uma compreensão simplificada da comunicação CAN e fornece uma plataforma prática para exercitar esses conhecimentos. Essas ferramentas simplificam o desenvolvimento e teste de aplicações relacionadas ao protocolo CAN, além de servirem como um ambiente seguro para prática e experimentação. Isso permite que os usuários melhorem suas habilidades em comunicação veicular de maneira eficaz e acessível, facilitando o desenvolvimento de profissionais qualificados na área.

2. Visão geral do SocketCAN

A implementação do `SocketCAN` no Linux fornece uma interface de programação de rede (similar a sockets de rede convencionais) para acessar o barramento CAN diretamente do espaço do usuário. Isso permite que os desenvolvedores criem aplicativos para ler e escrever dados no barramento CAN sem a necessidade de escrever drivers de dispositivo personalizados. Algumas características principais do `SocketCAN` incluem:

1. **Acesso de usuário**: O `SocketCAN` permite que os aplicativos acessem o barramento CAN diretamente do espaço do usuário, sem a necessidade de privilégios de superusuário.
2. **Suporte ao protocolo CAN**: Suporta os principais protocolos CAN, incluindo CAN 2.0A (Standard) e CAN 2.0B (Extended).
3. **Suporte a Sockets**: O `SocketCAN` é baseado no modelo de programação de sockets, tornando-o familiar para desenvolvedores que já trabalharam com comunicação de rede em sistemas `Unix-like`.
4. **Integração com o subsistema de rede do Linux**: Como parte do kernel do Linux, o `SocketCAN` é bem integrado ao subsistema de rede do sistema operacional, o que garante um desempenho eficiente e confiável.

3. Linguagem C e SocketCAN

Optei por utilizar a linguagem C neste projeto devido à natureza de baixo nível da API (Application Programming Interface) `SocketCAN`. Desenvolver em `C` proporciona um controle direto sobre essa interface, permitindo aos desenvolvedores otimizar o código para atender precisamente às demandas específicas do projeto. Além disso, minha experiência prévia com a linguagem durante minha graduação me confere uma maior familiaridade e habilidade para trabalhar de forma eficaz e produtiva com `C`.

3.1. Iniciando a comunicação

É preciso iniciar abrindo um soquete para comunicação na rede CAN. Como o `SocketCAN` implementa uma nova família de protocolos, é necessário passar `PF_ CAN` como o primeiro argumento para o `SystemCall socket()`. No momento, há dois

protocolos CAN disponíveis para seleção: o protocolo de soquete bruto e o gerenciador de transmissão em difusão (BCM). Assim, para abrir um soquete,de acordo com [Hartkopp et al. 2007], você escreveria:

```
int s;
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
// s recebe 1 se o system call for bem sucedido
// s recebe -1 se ocorrer algum erro no system call
```

É uma boa prática verificar se `s` é igual a -1 após a chamada à função `socket()` para garantir que o socket foi criado com sucesso antes de prosseguir com outras operações. Depois da criação do soquete, é comum usar o `SystemCall bind()` para associar o soquete a uma interface CAN (que difere do TCP/IP devido a endereços distintos). Pode ser feito, de acordo com [Hartkopp et al. 2007], da seguinte forma:

```
bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

`struct sockaddr` é uma estrutura em `C` usada para representar endereços de rede em programas que lidam com comunicação de rede, especialmente em sistemas baseados em soquetes. A ”sockaddr”é uma abreviação de ”socket address”e é construido, de acordo com [Hartkopp et al. 2007], da seguinte forma:

```
struct sockaddr_can {
    sa_family_t can_family;
    int         can_ifindex;
    union {
        /* transport protocol class address info (e.g.
           ISOTP) */
        struct { canid_t rx_id, tx_id; } tp;
        /* J1939 address information */
        struct {
            /* 8 byte name when using dynamic addressing */
            __u64 name;
            /* pgn:
            * 8 bit: PS in PDU2 case, else 0
            * 8 bit: PF
            * 1 bit: DP
            * 1 bit: reserved*/
            __u32 pgn;
            /* 1 byte address */
            __u8 addr;
        } j1939;
        /* reserved for future CAN protocols address
           information */
    } can_addr;
};
```

3.2. Criando e Manipulando a Mensagem

Depois de criar o soquete e associá-lo à interface CAN, podemos começar a criar e manipular mensagens CAN (CAN frames). Na linguagem C, os CAN frames têm um formato específico descrito pelo [Hartkopp et al. 2007]. Geralmente, você trabalhará com estruturas que representam esses frames.

```
struct can_frame {
    canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags
       */
    union {
        /* CAN frame payload length in byte (0 ..
           CAN_MAX_DLEN)
        * was previously named can_dlc so we need to carry
           that
        * name for legacy support */
        __u8 len;
        __u8 can_dlc; /* deprecated */
    };
    __u8    __pad;   /* padding */
    __u8    __res0;  /* reserved / padding */
    __u8    len8_dlc; /* optional DLC for 8 byte payload
       length*/
    __u8    data[8] __attribute__((aligned(8)));
};
```

Após criar uma mensagem CAN utilizando a estrutura adequada, você pode manipulá-la usando as funções disponíveis na biblioteca SocketCAN. Algumas das funções disponíveis para manipulação de mensagens CAN no SocketCAN incluem:

1. **Enviar mensagem CAN**: Para enviar uma mensagem CAN, você pode usar a função write() no descritor de arquivo do socket. Por exemplo:

   ```
   int write(int s, const void *msg, size_t len);
   ```

   Onde s é o descritor de arquivo do socket CAN, msg é um ponteiro para a mensagem que você deseja enviar e len é o tamanho da mensagem em bytes.

2. **Receber mensagem CAN**: Para receber uma mensagem CAN, você pode usar a função read() no descritor de arquivo do socket. Por exemplo:

   ```
   int read(int s, void *buf, size_t count);
   ```

   Onde s é o descritor de arquivo do socket CAN, buf é um ponteiro para o buffer onde a mensagem recebida será armazenada e count é o tamanho máximo do buffer em bytes.

3. **Configurar filtros CAN**: Você pode configurar filtros CAN para receber apenas mensagens específicas. Isso é feito usando a função setsockopt() com a opção SO_SET_FILTER. Por exemplo:

```
isetsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &
   filter
            ,sizeof(filter));
```

Onde s é o descritor de arquivo do socket CAN e filter é a estrutura que define os filtros a serem aplicados.

4. VCAN

Desenvolver e testar código `C` que utiliza o protocolo `SocketCAN` para comunicação com dispositivos CAN pode ser desafiador quando não se tem acesso imediato a um hardware CAN real. Neste cenário, uma solução alternativa comumente utilizada é simular o ambiente CAN através do `VCAN` (Virtual CAN), uma interface de rede virtual que emula um dispositivo CAN em sistemas Linux. Essa ferramenta permite criar interfaces de rede virtuais que se comportam como dispositivos CAN reais, facilitando a depuração e o teste de código CAN em sistemas Linux sem a necessidade de hardware físico. Os dados necessários para implementar o vcan foram pegos do reposítorio do [Guštin et al. 2020].

5. Configurando o Ambiente de Desenvolvimento

1. Abra o terminal do linux.
2. Verifique se o módulo `can` está instalado no kernel Linux. Normalmente, é incluído na maioria das distribuições modernas.
3. Se o módulo não estiver carregado, você pode carregá-lo manualmente usando o comando:

```
sudo modprobe can
sudo modprobe can_raw
sudo modprobe can_dev
```

4. Para tornar esses módulos permanentes, adicione-os ao arquivo `/etc/modules`:

```
sudo echo "can" >> /etc/modules
sudo echo "can_raw" >> /etc/modules
sudo echo "can_dev" >> /etc/modules
```

5. Instale o pacote de desenvolvimento SocketCAN `SocketCAN`:

```
sudo apt-get install libsocketcan-dev
```

6. Configure o ambiente de desenvolvimento C instalando seu compilador:

```
sudo apt-get install gcc
```

7. Crie um novo projeto C e Inclua o cabeçalho `socketcan/can.h` no seu código-fonte:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <net/if.h>
#include <linux/can.h>
#include <linux/can/raw.h>
```

8. Carregue o módulo vcan

```
sudo modprobe vcan
```

9. Crie uma interface de rede vcan (por exemplo, `vcan0`):

```
sudo ip link add dev vcan0 type vcan
```

10. Ative a interface vcan:

```
sudo ip link set up vcan0
```

6. Testes e Conclusões

O código usado para os testes se encontra no seguinte **link**. Este código em `C` cria um programa que lê dados da interface CAN no Linux. Ele estabelece uma conexão com a interface especificada, configura a comunicação e entra em um `loop` infinito para ler os quadros CAN recebidos. Para cada quadro lido, o programa exibe seu identificador (ID) e os dados contidos no quadro. Para o envio dos quadros é usando o seguinte comando no terminal:

```
cansend vcan0 123#112233
```

Esse comando envia um quadro CAN para a interface `vcan0` com ID `0x123` e dados `0x11 0x22 0x33`. Com isso temos o seguinte retorno:

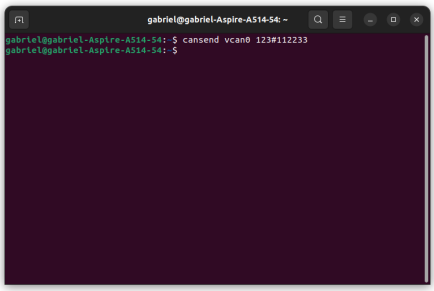

Figura 1. Terminal linux

Figura 2. Executável do código C

Este código pode ser utilizado para receber quadros CAN em uma interface específica e executar ações com base no conteúdo desses quadros. O loop while no final do código aguarda continuamente por quadros CAN recebidos na interface especificada (`vcan0`, no caso). No entanto, pode ser estendido para outras interfaces, como interfaces físicas conectadas a um veículo.

Referências

Guštin, M., Cloos, J., and Huth, P. (2020). Vcan. https://github.com/TheMatjaz/VCAN.

Hartkopp, O., Thuermann, U., and Kizka, J. (2007). kernel.org. https://www.kernel.org/doc/html/next/networking/can.html.