실습 6

Minsoo Ryu

Operating Systems and Distributed Computing Lab.
Hanyang University

msryu@hanyang.ac.kr

목차

- 실습
  - **Programming TCP/IP Sockets**
  - **Programming SocketCAN**
  - **More Than 8 Bytes with SocketCAN**

실습

1. Programming TCP/IP Sockets (1)

- 구현 목표
  - **Socket**을 이용하여 파일을 전송하는 **TCP/IP** 통신 프로그램을 구현한다
- 구현 조건
  - **Raspberry Pi**가 서버, **Linux** 호스트가 클라이언트로 동작한다
  - 클라이언트가 텍스트 파일을 서버에게 전송하고, 서버는 이를 수신하여 화면에 출력한다
  - 서버 측 프로그램 **"server_receiver.c"**을 작성한다
- 제공 자료
  - 클라이언트의 소스 코드 **"client_sender.c"**
    - 따라서 서버측 프로그램 "server_receiver.c"만 작성하면 되며, 제공되는 "client_sender.c"와 유사한 구조를 가지지만 (1) 대칭적으로 동작한다는 점과 (2) 서버 측에서 필요한 동작을 추가적으로 구현해야 한다는 점을 고려해야 함
    - 단, 제공된 코드에서 IP 주소를 본인이 사용하는 Raspberry Pi의 주소로 수정해야 됨 ("$ ifconfig" 명령으로 wlan0의 inet 정보 확인)
  - 클라이언트에서 보낼 파일 **"message.txt"**

1. Programming TCP/IP Sockets (2)

❒ 실행 결과 예시

```
msryu@rpi:~/Lab-RPi/61_Socket_Communication_TCP $ ./server_receiver.out
[STARTING] TCP File Server started.
[RECEVED] Hello World.
[RECEVED] Goodbye, see you later.
[CLOSING] Closing the server.
```

```
msryu@DESKTOP-AMQO5V8:~/Lab-RPi/61_Socket_Communication_TCP$ ./client_sender.out
[SENT] Hello World.
[SENT] Goodbye, see you later.
[CLOSING] Disconnecting from the server.
```

준비 작업:
CAN 디바이스 설정 및 연결

CAN 디바이스 설정 (1/3)

❒ $ sudo nano /boot/config.txt

- 파일에 아래의 내용을 삽입

```
dtoverlay=mcp2515-can0,oscillator=8000000,interrupt=25
```

- **(참고) /boot/config.txt는 Raspberry Pi만의 고유 기능**
  - Raspberry Pi는 /boot/config.txt 파일을 사용하여 부팅할 때마다 시스템을 설정
  - Raspberry Pi에서 dtoverlay는 boot firmware에게 새로운 device tree 정보를 전달
- **(참고) nano는 에디터이며, vi 에디터 또는 VS code를 사용해도 됨**

❒ $ sudo reboot –r now

- **raspberry pi 재부팅**

CAN 디바이스 설정 (2/3)

- **$ sudo /sbin/ip link set can0 up type can bitrate 500000**
  - 수동으로 **CAN** 디바이스를 활성화

- **(참고) 부팅시 자동으로 CAN 디바이스를 활성화하는 것도 가능**
  - **$ sudo nano /etc/network/interfaces**
  - 파일에 아래의 내용을 삽입

```
auto can0
iface can0 inet manual
    pre-up /sbin/ip link set can0 type can bitrate 500000 triple-sampling on restart-ms 100
    up /sbin/ifconfig can0 up
    down /sbin/ifconfig can0 down
```

CAN 디바이스 설정 (3/3)

- **$ ifconfig**
  - **CAN** 인터페이스가 정상적으로 설정되었는지 확인
  - 정상적으로 진행이 되었다면 아래와 같은 내용이 출력되어야 함

```
can0: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- **$ sudo apt-get install can-utils**
  - **CAN-utils** 설치

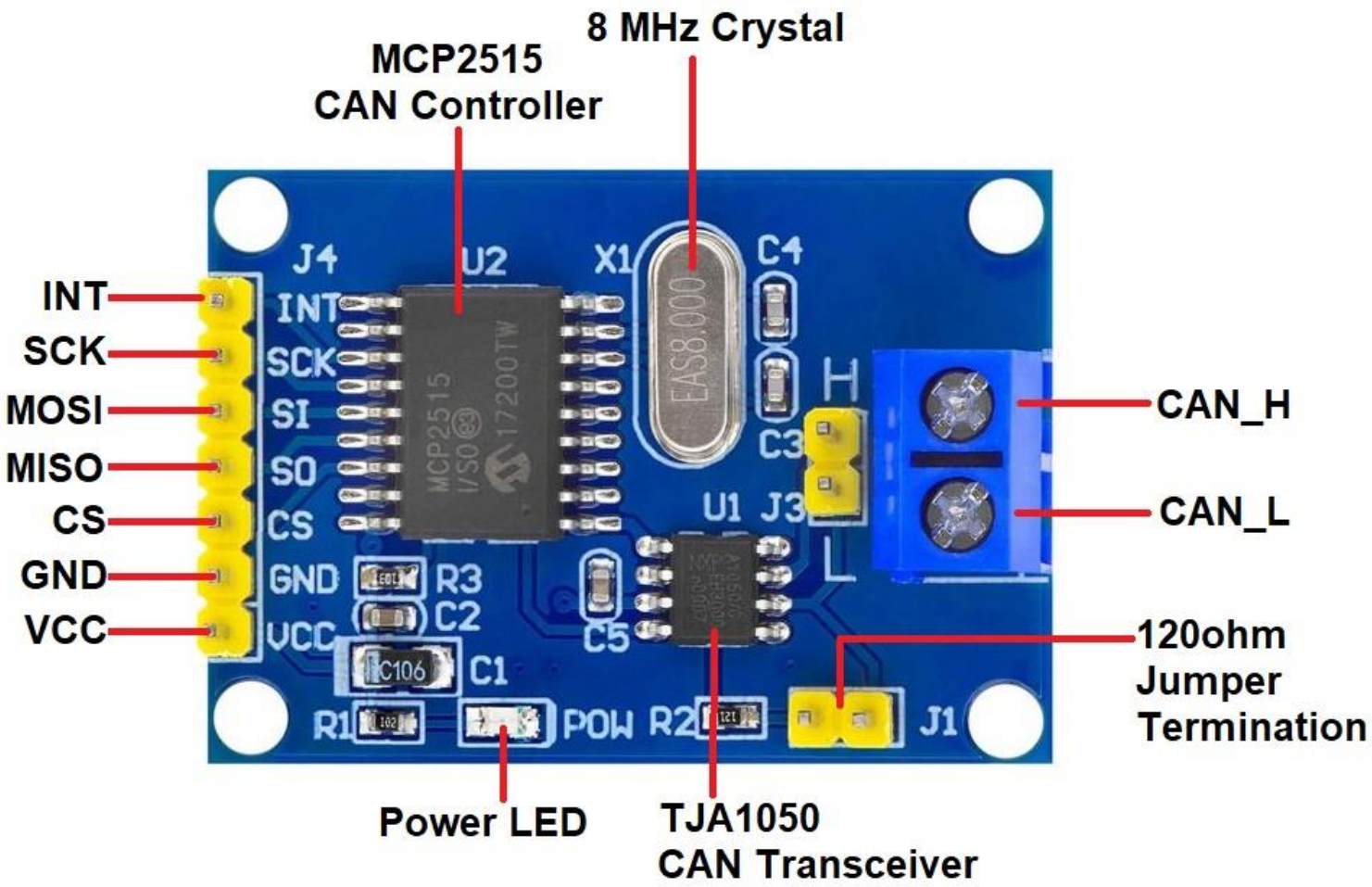
준비 작업: CAN-utils 동작 확인

- **$ candump can0**
  - **“can0”** 장치로 수신되는 메시지를 실시간으로 표시

- **$ cansend can0 123#1122334455667788**
  - **“can0”** 장치로 **“123”**을 **ID**로 하는 **“1122334455667788”** 데이터를 전송하라는 의미

Receiver	Sender
$ candump can0 **can0 12F [8] FF FF FF FF FF FF FF FF**	**$ cansend can0 12F#FF.FF.FF.FF.FF.FF.FF.FF**

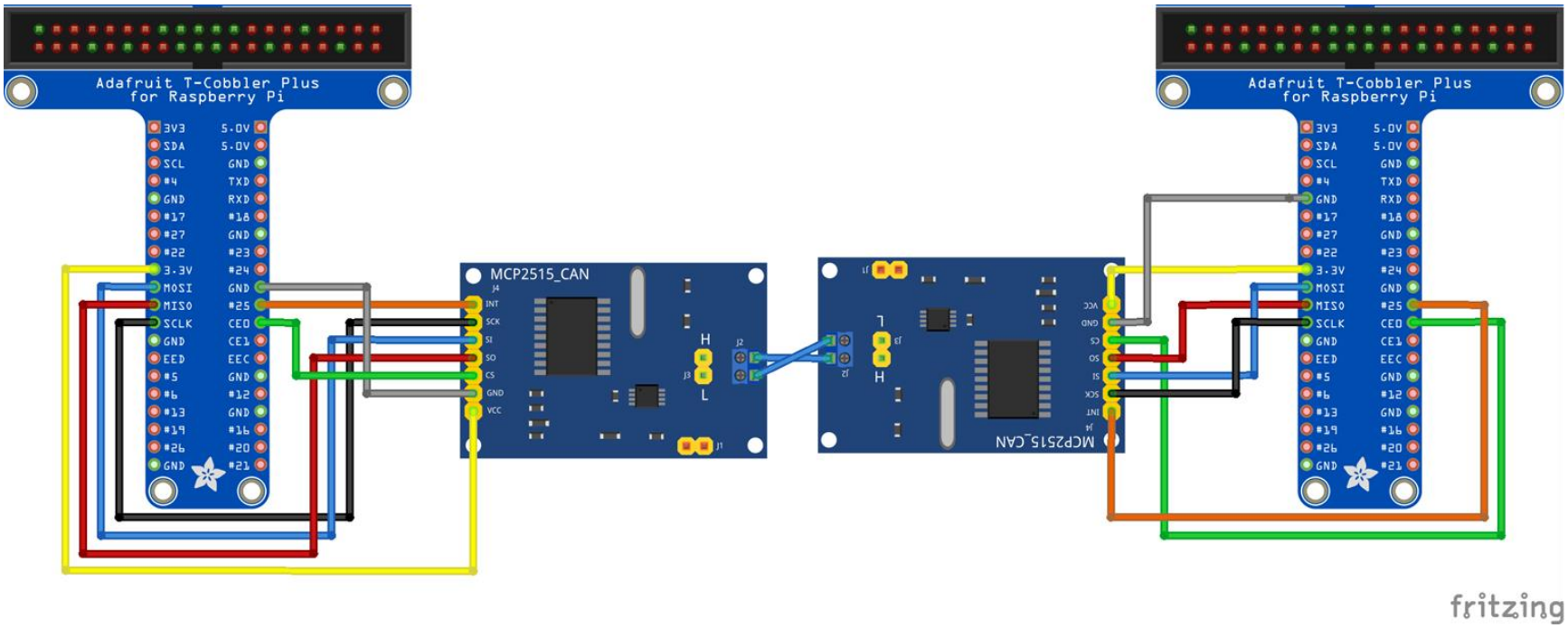
실습에 사용될 CAN 장치: MCP2515



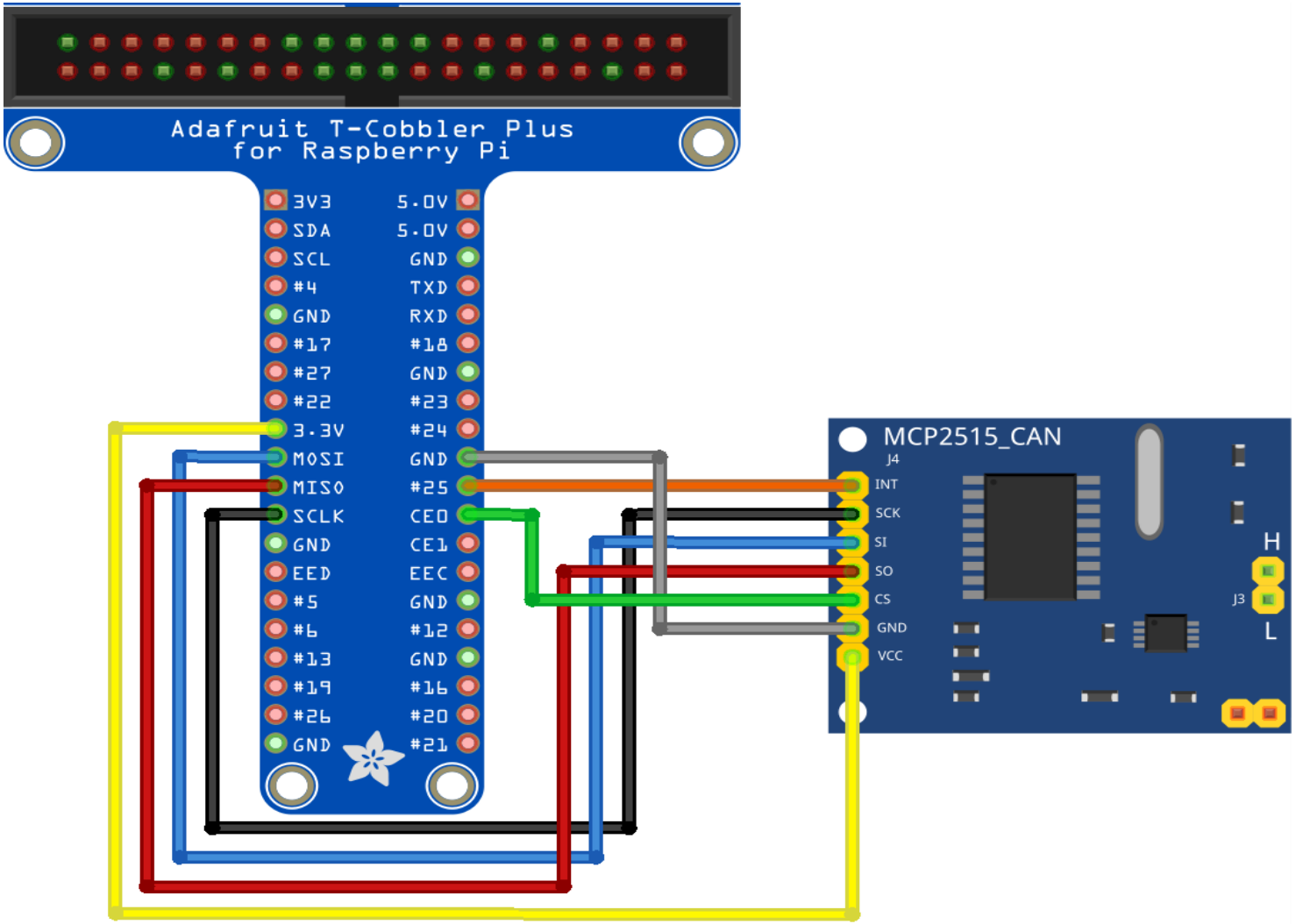
실습에 사용될 CAN 장치: MCP2515

- **Stand-Alone CAN Controller with SPI Interface**
  - **Implements CAN V2.0B at 1 Mb/s**
    - 0 to 8-byte length in the data field
    - Standard and extended data and remote frames
  - **Receive Buffers, Masks and Filters:**
    - Two receive buffers with prioritized message storage
    - Six 29-bit filters and two 29-bit masks
  - **Data Byte Filtering on the First Two Data Bytes**
    - Applies to standard data frames
  - **Three Transmit Buffers with Prioritization and Abort Features**
  - **High-Speed SPI Interface (10 MHz)**
    - SPI modes 0,0 and 1,1
  - **Interrupt Output Pin with Selectable Enables**
  - **Low-Power CMOS Technology**
    - Operates from 2.7V-5.5V, 5 mA active current (typical)
    - 1 μA standby current (typical) (Sleep mode)
  - **Temperature Ranges**
    - Industrial (I): -40°C to +85°C
    - Extended (E): -40°C to +125°C

MCP2515 연결


Adafruit T-Cobbler Plus
for Raspberry Pi
MCP2515_CAN
Adafruit T-Cobbler Plus
for Raspberry Pi
fritzing

MCP2515 연결


Adafruit T-Cobbler Plus
for Raspberry Pi
3V3
SDA
SCL
#4
GND
#17
#27
#22
3.3V
MOSI
MISO
SCLK
GND
EED
#5
#6
#13
#19
#26
GND
5.0V
5.0V
GND
TXD
RXD
#18
GND
#23
#24
GND
#25
CE0
CE1
EEC
GND
#12
GND
#16
#20
#21
MCP2515_CAN
J4
INT
SCK
SI
SO
CS
GND
VCC
H
J3
L

2. Programming SocketCAN (1)

❒ 구현 목표

- **SocketCAN**을 통해 정수값 **2**개를 송수신하는 프로그램을 구현한다

❒ 구현 조건

- 두 명이 한 조를 이루어 **2**대의 **Raspberry Pi** 보드를 **CAN**으로 연결한다
- 한 대는 **sender,** 다른 한 대는 **receiver**로 동작한다

❒ 제공 자료

- **“sender.c”** 소스 파일
- 따라서 수신측 프로그램 **“receiver.c”** 만 작성하면 되며, 제공되는 **“sender.c”**와 유사한 구조를 가지지만 대칭적으로 동작한다는 점을 고려해야 함

2. Programming SocketCAN (2)

- 제공받은 **“sender.c”** 동작 확인
  - 제공된 **“sender.c”** 프로그램을 컴파일하여 실행 파일 **“sender.out”**을 빌드
  - 수신을 담당할 **Raspberry Pi** 보드에서 아래의 명령을 실행
    - $ candump can0
  - 송신을 담당할 **Raspberry Pi** 보드에서 아래의 명령을 실행
    - $ sudo ./sender.out

```
msryu@rpi:~/Lab-RPi/62_SocketCAN $ ./sender.out
SocketCAN Sender
0x555 [8] 20 48
```

  - 수신 측에서 아래의 출력을 확인

```
hwkim@hwkim:~/Lab-RPi/62_SocketCAN $ candump can0
  can0  555   [8]  14 00 00 00 30 00 00 00
```

  - 만약 통신이 실패할 경우 두 보드를 모두 재부팅
    - $ sudo shutdown –r now

2. Programming SocketCAN (3)

❒ **“receiver.c” 구현 완료시 화면 출력 예시**

```
msryu@rpi:~/Lab-RPi/62_SocketCAN $ ./sender.out
SocketCAN Sender
0x555 [8] 20 48
```

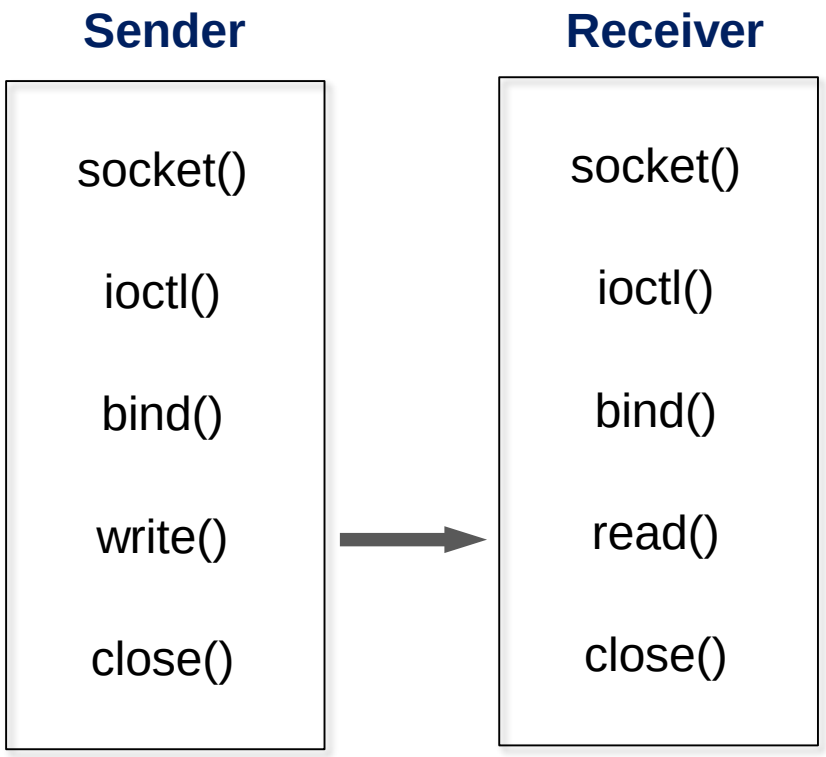
```
hwkim@hwkim:~/Lab-RPi/62_SocketCAN $ ./receiver.out
SocketCAN Receiver
0x555 [8] 20 48
```

- **(주의) “receiver.out”을 “sender.out”보다 먼저 실행해야 통신이 가능**

2. Programming SocketCAN (4)

❒ 참고

- **SocketCAN을 이용한 sender와 receiver간의 API 실행 흐름**



2. Programming SocketCAN (5)

❒ 참고

- **socket() 함수의 파라미터 설정**
  - 함수 형식: int socket(int domain, int type, int protocol)
  - domain = PF_CAN
  - type = SOCK_RAW
  - protocol = CAN_RAW
- **ioctl() 함수의 파라미터 설정**
  - 함수 형식: int ioctl(int fd, unsigned long request, ifreq *ifr)
  - fd = 소켓 디스크립터
  - request = SIOCGIFINDEX
    - ✓ S: system, IOC: ioctl 명령, G: get, IFINDEX: interface index
  - ifr = &ifr
    - ✓ 사전 선언: struct ifreq ifr
    - ✓ 사용: &ifr을 파라미터로 전달

2. Programming SocketCAN (6)

❒ 참고 (계속)

- **bind() 함수의 파라미터 설정**
  - 함수 형식: int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  - sockfd = 소켓 디스크립터 (socket 함수 리턴 값)
  - *addr = &addr
    - ✓ 사전 선언: struct sockaddr_can addr
    - ✓ 사용: &addr을 파라미터로 전달
  - addrlen = address 구조체 크기 = sizeof(addr)

```
struct sockaddr_can {
        sa_family_t can_family;
        int can_ifindex;
};
```

```
struct can_frame {
    canid_t can_id;
    __u8    can_dlc;
    __u8    data[8];
};
```

- **write() 함수의 파라미터 설정**
  - 함수 형식: ssize_t write(int fd, const void buf[], size_t count)
  - fd = 소켓 디스크립터
  - buf = 송신할 데이터를 저장할 배열의 포인터
    - ✓ 사전 선언: struct can_frame frame
    - ✓ 사전 작업: can_id 값(32-bit) 지정, can_dlc에 데이터 크기 지정 (= 8), data에 송신할 데이터 복사(memcpy 함수 등을 활용)
  - count: 송신할 데이터의 크기 (= 8)

2. Programming SocketCAN (7)

❒ 참고 (계속)

- **read() 함수의 파라미터 설정**
  - 함수 형식: ssize_t read(int fd, void buf[], size_t count);
  - fd = 소켓 디스크립터
  - buf = 수신할 데이터를 저장할 CAN frame의 포인터
    - ✓ 사전 선언: struct can_frame frame
    - ✓ 사용: &frame을 파라미터로 전달
  - count: 수신할 버퍼의 크기 (= 8)
- **close() 함수의 파라미터 설정**
  - 함수 형식: int close(int fd);
  - fd = 소켓 디스크립터

2. Programming SocketCAN (8)

❒ 참고 (계속)

- **“receiver.c”** 작성시 수신한 데이터를 버퍼로 옮길 때 아래의 **memcpy()** 함수를 사용
  - void *memcpy(void* dest, void* src, size_t n)
  - “memcpy” 함수는 src가 가리키는 주소로부터 n 바이트의 데이터를 dest가 가리키는 주소로 복사

3. More Than 8 Bytes (1)

- 구현 목표
  - **SocketCAN**을 통해 **8** 바이트 이상의 문자열을 송수신하는 프로그램을 구현한다
- 구현 조건
  - 두 명이 한 조를 이루어 **2**대의 **Raspberry Pi** 보드를 **CAN**으로 연결한다
  - 한 대는 **sender,** 다른 한 대는 **receiver**로 동작한다
  - **sender**는 사용자로부터 문자열을 입력받아 **receiver**에게 전송, **receive**는 수신한 문자열을 화면에 출력한다
  - 사용자가 **“q”**를 입력하면 **receiver** 프로그램이 종료하도록 작성한다
- 제공 자료
  - **“receiver.c”** 및 **while** 문 일부가 삭제된 **“sender.src”** 소스 파일
  - 따라서 송신측 프로그램 **“sender.c”** 만 완성하면 됨
    - 단, 사용자로부터 입력받는 데이터가 8 바이트보다 클 수 있으므로 데이터를 분할하여 전송해야 함

3. More Than 8 Bytes (2)

❒ 실행 화면 예시

```
msryu@rpi:~/Lab-RPi/63_SocketCAN_Multiple_Packets $ ./sender.out
SocketCAN Sender

Enter your text: Hello World
Your text: Hello World
Input Length = 12
Number of Packets to send = 2
Size of Last Bytes to send = 4

Enter your text: q
Your text: q
Input Length = 2
Number of Packets to send = 1
Size of Last Bytes to send = 2

QUIT COMMAND!
```

```
hwkim@hwkim:~/Lab-RPi/63_SocketCAN_Multiple_Packets $ ./receiver.out
SocketCAN Receiver
0x555 [8] Hello Wo
0x555 [4] rld

0x555 [2] q

QUIT COMMAND!
hwkim@hwkim:~/Lab-RPi/63_SocketCAN_Multiple_Packets $
```

3. More Than 8 Bytes (3)

❒ 참고: “sender.src”의 main() 함수에 선언된 변수 고려

- **int packetTotal; // Number of packets to send**
  - 입력받은 문자열을 전송하는데 필요한 write() 함수의 총 호출 횟수를 먼저 계산한다
  - 만약 packetTotal의 값이 5로 계산된다면 처음 4번을 8 byte로 전송
    - ✓ “frame.can_dlc = 8;” 로 설정한 후 write() 실행
- **Int lastPacketSize; // Size of last packet**
  - 마지막 write() 함수 호출시 보내야 할 바이트 개수를 계산한다
  - 만약 lastPacketSize의 값이 3이라면 마지막 write() 함수를 호출할 때는 3 byte를 전송
    - ✓ “frame.can_dlc = 3;”으로 설정한 후 write() 실행

thank you!