UNIVERSIDADE DE BRASILIA
FACULDADE DO GAMA

Fundamentos de Sistemas Embarcados

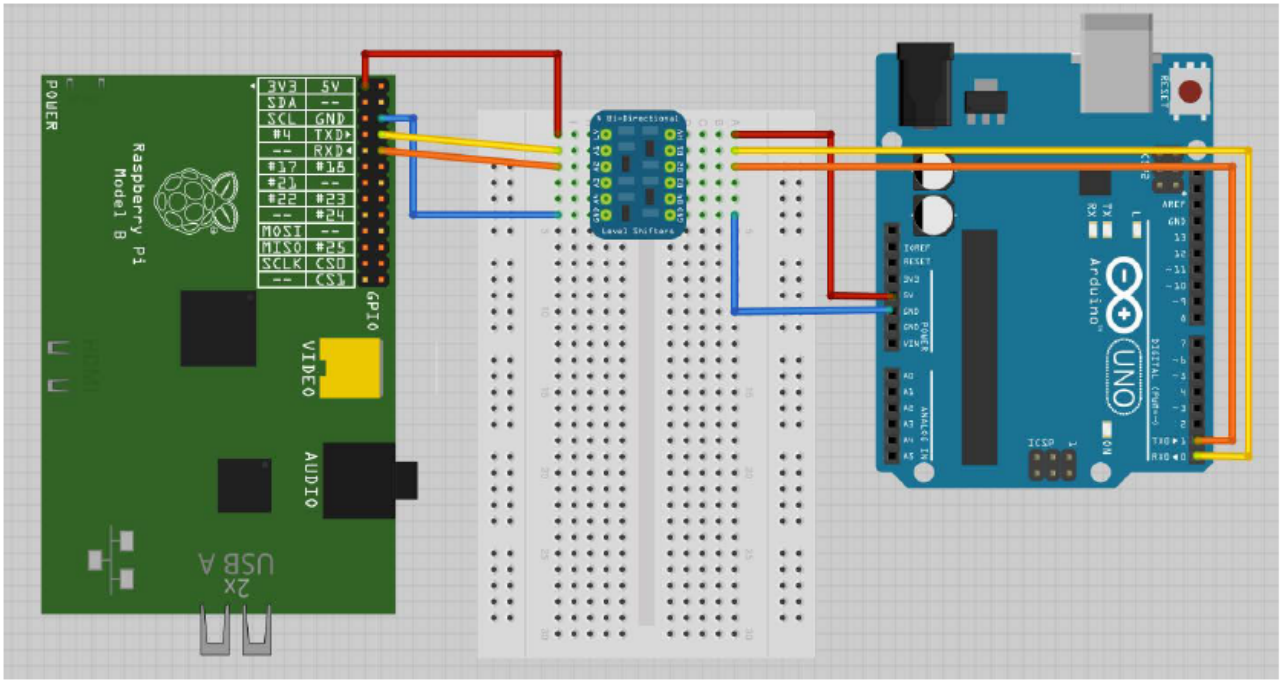
Exercício 1 - Comunicação UART

1 - OBJETIVOS

Neste trabalho o(a) aluno(a) irá exercitar o acesso a dispositivos (devices) utilizando as funções de acesso à arquivos do POSIX. No caso, uma comunicação serial (UART) terá que ser implementada entre a placa Raspberry Pi e um microcontrolador Arduino.

2 - CIRCUITO ESQUEMÁTICO

Para conectar o Raspberry Pi ao Arduino, será utilizada a porta serial através dos pinos 8 (UART_TXD) e 10 (UART_RXD) do Raspberry Pi e ps respectivos pinos RX, TX do Arduino. Porém, é necessário observar que o Raspberry Pi opera a uma tensão de 3.3V enquanto a maioria das placas Arduino operam em 5V. Neste caso será necessário a utilização de algum método de conversão de tensão (Level Shifter, Divisor de Tensão, Optoacopladores, etc.)



3 - ALGUMAS FUNÇÕES A SEREM USADAS

1. POSIX ***open()***: Para abertura do arquivo que apontará para a porta serial (UART):

```
int open(const char* path, int oflag, …)

Exemplo:

int fd;
fd = open("/tmp/teste.txt", O_WRONLY);
```

2. POSIX **close*()***: Para fechar o arquivo referente à porta serial:

```
int close(int fd);

Exemplo:

int fd;
close(fd);
```

3. POSIX ***write()***: Para escrever dados na porta serial

```
ssize_t write(int fildes, const void *buf, size_t nbyte);

Exemplo:

short siX16=0x7FFF;
int res = write(fid, &siX16, sizeof(short) );
```

4. POSIX ***read()***: Para ler dados da porta serial

```
ssize_t read(int fildes, void *buf, size_t nbyte);

Exemplo:

short siX16;
int res = read(fid, &siX16, sizeof(short) );
```

4 - ROTEIRO

1. Criar um programa em C, no Raspberry Pi, capaz de usar a comunicação com a porta serial UART para ler e escrever conforme o protocolo definido abaixo:
   a. **Escrita de dados**: Todas as mensagens de solicitação de dados enviadas pela porta serial devem seguir o padrão: Código do comando solicitado (Tabelas 1 e 2) + "Quatro últimos dígitos da matrícula" em formado **char**.
   Exemplo de Mensagem com o comando 0xA3 (Hexadecimal) = 163 (Decimal) e a matrícula "4521" ao final:

char[] = {163, 4, 5, 2, 1};

Já as mensagens de envio de dados deverão ser compostas por: **Comando** + **Dado** + **Matrícula**. No caso do envio de uma String, o formato deverá ser: **Comando** + **Tamanho da String** (1 byte) + **String** + **Matrícula.**

b. **Leitura de dados**: A leitura de dados deve seguir o padrão de retorno da Tabela 1. Para valores inteiros (int) ou reais (float), deverão ser lidos 4 bytes e armazenados em uma variável ***int*** ou ***float***, respectivamente. Para leitura de strings, a mensagem de retorno irá conter o número total de caracteres da string no primeiro byte (Valor entre 0 e 255), em seguida, é possível ler o conteúdo da mensagem que deverá ser armazenada em um array de char (char[]);
c. Cada tipo de mensagem recebida deverá ser impressa em tela (stdout).

Tabela 1 - Códigos do Protocolo de Comunicação - Solicitação de Informações

Código	Comando de Solicitação de Dados	Mensagem de Retorno
0xA1	Solicitação de dado inteiro: ***integer***	int (4 bytes)
0xA2	Solicitação de dado real: ***float***	float (4 bytes)
0xA3	Solicitação de dado do tipo string: ***char[]***	char (1 byte com o tamanho da string) + char[] (nbytes com o conteúdo da string)

Tabela 2 - Códigos do Protocolo de Comunicação - Envio de Dados

Código	Comando de Envio de Dados	Mensagem de Retorno
0xB1	Envio de um dado no formato ***integer***	int (4 bytes)
0xB2	Envio de um dado no formato ***float***	float (4 bytes)
0xB3	Envio de uma string: ***char[]***	char (1 byte com o tamanho da string) + char[] (nbytes com o conteúdo da string)

2. O programa deverá ser estruturado em funções e cada acesso à porta serial deve ser feito abrindo e fechando o arquivo referente ao *device* UART para evitar concorrência com outros usuários.
3. Finalmente, cada função de solicitação de dados ou de envio de comandos deverá ser acessada através de um menu na linha de comando.

4 - OBSERVAÇÕES

O código pode ser testado no Linux, porém terá que ser compilado para rodar no Raspberry Pi em sala de aula.

```
#include <stdio.h>
#include <unistd.h>           //Usado para a UART
#include <fcntl.h>            //Usado para a UART
#include <termios.h>          //Usado para a UART

int main(int argc, char ** argv) {

//-------------------------
//----- CONFIGURAÇÃO DA UART -----
//-------------------------
//    At bootup, pins 8 and 10 are already set to UART0_TXD, UART0_RXD (ie the
alt0 function) respectively
int uart0_filestream = -1;

// ABRIR A UART
// CONFIGURAÇÕES DEFINIDAS EM fcntl.h:
//    Modos de Acesso (utilize apensa 1 deles):
//     O_RDONLY - Abrir apenas para leitura.
//     O_RDWR   - Abrir para Leitura / Escrita
//     O_WRONLY - Abrir apenas para Escrita.
//
//     O_NDELAY / O_NONBLOCK (mesma função) - Habilita o modo não-blocante.
//    Quando configurado as solicitações de Leitura no arquivo podem retornar
//     imediatamente com erro quando não houverem dados disponíveis (Ao invés de
//    bloquear). Do mesmo modo, solicitações de escrita podem retornar erro
//    caso não seja possível escrever na saída.
//
//     O_NOCTTY - Quando definido e o caminho identificar um dispositivo de
//    terminal, a função open() não causará que este terminal obtenha controle
//    do processo terminal.
uart0_filestream = open("/dev/serial0", O_RDWR | O_NOCTTY |
O_NDELAY);        //Abrir no modo não blocante para Leitura / Escrita
if (uart0_filestream == -1)
{
      printf("Erro - Porta Serial nao pode ser aberta. Confirme se não está
sendo usada por outra aplicação.\n");
}

//CONFIGURAÇÕES DA  UART
//  Flags (Definidas em /usr/include/termios.h):
//    Baud rate:- B1200, B2400, B4800, B9600, B19200, B38400, B57600, B115200,
B230400, B460800, B500000, B576000, B921600, B1000000, B1152000, B1500000,
B2000000, B2500000, B3000000, B3500000, B4000000
//    CSIZE:- CS5, CS6, CS7, CS8
//    CLOCAL - Ignore modem status lines
//    CREAD - Enable receiver
//    IGNPAR = Ignore characters with parity errors
//    ICRNL - Map CR to NL on input
//    PARENB - Parity enable
//    PARODD - Odd parity
struct termios options;
tcgetattr(uart0_filestream, &options);
options.c_cflag = B115200 | CS8 | CLOCAL | CREAD;          // Set baud rate
options.c_iflag = IGNPAR;
options.c_oflag = 0;
options.c_lflag = 0;
tcflush(uart0_filestream, TCIFLUSH);
tcsetattr(uart0_filestream, TCSANOW, &options);
```

```
//-------------------------------------
//----- TX - Transmissão de Bytes -----
//-------------------------------------
      unsigned char tx_buffer[20];
      unsigned char *p_tx_buffer;

      p_tx_buffer = &tx_buffer[0];
      *p_tx_buffer++ = 'T';
      *p_tx_buffer++ = 'e';
      *p_tx_buffer++ = 's';
      *p_tx_buffer++ = 't';
      *p_tx_buffer++ = 'e';

      if (uart0_filestream != -1)
      {
      int count = write(uart0_filestream, &tx_buffer[0], (p_tx_buffer -
&tx_buffer[0]));        // Arquivo, bytes a serem escritos, número de bytes a
serem escritos;
      if (count < 0)
      {
            printf("Erro na transmissao - UART TX\n");
      }
      }


//-------------------------------------
//----- RX - Leitura de Bytes -----
//-------------------------------------
      if (uart0_filestream != -1)
      {
      // Ler até 255 caracteres da porta de entrada
      unsigned char rx_buffer[256];
      int rx_length = read(uart0_filestream, (void*)rx_buffer,
255); //Arquivo, buffer de saída, número máximo de caracteres a serem lidos
      if (rx_length < 0)
      {
            perror("Falha na leitura\n");
            return -1;
      }
      else if (rx_length == 0)
      {
            printf("Nenhum dado disponivel\n");
      }
      else
      {
            //Bytes received
            rx_buffer[rx_length] = '\0';
            printf("%i bytes lidos : %s\n", rx_length, rx_buffer);
      }
      }

      //----- CLOSE THE UART -----
      close(uart0_filestream);

      return 0;
}
```