Embedded Systems Programming
Assignment 5.1
RPi UART Communications

Steinarr Hrafn Höskuldsson

October 18, 2022

Part 1

The primary UART was configured using the `raspi-config` interface.
The symbol rate was found via the `stty` command to be 0.
After running minicom and lowering the baud rate to 1200. The `stty` command reports the baud rate on `/dev/ttyS0` to be 1200.
The difference between 1200 and 115200 baud is not noticeable when typing but obvious when pasting a long sentence into minicom.
Another way to detect the difference is by turning on local echo and typing at different speeds:

```
ttyyppiinngg  sslloowwllyy

ttyyppiningg  ffaastst..
```

After closing minicom the speed according to the `stty` command goes back to 0.

Part 2

Running the example given, the output was:

```
The following was read in [20]: Hello Raspberry Pi!
```

Adding a newline character to the end of the string results in almost the same output only a second empty line is also printed. This is because in the `printf` statement there is a newline character and the string received now also contains a newline character resulting in two consecutive newline characters.

The example was edited to read a line from stdin and send it to the serial port, the program can be seen in Appendix . For anything to be sent a newline has to be sent to stdin.

After exiting the program the baud rate stays at 57600 according to the `stty` command. Unlike when using `minicom`. The program does nothing to reset the settings on `/dev/ttyS0`.

Part 3

Two c programs were written, see Appendix . `part3_write` increments a counter once every second and writes its value to the serial port in binary format. `part3_read` uses a blocking read to read a `uint32_t` and prints it.

It took a bit of trial and error to get the read to always wait for 4 bytes. Before that was successfull the output looked like :

```
1
0
0
2
0
0
0
```

```
3
0
0
0
...
.
...
255
0
0
0
0
1
0
0
1
1
0
0
...
```

However after getting the port settings right the read call blocked until 4 bytes were read and then the output was as expected:

```
1
2
3
...
.
...
255
256
257
...
```

Appendix

Part 2

```c
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<termios.h>   // using the termios.h library
#include <signal.h>

static volatile int keepRunning = 1;

void intHandler(int dummy) {
    keepRunning = 0;
}



int main(){

    signal(SIGINT, intHandler);
   int fd, count;

   // Remove O_NDELAY to *wait* on serial read (blocking read)
```

```
   if ((fd= open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY))<0){
      perror("UART: Failed to open the file.\n");
      return -1;
   }

   struct termios options;       // the termios structure is vital
   tcgetattr(fd, &options);    // sets the parameters for the file

   // Set up the communication options:
   // 57600 baud, 8-N-1, enable receiver, no modem control lines
   options.c_cflag = B57600 | CS8 | CREAD | CLOCAL;
   options.c_iflag = IGNPAR | ICRNL;   // ignore partity errors
   tcflush(fd, TCIFLUSH);            // discard file information
   tcsetattr(fd, TCSANOW, &options); // changes occur immmediately

    char *line =NULL;
    size_t len = 0;

    ssize_t lineSize = 0;

    while (keepRunning)
    {
        lineSize = getline(&line, &len, stdin);
        // printf("You entered %s, which has %zu chars.\n", line, lineSize -1);

        if ((count = write(fd, line, lineSize-1))<0){         // transmit
            perror("Failed to write to the output\n");
            return -1;
        }

        usleep(100000);             // give the remote machine a chance to
    respond

        unsigned char receive[100] = {0}; //declare a buffer for receiving data

        if ((count = read(fd, (void*)receive, 100))<0){   //receive data
            perror("Failed to read from the input\n");
            return count;
        }

        if (count==0) printf("There was no data available to read!\n");
        // else printf("The following was read in [%d]: %s\n",count,receive);
        else printf("%s\n", receive);
    }

   close(fd);

   return 0;
}
```

Listing 1: part2.c

Part 3

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<termios.h>   // using the termios.h library
#include <signal.h>
#include <stdint.h>
```

```c
static volatile int keepRunning = 1;

void intHandler(int dummy) {
    keepRunning = 0;
}



int main(){

    signal(SIGINT, intHandler);
   int fd, count;

   // Remove O_NDELAY to *wait* on serial read (blocking read)
   if ((fd= open("/dev/ttyS0", O_RDONLY | O_NOCTTY))<0){
      perror("UART: Failed to open the file.\n");
      return -1;
   }

   struct termios options;       // the termios structure is vital
   tcgetattr(fd, &options);    // sets the parameters for the file

    cfmakeraw(&options); // set as raw

   // 57600 baud, 8-N-1, enable receiver, no modem control lines
   options.c_cflag |= B57600 | CS8 | CREAD | CLOCAL;

   tcflush(fd, TCIFLUSH);            // discard file information
   tcsetattr(fd, TCSANOW, &options); // changes occur immmediately

    uint32_t counter = 0;
    while (keepRunning)
    {

        if ((count = read(fd, &counter, 4))<0){   //receive data
            perror("Failed to read from the input\n");
            return count;
        }

        if (count==0){} // printf("There was no data available to read!\n");

        else printf("%d\n", counter);
    }

   close(fd);

   return 0;
}
```

Listing 2: part3_read.c

```c
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<termios.h>   // using the termios.h library
#include <signal.h>
#include <stdint.h>

static volatile int keepRunning = 1;
```

```c
void intHandler(int dummy) {
    keepRunning = 0;
}



int main(){

    signal(SIGINT, intHandler);

    // set stdin to canon
    // struct termios options;
    // tcgetattr(STDIN_FILENO, &options);
    // options.c_lflag &= ~ICANON;
    // tcsetattr(STDIN_FILENO, TCSANOW, &options);
   int fd, count;

   // Remove O_NDELAY to *wait* on serial read (blocking read)
   if ((fd= open("/dev/ttyS0", O_WRONLY | O_NOCTTY))<0){
      perror("UART: Failed to open the file.\n");
      return -1;
   }

   struct termios options;       // the termios structure is vital
   tcgetattr(fd, &options);    // sets the parameters for the file

   cfmakeraw(&options); // set as raw

   // 57600 baud, 8-N-1, enable receiver, no modem control lines
   options.c_cflag |= B57600 | CS8 | CREAD | CLOCAL;

   tcflush(fd, TCIFLUSH);            // discard file information
   tcsetattr(fd, TCSANOW, &options); // changes occur immmediately

    uint32_t counter = 0;
    while (keepRunning)
    {
        counter++;

        if ((count = write(fd, &counter, sizeof(counter)))<0){         //
    transmit
            perror("Failed to write to the output\n");
            return -1;
        }

        usleep(1000000);             // gsleep 1 second
    }

   close(fd);

   return 0;
}
```

Listing 3: part3_write.c