Criação de um driver para a interface serial do cMIPS

Luigi Muller Sousa Linhares, *Graduando de Ciência da Computação, UFRR,* Tarlison Sander Lima Brito, *Graduando de Ciência da Computação, UFRR.*

***Resumo*—O driver é responsável pelo tratamento dos dados recebidos pela unidade remota e pela informação que será gerada. A unidade remota irá receber um arquivo de entrada *serial.inp* e então transmite para a UART . O driver é divido em duas partes, em handler e um conjunto de funções que permitem ao programador enviar e receber através da interface serial. A este conjunto de funções é que chamaremos de driver. O handler irá receber os dados da UART e então poder ser acessado pelo conjuntos de funções para ser possível que o código de fibonacci seja executado e então depois a informação é passada para a fila de transmissão que manda a saída para UART. Enquanto o handler opera as funções não podem fazer nada e vice-versa e isso é controlado pelas funções enableInterr(void) e disableInterr(void).**

***Abstract*—The driver is responsible for handling the data received by the remote unit and for the information that will be generated. The remote unit will receive a textit serial.inp input file and then transmit to the UART . The driver is divided in two parts, in handler and a set of functions that allow the programmer to send and receive through the serial interface. To this set of functions we will call the driver. The handler will receive the data from the UART and then be accessible by the function sets to be possible for the fibonacci code to be executed and then the information is passed to the transmission queue that sends the output to the UART. While the handler operates the functions can do nothing and mutually and this is controlled by the enableInterr (void) and disableInterr (void) functions.**

***Index Terms*—driver, UART, fibonacci, handler, funções.**

I. Introdução

A interface serial cMIPS é um modelo VHDL sintetizável para os clássicos 5 estágios do pipeline (Busca da instrução, decodificação, execução, memória e escrita de volta).

A busca de instrução é a primeira etapa onde o processador acessa a memória e busca a instrução que será executada. Após isso temos a decodificação onde é identificado a operação que será realizada através dos bits que representam as instruções (opcode) e a mesma é executada com os operandos necessários na ULA. Caso necessário a operação é salva na memória e/ou em um dos registradores, caracterizando a escrita de volta (write back).

O driver irá operar junto a esta interface e gerar uma saída que possuirá o n-ésimo número de fibonacci dado pela entrada. O driver implementado é divido em duas partes: um tratador de interrupções, o handler, e um conjunto de funções.

A UART irá se comunicar com o handler que por sua vez se comunicará com o driver que irá devolver uma resposta ao handler e então essa informação voltará irá para a saída da UART. É possível o melhor entendimento olhando a figura no apêndice A.

A. Handler

O handler irá operar no meio do caminho entre a UART e o código em C. Aqui fica também dois contadores, o nrx que indica o número de caracteres disponíveis na fila de recepção e rtx que indica o número de espaços na fila de transmissão. Seu código é escrito em assembly e acrescentado ao arquivo handlers.s que servirá de biblioteca para o código em C.

Ele possui a fila de recepção e a fila de transmissão as quais, respectivamente, são responsáveis por receberem os dados da UART e receber os dados do driver. Os dados da fila de transmissão serão mandados de volta para a UART para que a saída seja exibida.

B. conjunto de funções: Driver

O tratador mantém dois contadores, nrx indica o número de caracteres disponíveis na fila de recepção, enquanto que ntx indica o número de espaços na fila de transmissão. O driver vai executar também um calcúlo de fibonacci e enviar esse valor para a fila de transmissão do handler.

1) int proberx(void): função responsável por retornar o valor de nrx.

```
int proberx(void){
    return Ud.nrx;
}
```

2) int probetx(void): função responsável por retornar o valor de ntx.

```
int probetx(void){
    return Ud.ntx;
}
```

3) Tstatus iostat(void): função responsável por retornar o conteúdo do registrador de status da UART. Segue aqui a sua estrutura para ser usada como base para poder fazer a função:

```
typedef struct status {
#if 0
    int s;
#else
    int ign : 24,
    cts     : 1,
```

```
    txEmpty : 1,
    rxFull  : 1,
    int_TX_empt: 1,
    int_RX_full: 1,
    ign2    : 1,
    framing : 1,
    overun  : 1;
#endif
}Tstatus;

int iostat(){
    return uart->cs.stat.s;
}
```

4) void ioctl (Tcontrol): função responsável por escrever o byte menos significativo no registrador de controle; Segue aqui a sua estrutura para ser usada como base para poder fazer a função:

```
typedef struct control {
  int ign   : 24,
    rts     : 1,
    ign2    : 2,
    intTX   : 1,
    intRX   : 1,
    speed   : 3;
} Tcontrol;

void ioctl(int ictl){
    Tcontrol ctrl;
    ctrl.speed = ictl & 0x7;
    ctrl.intRX = (ictl>>3) & 0x1;
    ctrl.intTX = (ictl>>4) & 0x1;
    ctrl.ign   = (ictl>>5) & 0x7;
    uart->cs.ctl = ctrl;
}
```

5) char Getc (void): retorna o caractere que está na cabeça da fila de recepção e decrementa nrx, ou EOF se a fila estiver vazia.

```
char getc(){
    char c = EOF;
    if(proberx() > 0){
        disableInterr();
        c = Ud.rx_q[Ud.rx_hd];
        Ud.rx_hd = (Ud.rx_hd+1)%16;
        Ud.nrx--;
        enableInterr();
    }
    return c;
}
```

6) int Putc(char): função responsável por inserir um caractere na fila de transmissão e decrementa ntx. Esta função deve verificar se ntx é maior que zero e então inserir o novo caractere em txreg. Se ntx = 0 Putc() retorna -1, senão, retorna 0.

```
void Putc(char c){
    if(probetx() > 0){
        disableInterr();
        Ud.tx_q[Ud.tx_tl] = c;
        Ud.tx_tl = (Ud.tx_tl+1)%16;
        Ud.ntx--;
        enableInterr();
    }
    else{
        writeFirstChar();
        Putc(c);
    }
}
```

II. Funcionamento

A execução começa com um arquivo chamado serial.inp esse arquivo possui inteiros representados em hexadecimal (para fins de comunicação serial) por uma sequência de caracteres terminadas por '\n'. Uma cadeia vazia '\n"\n' indica o final dos dados, por exemplo:

```
A 6 2 B 1 D C 1 \n = 2.787.843.521
4 2 \n = 0x42
\n \n = fim dos dados
```

O handler ao receber esse arquivo o coloca na fila de recepção para que assim seja possível que o driver com a função Getc() consiga receber os valores para então colocá-los em uma fila de cadeias e então converter de char para int e então indexar os valores em um vetor de inteiros.

Feito isso o vetor é convertido de int para char e então o Putc() do driver irá colocar os caracteres do vetor na fila de transmissão para que seja repassada pela UART.

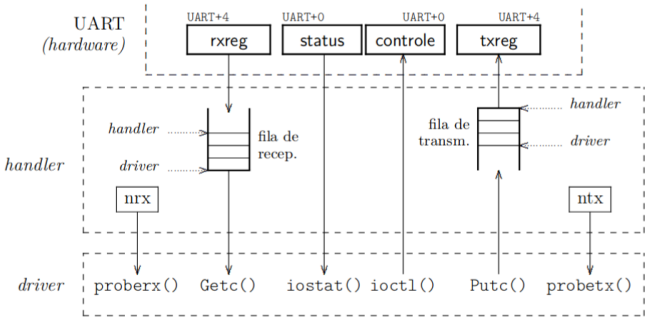
É importante ressaltar que o handler não pode operar ao mesmo tempo que o driver. As interrupções controladas pelas funções enableInterr(void) e disableInterr(void) estão definidas em include/handlers.s.

Unindo o código fonte em C e o código em assembly na aplicação, nós temos a execução do programa do driver de interface serial mostrado no gtkwave que nos informa como foi feito lá dentro da UART.

III. Conclusão

O driver serial é capaz de estabelecer uma comunicação com a UART, é responsável pelo tratamento dos dados recebidos pela unidade remota e pela informação que será gerada.

Através do handler é possível de controlar as interrupções e com o driver (conjunto de funções) é possível o gerenciamento de dados com interrupções no cMIPS o que indica como funcionam os drivers dentro de um sistema operacional e como ele trabalha com o processador.



APÊNDICE A
DIAGRAMA DE BLOCOS DO DRIVER DA UART

REFERÊNCIAS

[1] Roberto André Hexsel. 'Segmentação no cMIPS'. [online]. http://www.inf.ufpr.br/roberto/ci212/labSegm.html. [acessado 26-06-2019]

[2] Roberto André Hexsel. 'Primeiro Trabalho: Driver para a Interface Serial — 2017-1'. [online]. http://www.inf.ufpr.br/roberto/ci315/trab1-17pri.pdf. [acessado 26-06-2019]

[3] Roberto André Hexsel. 'cmpis'. [online]. https://github.com/rhexsel/cmips. [acessado 26-06-2019]

[4] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.