UFPR – Bacharelado em Ciência da Computação
CI315 – Projeto de Sistemas Operacionais
Primeiro Trabalho: Driver para a Interface Serial — 2017-1 11 de abril de 2017

Sua tarefa é implementar um *driver* para a interface serial do cMIPS. O *driver* deve funcionar por interrupções nos dois sentidos (rx e tx).

Você deve escrever um programa em C que retorna o i-ésimo número de Fibonacci, para $i \in [0, 100)$. O vetor de números está no arquivo `fib_vet.h`, que deve ser incluído de forma a que o vetor seja uma variável global.

Seu programa inicia a comunicação com a remota (fazendo RTS=1), e então recebe pela interface serial uma sequência de inteiros. Para cada inteiro recebido, seu programa transmite, também pela interface serial, o número de Fibonacci que lhe corresponde.

O circuito da interface serial (*Universal Asynchronous Receiver-Transmitter* ou UART) é aquele descrito na Seção 7.3 de [RH12] e nas notas de aula desta disciplina. Estude estes documentos com atenção. A UART deve ser programada para operar com caracteres de 8 bits, sem paridade e com 1 *stop-bit*. O dispositivo opera com *double buffering* nos dois sentidos de transmissão.

***Driver* para a UART** A Figura 1 mostra um diagrama de blocos de um *driver* simplificado para a UART. Este *driver* é dividido em duas partes: um tratador de interrupções (*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*.

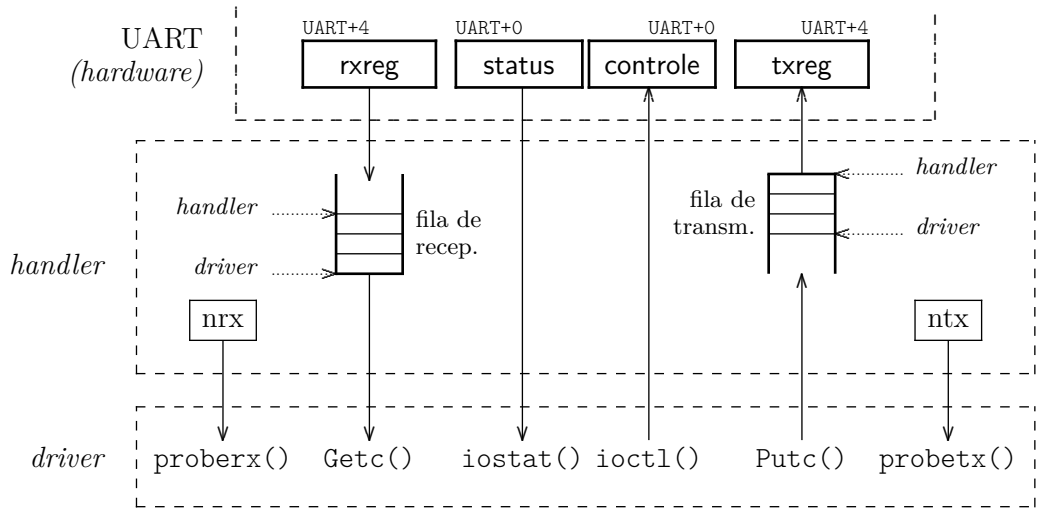

Figura 1: Diagrama de blocos do *driver* da UART.

O tratador de interrupções gerencia duas filas de caracteres, uma associada à recepção, e outra à transmissão, cada uma com capacidade para 16 caracteres. 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 tratador de interrupções deve ser escrito em *assembly* e seu código deve ser acrescentado à `include/handlers.s`.

As funções `proberx()` e `probetx()` retornam os valores de `nrx` e `ntx`, respectivamente. A função `iostat()` retorna o conteúdo do registrador de status da UART, e a função `ioctl()` permite escrever no seu registrador de controle.

A função `Getc()` retorna o caractere que está na cabeça da fila de recepção e decrementa `nrx`, ou `EOF` se a fila estiver vazia.

A função `Putc()` insere 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`.

Estas funções devem estar contidas em um único arquivo, que contém a função `main()`. Os protótipos são listados abaixo.

```
int  proberx(void);     // retorna nrx
int  probetx(void);     // retorna ntx
Tstatus iostat(void);   // retorna inteiro com status no byte menos sign.
void ioctl(Tcontrol);   // escreve byte menos sign no reg. de controle
char Getc(void);        // retorna caractere na fila, decrementa nrx
int Putc(char);         // insere caractere na fila, decrementa ntx
```

As funções `enableInterr()` e `disableInterr()` estão definidas em `include/handlers.s` e habilitam e desabilitam as interrupções; as duas retornam o conteúdo do registrador Status do processador *após* a alteração do bit Status.intEn. Estas funções devem ser usadas para impedir a execução concorrente do *driver* e do *handler*, quando o *driver* atualiza os ponteiros das filas circulares, `nrx` e `ntx`.

```
int  enableInterr(void);  // habilita interrupções, retorna Status
int  disableInterr(void); // desabilita interrupções, retorna Status
```

A Figura 2 mostra um diagrama com o fluxo dos dados do *driver* mais o tratador. A unidade remota lê o conteúdo do arquivo `serial.inp` e o transmite para a UART. O circuito de recepção da remota recebe os caracteres da UART e os exibe na saída padrão do simulador.

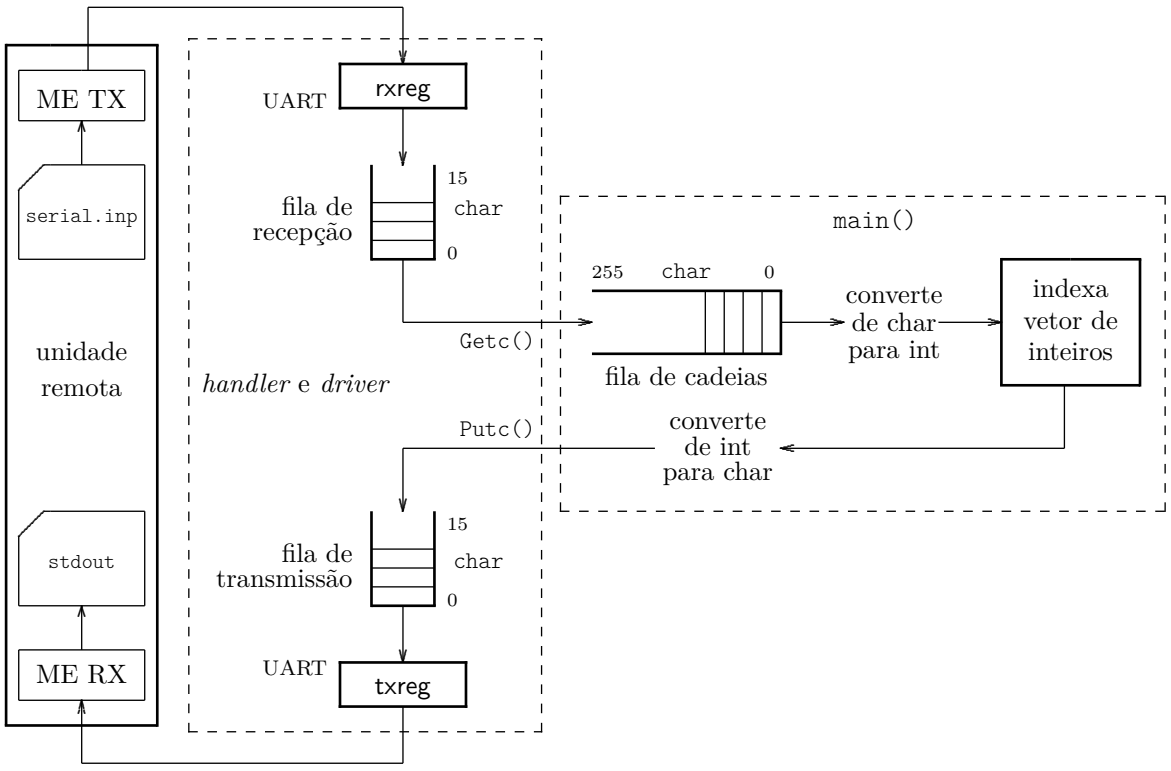

Figura 2: Fluxo de dados no *driver* e aplicação da UART.

O tratador da interrupção lê um caractere do registrador de dados da UART e o insere na fila de recepção.

A função `main()` remove um caractere da fila de recepção e o insere na fila de cadeias. Quando uma cadeia completa é recebida, esta cadeia é convertida para um inteiro N, e este é usado para indexar o vetor de Fibonacci. O Fibonacci(N) deve ser convertido para uma cadeia de caracteres e então transmitido, caractere a caractere através da fila de transmissão.

Quando uma cadeia vazia for recebida, o programa `main()` termina.

Interrupção de transmissão Para provocar a primeira interrupção da transmissão é necessário programar a UART para interromper na transmissão, *e então escrever o primeiro caractere a ser transmitido* diretamente no txreg – esta escrita provocará uma interrupção *após* o primeiro caractere ter sido copiado para o registrador de transmissão da UART.

Da implementação É recomendável que você des-comente o `.include` e trabalhe diretamente com `tests/handlerUART.s` porque este arquivo não faz parte do controle de versões e portanto não há risco de você perder seu trabalho quando ocorrer alguma alteração no modelo do cMIPS, nos programas de teste, ou no código do 'SO' primitivo do cMIPS.

As variáveis, filas e ponteiros, dos 'caminhos' de recepção e de transmissão estão declaradas no arquivo `include/handlers.s`. O código no arquivo com `main()` deve declarar estas variáveis com atributo **`extern`**. O vetor `_uart_buff` é o espaço reservado para salvar os registradores alterados pelo tratador da interrupção, e não é visível ao *driver*.

O Programa 1 mostra o leiaute da estrutura de dados na memória, como declarado em `include/handlers.s`.

Programa 1: Alocação da estrutura de dados para *driver* da UART.

```
        .global Ud, _uart_buff
Ud:
rx_hd:  .space 4        # index to queue head
rx_tl:  .space 4        # index to queue tail
rx_q:   .space 16       # reception queue
tx_hd:  .space 4        # index to queue head
tx_tl:  .space 4        # index to queue tail
tx_q:   .space 16       # transmission queue
nrx:    .space 4        # characters in RX_queue
ntx:    .space 4        # spaces left in TX_queue

_uart_buff: .space 16*4 # up to 16 registers to be saved here
```

Com base no Programa 1, uma estrutura de dados, que deve ser ser usada no *driver*, é mostrada no Programa 2

Programa 2: Definição da estrutura de dados para *driver* da UART.

```
typedef struct UARTdriver {
   int      rx_hd;     // reception queue head index
   int      rx_tl;     // reception queue tail index
   char     rx_q[16];  // reception queue
   int      tx_hd;     // transmission queue head index
   int      tx_tl;     // transmission queue tail index
   char     tx_q[16];  // transmission queue
   int      nrx;       // number of characters in rx_queue
   int      ntx;       // number of spaces in tx_queue
} UARTdriver;

extern UARTdriver Ud;
```

Da representação de cadeias e inteiros Para fins da comunicação serial, um inteiro é representado em hexadecimal por uma sequência de caracteres terminadas por `'\n'`. Os inteiros devem estar no intervalo $[0, 100)$. Uma cadeia vazia `'\n''\n'` indica o final dos dados. Abaixo são mostrados alguns dos valores válidos – os espaços não são transmitidos e servem apenas para facilitar a leitura.

`0 \n` $= 0$
`0 0 \n` $= 0$

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

Veja a página de manual da codificação ASCII para uma forma simples de conversão dos caracteres para os dígitos representados: `man ascii` .

Para simplificar a simulação e os testes, o conjunto de dados será pequeno – menor ou igual a dezesseis elementos – e os inteiros representados em até oito dígitos hexadecimais, mais o `'\n'`.

O compilador e demais ferramentas que geram código para o MIPS estão em
`/home/soft/linux/mips/cross/bin` .
Acrescente este caminho à sua variável de ambiente `PATH`.

Se for conveniente instalar as ferramentas de compilação em seu computador pessoal, siga as instruções em `cMIPS/docs/installCrosscompiler`. A instalação demora cerca de uma hora, mais o tempo para fazer o *download* dos 5 pacotes necessários.

Especificação:

1. O trabalho pode ser efetuado em duplas;
2. o arquivo com os produtos deve ser nomeado `xx-yy.tgz` sendo `xx` e `yy` os *usernames* dos componentes do grupo, e todos os arquivos relevantes deverão estar abaixo do diretório `xx-yy`;
3. PLÁGIO NÃO SERÁ TOLERADO. É interessante que os alunos conversem sobre o projeto mas cada grupo deve escrever seu próprio código.

Produtos:

1. Relatório em papel A4, com letras em 11 pontos, espaço simples, formatação simples, contendo os nomes dos componentes do grupo, e o código do tratador de interrupções em *assembly*.
2. todo o código C e *assembly* do programa de comunicação deve ser entregue no *tarball*.
3. arquivo enviado por e-mail para `rhexsel@gmail.com` contendo todos os arquivos fonte necessários para testar o programa de comunicação. Projetos com arquivos faltando e que não possam ser testados receberão nota zero. Todos os programas serão recompilados antes de simulados na avaliação, e os arquivos de teste serão alterados para a verificação do trabalho;
4. presença dos membros do grupo na data e hora marcadas para a apresentação.

Sugestões:

1. Assegure-se de que entendeu a especificação antes de iniciar o projeto do *driver*;
2. assegure-se de que entendeu as notas de aula sobre a UART antes de escrever qualquer linha de código;
3. escreva as funções de recepção (use o código em `include/handlers.s` como modelo); isso pronto, escreva as funções para a transmissão; isso pronto, escreva um programa de testes que faz eco e somente repete na saída a entrada; com isso funcionando corretamente, e só então, escreva o código das funções de conversão de cadeias para inteiros.

Histórico das Revisões:

11abr	correção aos comentários do Programa 1, índices e não pointers;
06abr	acrescentada unidade remota à Figura 2, tipos de `ioctl()` e `iostat()`;
03abr	vetor de Fibonacci é uma constante;
28mar	publicação.

Referências

[RH12] *Sistemas Digitais e Microprocessadores*, R.A.Hexsel, 2012, Editora da UFPR.

[RH01] *Redes de Dados: Tecnologia e Programação*, R.A.Hexsel, 2001, Relatório Técnico do Depto. de Informática da UFPR, RT-DInf 005-2001, `http://www.inf.ufpr.br/roberto/rt005_2001.pdf`