Projecto SO

Carlos Manuel Sanches Leocádio (ee09313)

Bruno Emanuel Cardoso Miranda Gonçalves (ee09274)

Introdução

Este documento pretende dar uma perspectiva geral sobre a implementação de um char device driver para Linux, baseando-se em algum do código do driver scullc. Este o projecto está dividido em duas pastas correspondentes ao driver em modo "polling" – serp – e numa outra correspondente ao driver com suporte a interrupções – seri.

Como o código correspondente ao driver seri é uma evolução do predecessor serp, este documento apresentará as funções implementadas no primeiro, bem como as respectivas melhorias.

1. **seri_init()**
    a. Esta função é responsável pela inicialização dos parâmetros da comunicação UART, como por exemplo velocidade da ligação, bits de paridade considerados, e capacidade de geração de interrupções. Estes parâmetros são definidos actuando directamente nos registos correspondentes, por exemplo, UART_LCR, UART_DLL, UART_IER, entre outros.
    b. É também activado o suporte para FIFO's ao nível do hardware;
    c. É invocada a função *alloc_chrdev_region()* que procede ao registo ao nível do kernel do número de device drivers pretendidos;
    d. É efectuada a alocação de memória limpa ao nível do kernel onde serão alocados os vários devices registados, recorrendo para tal à função kzalloc(); após este procedimento são inicializados adequadamente os elementos da estrutura que corresponde ao device (a estrutura será apresentada adiante).
    e. Por fim é configurada a funcionalidade das interrupções recorrendo à função request_irq(), e é reservada a região de I/O correspondente à UART por invocação da função request_region().
2. **seri_open()**
    a. Esta função é responsável pela abertura do device, quando invocada.
    b. Encontra-se implementada de forma a evitar problemas de concorrência, ou seja, apenas pode haver um utilizador a utilizar o device num dado instante.
    c. A região em que se procede à verificação do número de utilizadores está protegida por intermédio do mecanismo de spin_lock, pois representa uma zona crítica.
    d. Aqui, o device é inicializado em modo nonseekable, através da função nonseekable_open();
3. **seri_write()**

a. Esta função é responsável pelo envio de caracteres quando utilizando o device;
b. Para tal é alocado dinamicamente um buffer de chars, ao nível do kernel - echoKernelBuf -, de dimensão correspondente a *count* que recebe como argumento;
c. A função copia de user space (conteúdo presente em buff), para o buffer em kernel space alocado previamente recorrendo para tal à função copy_from_user(echoKernelBuf, buff, count);
d. Os caracteres presentes em echoKernelBuf são posteriormente enviados, um a um;
e. Caso seja detectado que o caractere não pode ser enviado, a função dorme, por invocação da função msleep_interruptible(); A utilização desta função tem em conta o facto de o driver estar implementado de forma a suportar interrupções;
f. Por fim, a função limpa o buffer previamente alocado, e procede ao retorno do número de caracteres escritos.

4. **seri_read()**
    a. Esta função começa por alocar buffer com o tamanho adequado à informação que espera receber, com base no parâmetro count; este buffer é uma variável global;
    b. Para evitar espera activa, a função é colocada a dormir utilizando para tal wait_event_interruptible(wq, flag); em consequência, o restante procedimento de leitura é tratado pela função IH;
    c. A função IH, por sua vez, procede à leitura do registo adequado à recepção, armazena no buffer o caractere recebido, e procede à ativação da flag e consequente acorda a função read();
5. **seri_release()**
    a. Esta função quando invocada, liberta o device, atualizando o valor correspondente ao número de utilizadores, que se encontra na estrutura;
    b. Esta acção, por corresponder a uma zona critica, está protegida pela utilização do mecanismo de spin_lock;
6. **seri_exit()**
    a. Através desta função é libertado o espaço previamente alocado para os devices na função init();
7. **interrupt_handler()**
    a. Esta é a função responsável pelo tratamento das interrupções quando estas ocorrem, quer durante os procedimentos de escrita quer de leitura;
    b. No caso da função read(), o disparo da flag e consequente wake_up_interruptible() é invocado assim que um caractere é recebido; o buffer é então retornado para user space quando o utilizador prime ENTER;
    c. No caso da função write, o IH é invocado em consequência do adormecimento causado pela invocação da função msleep_interruptible(); Aqui os caracteres são enviados assim mal haja disponibilidade.
8. **seri_ioctl()**
    a. Esta função permite alterar as configurações da comunicação, nomeadamente bit de paridade, stop bit, velocidade de ligação, entre outros;
    b. O ficheiro de testes desenvolvido permite alterar e confirmar a correcta alteração sobre estes parâmetros.

9. **O_NONBLOCK**
    a. Foi implementado no seri nas funções de read (linha 436) e write (linha 397).
10. **FIFO**
    a. Implementado no seri, é ativado na função init (linha 267), no entanto visto não ser suportado pelo funcionamento da VM, não foi possível testá-lo de forma conclusiva.

Consolidando os contributos de ambos os elementos do grupo para o resultado final do trabalho, a distribuição é a seguinte:

- Carlos Leocádio 60%

- Bruno Gonçalves 40%