

Dicas NetOS 7.x – Mecanismo de atualização no bootloader

Introdução

Desde suas versões anteriores e agora de maneira mais abrangente nas versões 7.x, o bootloader padrão das aplicações NetOS já conta com alguns mecanismos de recuperação de firmware, entretanto, estes mecanismos só entram em ação, nos casos em que o programa carregado na flash esteja corrompido por algum motivo, e não por intervenção direta do usuário.

Este artigo tem como objetivo fornecer dicas, para que o bootloader padrão possa ser alterado, de forma a permitir a criação de um modo de manutenção, em que seja possível incluir diagnósticos e mecanismos de correção de problemas. Não se trata de uma solução completa, uma vez que as necessidades desta tela de manutenção tendem a ser específicas para cada aplicação. Nosso objetivo aqui é, sugerir um guia para que o desenvolvedor possa compreender a mecânica das customizações no bootloader.

Objetivo das alterações propostas

As dicas aqui apresentadas, têm como objetivo final monitorar durante o boot os pinos 18 e 20 do Digi ConnectME, executando tarefas específicas caso estejam conectados à terra.

O pino 20 (denominado “/init” ou “Software Reset”) foi previsto para que seja conectado a um *push botom* ou jumper de reset, neste exemplo ele será utilizado para colocar as configurações de rede e senha do equipamento em padrões conhecidos, muito útil em situações em que se deseja corrigir problemas de configuração do equipamento.

O pino 18 é reservado pelo fabricante para testes de fábrica e por este motivo, fica conectado na placa de desenvolvimento ao pino 3 do conector MFG (P5). Neste exemplo ele será utilizado para enviar para a “SERIAL A” uma tela com opções de manutenção.

Nosso foco nesta tela de manutenção é fornecer uma opção que permita ao módulo, buscar uma imagem de firmware via TFTP (Trivial File Transfer Protocol) em um servidor remoto e gravá-la em sua memória Flash.

Dicas para implementação das funcionalidades

As alterações apresentadas devem ser feitas no arquivo “blmain.c”.

No início da função “int NABlMain(void)”, antes de “#ifdef BL_TFTP_DEBUG_MENU” e depois da inicialização do driver simplificado de serial (setupSimpleSerial) e da inicialização da flash (NAFlashInit), adicionar o código que testa os jumpers no boot.

```
.....  
  
if( !(narm_read_reg(NARM_PORTX_REG, NARM_PORTC_ADDR, data) & BIT_1) )  
{  
    /*  
    Este ponto é atingido se durante o boot existir um jumper entre o pino 18 e o  
    terra (pinos 3 e 4 de P5)  
  
    Rotinas para a tela de manutenção (A)  
  
    */  
}  
else if( !(narm_read_reg(NARM_PORTX_REG, NARM_PORTC_ADDR, data) & BIT_5) )  
{  
    /*  
    Este ponto é atingido se durante o boot existir um jumper do pino 20 para o  
    terra  
  
    Rotinas para restauração dos padrões de fábrica (B)  
  
    */  
}  
  
.....
```

A) Tela com opções de manutenção (pino 18)

Uma vez acionada pela colocação do jumper entre os pinos 3 e 4 do conector **P5**, o programa precisa montar e enviar para a serial a tela com as opções de manutenção.

Neste ponto estamos ainda no bootloader, e a maioria das funcionalidades e drivers do sistema operacional ainda não foram carregadas, sendo assim, precisamos utilizar funções preparadas especificamente para utilização dentro deste.

Um exemplo destas funções específicas é o "`bsp_printf`" que aqui, será utilizada em conjunto com o driver "simpleSerial" para enviar a tela de manutenção para a serial, como segue:

```
.....
bsp_printf("Opções de recuperação\n");
bsp_printf("\t[1] para download de firmware via TFTP\n");
bsp_printf("\t[2] para OPCA0 2\n");
bsp_printf("\t[N] para OPCA0 N\n");
.....
```

Depois de enviada a tela para a serial, o programa deve aguardar em *loop*, a escolha de uma das opções apresentadas. A escolha do usuário pode ser recuperada com a função "`bsp_fgetc`", em substituição a `getc()` da biblioteca padrão. Quando uma opção for selecionada, o programa deve setar uma flag informativa, para que em uma etapa posterior possamos identificar a seleção do usuário.

Se a opção selecionada pelo usuário for a tela de download de firmware, deverão ser solicitados os seguintes parâmetros auxiliares: IP do servidor TFTP, IP do Módulo, Máscara de sub-rede e IP do Gateway. Uma sugestão para permitir a entrada cada parâmetro auxiliar é receber os caracteres digitados um a um em *loop* com a função "`bsp_fgetc`", até receber um `0x0d`, passando então para a coleta do próximo parâmetro e assim por diante.

Depois de coletados todos os dados auxiliares, será necessário adicionar cláusulas "`else if`" em "`if (shouldDownloadImage())`", que sejam executadas de acordo com as flags setadas nas opções fornecidas no menu de recuperação.

```
.....
if(shouldDownloadImage()) {
    .....
} else if (DOWNLOAD_TFTP) {
    // Código para a atualização de firmware por TFTP
    .....
} else if (OPCA0_N) {
    // Código para implementar a opção N da tela de manutenção
    .....
}
.....
```

Dentro da cláusula "else" correspondente a atualização de firmware por TFTP, pode-se fazer uso das funções "`blUdpInit`" e "`blTftpGetFile`" para receber na RAM o arquivo "image.bin" contendo o novo firmware a ser gravado na FLASH. Depois de recebido na RAM, para evitar a gravação de imagens corrompidas na flash, o arquivo precisa ser validado, utilizando-se a função "`isImageValid`".

Depois da validação, o arquivo agora precisa ser gravado na memória FLASH. Para isso, vamos utilizar a função "`blWriteToFlash`", gravando os dados a partir do offset "`NAAppOffsetInFlash`" com o tamanho do arquivo recebido. Após a gravação, chamar a função "`customizeReset()`" para reinicializar o módulo, já agora com o novo firmware instalado.

B) Restauração das configurações de fábrica (pino 20)

O intuito deste mecanismo é de agir como um “reset” das configurações do módulo, forçando o mesmo a assumir valores conhecidos. Para isso, em primeiro lugar será necessário ler os parâmetros atuais de configuração da NVRAM para uma estrutura em memória, utilizando a função “[customizeReadDevBoardParams](#)”, em seguida será necessário localizar as configurações da interface de rede e alterá-las, como mostrado no trecho de código a seguir:

```
.....
// Localiza as configurações da interface de rede
interface = customizeIamFindInterfaceConfig(BP_ETH_INTERFACE, &nvParams.iamParamsInfo);

// Ajusta os parâmetros para valores conhecidos
interface->staticParams.IPV4_ADDR(ipAddress) = NAIInet_addr("192.168.0.1");
interface->staticParams.subnetMask = NAIInet_addr("255.255.255.0");
interface->staticParams.IPV4_ADDR(gateway) = NAIInet_addr("192.168.0.1");
interface->staticParams.isEnabled = TRUE;
.....
```

Uma vez reconfigurados os parâmetros da interface de rede, uma boa idéia seria reconfigurar também a senha de *root* do módulo:

```
.....
// Reset na senha de root
strcpy(nvParams.rootPassword, "Netsilicon");
.....
```

Finalmente, vamos devolver as configurações para a NVRAM do equipamento:

```
.....
status = customizeWriteDevBoardParams(&nvParams);

if (status != BP_SUCCESS) {
    bsp_printf("\r\n\r\nErro no WRITE\r\n");
} else {
    bsp_printf("\r\n\r\nSucesso no WRITE.\r\n");
}
.....
```

Para indicar ao usuário que o processo de gravação foi finalizado, e que o jumper do pino 20 pode ser retirado, vamos utilizar o Led de atividade de rede (verde) do *ConnectME* para piscar em um padrão conhecido, até que se retire o jumper:

```
.....
while (1){
    // Intervalo de 5 ticks
    NAWait(5);
    // Um ciclo longo
    NALedBlink(LED_ETHERNET, 1, 150);
    // Intervalo de 5 ticks
    NAWait(5);
    // Cinco ciclos curtos
    NALedBlink(LED_ETHERNET, 5, 75);
    if( (narm_read_reg(NARM_PORTX_REG, NARM_PORTC_ADDR, data) & BIT_5) ){
        break;
    }
}
.....
```

Com isto encerramos nossas dicas para alteração do bootloader, lembrando que os códigos apresentados são apenas sugestões de implementação e que cada desenvolvedor está livre para fazer as alterações da maneira mais conveniente, de acordo com suas necessidades específicas.