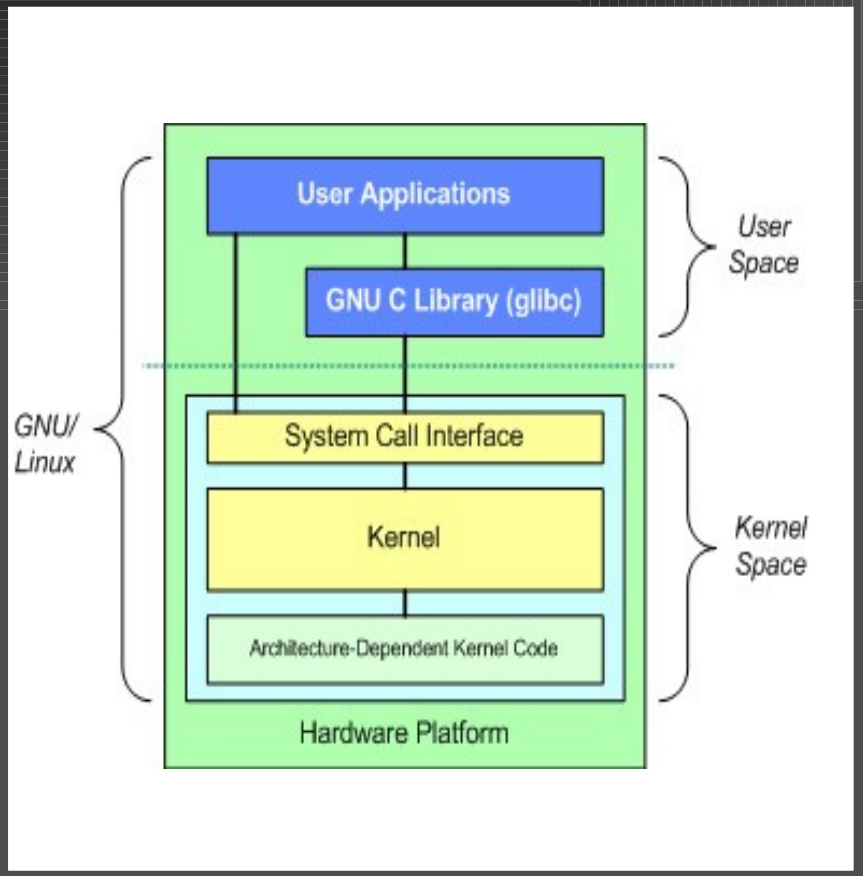
Desenvolvimento de um Device Driver para GNU/Linux – Plataforma ARM

Luiz de Barros Oliveira Neto

Lboneto@gmail.com

14/11/2009

Arquitetura do GNU/Linux



Sistema Operacional de Código aberto, disponibilizado sob licensa GPL V2.0, formados pelos seguintes componentes:

- Linux: É o núcleo do sistema operacional e onde estão implementadas as funcionalidades dependentes da arquitetura de hardware, comunicação, rede.
- GNU: É composto pela ampla gama de aplicativos que compoem, juntamente com o kernel, o sistema operacial, incluindo editores, compiladores, interpretadores de comandos, bibliotecas.

Por que utilizar GNU/Linux Embarcado?

- Ampla gama de opções de desenvolvimento (C, C++, Java, Perl, PHP, Python);
- Grande variedade de aplicativos (telnet, ftp, http, ssl, sql, correio eletrônico, compiladores, aplicativos multimedia e interface com usuário);
- Disponibilidade de pilhas de protocolo completas para TCP/IP, I2C, SPI, Sistema de Arquivos, Comunicação sem fio, captura e disponibilização de audio, tratamento de vídeo, criptografia;
- Disponibilidade de Drivers para a maioria dos dispositivos disponíveis na maioria dos SoCs;
- Menor TCO (inexistência de custos com licensas ou royalities);
- Foco no negócio e não no ambiente.

Quem usa Linux Embarcado

- Segundo pesquisa realizada pela a Embedded Systems Design Jul/ Ago 2009, 25% dos entrevistados consideram usar Linux Embarcado no seu próximo projeto;
- Segundo o site linuxfordevices, 32% dos netbooks em 2009 são equipados com Gnu/Linux. A previsão para 2013 é este número ultrapassar o número de netbooks com Windows;
- O linux é utilizado hoje em dispositivos com alto volume de produção como: telefones celulares, pdas, dispositivos de entretenimento de bordo, settop boxes, internet tablets.

Kernel Space x User Space

Kernel Space

- API Limitada (não tem memset, malloc, etc) implementada diretamente no kernel
- Não é SWAPPABLE
- C e Assembly (NO C++)
- Controle limitado de excessões
- Evitar ponto flutuante
- Stack limitado (4K, 8K)
- Proteção de memória limitada

User Space

- Utiliza APIs do KERNEL
- Roda sob o controle do kernel
- Processo não tem muito controle de sua alocação de CPU
- Processo pode usar memória física ou virtual
- Melhor controle de excessões
- Pode usar ponto flutuante
- Stack pode crescer dinâmicamente
- Proteção de memória/processo

Loadable Kernel Module

- Adicionam uma dada funcionalidade ao kernel
- Podem ser carregados ou removidos a qualquer momento
- Tem acesso total ao espaço de memória do kernel
- Facilitam o desenvolvimento de driver, sem precisar rebootar constantemente a plataforma

Um módulo básico

```
/* hello.c */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
static int __init hello_init(void)
{
    printk(KERN_ALERT "Good morrow");
    printk(KERN_ALERT "to this fair assembly.\n");
    Return 0;
}
static void __exit hello_exit(void)
{
    printk(KERN_ALERT "Alas, poor world, what treasure");
    printk(KERN_ALERT "hast thou lost!\n");
}
module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module");
MODULE_AUTHOR("William Shakespeare");
```

http://freeelectrons.com/doc/c/hello.c

__init:

removed after initialization (static kernel or module).

__exit: discarded when module compiled statically into the kernel.

http://freeelectrons.com/doc/c/hello.c

Funções de Inicialização e Encerramento

module_init()

- Macro que define o ponto de entrada;
- Acionado quando o módulo é carregado;
- Em caso de falha, o módulo será removido;
- Pode haver apenas uma por módulo.

module_exit()

- Macro que define o ponto de saída;
- Acionado quando o módulo é removido;
- Não possui valor de retorno;
- Remove o módulo da memória na saída
- Pode haver apenas uma por módulo.

Licença

MODULE_LICENSE

Comunica ao kernel o tipo de licensa do módulo.

Alguns símbolos do kernel e informações de debug podem não ser expostas para módulos não GPL.

Exemplos:

MODULE_LICENSE("GPL");

MODULE_LICENSE(“Copyright (c) 2007...”);

Descrição e Autor

MODULE_DESCRIPTION("Greeting module");

Identifica a descrição do módulo.

MODULE_AUTHOR("William Shakespeare");

Identifica o autor do módulo.

O Makefile do Módulo

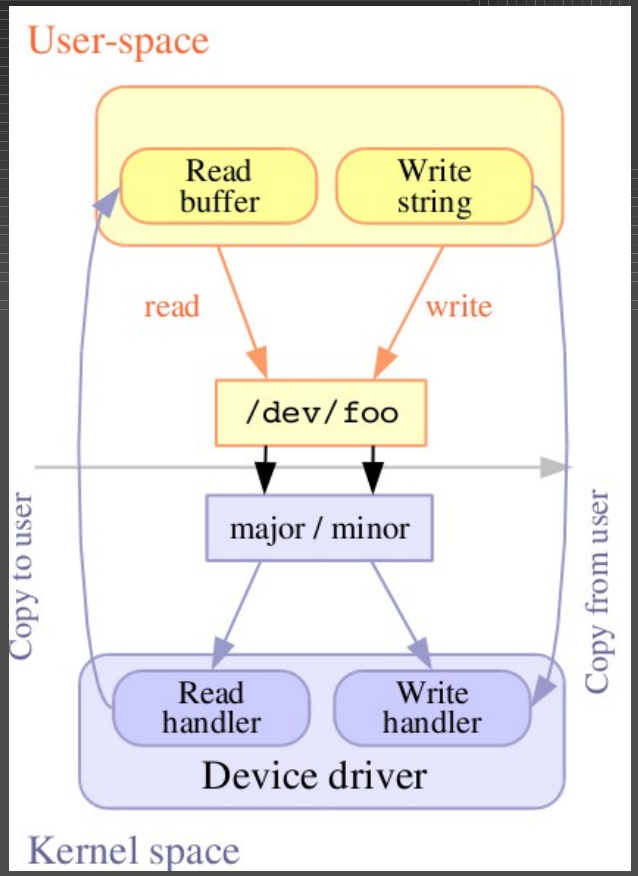
```
# Makefile for the hello module
obj-m := hello.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

- Diretório fonte do kernel (configurado e compilado)

- ou somente o diretório dos headers

http://free-electrons.com/doc/c/Makefile

Comunicação Kernel <=> User Space utilizando Character Devices



- Realizada por meio de interface similar a um arquivo (open, read, write, close...) ou pipe (tudo que o userspace precisa para realizar o acesso é o nome do ”arquivo de dispositivo”.
- O kernel identifica o arquivo de dispositivo por meio de um par ”major/minor”.
- O driver deve ter manipuladores para ”File Operations” correspondentes às chamadas Open,Read,Write,Close do Userspace.
- Toda troca de dados entre userspace e kernel space deve ser feita com funções específicas (copytouser e copyfromuser).
- Memória do kernel não pode ser acessada do user space e vice-versa.

Registrando um Character Device

- Registro de número do dispositivo (major e minor). Um driver pode registrar um ou mais manipuladores de dispositivo caracter (dependendo das funcionalidades que implemente).
- Registro das funções de file operations que serão acionadas quando forem chamadas as apis correspondentes no userspace.
- Não devem ser realizadas a partir dos manipuladores de file operations operações complexas de acesso ao dispositivo.

Abrir/Fechar Device

```
/* Chamada quando o userspace abre um device node */
int device_open(struct inode *inode,        /* pointer do inode structure
*/
                struct file *file)          /* pointer do file struct */


/* Chamada quando o userspace fecha um device node */
int device_release(struct inode *inode,     /* pointer do inode structure
*/
                struct file *file)          /* pointer do file struct */
```

Ler/Escrever no Device

```
/* Chamada quando o userspace lê um arquivo de dispositivo*/
ssize_t (*read) (
   struct file *,                    /* pointer do file struct */
   __user char *,                    /* buffer do user-space*/
   size_t,                           /* tamanho do buffer do user-space */
   loff_t *);                        /* Offset no arquivo de dispositivo */


/* Chamada quando o userspace escreve em um arquivo de dispositivo*/
ssize_t (*write) (
   struct file *,                    /* pointer do file struct */
   __user const char *,              /* buffer do user-space*/
   size_t,                           /* tamanho do buffer do user-space */
   loff_t *);                        /* Offset no arquivo de dispositivo */
```

Declarando o File Operations

```
struct file_operations StFops =
{
    read:    device_read ,
    open:    device_open ,
    write:   device_write,
    release: device_release
};
```

Registrando/Eliminando um File Operations

```
int register_chrdev(unsigned int,                    /* major number do device */
                    const char *,                    /* Nome do dispositvo*/
                    const struct file_operations *);  /* File Operations Pointer */
```

```
unregister_chrdev (unsigned int,                    /* major number do device */
                   const char *);                   /*  Device name string */
```

Trocando Informações Kernel ⇔ User-Space

```
#include <asm/uaccess.h>

unsigned long copy_to_user (void __user *to,     /* Buffer destino do usuário */
                            const void *from,    /* Buffer origem no kernel */
                            unsigned long n);    /* tamanho a copiar */


unsigned long copy_from_user (void *to,              /* Buffer destino no kernel */
                              const void __user *from,     /* Buffer origem do usuário */
                              unsigned long n); /* tamanho a copiar */
```

Aguardando um Evento

```
init_waitqueue_head(&sWaitQ) ;        /* Inicializa a fila de espera */

wake_up (&sWaitQ);                    /* Desbloqueia fila de espera  */

sleep_on (&sWaitQ);                   /* Espera por desbloqueio da fila */
```

Processamento Periódico

- Como fazer o pooling periódico de um recurso?
- O kernel dispõe de algum ”hook” para processamento periódico?
- Posso usar interrupções para fazer a leitura de um recurso?
- Como faço uma temporização de alta resolução?

Usando a Interface do Jiffies

```
 init_timer(&TickTimer);                          /* Inicializa o timer */
TickTimer.function = scheduled_routine;           /* Função callback */
TickTimer.data = 0;
TickTimer.expires = jiffies + 1;                  /* Quando o timer expira */
add_timer (&TickTimer);                           /* Adiciona o timer   */

void
scheduled_routine (unsigned long vData)           /* Procedimento  Periódico     */
{
    TickTimer.expires = jiffies + 1;              /*  Registrando proxima chamada */
    add_timer (&TickTimer);                       /* Adiciona o timer   */
    faz_alguma_coisa();
}
```

Acessando o GPIO

- A maioria dos microcontroladores embarcados dispõe de macros ou funções para acesso direto de leitura e escrita nos registradores de GPIO;
- O acesso normalmente é feito utilizando memory mapped I/O.

Funções de GPIO na família AT91RM9200 Atmel

- Exportadas a partir do arquivo ***arch/arm/mach-at91/gpio.c***

```
int at91_set_gpio_input (unsigned pin, int use_pullup);
int at91_set_gpio_output(unsigned pin, int value);
int at91_set_gpio_value(unsigned pin, int value);
int at91_get_gpio_value(unsigned pin);
```

Funções de Apoio

- printk : Semelhante ao printf do user space;
- kmalloc: Alocador dinâmico de memória do kernel;
- Buffers: Pode ser necessário utilizar buffers ou filas circulares para armazenar temporariamente dados a serem comunicados com o userspace;

Analisando um Driver Simplificado

- Identificando as seções no código;
- Compilando;
- Carregando;
- Interagindo com o Driver;
- Removendo Driver.

Referências:

- Linux Device Drivers 3rd Edition:

http://lwn.net/Kernel/LDD3/

- Embedded Linux Kernel and Driver Development:

http://free-electrons.com/doc/embedded_linux_kernel_and_drivers.pdf

Obrigado

Luiz de Barros Oliveira Neto

E-mail: Lboneto@gmail.com