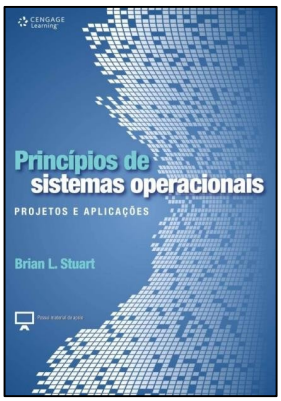
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
SISTEMAS OPERACIONAIS – Aula 10 – 2º SEMESTRE/2019
PROF. LUCIANO SILVA

TEORIA: GERENCIADOR DE DISPOSITIVOS (PARTE II)

Nossos **objetivos** nesta aula são:

- conhecer a estrutura geral de device drivers no sistema operacional MINIX 3.0
- programar exemplos de device drivers em MINIX 3.0



Para esta aula, usamos como referência o **Capítulo 15 (Gerenciamento de Dispositivos)** do nosso livro-texto:

STUART, B.L., **Princípios de Sistemas Operacionais: Projetos e Aplicações**. São Paulo: Cengage Learning, 2011.

Não deixem de ler este capítulo depois desta aula!

DEVICE DRIVER HELLO

- Nosso primeiro exemplo de device driver será para o dispositivo hello, que apenas apresenta a mensagem hello quando o driver é carregado.

- O primeiro passo é criar um diretório com os comandos abaixo:

```
# cd /usr/src/minix/drivers/examples
# mkdir hello
# cd hello
```

- Para compilar o driver, vamos precisar de um Makefile dentro do diretório criado:

```
# Makefile for the hello driver.
PROG=   hello
SRCS=   hello.c

FILES=${PROG}.conf
FILESNAME=${PROG}
FILESDIR= /etc/system.conf.d

DPADD+= ${LIBCHARDRIVER} ${LIBSYS}
LDADD+= -lchardriver -lsys

MAN=

.include <minix.service.mk>
```

- A estrutura básica da implementação do driver é mostrada abaixo:

```
#include <stdio.h>
#include <stdlib.h>
#include <minix/syslib.h>

int main(int argc, char **argv)
{
    sef_startup();

    printf("Hello, World!\n");
    return EXIT_SUCCESS;
}
```

- A chamada sef_startup() registra o programa como um driver e, normalmente, é a primeira chamada dentro da função principal.

- Para compilar e instalar o driver, utilizamos os comandos abaixo:

```
# make clean
# make
# make install
```

- Antes de utilizar o driver, devemos definir as suas permissões, construindo o arquivo hello.conf, conforme mostrado no código abaixo:

```
service hello
{
        system
                UMAP            # 14
                IRQCTL          # 19
                DEVIO           # 21
        ;
        ipc
                SYSTEM PM RS LOG TTY DS VM VFS
                pci inet amddev
                ;
    uid 0;
};
```

- Para verificar o funcionamento, utilizamos o comando abaixo, que "sobe" o módulo do driver no gerenciador de dispositivos:

```
#minix-service up /service/hello
Hello, World!
RS: restarting /service/hello, restarts 0
```

- Para remover o driver do gerenciador de dispositivos, fazemos:

```
#minix-service down hello
```

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

(a) Existe alguma entrada no diretório /dev representando o dispositivo ?

(b) Módulos de drivers podem ser usados sem entrada no diretório /dev ?

DEVICE DRIVER /dev/hello

- Neste segundo exemplo, vamos associar uma entrada no diretório /dev (/dev/hello) para representar o dispositivo hello, que será do tipo caracter (c):

- O primeiro passo é criar uma entrada no diretório /dev com indicação de um major number (17) e um minor number (0). O major number é o número que identifica o driver. Como um mesmo driver pode gerenciar vários dispositivos, o minor number permite diferenciá-los.

```
# mknod /dev/hello c 17 0
```

- Nosso driver agora terá um arquivo .h (hello.h):

```
#ifndef __HELLO_H
#define __HELLO_H

/** The Hello, World! message. */
#define HELLO_MESSAGE "Hello, World!\n"

#endif /* __HELLO_H */
```

- A estrutura do módulo hello.c, agora, possui uma estrutura mais complexa:

```
#include <minix/drivers.h>
#include <minix/chardriver.h>
#include <stdio.h>
#include <stdlib.h>
#include <minix/ds.h>
#include "hello.h"

/*
 * Function prototypes for the hello driver.
 */
static int hello_open(devminor_t minor, int access, endpoint_t user_endpt);
static int hello_close(devminor_t minor);
static ssize_t hello_read(devminor_t minor, u64_t position, endpoint_t
endpt, cp_grant_id_t grant, size_t size, int flags, cdev_id_t id);

/* SEF functions and variables. */
static void sef_local_startup(void);
static int sef_cb_init(int type, sef_init_info_t *info);
static int sef_cb_lu_state_save(int, int);
static int lu_state_restore(void);
```

```c
/* Entry points to the hello driver. */
static struct chardriver hello_tab =
{
    .cdr_open   = hello_open,
    .cdr_close = hello_close,
    .cdr_read   = hello_read,
};

/** State variable to count the number of times the device has been opened.
 * Note that this is not the regular type of open counter: it never
decreases.
 */
static int open_counter;

static int hello_open(devminor_t UNUSED(minor), int UNUSED(access),
    endpoint_t UNUSED(user_endpt))
{
    printf("hello_open(). Called %d time(s).\n", ++open_counter);
    return OK;
}

static int hello_close(devminor_t UNUSED(minor))
{
    printf("hello_close()\n");
    return OK;
}

static ssize_t hello_read(devminor_t UNUSED(minor), u64_t position,
    endpoint_t endpt, cp_grant_id_t grant, size_t size, int UNUSED(flags),
    cdev_id_t UNUSED(id))
{
    u64_t dev_size;
    char *ptr;
    int ret;
    char *buf = HELLO_MESSAGE;
    printf("hello_read()\n");

    /* This is the total size of our device. */
    dev_size = (u64_t) strlen(buf);

    /* Check for EOF, and possibly limit the read size. */
    if (position >= dev_size) return 0;              /* EOF */
    if (position + size > dev_size)
        size = (size_t)(dev_size - position);/* limit size */

    /* Copy the requested part to the caller. */
    ptr = buf + (size_t)position;
    if ((ret = sys_safecopyto(endpt, grant, 0, (vir_bytes) ptr, size)) != OK)
        return ret;

    /* Return the number of bytes read. */
    return size;
}


static int sef_cb_lu_state_save(int UNUSED(state), int UNUSED(flags)) {
/* Save the state. */
```

```
    ds_publish_u32("open_counter", open_counter, DSF_OVERWRITE);

    return OK;
}

static int lu_state_restore() {
/* Restore the state. */
    u32_t value;

    ds_retrieve_u32("open_counter", &value);
    ds_delete_u32("open_counter");
    open_counter = (int) value;

    return OK;
}

static void sef_local_startup()
{
    /*
     * Register init callbacks. Use the same function for all event types
     */
    sef_setcb_init_fresh(sef_cb_init);
    sef_setcb_init_lu(sef_cb_init);
    sef_setcb_init_restart(sef_cb_init);

    /*
     * Register live update callbacks.
     */
    sef_setcb_lu_state_save(sef_cb_lu_state_save);

    /* Let SEF perform startup. */
    sef_startup();
}

static int sef_cb_init(int type, sef_init_info_t *UNUSED(info))
{
/* Initialize the hello driver. */
    int do_announce_driver = TRUE;

    open_counter = 0;
    switch(type) {
        case SEF_INIT_FRESH:
            printf("%s", HELLO_MESSAGE);
        break;

        case SEF_INIT_LU:
            /* Restore the state. */
            lu_state_restore();
            do_announce_driver = FALSE;

            printf("%sHey, I'm a new version!\n", HELLO_MESSAGE);
        break;

        case SEF_INIT_RESTART:
            printf("%sHey, I've just been restarted!\n", HELLO_MESSAGE);
        break;
    }
```

```
    /* Announce we are up when necessary. */
    if (do_announce_driver) {
        chardriver_announce();
    }
    /* Initialization completed successfully. */
    return OK;
}

int main(void)
{
    /* Perform initialization. */
    sef_local_startup();

    /* Run the main loop. */
    chardriver_task(&hello_tab);
    return OK;
}
```

- Compilamos e instalamos o novo device driver como fizemos anteriormente:

```
# make clean
# make
# make install
```

- Para subir o serviço do driver, fazemos como abaixo. Observe o uso do /dev/hello.

```
# service up /usr/sbin/hello -dev /dev/hello
Hello, World!
```

- Para testar o novo device, fazemos:

```
# cat /dev/hello
hello_open()
hello_read()
hello_read()
hello_close()
Hello, World!
#
```

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Por que vimos a mensagem Hello, World! Na saída acima ?

EXERCÍCIO EXTRA-CLASSE

Pesquise outro dispositivo para o sistema operacional MINIX 3.0 e descreva a produção de um device driver para ele, conforme mostrado no exemplo em classe (hello e /dev/hello).