PCS-2042 – Sistemas Operacionais

Projeto 3: Drivers

Mark Hodgkin
Pedro d'Aquino
Rafael da Silva
13/07/2008

CONTEÚDO

OBJETIVO

A experiência 8 do lab. de microprocessadores tem como objetivo apresentar o que é um driver em linux. Em linux, um driver é um conjunto de rotinas que implentam funções padronizadas como read e write. No caso do minix, um driver é um processo que responde a mensagens padronizadas. Crie um driver muito simples em minix que apenas imprime mensagens quando recebe mensagens padronizadas de outro processo. O driver pode ser associado a devteste. Declare devteste associando um major e minor number e o código do processo associado ao driver. Use service para rodar o driver. Gere um relatório sobre isso. As mensagens que o driver deve responder são as mesmas que o driver da impressora respondem, ou seja

```
   m_type      TTY_LINE   PROC_NR    COUNT    ADDRESS
-------------+---------+---------+---------+---------
 DEV_OPEN
-------------+---------+---------+---------+---------
 DEV_CLOSE             proc nr
-----------------------------------------------------
 HARD_INT
-------------+---------+---------+---------+---------
 SYS_EVENT
-------------+---------+---------+---------+---------
 DEV_WRITE   minor dev proc nr   count   buf ptr
-------------+---------+---------+---------+---------
 CANCEL      minor dev proc nr
-----------------------------------------------------
```

Envie mensagens a devteste como em `echo “lixo” > devteste` e veja a mensagem que é impressa pelo seu driver. Faça o driver imprimir a string que recebeu (lixo).

DRIVERS NO MINIX

Driver é um programa que realiza a comunicação com um determinado dispositivo, realizando operações de abrir (inicializar), ler, escrever e fechar.

Os Drivers no MINIX não tem privilégio de Kernel e não podem controlar seu dispositivo diretamente. A comunição com os dispositivos é feita, portanto, através de system calls.

CRIANDO UM DRIVER

Para carregar um driver na memória é necessário executá-lo no modo kernel, já que drivers não rodam no modo usuário. A system call responsável por isso é a *service up,* que pode ser usada da seguinte maneira:

```
service up <service> [-args args] [-dev special] [-period ticks]
```

Esse comando carrega o driver executável na memória e o linka com o arquivo special em /dev. O arquivo special pode ser aberto em modo usuário, assim tornando disponível para o usuário ler ou escrever no driver. A maneira de criar o arquivo special é usando o comando mknod, como no exemplo:

```
mknod /dev/audio c 13 0
```

onde sera criado um novo arquivo special áudio em /dev. O parâmetro c indica que o dispositivo não lidará com acesso randômico, enquanto 13 é o seu major number e 0 o seu minor number. Diversos arquivos relacionados ao mesmo driver podem ser criados com o mesmo major number, porém seus arquivos special's serão diferenciados pelo minor number.

O driver criado foi baseado no driver priter.c em (/usr/src/drivers/printer/), assim apresentando as mesmas mensagens de resposta que o driver da impressora. O Código feito em C segue abaixo:

```
#include "../drivers.h"
#include <stdio.h>

void do_open( message *m_ptr );
void do_hard_int( message *m_ptr );
void do_sys_event( message *m_ptr );
void do_write( message *m_ptr );
void do_cancel( message *m_ptr );
void do_close( message *m_ptr );
void reply(int code, int replyee, int process, int status);

static int jaImprimiu = 0;

PUBLIC void main(void)
{
    message pr_mess;

    while (TRUE) {
        receive(ANY, &pr_mess);
        switch(pr_mess.m_type) {
            case DEV_OPEN: do_open(&pr_mess); break;
            case HARD_INT: do_hard_int(&pr_mess); break;
            case DEV_WRITE:    do_write(&pr_mess);    break;
            case CANCEL: do_cancel(&pr_mess); break;
            case DEV_CLOSE: do_close(&pr_mess); break;
        }
      reply(TASK_REPLY, pr_mess.m_source, pr_mess.IO_ENDPT, OK);
    }
}

void do_open(register message *m_ptr) {
    printf("do_open\n");
}

void do_hard_int(register message *m_ptr) {
    printf("do_hard_int\n");
}

void do_sys_event(register message *m_ptr) {
    printf("do_sys_event\n");
}

void do_cancel(register message *m_ptr) {
```

```
    printf("do_cancel\n");
}

void do_close(register message *m_ptr) {
    printf("do_close\nDEV_WRITE enviado %d vezes\n",jaImprimiu);
    jaImprimiu = 0;
}


void do_write(register message *m_ptr){
    char* buf;
    if(jaImprimiu++) {
        return;
    }
    buf = (char*) malloc(m_ptr->COUNT + 1);
    buf[m_ptr->COUNT] = '\0';
    if(!buf) printf("malloc falhou\n");
    if (sys_datacopy(m_ptr->IO_ENDPT, (vir_bytes) m_ptr->ADDRESS, SELF, (vir_bytes) buf,
m_ptr->COUNT) != OK) {
          printf("sys_datacopy falhou\n");
    } else {
        printf("%s\n",buf);
    }
}


void reply(int code, int replyee, int process, int status)
{
    message pr_mess;
    pr_mess.m_type = code;
    pr_mess.REP_STATUS = status;
    pr_mess.REP_ENDPT = process;
    send(replyee, &pr_mess);
}
```

Para a implementação do código foi necessário usar a chamada de sistema sys_datacopy para copiar a string do espaço de endereçamento do processo remetente ao nosso espaço de endereçamento.

A função reply é necessária para nem o FS (que gerencia os dispositivos) nem o processo que nos solicitou fiquem bloqueados, uma vez que no MINIX, todo driver precisa responder às suas requisições. Nesse caso, é respondido OK para tudo que for recebido.

Para compilar e instalar, os seguintes passos foram realizados:

Foi criado a pasta /usr/src/drivers/driverteste que continha os arquivos driverteste.c além do Makefile e do .depend. Foi executado o comando mknod /dev/devteste c 30 0 para criar o devteste em /dev e em

seguida make install em /usr/src/drivers/driverteste. Por fim service up /usr/sbin/driverteste -dev /dev/devteste para iniciar o driver e associá-lo ao dispositivo devteste.

Para testar foi executado echo teste > /dev/devteste e obteve-se os seguintes resultados:

```
# echo "teste" > /dev/devteste
do_open
teste

do_close
DEV_WRITE enviado 11 vezes
# _
```

BIBLIOGRAFIA

- TANENBAUM, A.S. e WOODHULL, A. S.; Sistemas Operacionais: Projeto e Implementação – Terceira Edição.
- BRONWASSER, Laurens; Audio Driver in MINIX; http://www.cs.vu.nl/~lmbronwa/es1371/bachelorES1371.pdf, visitado em 13/07/08.