**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**
**FACULDADE DE COMPUTAÇÃO**
**Curso de Graduação em Ciência da Computação**

**Trabalho de conclusão de Disciplina**

Grupo 8

**Professor Rivalino Matias Júnior**

**Esdras de Lima Chaves –** 11511BCC015
**Tiago Pereira de Faria –** 11511BCC001
**Márcio Antônio de Freitas Júnior –** 11511BCC026

**Uberlândia, 10 de dezembro de 2016**

# Sumário

# 1 Introdução

Este relatório tem o objetivo de detalhar o trabalho realizado para a disciplina de Sistemas Operacionais (GBC045) do curso de Ciência da Computação da Universidade Federal de Uberlândia.

O trabalho se relaciona com alguns conceitos trabalhados na disciplina, principalmente com a parte de interrupções de hardware e como tratá-las, para isso, o objetivo do trabalho foi analisar, projetar e implementar um *device driver* para o kernel Linux, utilizando a plataforma x86, para proporcionar uma experiência para os alunos descobrirem como o Kernel de um sistema operacional moderno é organizado e como é realizado a programação no Kernel Space.

O relatório irá tratar sobre o que e como são tratadas interrupções de Hardware, mais especificamente as de teclado, o funcionamento de um Timer e o processo de implementação do Device Driver.

# 2 Problema Proposto

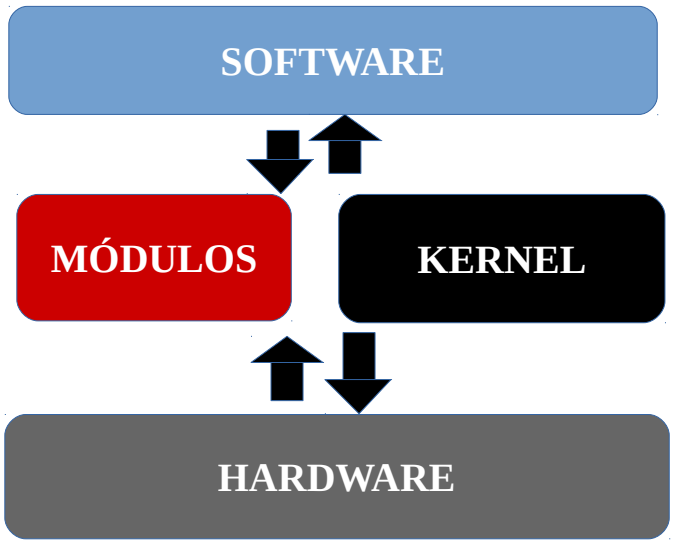
Projetar e Implementar um device driver para o kernel Linux, plataforma x86, o qual será executado como módulo. O device driver a ser implementado para monitorar o teclado e, ao detectar uma determinada combinação de teclas, deve desligar o computador (*shutdown*). Esta operação deve ocorrer de duas formas independentes: imediata (ao perceber a combinação de teclas definida) ou programada (tempo futuro). A definição dos parâmetros de entrada e valores de retorno fazem parte da etapa de projeto do *device driver*.

# 3 Fundamentos Teóricos

## 3.1 Módulos

Os módulos são pedaços de códigos que podem ser carregados para o *kernel* e descarregados do mesmo sob demanda. Isso faz deles códigos que estendem as funcionalidades do kernel, passando a atuar na memória principal somente quando os registramos. Assim, não é necessário que ele fique no *kernel* a todo momento, o que pode acarretar problemas de desempenho e tornaria necessário a reinicialização do sistema sempre que alguma alteração no mesmo fosse necessária.

Geralmente os módulos são criados para realizarem a comunicação entre os softwares e alguns dispositivos de hardware, como dispositivos de memória, vídeo, etc. Abaixo encontra-se um esquema do que foi falado:




A programação do módulo deve conter as seguintes definições de funções:

*int init_module (void)*

Retorna 0 para sinalizar sucesso, chamada quando o módulo é carregado.

*void cleanup_module (void)*

Chamado quando o módulo é descarregado.

MACROS:

```
#include <linux/module.h>
MODULE_LICENSE(license);                    // Licença do módulo
MODULE_AUTHOR(autor_str);                   // Nome e contato do autor
MODULE_DESCRIPTION(description_str);        // Descrição do módulo
```

Exemplo de função chamadas quando o módulo é registrado e removido:

```
static int __init modulo_init(void) {
        printk(KERN_INFO "Módulo registrado\n");
}
static void __exit modulo_exit(void) {
        printk(KERN_INFO "Módulo removido\n");
}
module_init(modulo_init);          /* Define as funções (callbacks) a serem chamadas */
module_exit(modulo_exit);          /* no init e no exit */
```

## 3.2 Interrupções de hardware (teclado)

A comunicação entre a CPU e o resto do Hardware pode ocorrer, em linhas gerais, de duas maneiras:

1- CPU manda o Hardware fazer alguma coisa;

2- Hardware precisa comunicar a CPU sobre algo;

Essa segunda abordagem é o que chamamos de interrupção.

O Linux trata as interrupções de Hardware como IRQ's (*Interrupt Request)*. Quando a CPU recebe essa interrupção, ela para o que está fazendo e chama um *interrupt handler* (responsável por tratar a interrupção). Para a implementar esse "tratamento" no Linux, precisamos chamar a função *request_id();* que será explicada mais adiante. Vamos seguir com a arquitetura do teclado.

Quando uma tecla é pressionadaum sinal é enviado da controladora de teclado e o IRQ 1 é recebido (IRQ da controladora das arquiteturas Intel). Quando a interrupção é recebida é possível ler o *scancode*, que é o valor retornado pelo teclado, através da inb(0x60) e o status, através de inb(0x64). Basicamente, o scancode é uma sequência de 1 a 3 bytes enviada pelo teclado onde o bit mais significativo mostre se a tecla foi pressionada ou solta, enquanto os bits restantes referenciam a tecla em si.

- *Int request_irq (unsigned int irq, irq_handler_t handler, unsigner long irqflags, const char* devname, void* dev_id);*
    - Essa função aloca uma linha de interrupção. Recebe como parâmetro a linha para alocar, a função a ser chamada caso o IRQ ocorra, tipo das flags, um nome ascii pro dispositivo e um cookie que passa pro handler;
- *void free_irq(unsigned int irq, void* dev_id);*
    - Tenta liberar a linha de interrupção. Recebe como parâmetro a linha e a identidade do dispositivo;

## 3.3 Timers

Muitas vezes é importante que haja um controle de tempo nos device drivers, tanto para executar um segmento de código em um momento apropriado ou até mesmo parar de executar algo, para isso são usado os *timers*. O Linux disponibiliza uma API para a construção e gerenciamento de *timers*, que serão usados nesse trabalho apenas para fazer uma "contagem regressiva", mas são importantes e precisam ser explicados.

Os *timers* no Linux são definidos por uma estrutura chamada timer_list onde se encontram os dados necessários para a implementação dos mesmos, sendo eles: O tempo de expiração, a função que irá executar quando chegar esse tempo (chamada callback). Existem várias funções que podem configurar esses timers, abaixo estão explicadas as que foram utilizadas no trabalho:

- *setup_timer (struct timer_list *timer, void (*function)(unsigned long), unsigned long data);*
  - Usada para configurar o timer, passa como parâmetro a estrutura timer_list criada, a função de callback e os parâmetro para a mesma.
- *int mod_timer( struct timer_list *timer, unsigned long expires );*
  - Usado para definir o tempo de expiração do timer, recebe a estrutura como parâmetro e o tempo desejado somado com *jiffies* para falar que o contador começa do instante atual.
- *try_to_del_timer_sync (struct timer_list *timer);*
  - Tenta deletar o timer criado;

# 4 Implementação do Device Driver

Segue abaixo a Implementação do device driver :

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/interrupt.h>
#include <asm/io.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/reboot.h>
#include <linux/timer.h>
#include <linux/workqueue.h>
#include <linux/sched.h>
```

```
MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("Esdras de Lima Chaves <esdraslchaves@gmail.com>");
MODULE_DESCRIPTION("Modulo que desliga o computador de maneira imediata ou programada a partir de teclas pressionadas");

// Definições de quais índices do vetor tecla[] representam quais teclas
#define T_shift 0
#define T_c 1
#define T_t 2

// Definição dos scancodes das teclas "chaves"
#define SHIFT 0x2A
#define C 0x2E
#define T 0x14

// Definição das váriaveis de "transformação" do scancode e status
#define scan_const 0x7F
#define stat_const 0x80


// Definição das teclas que representam segundos, minutos e horas
#define seg 0x1F
#define mnt 0x32
#define hor 0x23

// Definição das teclas numéricas
#define k0 0x0b
#define k1 0x02
#define k2 0x03
#define k3 0x04
#define k4 0x05
#define k5 0x06
#define k6 0x07
#define k7 0x08
#define k8 0x09
#define k9 0x0A

// Declaração da estrutura do timer a ser utilizado
static struct timer_list my_timer;
```

```
// Declaração e inicialização de variáveis de controle
int tecla[3] = {0, 0, 0};
int tempo = 0;
int tempoaux = 0;
int set_timer = 0;

// Procedimento que desliga o computador
void computer_shutdown(void) {
   orderly_poweroff(true);
}

/*
*  Procedimento que executa quando o timer é setado,
*  impede que o desligamento programado seja chamado novamente e
*  programa o computar para desligar
*/
void my_timer_callback(unsigned long data) {
   computer_shutdown();
}


// Handler de interrupção do teclado
irq_handler_t irq_handler(int irq, void *dev_id, struct pt_regs *regs)
{
   static unsigned char scancode;
   unsigned char status;

   status = inb(0x64);
   scancode = inb(0x60);

   // Guarda quais teclas estão sendo pressionadas
   if((scancode  & scan_const) == SHIFT  && (scancode & stat_const) == 0)
      tecla[T_shift] = 1;

   if((scancode  & scan_const) == SHIFT && (scancode & stat_const) != 0)
      tecla[T_shift] = 0;

   if((scancode  & scan_const) == C && (scancode & stat_const) == 0)
   tecla[T_c] = 1;
```

```
if((scancode  & scan_const) == C && (scancode & stat_const) != 0)
tecla[T_c] = 0;

if((scancode  & scan_const) == T && (scancode & stat_const) == 0)
tecla[T_t] = 1;

if((scancode  & scan_const) == T && (scancode & stat_const) != 0)
tecla[T_t] = 0;

// Lê o tempo desejado
if (set_timer == 1 && (scancode & stat_const) == 0) {

   switch(scancode & scan_const) {
      case k0:
        tempoaux *= 10;
      break;
          case k1:
        tempoaux *= 10;
        tempoaux += 1;
      break;
          case k2:
              tempoaux *= 10;
              tempoaux += 2;
      break;
          case k3:
              tempoaux *= 10;
              tempoaux += 3;
      break;
          case k4:
              tempoaux *= 10;
              tempoaux += 4;
      break;
          case k5:
        tempoaux *= 10;
        tempoaux += 5;
      break;
          case k6:
              tempoaux *= 10;
```

```
            tempoaux += 6;
        break;
            case k7:
                tempoaux *= 10;
                tempoaux += 7;
        break;
            case k8:
          tempoaux *= 10;
                tempoaux += 8;
        break;
            case k9:
                tempoaux *= 10;
                tempoaux += 9;
        break;
            case seg:
                    tempo += tempoaux * 1000;
                tempoaux = 0;
                    set_timer = -1;
          setup_timer(&my_timer, my_timer_callback, 0);
          mod_timer(&my_timer, jiffies + msecs_to_jiffies(tempo));
        break;
        case mnt:
                tempo += tempoaux * 60000;
          tempoaux = 0;
        break;
            case hor:
          tempo = tempoaux * 3600000;
          tempoaux = 0;
        break;
            default:
          set_timer = 0;
        break;
    }
}

// Verifica se é um desligamento imediato ou programado
if(tecla[T_shift] == 1 && tecla[T_c] == 1 && tecla[T_t] == 0)
        computer_shutdown();
else if(set_timer == 0 && tecla[T_shift] == 1 && tecla[T_c] == 0 && tecla[T_t] == 1){
```

```
            printk(KERN_INFO "Teste");
            set_timer = 1;
    }
    return (irq_handler_t)IRQ_HANDLED;
}
// Função chamada quando módulo é inserido no kernel (insmod)
static int __init keyshut_init(void) {
    printk(KERN_INFO "keyshut device has been registered\n");
    request_irq(1, (irq_handler_t) irq_handler, IRQF_SHARED, "keyboard_stats", (void *) (irq_handler));

    return 0;
}

// Procedimento chamado quando o módulo é removido do kernel (rmmod)
static void __exit keyshut_exit(void) {
    try_to_del_timer_sync (&my_timer);
    free_irq(1, (void *) (irq_handler));
    printk(KERN_INFO "keyshut device has been unregistered \n");
}

// Define as função (callbacks) a serem chamadas no init e no exit
module_init(keyshut_init);
module_exit(keyshut_exit);
```

# 4.1 Explicação do algoritmo usado

Nós escolhemos usar o comando shift + c para um reinicio imediato e shift + t para um reinicio com tempo marcado. Para o segundo caso, o usuário deve apertar e soltar shift + t e então digitar uma sequência que represente o tempo que deseja esperar antes do desligamento. Isso deve ser feito da seguinte forma: 2h20m0s. É importante ressaltar que o programa usa a letra s como um sinal de que o usuário acabou de digitar, então é sempre necessário apertar tal tecla

Sempre que uma IRQ é recebida, o programa checa o scancode e faz as seguintes ações dependendo do que for lido:

Se for uma tecla pertencente ao comando e a tecla tiver sido apertada, uma variável referente a tal tecla é marcada como true, se a tecla tiver sido solta, a variável será marcada como false. Se em algum momento as variáveis referentes a todas as teclas da sequência combinada estiverem marcadas como true, executamos o comando

Caso o comando for shift + c, a função que desliga o computador é chamada, caso contrário, o programa se prepara para receber o tempo

# 6 Bibliografia


- THE LINUX KERNEL API. Disponível em: <https://www.kernel.org/doc/htmldocs/kernel-api/index.html> Acesso em 05 de dezembro de 2016.
- SALZMAN, Peter; BURIAN, Michael; POMERANTZ, Ori. The Linux Kernel Module Programming Guide – ver 2.6.4 – 2007. Disponível em: <http://www.linuxtopia.org/online_books/linux_kernel/linux_kernel_module_programming_2.6/index.html> Acesso em 05 de dezembro de 2016.
- KERNER APIs, PART 3: TIMERS AND LISTS IN THE 2.6 KERNEL. Disponível em: <https://www.ibm.com/developerworks/library/l-timers-list/> Acesso em 05 de dezembro de 2016.
- KEYBOARD SCANCODE. Disponível em: <https://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html> Acesso em 05 de dezembro de 2016.
- LINUX CROSS REFERENCE. Disponível em: <http://lxr.free-electrons.com/source/kernel/> Acesso em 05 de dezembro de 2016.