- 52 - Den Kernel neu kompilieren

Den SuSE Kernel neu konfigurien und kompilieren

- Pakete `kernel-source` , `make` und `gcc` mit YaST installieren
- In `/usr/src/linux/Makefile` die Variable
  `HIGHMEMVERSION=-4GB` ändern nach
  `HIGHMEMVERSION=-4GBa` oder so was.
- Aktuelle Kernel konfig datei kopieren

```
cp /boot/vmlinuz.config /usr/src/linux/.config
```

- Kernel compilation

```
cd /usr/src/linux
make xconfig      (kernel konfigurien)

make dep
make clean
make bzImage
make modules
make modules_install
oder
make dep && make clean && make bzImage && make modules \
&& make modules_install
```

- Neue Kernel kopieren

```
ls -l arch/i386/boot/
cp -v arch/i386/boot/bzImage   /boot/vmlinuzN
```

- Erzeugung von neue `initrd` für neue Kernel

```
mk_initrd -k "vmlinuzN" -i "initrdN"
```

- Nur für LILO
  - Datei `/etc/lilo.conf` editieren:

```
        #Extra Eintrgäge
        label = new
        root  = /dev/hda6    (nur z.B. hda6)
        image = /boot/vmlinuzN
        initrd= /boot/initrdN
```

  - LILO installieren:

```
lilo  <Enter>
  Added linux *
  Added new
  Added win
```

- Nur für GRUB

```
title linuxNew
kernel (hd0,2)/boot/vmlinuzN apm=on acpi=off apic \
        root=/dev/hda3 vga=791 splash=silent showopts
initrd (hd0,2)/boot/initrdN
```

- Neu booten (`reboot` oder *<Str> <Alt> <Entf>*)
- Neu Kernel probieren(`linuxNew` wahlen)

- **Linux/UNIX Kernel:**

In UNIX or Linux Kernel version 1.x.x the kernel must be recompiled for new features or device drivers.
From Linux Kernel version 2.x.x external modules can be compiled separately from the kernel and dynamically loaded or unloaded. They are called Kernel Modules.

- **Kernel options at boot time**

The list of options supported by the current kernel can be found in :
`/usr/src/linux/Documentation/kernel-parameters.txt`.

- **Kernel Modules:**

The kernel modules are normally located in:
`/lib/modules/`*`kernel-version/`*`*` or
`/lib/modules/$(uname -r)/*`   `$(uname -r)` = kernel version

- Modules are files with the extention '`.o`' eg. `serial.o`
- Modules can depend on other modules to be loadable. The list of modules dependencies is located at: `/lib/modules/`*`kernel-version/`***`modules.dep`**
  This file is produced by running the command: `depmod`. `depmod` will also generate various map files in this directory, for use by the *hotplug* infrastructure.
- Modules can be loaded in 2 different ways:
  - Manualy. The commands `insmod` and `modprobe` are used:

`insmod` *`modulename`*	Loads the module without checking for dependencies.
`modprobe` *`modulename`*	Checks the module's dependencies. Loads all the dependencies if needed and then loads the module.

  - Automatically via:
    - The hotplug infrastructure (see LPI-101 Hardware section) (for filesystems etc.)
    - The `devfsd` daemon and an `alias` entry in `/etc/modules.conf`
      `devfsd` will load the module each time the device is accessed

```
syntax:    alias /dev/devicefile modulename
eg.        alias /dev/net/tun    tun
```

    - The kmod support in kernel (`CONFIG_KMOD`) and an `alias` entry in `/etc/modules.conf`. kmod uses `modprobe` to load modules.

```
Syntax:   alias DeviceInternalName  modulename
          alias block-major-NN[-nn] modulename
          alias char-major-NN-[nn]  modulename

eg.       alias eth0                8139too
          alias block-major-58      lvm-mod
          alias char-major-10-134   apm
          alias char-major-81       bttv
```

NN is The Device Major Number and the *nn* is the minor number.

eg. `ls -l /dev/apm_bios`

```
crw-rw----  1 root root 10, 134 Jan 18 11:26 apm_bios
```

Entry in `modules.conf`: `alias char-major-10-134 apm`

To create new devices in `/dev` directory use the following format:

```
    mknod -m modes /dev/newdev {c|b} majorNr. minorNr.
eg. mknod -m 644   /dev/ttyS4   c      4        67
```

or use the script `MAKEDEV:`

```
eg.   cd /dev ; ./MAKEDEV ttyS
```

- A runlevel script. The script can issue `modprobe` commands when the system boots-up to load modules ready to use.

- Note: the file `/etc/modules.conf` and `/etc/conf.modules` are the same. Which filename is used varies between distributions but `modules.conf` is newer.

- For a module to dynamically link to the kernel, a kernel symbol table with memory pointers is used. Such table can be seen at `/proc/ksyms`.

- **Programs used to control modules:**

Note: The *modulename* never contains the '`.o`' extention of its filename.

`lsmod` Lists the loaded modules. Same result as `cat /proc/modules`
Syntax: `lsmod`

`insmod` Loads a module (no dependency check)
Syntax: `insmod` *modulename* [*module_parameters*]
eg. `insmod ne io=0x300 irq=5`

`modprobe` Loads/Removes a module(with dependency check)
`modprobe` expects an up-to-date `modules.dep` file,
as generated by `depmod`.
Syntax: `modprobe [-vcniqo]` *module* [*module_params*]
eg. `modprobe [-l] [-t dir.] [-a] [`*wildcard*`]`
`modprobe -r` *module1* [*module2*] `...` (`-r` = remove)
Automatic try of all network card modules until success:
`modprobe -t net \*`

`rmmmod` Removes a module.
Syntax: `rmmod [-r]` *module1* [*module2*] `....`
`-r` = Removes recursively through dependencies

`depmod` Determines module dependencies and writes `modules.dep` file.
Syntax: `depmod [-abeFAn]` (`-n`=Writes only to screen).
eg. `depmod -av` Checks all and writes `modules.dep`
Note: Run `depmod -a` after changes in `/etc/modules.conf`

`modinfo` Prints information about a module.
Syntax: `modinfo [-adlpn] [-F` *field*`]` *modulename*
`-n` = */path/filename* of module's file
`-F --field` Only print this field value, one per line.
Field names:
a`uthor(-a)`, **d**`escription(-d)`, **l**`icense(-l)`,
`depends`,and `alias`.
p`aram(-p) :` Shows which parameters are supported.
Output format of `-p`:
option type (*valid-values*) *description*
Options `[-adlp]` are shortcuts for these above fields.

The file `/etc/modules.conf` (or `/etc/conf.modules`):

This file is used by `kmod` to load the right modules automatically when certain devices are accessed or by `modprobe` to add needed options to modules and possibly run certain commands before and/or after loading and/or unloading modules. It can contain the following information:

- Module Parameters(options)

```
Syntax:    options modulename  options
eg.        options ne          io=0x300 irq=5
```

- Alias names for modules: Modules is then having 2 names.

```
Syntax:    alias aliasname modulename
eg.        alias eth0      3c509
           Makes it possible to do a -------> modprobe eth0
           which has the same result as --> modprobe 3c509
```

- Commands that should be run before and/or after a module is loaded

```
Syntax:    pre-install  modulename  command
           post-install modulename  command
eg.        post-install bttv        insmod tuner
```

- Commands that should be run before and/or after a module is un-loaded

```
Syntax:    pre-remove  modulename   command
           post-remove modulename   command
eg.        post-remove bttv         rmmod tuner
```

The command 'uname'

Syntax: `uname options`

This command is used to display information about the current system.

`uname -a`	Shows all information *in the following order*:
`-s, --kernel-name`	Print the kernel name
`-n, --nodename`	Print the network node hostname
`-r`	Print the current kernel release. eg. `/lib/modules/$(uname -r)/......` or ``/lib/modules/`uname -r`/.......``
`-v, --kernel-version`	Print the kernel version (Build date)
`-m, --machine`	Print the hardware machine name
`-p, --processor`	Print the processor type
`-i, --hardware-platform`	Print the hardware platform
`-o, --operating-system`	Print the operating system

Configuring and compiling the KERNEL

- **Pre-Requirements**

  - Source code installed from CD

  - Kernel versions that has the <u>second digit</u> with an even number are stable and the odd numbers are for the development versions (unstable)

- **Kernel Source code**

  The kernel source code is normally located in `/usr/src/linux/*`.
  Normally this directory is a symbolic link to `/usr/src/`*`kernelname/`* directory.
  It contains also all the configuration files necessary to compile the kernel.

- **Configuring the kernel:**

  - <u>Getting the source code and the current kernel configuration file.</u>
    The source code is normally available from the current distribution disks or from the internet(`www.kernel.org`)

  - <u>Using the present kernel configuration file as a start template.</u>
    On SuSE the current kernel configuration file is located in:

    ```
    /proc/config.gz
    ```

    To use this file as a start template before making changes do the following:d /

    ```
    usr/src/linux
    zcat /proc/config.gz > .config
    make oldconfig
    ```

    After getting the source code installed in the system, the kernel needs to be configured before compiling it. This configuration process wil create the configuration file : `/usr/src/linux/.config`
    We have the choice of using an older configuration file as a template or create a totally new one from scratch.(not recommended)

    Before issuing any commands we need to change to the source code directory:

    ```
    cd /usr/src/linux
    ```

  - **Preparing the an old `.config` for a new kernel source.**
    Copy the old .config to `/usr/src/linux/` directory and run the command:
    `make oldconfig`
    This will scan the file and add the new items that were not existing in the old kernel but present in the new kernel.

  - **Configuration programs**
    The following 3 commands start programs that read the `.config` file, allow for changing the configuration and when finished, saves the new configuration in the same `.config` file, replacing the original.

    ```
    make config          Older questions oriented.
    make menuconfig      Text/Menu oriented.
    make xconfig         Menu/Buttons Graphic Program
    ```

    <u>Note:</u>Because the configuration with `make xconfig` is not as well maintained as the other configuration possibilities, run the command `make oldconfig` after using this configuration method.

The main work of kernel configuration is to decide for:
- Which features are supported in the kernel.
- Which modules will be either:
    - Integreated in the kernel or
    - Compiled as separate loadable modules
    - Not compiled seperately and not integrated in the kernel.

- **Preparing the compilation**
  Since 'make' doesn't compile already compiled parts of the kernel, in order to create completely new ones, some already compiled need to be deleted by issuing the command : `make clean`

  Before compiling the kernel, the dependencies file need to be created.
  This file is named: `/usr/src/linux/.depend`
  The command: `make dep`

- **Compiling the kernel**
  The long and complex compiling process can now start by issuing one of the following commands:
  make zimage Old command to create a small jernel which will be saved as: `/usr/src/linux/arch/i386/boot/zImage`
  `make zdisk` Old command that once compiled the kernel will be saved in a floppy as a boot floppy.
  `make bzImage` New command to create a big kernel which will be saved as: `/usr/src/linux/arch/i386/boot/bzimage`
  `make bzdisk` New command that once compiled the kernel will be saved in a floppy as a boot floppy.

- **Compiling the modules**
  The compiling of the modules is made by issuing the command:(also long)
  ```
  make modules
  ```

- **Installling the modules**
  Once compiled the modules need to be installed in the directory `/lib/modules/`*`kernelversion/`* by issuing the command:
  ```
  make modules_install
  ```
  The command `depmod -a` will be automatically run by this above command.
  To produce a System Map file:
  ```
  /sbin/depmod -ae -F System.map kernelversion
  ```

- **Installing the new kernel**
  Once compiled the kernel and the system map file need to be copied to `/boot` directory and the boot manager config. file modified to reflect the changes.
  This can be achieved by issuing the following commands:
  ```
  cp /usr/src/linux/arch/i386/boot/bzimage /boot/vmlinuz
  cp /usr/src/linux/System.map /boot/System.map.$(uname -r)
  ```
  The file `/boot/System.map` contains kernel symbols required by the modules to ensure successful launching of kernel functions. This file depends on the current kernel.

- **The boot file `initrd`** (Init RAM DISK)
  This file is normally loaded by the boot manager and is used by the kernel to load

modules contained in it that are needed during boot time. This is the alternative to compiling these needed modules inside the kernel.
If an `initrd` is needed then issuing the following command will create it:
In SuSE the file /etc/sysconfig/kernel contains the directives that are taken for account when running this command. This is here we can enter the list of mokdules that should be integrated in the `initrd` file.
`mkinitrd` *`Options-needed`*

If using LILO	If using GRUB
`vi /etc/lilo.conf`	`vi /boot/grub/menu.lst`
`lilo`	

- **All kernel compiling commands in short**:
  - Install the kernel source in `/usr/src/linux/` directory
  - Copy the `.config` file from the current kernel in `/usr/src/linux/` directory
  - `make clean`   Deletes all already compiled modules from source tree
  - `make oldconfig`   Uses the current `.config` and creates a new one
  - `make xconfig` or `make menuconfig` or `make config`
    To configure the kernel options before compiling
  - `make dep`   Creates the dependencies file `.depend`
  - `make bzImage`   Compile the kernel
  - `make modules`   Compile the modules
  - `make modules_install`
    Install modules in `/lib/modules/kernelversion/`
  - `cp /usr/src/linux/arch/i386/boot/bzimage /boot/vmlinuz`
    Copies the kernel in the `/boot` directory
  - `cp /usr/src/linux/System.map /boot/System.map.$(uname -r)`
    Copies the `.map` file in the `/boot` directory
  - If using an `initrd` file when booting: `mkinitrd` *`Options`*
  - If using LILO: `vi /etc/lilo.conf` (Edit `lilo.conf`) then `lilo`
  - if using GRUB: `vi /boot/grub/menu.lst` or
    `vi /boot/grub/grub.conf`

- **Safeguard against a non-working new kernel**:
  To make sure that the the old kernel is saved as an alternative to boot, in the case of the new kernel not working, it is advisable to change the name of the older kernel, its `initrd`, and *`System.map.`*`$(uname -r)`, and its `/lib/modules/`*`kernelversion/`* directory before copying the kernel or issuing the comand `make modules_install`.
  An alternative menu item in the boot manager config file for being able to boot the older kernel is also advisable.

- **Modifying the Operating Voltages of a Pentium M (730) for SuSE 9.3**

  - **Note:** The following information was taken from the web site:

  - Install the kernel source
  - Edit the file:
    `/usr/src/linux/arch/i386/kernel/cpu/cpufreq/speedstep-centrino.c`

  - Then locate this part of the code:

Code

```
static int centrino_cpu_init(struct cpufreq_policy *policy)
{
        struct cpuinfo_x86 *cpu = &cpu_data[policy->cpu];
        unsigned freq;
        unsigned l, h;
        int ret;
```

And right before it add the following lines: (on next page)
Just cut and paste the code.

Code

```
static int centrino_target (struct cpufreq_policy *policy,
                            unsigned int target_freq,
                            unsigned int relation);

/************************* sysfs interface for user defined voltage table ***********************/
static ssize_t show_user_voltage (struct cpufreq_policy *policy, char *buf)
{
        ssize_t       bytes_written = 0;
        unsigned int  cpu          = policy->cpu;
        unsigned int  op_index     = 0;
        unsigned int  voltage      = 0;

        //dprintk("showing user voltage table in sysfs\n");

        while (centrino_model[cpu]->op_points[op_index].frequency != CPUFREQ_TABLE_END)
        {
                //dprintk("getting state %i \n", i);
                voltage = centrino_model[cpu]->op_points[op_index].index;
                voltage = 700 + ((voltage & 0xFF) << 4);
                //dprintk("writing voltage %i: %u mV \n", i, voltage);
                bytes_written += snprintf (&buf[bytes_written],PAGE_SIZE, "%u",voltage);
                op_index++;
                if (centrino_model[cpu]->op_points[op_index].frequency != CPUFREQ_TABLE_END)
                        bytes_written += snprintf (&buf[bytes_written],PAGE_SIZE, ",");
                else
                        bytes_written += snprintf (&buf[bytes_written],PAGE_SIZE, "\n");
        }
        buf[PAGE_SIZE-1] = 0;
        return bytes_written;
}

static ssize_t
store_user_voltage (struct cpufreq_policy *policy, const char *buf, size_t count)
{
        unsigned int  cpu;
        const char   *curr_buf;
        unsigned int  curr_freq;
        unsigned int  op_index;
        int           i;
        int           isok;
        char         *next_buf;
        unsigned int  op_point;
        ssize_t       retval;
        unsigned int  voltage;

        static struct cpufreq_frequency_table **original_table = NULL;

        if (!policy)
            return -ENODEV;
        cpu = policy->cpu;
        if (!centrino_model[cpu] || !centrino_model[cpu]->op_points)
            return -ENODEV;

        if (!original_table)
        {
                original_table = kmalloc(sizeof(struct cpufreq_frequency_table *)*NR_CPUS, GFP_KERNEL);
                for (i=0; i < NR_CPUS; i++)
                {
                        original_table[i] = NULL;
                }
        }

        if (!original_table[cpu])
        {
                /* Count number of frequencies and allocate memory for a copy */
                for (i=0; centrino_model[cpu]->op_points[i].frequency != CPUFREQ_TABLE_END; i++);
                /* Allocate memory to store the copy */
                original_table[cpu] = (struct cpufreq_frequency_table*) kmalloc(sizeof(struct cpufreq_frequency_table)*(i+1), GFP_KERNEL);
                /* Make copy of frequency/voltage pairs */
                for (i=0; centrino_model[cpu]->op_points[i].frequency != CPUFREQ_TABLE_END; i++)
                {
                        original_table[cpu][i].frequency = centrino_model[cpu]->op_points[i].frequency;
                        original_table[cpu][i].index = centrino_model[cpu]->op_points[i].index;
                }
                original_table[cpu][i].frequency = CPUFREQ_TABLE_END;
        }

        op_index = 0;
        curr_buf = buf;
        next_buf = NULL;
        isok     = 1;

        while ((centrino_model[cpu]->op_points[op_index].frequency != CPUFREQ_TABLE_END)
                && (isok))
        {
                voltage = simple_strtoul(curr_buf, &next_buf, 10);
                if ((next_buf != curr_buf) && (next_buf != NULL))
                {
                        if ((voltage >= 700) && (voltage<=1600))
                        {
                                voltage = ((voltage - 700) >> 4) & 0xFF;
                                op_point = (original_table[cpu])[op_index].index;
                                if (voltage <= (op_point & 0xFF))
                                {
                                        //dprintk("setting control value %i to %04x\n", op_index, op_point);
                                        op_point = (op_point & 0xFFFFFF00) | voltage;
                                        centrino_model[cpu]->op_points[op_index].index = op_point;
                                }
                                else
                                {
                                        op_point = (op_point & 0xFFFFFF00) | voltage;
                                        dprintk("not setting control value %i to %04x because requested voltage is not lower than the default value\n", op_index, op_point);
                                        //isok = 0;
                                }
                        }
                        else
                        {
                                dprintk("voltage value %i is out of bounds: %u mV\n", op_index, voltage);
                                isok = 0;
                        }
                        curr_buf = next_buf;
                        if (*curr_buf==',')
                                curr_buf++;
                        next_buf = NULL;
                }
                else
                {
                        dprintk("failed to parse voltage value %i\n", op_index);
                        isok = 0;
                }
                op_index++;
        }

        if (isok)
        {
                retval = count;
                curr_freq = cpufreq_get(policy->cpu);
                centrino_target(policy, curr_freq, CPUFREQ_RELATION_L);
        }
        else
        {
                retval = -EINVAL;
        }

        return retval;
}

static struct freq_attr centrino_freq_attr_voltage_table =
{
        .attr = { .name = "voltage_table", .mode = 0644, .owner = THIS_MODULE },
        .show = show_user_voltage,
        .store = store_user_voltage,
};
```

Then locate these lines:

Code

```
static struct freq_attr* centrino_attr[] = {
        &cpufreq_freq_attr_scaling_available_freqs,
        NULL,
};
```

And replace them by these ones:

Code

```
static struct freq_attr* centrino_attr[] = {
        &cpufreq_freq_attr_scaling_available_freqs,
        &centrino_freq_attr_voltage_table,
        NULL,
};
```

That's it. Rebuild your kernel, install it and reboot.

Getting the current voltage settings

To get the current voltage read the content of the file voltage_table in /sys/devices/system/cpu/cpu0/cpufreq. The content of this file is the list of all the voltages currently stored in the CPU frequency table, in mV. There is one value for each entry in the frequency table of the cpu.

Sample reading of current settings

```
quasar b12 # cat /
sys/devices/system/cpu/cpu0/cpufreq/voltage_table
1356,1356,1356,1356,1356,1356,1356,1244,1116,988
```

The sample above is on a MSI S260 laptop with a 1.6 GHz Pentium M 730. The first 7 duplicated value are for 1.6 GHz. The next values are for 1.3 GHz, 1 GHz and 800 MHz. Depending on your CPU and laptop model your mileage may vary.

Changing the voltage setings

To change the current voltage settings write the content of the file voltage_table in `/sys/devices/system/cpu/cpu0/cpufreq`. You need to use the same format as what you get when reading the file (same number of values separated by "`,`" on one single line")

Be carefull when doing this. There are some protections in the code to block you from setting voltage values that are greater than the default settings. But if you set values that are too low your CPU will freeze and you will have to reboot your computer.

Sample modification of voltages

```
echo "1084,1084,1084,1084,1084,1084,1084,988,908,860" >
                       /sys/devices/system/cpu/cpu0/cpufreq/voltage_table
```

The sample above is on a MSI S260 laptop with a 1.6 GHz Pentium M 730. *The first 7 duplicated value are for 1.6 GHz. The next values are for 1.3 GHz, 1 GHz and 800 MHz. Depending on your CPU and laptop model your mileage may vary.*

After changing the values you can read the file again to check that all your settings have been taken into account. If you don't get the same thing as what your have writen in the file have a look at dmesg. If you have enabled cpufreq debug messages in your kernel you will see errors like the following that will give you a hint on what you did wrong:

Sample error codes

```
speedstep-centrino: voltage value 0 is out of bounds: 1724 mV
speedstep-centrino: not setting control value 1 to 0c36 because requested voltage is not lower than the default value
speedstep-centrino: failed to parse voltage value 2
```

It may also be that the voltages have been rounded to a 16 mV multiple.

Finding the best voltage settings

Smallest voltages before the CPU freeze

If you are not affraid to crash your system a few times you can quickly find the lowest voltages that your CPU can acheive using the following procedure:

1. Set your CPU to the first frequency you want to test using the userspace governor
2. Run the voltage-ramp-down script (the script is at the end of this page)
3. Wait until your system freeze
4. Write down the voltage value of the current frequency that was displayed just before the last "OK"
5. Reboot
6. If you want to crash your system again set your CPU frequency to the next frequency and go back to step 2

If you are lucky your CPU will be able to run at ist lowest frequency using 700 mV.

It is strongly recommended to perform this procedure in console mode with the minimum sofware running (shut down all the services you can).

Also it not recommended to do this if you do not have a journalized file system. And even with that there are still chances that your file system gets corrupted if you have application writing to the disk

Note that the voltages you will find with this method **are not safe**. You CPU is most likely to make calculation errors with these settings or even to freeze after a few minutes. But it will quickly give you a good starting point to find the best settings (see next chapter)

The table below shows the voltage values that have been found by some users of this method:

Who	CPU	600 MHz	800 MHz	1 GHz	1.2 GHz	1.33 GHz	1.4 GHz	1.6 GHz	1.8 GHz	2 GHz	2.2 GHz
Michel	**P-M 730**	**-**	**700 mV**	**748 mV**	**-**	**828 mV**	**-**	**908 mV**	**-**	**-**	**-**
pumpkin0	P-M 725	-	716 mV	798 mV	860 mV	-	956 mV	1036 mV	-	-	-
You?	P-M ?	? mV	? mV	? mV	? mV	? mV	? mV	? mV	? mV	? mV	? mV

ToDo: Add samples of min voltages of other people with same and different Pentium M model

Safe voltages

ToDo: add procedure to find safe voltages settings using a tool like mprime

The table below shows the voltage values that are considered safe by some people that have undervolted their CPU:

Who	CPU	600 MHz	800 MHz	1 GHz	1.1 GHz	1.2 GHz	1.33 GHz	1.4 GHz	1.6 GHz	1.8 GHz	2 GHz
rschwarze	P-M 710	700 mV	700 mV	780 mV	812 mV	860 mV	-	940 mV	-	-	-
bdz	**P-M 730**	**-**	**700 mV**	**764 mV**	**-**	**-**	**860 mV**	**-**	**956 mV**	-	-
dgaffuri	P-M 750	700 mV	700 mV	764 mV	-	844 mV	-	924 mV	1004 mV	1084 mV	1196 mV
pumpkin0	P-M 725	700 mV	748 mV	828 mV	-	892 mV	-	988 mV	1100 mV	-	-
You?	P-M ?	? mV	? mV	? mV	? mV	? mV	? mV	? mV	? mV	? mV	? mV

ToDo: Add samples safe voltages of other people with same and different Pentium M model

Some handy scripts

As lowering the voltage is likely to crash the system it is safer to perform the tests in console mode to minimize the risks of file system corruption. (Using a journalized file system is also a good idea)

To perform tests in console mode you may want to use some scripts to help you change the voltages and monitor the CPU temperature and the current drawned from the battery. This chater contains some very basic sample scripts you may want to use

Finding the lowest voltages before the CPU freeze

This script automatically decreases the voltages settings by 16 mV every 2 seconds until the CPU freeze or the minimum value of 700 mV is reached.

Use with caution. It will crash your system!

voltage-ramp-down.sh

```
#!/bin/bash

# Min voltage value
# Voltages will not be set below than this value
Vmin=700

# Delay before assuming hat the new voltage is ok and trying the new one (in seconds)
Period=2

while [ 0 ]; do
  # Get current votages
  CurVoltages=$(cat /sys/devices/system/cpu/cpu0/cpufreq/voltage_table \
                | cut -d"," --output-delimiter=" " --fields=1-)
  NewVoltages=""

  # Compute new voltages as current - 16 mV
  for V in $CurVoltages; do
    V=$(($V - 16))
    # Ensure that voltage is not below min value
    if [ $V -lt $Vmin ]
    then
      V=$Vmin
    fi
    NewVoltages="$NewVoltages,$V"
  done

  # Display current settings from the sysfs file
  echo " "
  echo "Current settings:   "$(cat /sys/devices/system/cpu/cpu0/cpufreq/voltage_table)

  # Display the new settings that we are going to write to the sysfs file
  echo "Requested settings: $NewVoltages"

  # Force the kernel to write its buffers to the hard disk
  # to reduce the risks of file system corruption in case of CPU freeze
  sync

  # Apply new settings
  echo "$NewVoltages" > /sys/devices/system/cpu/cpu0/cpufreq/voltage_table

  # wait some time to see if the CPU freezes
  sleep $Period

  echo OK
done
```

Script for Displaying the CPU Frequency, Throttle, Temperature and Voltage every second.

```
#!/bin/bash

# Name:     cpustat
# Purpose:  Displays every second the CPU Speed, CPU Throttle State, CPU
#           temperature, CPU Voltage
# Syntax:   cpustat
# Note:     It only runs with the patched Kernel Module speedstep-centrino
#------------------------------------------------------------------------

watch -tn1 'Freq=$(cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq);
echo "Frequency:   $Freq";echo "Throttling:  $(grep \\*T[0-7]:  /
proc/acpi/processor/CPU0/throttling|cut -d: -f2|tr -d  " ")"; echo "Temperature:
$(acpi -t -B | cut -d" " -f9| cut -d. -f1) °C" ; Table=$(cat /
sys/devices/system/cpu/cpu0/cpufreq/voltage_table);echo -n "Voltage:     ";V1=$
(echo $Table | cut -d, -f4);V2=$(echo $Table | cut -d, -f3);V3=$(echo $Table |
cut -d, -f2);V4=$(echo $Table | cut -d, -f1);case $Freq in 800000) echo $V1
mv; ;; 1067000) echo $V2 mv; ;; 1333000) echo $V3 mv; ;; 1600000) echo $V4 mv; ;;
esac'
```