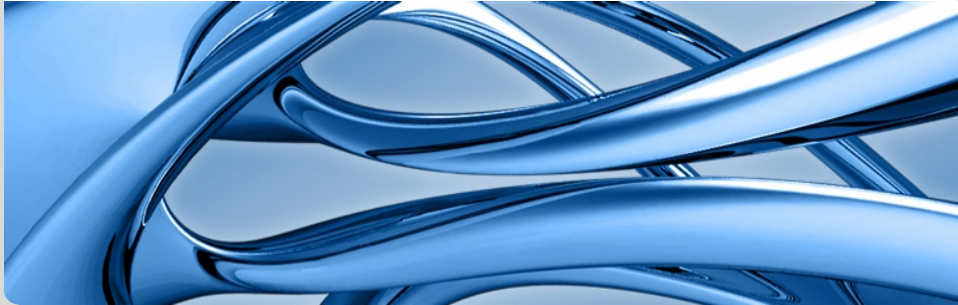


Power Management Praktikum

Einführung

Thorsten Gröninger | 2. Dezember 2014

OPERATING SYSTEMS GROUP DEPARTMENT OF COMPUTER SCIENCE



- Praktische Anwendung des Stoffes der Vorlesung
- Umsetzung eines Power-Management-Konzepts in Linux
- Erfahrung mit Systemprogrammierung sammeln

- Einteilung in Gruppen zu je 2–3 Personen
- Bearbeitung eines Themas bis zum Ende des Semesters
- Präsentation am Ende der Vorlesungszeit
- Praktikumstermine siehe Kalender in Ilias

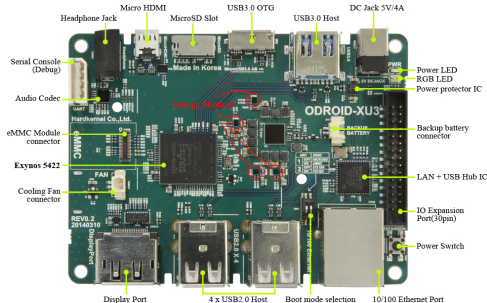
- Vorführen der Implementierung
 - Keine 100 % perfekte Implementierung erwartet
 - Aber Bemühung um die Lösung des Problems muss erkennbar sein
- Abgabe des Quellcodes (Änderungen am Linuxquellcode als Patch)
- Kurzes Vorstellen der Quellcodes am Ende des Semesters
- Präsentation des Designs, erste Ergebnisse, Messungen, ...

- Programmierung im Poolraum 149
 - Geöffnet Mo–Fr, ca. 9–19h
- „Offizielle“ Praktikumstermine
 - Fragen, Hilfestellungen, etc.
 - Genaue Termine im Netz
- Vorstellung der Ergebnisse
 - 2. Februar 2015
 - 3. Februar 2015

- Disk Power Management 2x PC (Intel Atom N270 mit HDD und SSD)
 - Energiegewahres Dateisystem
- CPU Power Management 3x PC (i7-2600K Sandy Bridge)
 - Energieabschätzung und Drosselung
- Energiemessung 3x Ordoid XU3 (Samsung Exynos 5422)
 - Analyse der Leistungsmerkmale des Big/Little Octacores
 - Drosselung des Gesamtsystems

■ Energiemessung und Management

- Energieverbrauch der unterschiedlichen Kerne ermitteln
- Prozessor drosseln, wenn Energiekontingent erschöpft
- ...



- Ereigniszähler des Prozessors nutzen
- Energie berechnen / auslesen (RAPL)
- Energieprofile von Tasks erstellen / vergleichen



Ereignis	Gewicht [nJ]
INST_RETIRED	-0,224
CPU_CLK_UNHALTED	4,546
DSB2MITE_SWITCHES	-5,972
DSB_FILL:ALL_CANCEL	64,240
ILD_STALL:IQ_FULL	-1,425
L2_RQSTS:PF_HIT	-22,837
LD_BLOCKS:ALL_BLOCK	5,504
LD_BLOCKS:DATA_UNKNOWN	-6,281
UOPS_DISPATCHED:STALL_CYCLES	-2,508
BR_INST_RETIRED:FAR_BRANCH	-18031,295

- Energiekontingent für jeden Prozess
 - Prozessor drosseln, wenn Energiekontingent erschöpft
 - Rechenintensive Prozesse drosseln, interaktive nicht benachteiligen
 - ...

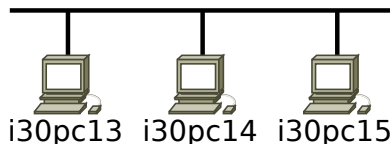


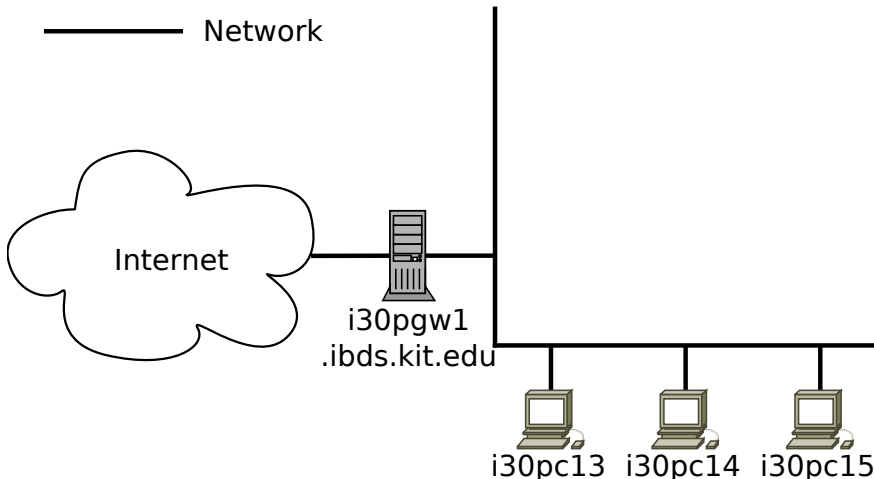
- Stromsparendes Dateisystem mit zwei Speichermedien
 - HDD (Hintergrundspeicher)
 - Solid State Disk (SSD) (energiesparender Cache)

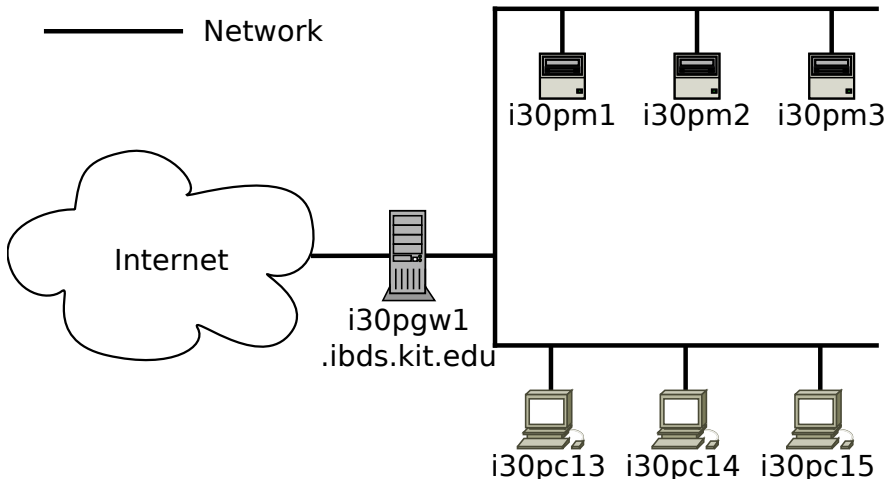


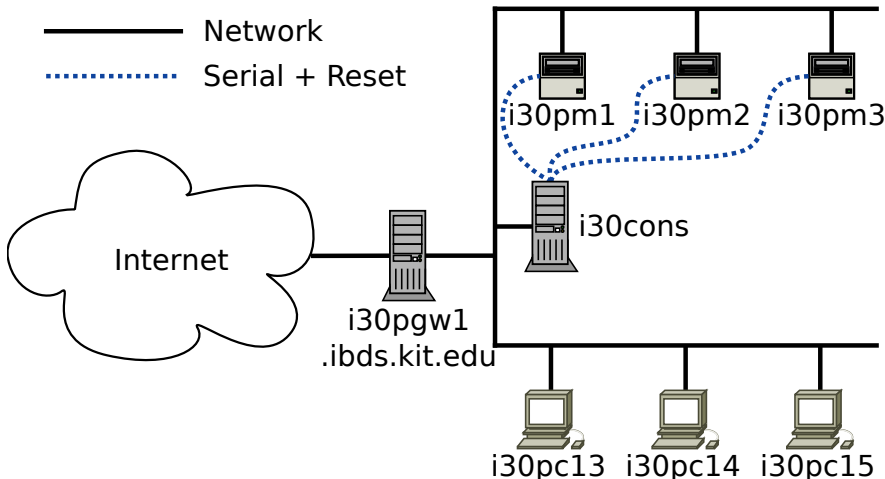
- Entwicklungsrechner im Poolraum
 - Editieren von Quellcode
 - übersetzen (bei ATOM Rechnern)
 - Internetrecherche
- Zielsystem läuft auf Testrechner im Rechnerraum
 - pm2, ... , pm10
 - Ausgabe über serielle Konsole (pm2-10)
 - Ausgabe über SSH (i30pm2-10)
- Konsolenrechner
 - Fernsteuerung der Testrechner über Minicom
 - Remote-Reset
- Messsystem für Leistungsmessungen HDD (i30pm1)

— Network









- Konsolenrechner: `i30cons`
- Zugang zum Praktikumssystem von außerhalb über `i30pgw1.itec.kit.edu`
- Zugriff jeweils über `ssh` und Gruppenaccount

- pm2 bis pm10
- zugeordnet zum jeweiligen Thema
- Gruppenaccounts
 - Login: pm<nr>
 - Passwort: <wird jeder Gruppe zugeteilt>
 - bitte ändern mit passwd

- Testrechner: i30pm<nr>
- Zugang
 - Login: pm<nr>
 - Passwort: pm<nr>
 - ➔ unter Umständen ändern passwd

- Per ssh auf i30cons einloggen
- minicom <Rechnername> aufrufen
 - Beispiel: `minicom pm2`
- Einloggen auf serieller Konsole oder per ssh
 - Neustart durch `sudo shutdown -r now` oder `sudo reboot`
- Aus minicom jederzeit möglich: Reset mit Ctrl+a r
- Konsolenausgabe erscheint (Kernmeldungen etc.)
- Wurde der Testrechner erfolgreich gebootet, einloggen ebenfalls über ssh
 - Beispiel: `ssh pm2@i30pm2`
- Live-Demo nächstes Mal

- Kernquellen unter `/home/power/pub` oder auf `kernel.org`
- Entpacken z.B. in `~/src` mit `tar xjvf <Dateiname>`
- oder `git clone <url>`
- Konfigurieren mit `make menuconfig`
 - Standardkonfiguration kann verwendet werden
 - oder entsprechende config aus `/home/power/pub/` verwenden
- übersetzen mit `make`
- Abweichung je nach System (z.B. Odroid XU3) möglich

- Kompilierter Kern liegt im Verzeichnis `arch/x86/boot/bzImage`
- Kern wird mit grub übers Netzwerk geladen (bei ATOM-Rechern)
- Ablegen/verlinken in `~/boot`
- Rechner neustarten

- cscope
 - Code Browser
- ctags
 - Erzeugt tags zur Navigation in Editoren (z.B. vi)
- `http://lxr.free-electrons.com/`
- `http://lxr.linux.no/` (falls erreichbar)
 - Linux-Kerne über HTML verlinkt

```

Linux/kernel/sched.c - Firefox
File Edit View Go Bookmarks Tools Help
http://lxr.linux.no/source/kernel/sched.c#L2662 Go
2658
2659 /*
2660  * schedule() is the main scheduler function.
2661  */
2662 asmlinkage void __sched schedule(void)
2663 {
2664     long *switch_count;
2665     task_t *prev, *next;
2666     runqueue_t *rq;
2667     prio_array_t *array;
2668     struct list_head *queue;
2669     unsigned long long now;
2670     unsigned long run_time;
2671     int cpu, idx;
2672
2673     /*
2674      * Test if we are atomic. Since do_exit() needs to call into
2675      * schedule() atomically, we ignore that path for now.
2676      * Otherwise, whine if we are scheduling when we should not be.
2677      */
2678     if (likely(!current->exit_state)) {
2679         if (unlikely(in_atomic())) {
2680             printk(KERN_ERR "scheduling while atomic: "
2681                  "%s/0x%08x/%d\n",
2682                  current->comm, preempt_count(), current->pid);
2683             dump_stack();
2684         }
2685     }
2686 }
Done

```

- Kconfig in jedem Unterverzeichnis
 - Wurzel in arch/x86/Kconfig oder arch/arm/Kconfig
 - Format dokumentiert in
Documentation/kbuild/kconfig-language.txt
- Beispiele für Konfiguration (in Datei .config)
 - #CONFIG_FAT_FS is not set
CONFIG_PROC_FS = y
CONFIG_EXT3_FS = m //Kern-Modul
 - Welche Dateien übersetzen? (im Makefile)
subdir-y := parport char block net sound
subdir-\$(CONFIG_SCSI) += scsi
 - Welche Code-Teile übersetzen?
#ifdef CONFIG_PROC_FS
...
#endif

■ Text auf Konsole ausgeben

- `int printk(KERN_ALERT const char *fmt, ...)` (nicht im Scheduler)
- Verschiedene Stufen der Dringlichkeit

■ Speicher reservieren / freigeben

- `void *kmalloc(size_t size, gfp_t flags)`
- `void kfree(void *__ptr)`
- Beispiel

```
struct acpi_fan *fan = NULL;
fan = kmalloc(sizeof(struct acpi_fan), GFP_KERNEL);
if (!fan) return -ENOMEM;
...
kfree(fan);
```

- Unterbrechungen aus-/einschalten
 - `local_irq_save()`, `local_irq_restore()`
- Einzelne Werte (z.B. int, long) vom/zum Userspace kopieren
 - `put_user()`, `get_user()`
- Beliebige Datenmengen vom/zum Userspace kopieren
 - `copy_to_user()`, `copy_from_user()`

Beispiel: Helloworld-Modul (Code)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int helloworld_init(void) {
    printk(KERN_ALERT "Hello World!");
    return 0;
}

static void helloworld_exit(void) {
    printk(KERN_ALERT "Goodbye World!");
}

module_init(helloworld_init);
module_exit(helloworld_exit);
```

Beispiel: Helloworld-Modul (Code)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int helloworld_init(void) {
    printk(KERN_ALERT "Hello World!");
    return 0;
}

static void helloworld_exit(void) {
    printk(KERN_ALERT "Goodbye World!");
}

module_init(helloworld_init);
module_exit(helloworld_exit);
```

Beispiel: Helloworld-Modul (Code)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int helloworld_init(void) {
    printk(KERN_ALERT "Hello World!");
    return 0;
}

static void helloworld_exit(void) {
    printk(KERN_ALERT "Goodbye World!");
}

module_init(helloworld_init);
module_exit(helloworld_exit);
```

Beispiel: Helloworld-Modul (Code)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int helloworld_init(void) {
    printk(KERN_ALERT "Hello World!");
    return 0;
}

static void helloworld_exit(void) {
    printk(KERN_ALERT "Goodbye World!");
}

module_init(helloworld_init);
module_exit(helloworld_exit);
```

- Symbole exportieren
 - `EXPORT_SYMBOL(helloworld);`
- Makefile
 - `obj-m := helloworld.o`
- make aufrufen
 - `make -C /path/to/kernel-source SUBDIRS=$PWD modules`

■ Definitionen

```
■ struct list_head {  
    struct list_head *next, *prev;  
};
```

■ Initialisierung

```
■ #define LIST_HEAD(name) \  
    struct list_head name = { &(amp;name), &(name) }
```

■ Hinzufügen/Entfernen von Elementen

```
■ void list_add(struct list_head *new,  
    struct list_head *head);  
■ void list_del(struct list_head *entry);
```


■ Initialisierung

```
■ struct apm_user {  
    struct list_head list;  
    ...  
};  
■ static LIST_HEAD(apm_user_list);  
   struct apm_user *u;  
   list_add(&u->list, &apm_user_list);
```

■ Schleife über alle Elemente

■ `struct list_head *l;`

```
list_for_each(l, apm_user_list) {  
    struct apm_user *a =  
        list_entry(l, struct apm_user, list);  
    ...  
}
```

- Verzögerte Ausführung von Code
- Erzeugen mit
 - `struct workqueue_struct`
`*create_workqueue(const char *queue);`
- Tasks müssen in `struct work_struct` verpackt werden
 - `DECLARE_WORK(name, (*function)(void *));`
- Einhängen in Workqueue
 - `int queue_work(
 struct workqueue_struct *queue,
 struct work_struct *name);`

- Einfacher Synchronisationsmechanismus
- Warten, bis ein anderer Thread die entsprechende Aufgabe erledigt hat
- Deklaration
 - `DECLARE_COMPLETION(my_comp);`
- Auf Ereignis warten
 - `wait_for_completion(&my_comp);`
- Aufwecken (anderer Thread)
 - `complete(&my_comp);`
- Struktur sollte vor jeder Verwendung neu initialisiert werden
 - `INIT_COMPLETION(my_comp)`

- Warten, bis eine bestimmte Bedingung erfüllt ist
- Deklaration
 - `DECLARE_WAIT_QUEUE_HEAD(queue);`
- Warten
 - `wait_event(queue, condition);`
- Aufwecken (anderer Thread)
 - `wake_up(&queue);`
- Weitere Varianten siehe `include/linux/wait.h`

- Zugriff auf Attribute von Kernobjekten, z.B. von Geräten
- Ersetzt das procfs in großen Teilen
- Initialisierung
 - ```
#include <linux/device.h>
DEVICE_ATTR("energy", 0644, show_energy,
 store_energy);
device_create_file(device, &dev_attr_energy);
...
device_remove_file(device, &dev_attr_energy);
```

## ■ lesen

```
ssize_t show_energy(struct device *dev,
 struct device_attribute *attr, char *buf) {
 return sprintf(buf, "%i\n", dev->energy);
}
```

## ■ Kern reserviert immer Puffer der Größe PAGE\_SIZE

## ■ schreiben

```
ssize_t store_energy(struct device *dev, struct
 device_attribute *attr, char *buf, size_t count) {
 sscanf(buf, "%i", &dev->energy);
 return count;
}
```

- Relevant für Prozessinformationen

- Initialisierung

- ```
struct proc_dir_entry *entry =  
    create_proc_entry("cpufreq",  
        S_IRUGO|S_IWUSR, &proc_root);
```

```
if (!entry) ...
```

```
entry->read_proc = cpufreq_proc_read;  
entry->write_proc = cpufreq_proc_write;  
...  
remove_proc_entry("cpufreq", &proc_root);
```






```
■ static int cpufreq_proc_read(  
    char *buffer, char **start, off_t off, int count,  
    int *eof, void *data) {
```

```
    int len = 0;  
    while (len + off + 80 < count)  
        len += sprintf(buffer + len + off, "...");  
    *eof = 1; /* alle Daten geschrieben */  
    return len;  
}
```

- Siehe Kommentar „How to be a proc read function“ in `fs/proc/generic.c`

```
■ static int cpufreq_proc_write(  
    struct file *file, const char *buffer,  
    unsigned long count, void *data) {  
  
    char tmp[100];  
    if (count > sizeof(tmp)) return -EINVAL;  
    if (copy_from_user(tmp, buffer, count))  
        return -EFAULT;  
    ...  
    return count;  
}
```

-  BOVET, CESATI: Understanding the Linux Kernel (3rd Edition)
-  MAURER: Linux Kernelarchitektur: Konzepte, Strukturen und Algorithmen von Kernel 2.6
-  CORBET, RUBINI, KROAH-HARTMAN: Linux device drivers (3rd Edition)
 - PDF-Version unter `/home/power/pub/documents`

- Einloggen und mit Testrechner verbinden
- Versionsverwaltung aufsetzen
- Arbeitsversion des Linux-Kerns auswählen/einchecken
- Minimale Kernkonfiguration anpassen
- „Hallo Welt“ im Kern
 - Kern übersetzen
 - Testrechner mit neugebautem Kern starten
 - Kleine Änderung am Kern vornehmen (z.B. Startmeldung)
 - Neu bauen, Testrechner neu starten, Auswirkung der Codeänderung kontrollieren

Energiegewahres Dateisystem (pm2/pm3)

- Stromsparendes Dateisystem mit zwei Speichermedien
 - HDD (Hintergrundspeicher)
 - Solid State Disk (SSD) (energiesparender Cache)



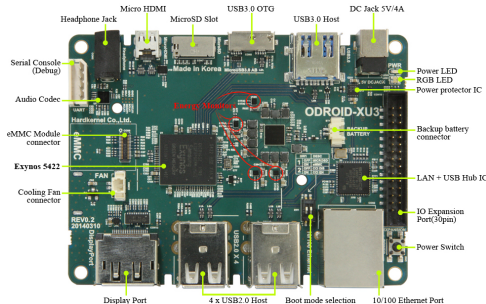
- Ereigniszähler des Prozessors nutzen
- Energie berechnen / auslesen (RAPL)
- Energieprofile von Tasks erstellen / vergleichen
- Drosseln



Ereignis	Gewicht [nJ]
INST_RETIRED	-0,224
CPU_CLK_UNHALTED	4,546
DSB2MITE_SWITCHES	-5,972
DSB_FILL:ALL_CANCEL	64,240
ILD_STALL:IQ_FULL	-1,425
L2_RQSTS:PF_HIT	-22,837
LD_BLOCKS:ALL_BLOCK	5,504
LD_BLOCKS:DATA_UNKNOWN	-6,281
UOPS_DISPATCHED:STALL_CYCLES	-2,508
BR_INST_RETIRED:FAR_BRANCH	-18031,295

■ Energiemessung und Management

- Energieverbrauch der unterschiedlichen Kerne ermitteln
- Prozessor drosseln, wenn Energiekontingent erschöpft
- ...



- 9. Dezember 2015
- R149 (Poolraum) im Informatik-Gebäude
 - Kern bauen, falls noch nicht geschehen
 - Linuxkern Quellcode lesen, Design überlegen, Experimente, etc.
 - Fragen stellen
 - ...