Einführung in das PM Praktikum

Power Management Praktikum

Simon Kellner, Raphael Neider

System Architecture Group

WS 10/11

Zielsetzung

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

Organisation

- Einteilung in Gruppen zu je 2–3 Personen
  - erfolgt beim nächsten Praktikumstermin
- Bearbeitung eines Themas bis zum Ende des Semesters
- Zunächst hauptsächlich Vorlesungstermine, später mehr Praktikumstermine

Praktikumsschein (Informatik)

- Vorführen der Implementierung
  - Keine 100 % perfekte / vollständige Implementierung erwartet
  - Aber Bemühung um die Lösung des Problems muss erkennbar sein
- Abgabe des Quellcodes als Patch
- Kurzes Vorstellen der Lösung am Ende des Semesters

Benotete Leistung (Informationswirte)

- Voraussetzungen wie Praktikumsschein, aber Benotung
- Gewichtung
  - 70 % Design und Implementierung
  - 30 % Präsentation

Ablauf

- Programmierung im Poolraum 149
  - Geöffnet Mo–Fr, ca. 9–19h
- „Offizielle“ Praktikumstermine
  - Fragen, Hilfestellungen, etc.
  - Abwechselnd mit Vorlesungsterminen
  - Genaue Termine im Netz
- Vorstellung der Ergebnisse
  - 8. Februar
  - 10. Februar

Zielsystem

- Linux
  - PC (Pentium 4)
  - Low-Power-System (Atom)

Aufgaben

- PC
  - Energieabschätzung mit Ereigniszählern (2x)
- Atom
  - Process Cruise Control (2x)
  - PAST-Algorithmus (2x)
  - Energiegewahres Dateisystem
  - Thermal Management
  - Spindown-Strategie (2x)

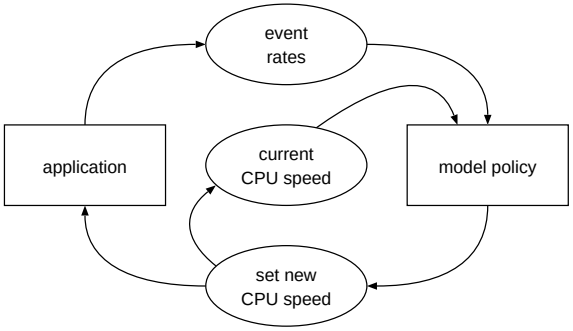
Energieabschätzung

- Ereigniszähler des Prozessors nutzen
- Energie berechnen
- Energieprofile von Tasks erstellen

Ereignis	Zählregister	Gewicht [nJ]
time stamp counter	MSR_IA32_TS_COUNTER	8,14
unhalted cycles	MSR_IA32_BPU_COUNTER0	11,5
µop queue writes	MSR_IA32_MS_COUNTER0	7,71
retired branches	MSR_IA32_MS_COUNTER1	0,914
mispred branches	MSR_IA32_IQ_COUNTER0	552
mem retired	MSR_IA32_IQ_COUNTER1	2,81
mob load replay	MSR_IA32_BPU_COUNTER1	48,6
ld miss 1L retired	MSR_IA32_IQ_COUNTER2	22,1
floating point	MSR_IA32_FLAME_COUNTER0	0,697

Process Cruise Control

- Optimale Frequenz taskspezifisch wählen
  - rechenintensive Tasks ➜ hohe Frequenz
  - speicherintensive Tasks ➜ niedrige Frequenz



PAST-Algorithmus

- Idle-Phasen vermeiden
- Aktive Phasen durch Verringerung der Frequenz strecken

```
idle_cycles = hard_idle + soft_idle;
run_cycles += excess_cycles;
run_percent = run_cycles /
   (idle_cycles + run_cycles);

next_excess = run_cycles -
   speed * (run_cycles + soft_idle)
IF excess_cycles < 0. THEN
   excess_cycles = 0.

energy = (run_cycles - excess_cycles) *
   speed * speed;
```

```
IF excess_cycles > idle_cycles THEN
   newspeed = 1.0;
ELSEIF run_percent > 0.7 THEN
   newspeed = speed + 0.2;
ELSEIF run_percent < 0.5 THEN
   newspeed = speed -
      (0.6 - run_percent);

IF newspeed > 1.0 THEN
   newspeed = 1.0;
IF newspeed < min_speed THEN
   newspeed = min_speed;

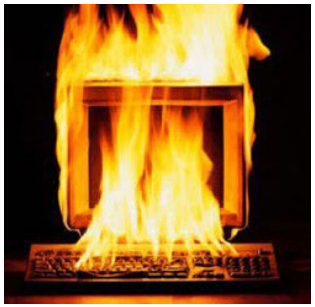
speed = newspeed;
excess_cycles = next_excess;
```


Energiegewahres Dateisystem

- Stromsparendes Dateisystem mit zwei Speichermedien
  - Platte
  - Flash

Thermal Management

- Überhitzen des Prozessors verhindern
  - Prozessor drosseln
  - Rechenintensive Prozesse drosseln, interaktive nicht benachteiligen



Spin-Down Strategie

- Festplatte abschalten, wenn voraussichtlich in naher Zukunft nicht mehr darauf zugegriffen wird

DDT-Algorithmus

Festplatte in den Standby-Modus setzen, falls

$$t_{la} + t_{be} \leq t$$

Die Variablen bedeuten:

t : die aktuelle Zeit

t_{la} : wann wurde zuletzt auf die Festplatte zugegriffen (last_access)

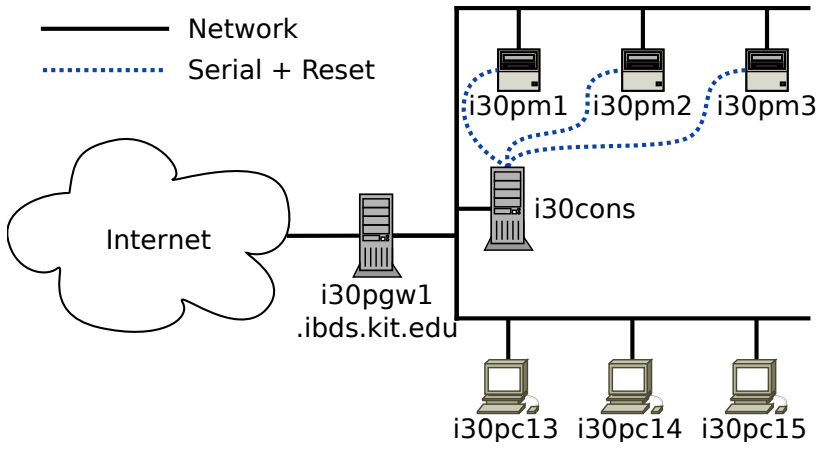
t_{be} : die break-even-Zeit

- Erweiterung: DDT-ES

Praktikumsumgebung: Rechner

- Entwicklungsrechner im Poolraum
  - Editieren von Quellcode
  - Übersetzen
- Zielsystem läuft auf Testrechner im Rechnerraum
  - pm1, . . . , pm10
  - Ausgabe über serielle Konsole
- Konsolenrechner
  - Fernsteuerung der Testrechner über Minicom
  - Remote-Reset
- Messsystem für Leistungsmessungen

Praktikumsumgebung: Netzwerk



Testrechner booten

- Per ssh auf i30cons einloggen
- `minicom <Rechnername>` aufrufen
  - Beispiel: `minicom pm1`
- Einloggen auf serieller Konsole oder per ssh
  - Neustart durch `shutdown -r now`
- Aus minicom jederzeit möglich: Reset mit Ctrl+a r
- Konsolenausgabe erscheint (Kernmeldungen etc.)
- Wurde der Testrechner erfolgreich gebootet, einloggen ebenfalls über ssh
  - Beispiel: `ssh i30pm1`
- Live-Demo nächstes Mal

Kernkonfiguration/Build-Prozess

- Kernquellen unter `/home/power/pub`
- Entpacken z.B. in `~/src` mit `tar xjvf <Dateiname>`
- Konfigurieren mit `make menuconfig`
  - Standardkonfiguration kann verwendet werden
- Übersetzen mit `make`

Booten des neuen Kerns

- Kompilierter Kern liegt im Verzeichnis `arch/i386/boot/bzImage` bzw. `arch/x86/boot/bzImage`
- Kern wird mit grub übers Netzwerk geladen
- Ablegen/verlinken in `~/boot`
- Eintragen in `~/boot/menu.lst` analog zum vorgegebenen Eintrag
  - Dateisystem auf lokaler Platte im Testrechner
  - Ausgabe der Kernmeldungen auf serieller Schnittstelle

Hilfsmittel zur Kernprogrammierung

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

linux.no

Linux/kernel/sched.c - Firefox

File Edit View Go Bookmarks Tools Help

http://lxr.linux.no/source/kernel/sched.c#L2662

```
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                 }
```

Done

Konfiguration

- `Kconfig` in jedem Unterverzeichnis
  - Wurzel in `arch/x86/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
    ```

Bibliotheksfunktionen

- Text auf Konsole ausgeben
  - `int printk(const char *fmt, ...)`
- 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);
```


Bibliotheksfunktionen (2)

- 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 (Build)

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

Kernstrukturen: Listen

- Definitionen
  - ```
    struct list_head {
      struct list_head *next, *prev;
    };
    ```
- Initialisierung
  - ```
    #define LIST_HEAD(name) \
        struct list_head name = { &(name), &(name) }
    ```
- Hinzufügen/Entfernen von Elementen
  - ```
    void list_add(struct list_head *new,
        struct list_head *head);
    ```
  - ```
    void list_del(struct list_head *entry);
    ```


Kernstrukturen: Listen (2)

- 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);
    ```

Kernstrukturen: Listen (3)

- 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);
  ...
}
```


Kernstrukturen: Workqueues

- 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);`

Kernstrukturen: Completions

- 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)`

Kernstrukturen: Warteschlangen

- 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`

sys-Dateisystem

- Zugriff auf Attribute von Kernobjekten, z.B. von Geräten
- 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);
    ```


sys-Dateisystem, Zugriff

- 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` (=4096)
- schreiben

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

proc-Dateisystem

- 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);
    ```


proc-Dateisystem, lesen

- ```
  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`

proc-Dateisystem, schreiben

```
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;
}
```


Notes

Literatur

- 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

Nächster Termin

- Wird noch bekanntgegeben
- R149 (Poolraum) im Informatik-Gebäude
  - Gruppeneinteilung
  - Verteilung der Themen
  - Kern übersetzen, booten, etc.