Generalizing the SPUFS concept – a case study towards a common accelerator interface

Andreas Heinig, René Oertel, Jochen Strunk, Wolfgang Rehm, Heiko Joerg Schick

andreas.heinig@cs.tu-chemnitz.de

Department of Computer Science
Chemnitz University of Technology

April 2nd 2008
Different accelerators are available on the market
They all are integrated through different ways into the Linux environment, e.g.:

<table>
<thead>
<tr>
<th>Accelerator</th>
<th>Integration</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell/B.E.</td>
<td>VFS</td>
<td>SPUFS</td>
</tr>
<tr>
<td>FPGA</td>
<td>Char. Dev</td>
<td>OpenFPGA, AAL</td>
</tr>
<tr>
<td>(GP)GPU (Tesla)</td>
<td>Char. Dev</td>
<td>CUDA</td>
</tr>
</tbody>
</table>

⇒ No common attach point inside the Kernel is available
⇒ This is disadvantageous for application- and library programmers
  ⇒ Every interface has other syntax and semantics
We propose a common interface concept (based on "SPUFS") called "ACCFS"
The way to ACCFS

- **Introduction**
- **Cell/B.E.**
- **SPUFS**
- **ACCFS**
- **Conclusion**

**Current Situation**

**Proposal**

**Virtualization**

**VFS-Interface**

---

**The way to ACCFS**

**User Level**

- Application
- `libspe *`
- System call interface

**Kernel Level**

- `libfs`
- `spufs *`
- Character / Network / Block device drivers

**Hardware Level**

- `PPE` (Cell/B.E.)
- `CPU` (case study: Opteron)
- ACCelerator: e.g. FPGA

**Andreas Heinig**

Generalizing the SPUFS concept
Integration Concepts

1. Virtualization of the accelerators
2. VFS Context access
3. Separation of the functionalities
Virtualization of the Accelerator

Why Virtualization?

- Accelerators are mostly only exclusively usable
- In a multi-user system several applications may need the same accelerator
  ⇒ The system can dead-lock
  ⇒⇒ Virtualization avoid resource conflicts

Abstraction

We abstract the physical accelerator with accelerator context
Why VFS?

Other approaches to integrate accelerators into Linux:

1. **Character Devices**
   - Simply to implement
   - Controlled via `read`, `write` and `ioctl`
   - Hard to virtualize

2. **New thread space**
   - The *physical accelerator* gets abstracted with *process running on the accelerator*
   - These processes can be scheduled by the kernel
   - Hard to implement
     - Dublicated kernel infrastructure
     - Possibly large number of new system calls
     - Changes of all system calls dealing with the PIDs
VFS Interface (II)

- A virtual file system is a middle way
  ⇒ Easy to virtualize
  ⇒ Not so hard to implement

Main advantage

VFS uses well known system calls like open/close, read/write and mmap

With SPUFS we have a working solution which uses the VFS and the virtualization concepts.
Cell Broadband Engine Architecture

- Corporate development of Sony, Toshiba and IBM ("STI")
- Based on the Power Architecture
- Extended with eight cooperative offload processors
- Designed for the next generation of entertainment devices

Features:
- Heterogeneous multiprocessor
- High-speed bus interface
- On-chip memory controller
Block Diagram

Andreas Heinig
Generalizing the SPUFS concept
Introduction

- SPUFS = Synergistic Processing Unit File System
- Virtual File System (VFS) mounted on "/spu" by convention
- Integrates the SPUs in the Linux environment
SPUFS - Concepts

1. Virtualization of the SPE
2. VFS Context Access
Virtualization of the SPE

- SPUFS manages SPE Contexts
- The context contains all data for suspending and resuming SPU program execution
- The context gets bounded on a physical SPE by a scheduler
**VFS Interface**

**Only two new system calls**

1. `sys_spu_create`
   - Creates a new SPU context, reflecting in a new directory

2. `sys_spu_run`
   - Starts SPU-code execution on the specified context
   - Synchronously ⇒ the calling thread blocks until completion

**Context directory entries**

<table>
<thead>
<tr>
<th>File</th>
<th>Perm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbox</td>
<td>r</td>
<td>the first SPU to CPU communication mailbox</td>
</tr>
<tr>
<td>ibox</td>
<td>r</td>
<td>the second SPU to CPU communication mailbox</td>
</tr>
<tr>
<td>wbox</td>
<td>-</td>
<td>CPU to SPU communication mailbox</td>
</tr>
<tr>
<td>mem</td>
<td>r w</td>
<td>local-store memory</td>
</tr>
<tr>
<td>regs</td>
<td>r w</td>
<td>register file</td>
</tr>
</tbody>
</table>
**Introduction**

- **ACCFS = ACCelerator File System**
- Virtual File System (VFS) mounted on "/acc" by convention
- Proposal for integrating different kinds of accelerators into the Linux environment

---

**Andreas Heinig**

Generalizing the SPUFS concept
ACCFS consists of two parts

1. **Top half: accfs**
   - VFS implementation
   - Provides the user/VFS interface
   - Provides the vendor interface

2. **Bottom half: device handler**
   - Vendor specific part
   - Integrates the accelerator

<table>
<thead>
<tr>
<th>Structure</th>
<th>ACCFS Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top Half</td>
<td>accfs</td>
</tr>
<tr>
<td>Bottom Half</td>
<td>device handler</td>
</tr>
</tbody>
</table>

```
| Vendor Interface | "VFS Interface" |
| User Interface (Syscalls) | |
```

Andreas Heinig
Generalizing the SPUFS concept
User/VFS Interface

Two new system calls

1. `sys_acc_create`
   - Creates a new ACCFS context, reflecting in a new directory

2. `sys_acc_run`
   - Starts the accelerator
   - Synchronously ⇒ the calling thread blocks until completion

Context directory entries (first proposal)

<table>
<thead>
<tr>
<th>File</th>
<th>Perm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regs</td>
<td>r w</td>
<td>register file</td>
</tr>
<tr>
<td>message</td>
<td>r w</td>
<td>message interface</td>
</tr>
<tr>
<td>memory/</td>
<td>r w</td>
<td>exported memories</td>
</tr>
<tr>
<td>semaphore/</td>
<td>r w</td>
<td>semaphores</td>
</tr>
</tbody>
</table>
Vendor Interface

Loading (registration) of a device handler
accfs_register (const struct accfs_vendor *)

Unloading of the device handler
accfs_unregister (const struct accfs_vendor *)

The structure accfs_vendor contains:
- Callback functions (create, destroy and run context; DMA; ...)
- Accelerator Identifier
  - Unique number identifying the accelerator
  ⇒ Has to be specified when calling sys_acc_create
Device Handler – Tasks

Managing the accelerator

- Establish the interconnection
- Initialize the hardware
- Configure memory mappings
- ...

Provide the virtualization

- Only the device handler "knows" what regions of the accelerator has to be saved
- If the accelerator does not support virtualization, only the amount of physical accelerator units can be allocated when calling `sys_acc_create`
Possible Benefits

1. Device handlers have only to concentrate on the hardware integration
   - No management of operation system structures
   - No providing of a whole user interface

2. Ease the development of library programing
   - Well known interface
     ⇒ No non-standard ioctl differing from one accelerator to another
   - Better usability of the accelerator
     ⇒ Always the same usage ”protocol”:
       1. Create the Context
       2. Upload the Code
       3. Execute the Context
       4. Destroy the Context

3. The accelerator becomes better exchangeable
There is no well-defined interface to integrate accelerators into the Linux kernel
We propose a common interface based on SPUFS
Open to different kinds of accelerators

Further Work
Finish the interface implementation of ACCFS
Porting SPUFS to an ACCFS device handler for SPEs
Implementing device handlers for the first accelerators other than Cell
References

J. A. Kahle et al.

A. Bergmann

A. Heinig

Heterogeneous Multiprocessing On a tightly coupled Opteron Cell evaluation platform. *Poster presented at CASCON, Dublin, October 2007.*
Thank you for your attention!

Questions?