A design proposal for a shareable USB server in a microkernel environment

Daniel Ernst   Matthias H. F. Jurisch

Hochschule RheinMain
Fachbereich Design Informatik Medien
Unter den Eichen 5
D-65195 Wiesbaden

First Wiesbaden Workshop on Advanced Microkernel Operating Systems

Overview

1. Introduction
   - Application Example
   - USB
2. OS-Level USB Support
   - Linux
   - Microkernel
3. Bus Access Scheduling
   - Scheduling Algorithm comparison
   - Selected Approach
4. Access Rights Management
   - ACLs and Capabilities
   - Hotplugging
5. Server Design Overview
6. Conclusion

Introduction

- USB is very popular in the desktop market
- Nowadays also used in smaller devices (e.g. the Raspberry Pi)
- Used to tether multiple peripherals
- Accessed through the USB host controller

Problem

USB host controller is a shared resource

- Trend in automotive industry: Using Android as media center
- A USB-thumbdrive with MP3s can be plugged into the media-console
- Also, other devices can use the USB-host of the media center, e.g. a tachometer

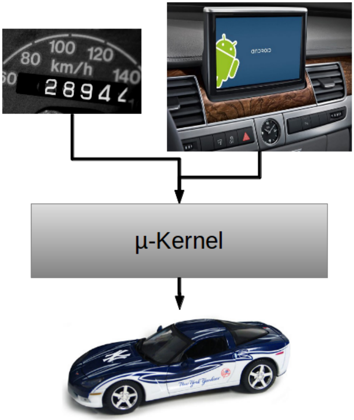

Figure: The application scenario.

- Universal Serial Bus
- Strictly hierarchical and host-centric (everything routes through the USB-host)
- Device descriptors for each device
- Descriptor-hierarchy: Device descr. $\Rightarrow$ configuration descr. $\Rightarrow$ interface descr. $\Rightarrow$ endpoint descr.
- Every endpoint is the end of a unidirectional pipe to the USB host

Descriptors

- idVendor: Samsung Electronics Co.
- iProduct: Galaxy Nexus
- bDeviceClass: Mass-storage-device

OS-Level USB Support

Linux

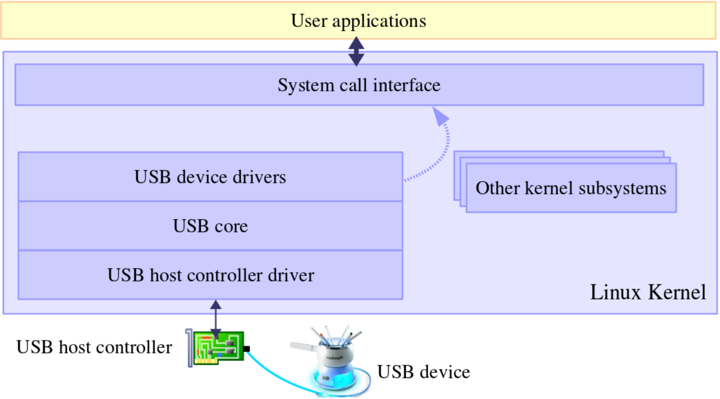

Figure: USB in the Linux Kernel[1]

[1] http://free-electrons.com/doc/linux-usb.pdf

OS-Level USB Support

Linux

- Communication between layers 2 and 3: *USB Request Blocks* (URBs)
- URB encapsulates USB requests, contains:
  - Device and endpoint identification
  - Pointer to memory with payload buffer
  - Pointer to completion handler
- When URB is submitted, it can be passed to the host controller
- Model is sufficient, because USB is host centric (all transfers are started by the host)

OS-Level USB Support

Microkernel

- Not Acceptable: give hardware acces to all clients
- Key question: Abstraction level?
- Should we provide a function-like interface?

Sending URBS

- Idea: A server allows clients to send URB-like datastructures.
- Provide USB library that encapsulates sending URBs

Problems

- If we have more than one pending URB: what to process next?
- How to decide which clients can submit URBs concerning specific devices?

Bus Access Scheduling

- Scheduling
  - Determining which task is allowed acces to a resource at a given time.
- Task: URB
- Resource: Forwarding requests to actual host controller driver
- Scheduling will determine, given a set of URBs, which URB will be processed next

Bus Access Scheduling

Comparison

Naive: *First in first out* (FIFO)

- Simple FIFO datastructure, first submitted URB is processed first
- Blocking USB bus is possible

Popular: *Earliest Deadline First* (EDF)

- Task has Deadline d
- Next scheduled: Task with earliest d
- Optimal for single resource scheduling

Multi-Resource: *Least Laxity First* (LLF)

- Task has Deadline d and execution time c
- Compute Laxity $l = (d - t) - c$
- Optimal for single resource scheduling

Fixed Priority Scheduling

- Each task has a fixed priority
- Task with highest priority gets scheduled next
- URBs would receive priority of client

- EDF application:
  - Non-rt URBs with deadline ∞
  - Would be scheduled with FIFO
  - Specific non-rt scheduling usefull
- LLF: We don't know the execution time

Bus Access Scheduling

Selected Approach

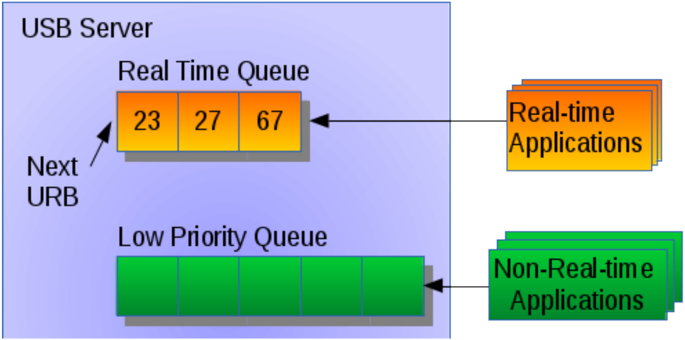

Figure: Selected Approach[2]

[2]Numbers represent the deadline of the respective URB

Access Rights Management

Why is it necessary?

- Application scenario: Android vs. real-time application accessing USB-devices
- Malicious example: Bad application sending/receiving tachometer information

Conclusion

We need to restrict which application may access which device

Access Rights Management

Protection Matrix

	Object							
Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Figure: A protection matrix[3].

- Domain: Application like Android/real-time-application/etc.
- Object: USB-device like USB-thumbdrive/fondue-pot/etc.
- Empty cells mean there are *no* rights (whitelisting, principle of least privilege)

[3] A. S. Tanenbaum. Modern operating systems

Access Rights Management

ACL vs. Capabilities

- Capabilities (Caps): Domains hold which objects they may access
- Access Control List (ACL): Objects hold which domains may access them
- Both have their pros and cons. . .

Capabilities' main issue

- Domains granted access on mere possession of a Cap
- Domains need to manage their Caps $\Rightarrow$ API needs to be changed!

Proposal: ACLs

- Easily implemented in a centric and isolated environment (the server)
- Issues of ACLs are acceptable for our use-case

Access Rights Management

Hotplugging

- What if an unknown device is hotplugged? — No known Access Control Entries (ACE) for that device
- Idea 1: Block hotplugged devices ⇒ USB-thumbdrive with MP3's is useless!
- Idea 2: Don't block hotplugged devices ⇒ Malicious!
- Idea 3: Some sort of authorization for applications and USB-devices ⇒ API changes!

Proposal

- Static configuration with whitelisted devices
- Use USB-device-descriptors for specific or role-based whitelisting
- e.g. *mass-storage-device* (bDeviceClass) or *Samsung Electronics Co., Ltd* (idVendor)

Conclusion

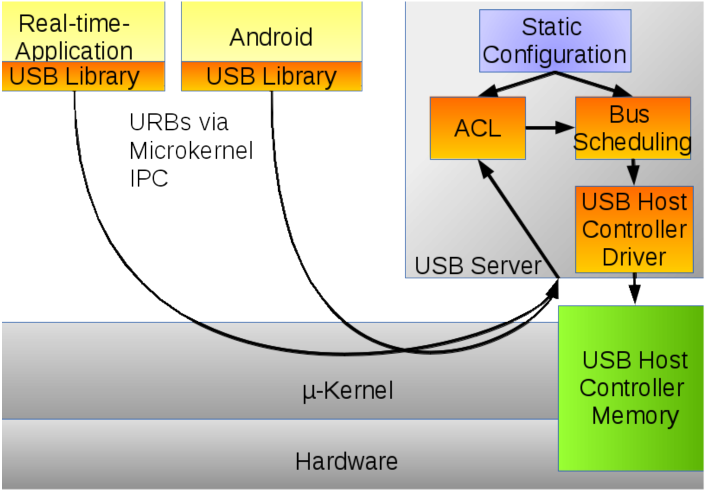

Figure: Server Design

Conclusion

- USB as shareable resource
- Application scenario: Multiple applications, mircokernel, multiple USB devices
- Leverage of existing code and protecting real-time-applications has high priority

Major issues

- Bus scheduling for real-time and non-real-time simultaneously
- Access rights management — which application may use which device

Proposals

- A scheduling algorithm combining deadline- and priority-scheduling
- An Access Control List with a group-based whitelist via static configuration