TUXERA

Make it work.

SMB Server
Kernel versus User Space Learnings

Oleg Kravtsov

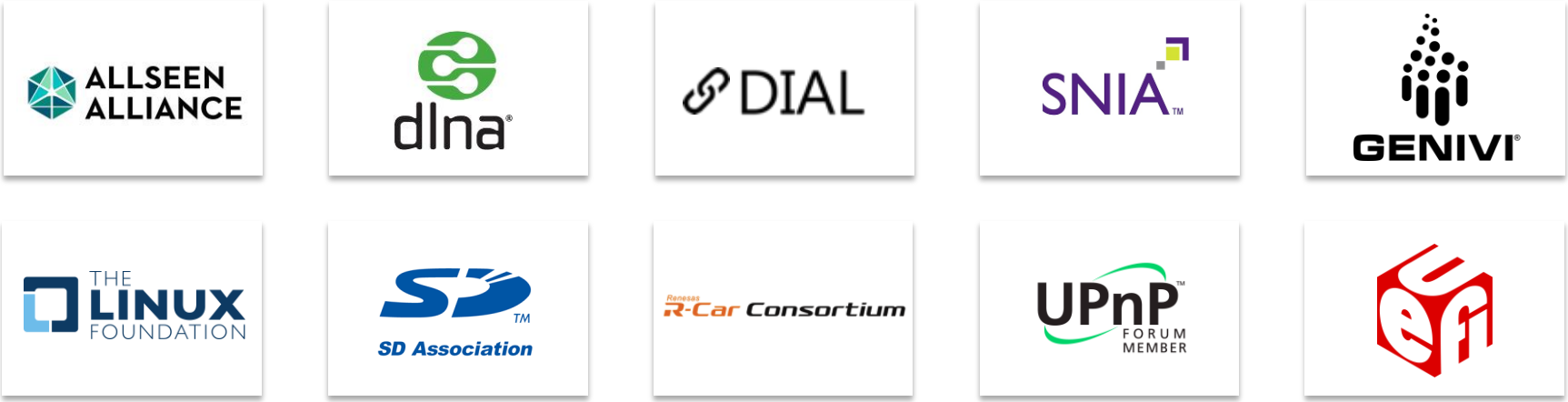
Lead Software Developer

Tuxera is the leading provider of file systems software and networking technologies.

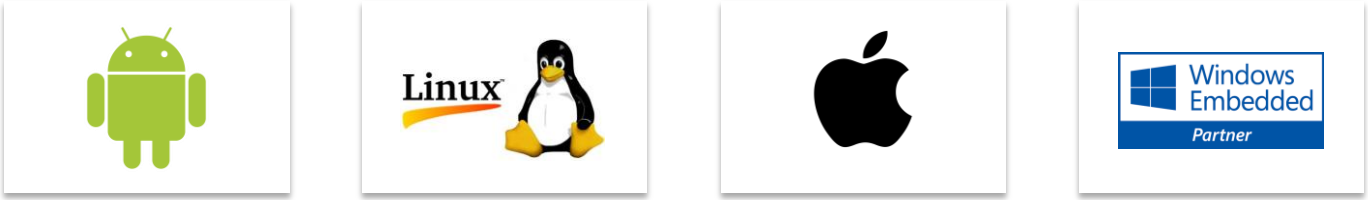
We enable people and businesses to reliably store, access, and share any type of data on any device.

Strong partnerships ensure interoperability

We're active members in key industry associations

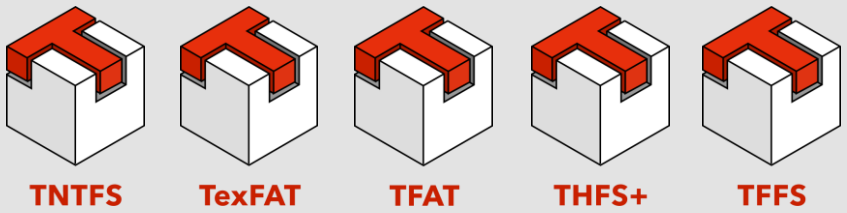


Selected software partners



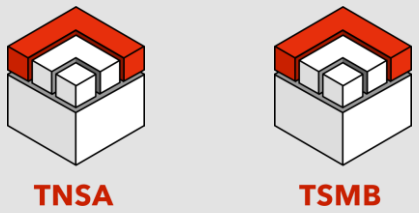
Challenges solved by Tuxera's technology

Fully interoperable data storage support



Tuxera's file system solutions

High-performance network storage



Tuxera's network technology

Multiscreen media streaming

Tuxera's SDKs

Data storage and streaming for the connected home

Tuxera's consumer apps

Motivation – embedded vs. enterprise

Each use case has it's own unique risks, complexities, and constraints

vs.

Motivation – embedded platforms

General characteristics

OS: Mostly Linux

Use: Routers, set-top boxes, etc. in home networks

Server paradigm: Both user and kernel

Hardware constraints

- Limitation on CPU and processor capacity (low clock speed, etc.)
- Resource use by other parallel processes
- Limited memory

Software characteristics

- Highly dependent on underlying HW
- Code on-disk and in-memory size
- Energy-efficient
- Increased performance expectation under resource constraints
- Simple security requirements
- Typically less number of clients

Flexibility in development

- Dependent on toolchain capabilities
  - Perhaps limited library set

Motivation – enterprise platforms

General characteristics

OS: Manufacturer-specific

Use: Large-scale server farms, data centers

Hardware constraints

- Mostly large-scale servers
- No or very low hardware and resource constraints

Software characteristics

- Performance
- Scalability: scale-up or scale-out + large number of simultaneous client support
- Robustness
- Availability
- Complex security configurations
- Enterprise jargons: multi-tenancy, de-dup, etc.

Flexibility in development

- If not based on Linux, need implementation of OS-specific function calls

Motivation – common expectations

- Real-time behavior for events and expected outcome
- High demands on availability, security, and interoperability
- Usually long-lived systems, hence reliability is very important
- Multi-client application

User space vs. kernel space – user space

Reasons we love user space...

- Highly flexible compared to kernel space
- Better sanity checks and fault tolerance
- No system freezes due to sub-optimal or error-prone code
- Development in kernel avoids unnecessary data copies
  - Sendfile, recvfile (unofficial), etc.
- Debugging and monitoring are easy with a plethora of tools

Reasons user space sucks…

- Overhead added due to:
  - Context switches
  - Making system calls
  - Data copy between user and kernel spaces
- Additional CPU consumption
  - TLB misses
  - Memory page swapping, etc.

User space vs. kernel space – kernel space

Reasons we love kernel space...

- Tight integration with VFS and network stack
- Avoids duplicate caching compared to user space
- Simpler zero-copy read and write interface
- Typically better performance than user space
- Because it's just cool...

Reasons kernel space sucks…

- Steep learning curve
- Requires kernel know-how for efficient programming
- Debugging and monitoring not as easy as user-space
- Some utilities require provisioning kernel with appropriate config options
- If you screw up, deadlocks, panic, and freezes are possible

General design goals

- **Modularity**

  Important services and core functionalities divided into sub systems

- **Hierarchical**

  Lower-level subsystems provide services to upper level

- **Multi-threaded and multi-process hybrid architecture**
- **Maximum interrupt spread on multi-core architecture**

  Controlling thread (process) affinity?

- **Design based on minimizing cache misses?**

  We are being dreamy here...

Space-specific design goals

User space

- Reduced penalty due to boundary crossing and buffer copies
- Memory mapped IO?

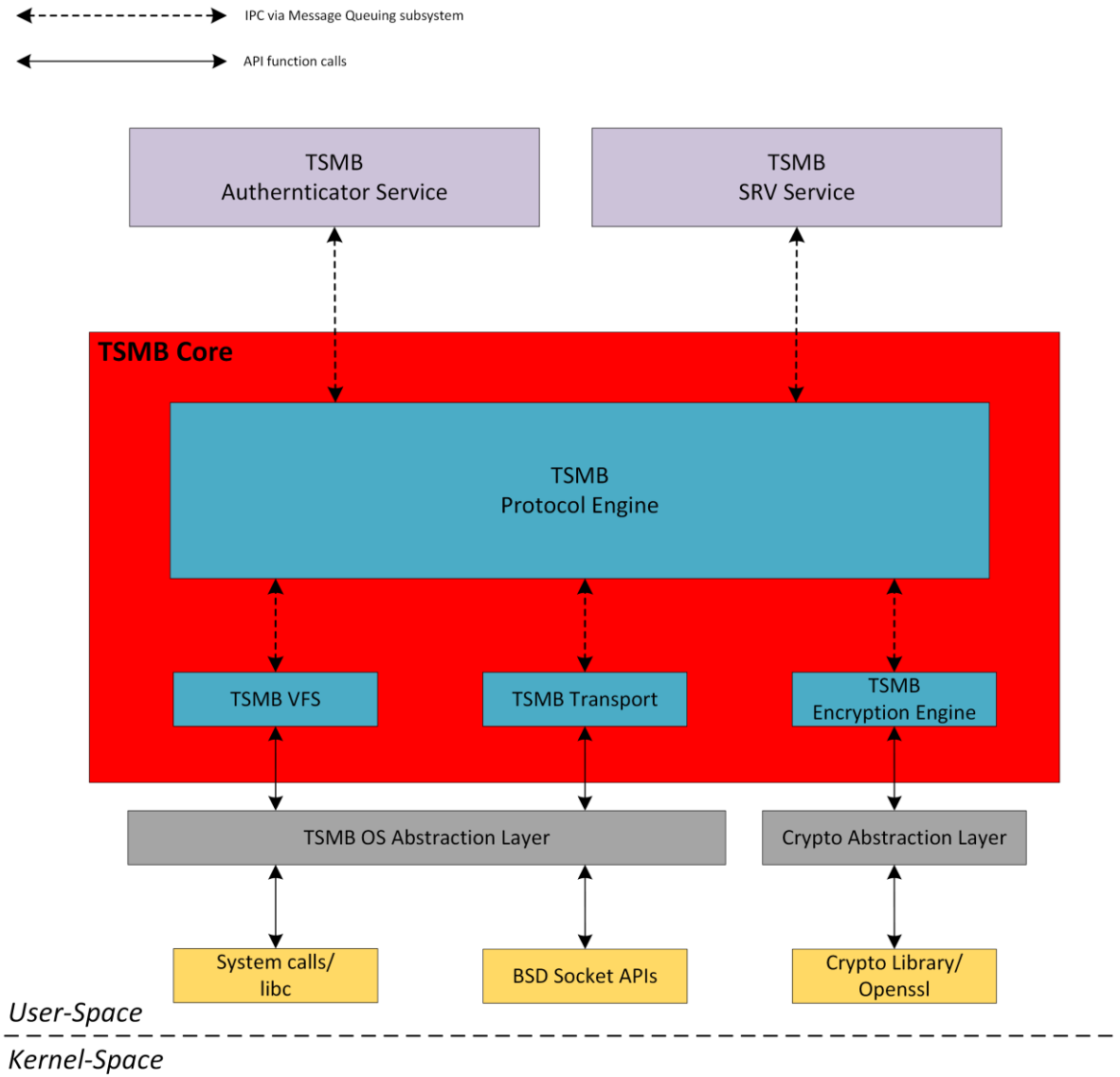
Kernel space

- Keep it comparatively light weight

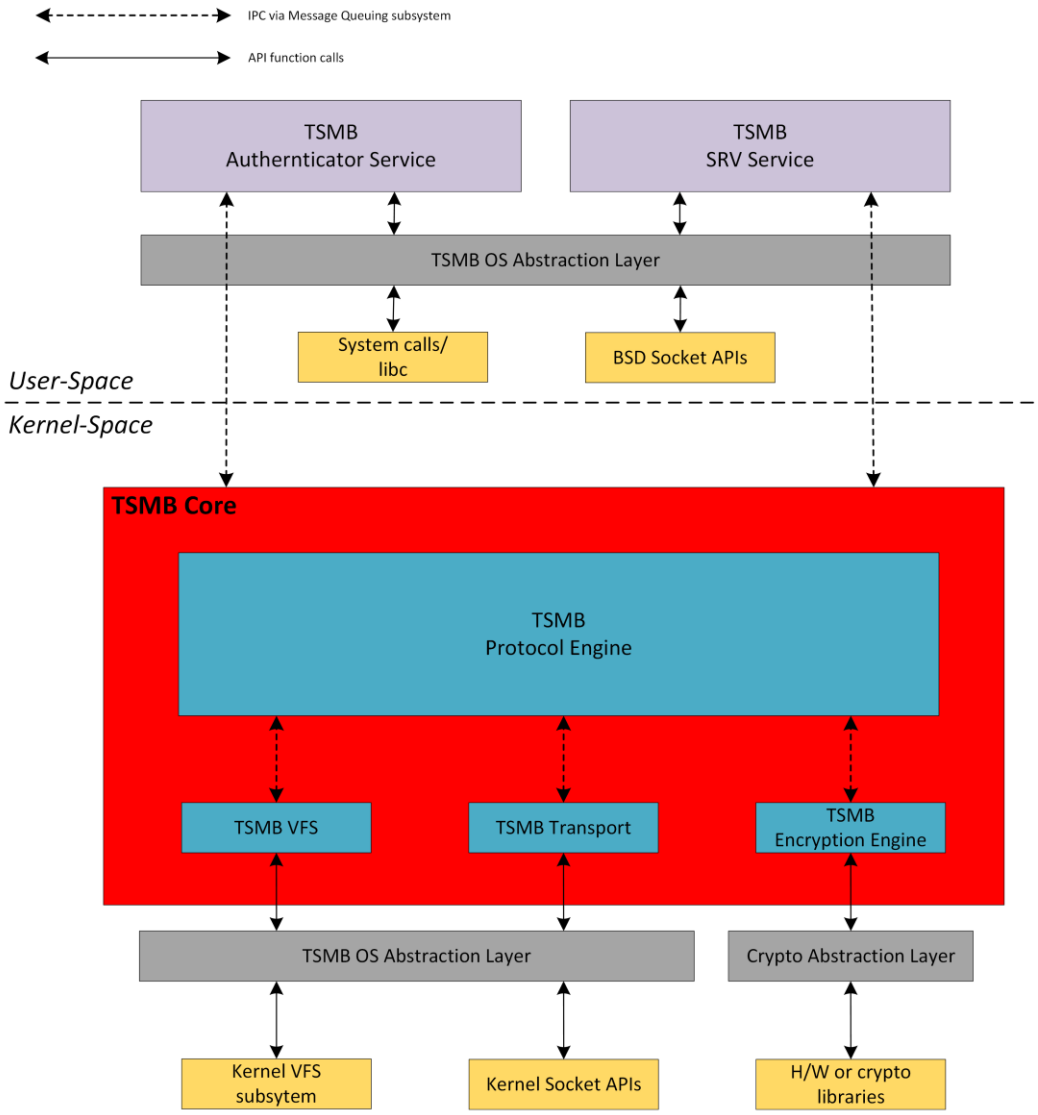
Architecture

Server Format	No. Of Binaries	Binary type	Server Mode
User space	1	Server binary	Single process (Threaded)
			Multi process (Forked)
Kernel space	2	Server binary	Single process (Threaded)
			Multi process (Forked)
		Kernel module	Kernel threads

Architecture – user-space format



Architecture – kernel-space format



Architecture – abstractions, everywhere... (1 of 2)

Hybrid architecture made possible with strong abstraction layer

Architecture – abstractions, everywhere... (2 of 2)

Threads/processes

- User-space (pthreads and fork)
- Kernel threads

Memory management

- Libc-malloc
- Kmalloc
- Memory pools?

Timer

- Posix timer, kernel timer

Network

- Socket descriptor, struct sock

Locks

- Pthread locks, mutex, spinlocks

File system I/O operations

- Streams, descriptors, struct file, etc.
- Allows pluggable VFS modules

In-house message queuing subsystem

Architecture – components

Main components

- TSMB core (user space binary or kernel module)
  - Transport
  - VFS
  - Encryption Engine
  - Protocol Engine
- TSMB Authenticator (service in user space)
- TSMB SRV (Service in user space)

- **Same code for all ports**
  - Most optimizations are in abstraction layer for different ports
- **Core can reside in kernel or user space depending on compiled format**

Architecture – TSMB core

- In user format → lives in user-space
- In kernel format → resides as a kernel module
- Protocol Engine
  - Lock less, sleep less thread
  - Receives and forwards packet from and to transport sub-component
  - Receives and forwards packets from and to VFS
- VFS
  - Threaded component
  - Handles file system operations
- Transport
  - Threaded component
  - Handles Rx/Tx of SMB PDUs from/to authenticated clients
  - SMB Direct (RDMA) – new transport is as simple as a pluggable abstraction
- Encryption engine
  - Threaded component
  - Encrypts and decrypts SMB PDU
  - Can either use in-house encryption and signing libraries or link to Openssl (when in user space)

Architecture – service components

- Always in user space
- Runs as a separate process in user space
- Supplements the TSMB core
- Authenticator service
  - Authenticates sessions from new clients
  - In kernel format, TSMB-core sends an up-call to get security context of users, UID, GIDs, etc.
- SRV service
  - Server management service
  - House the DCE/RPC component
    - Supports RPC endpoints srvsvc, spoolss, etc.
- Service components are pluggable
- Maintaining services in user space reduces server bulkiness
- Decision to maintain non-performance critical components in user space
- New services like clustering, witness may be added as services

What did we learn?

1. Abstractions are important

- Clean abstraction can separate core components from OS-specific calls
- Abstraction provided clean interface among different components
- Only learning curve is the knowledge of local abstracted syscall
  - Divide and conquer
  - Code easily manageable

What did we learn?

2. Deadlock nightmares

- Code in interrupt context should be kept extremely light and lock free
  - Obviously not always the case
- Appropriate locking mechanism **must** be considered depending on the following:
  - The context in which a particular code is acquiring or releasing the locks
  - The context in which the data being locked is accessed
- The contexts may be **Interrupt (or) Process**
- Properly handle disabling pre-emption or disabling interrupts

What did we learn?

3. Performance gain (is it a myth?)

- Embedded hardware shows a significant performance gain when in kernel space compared to user space
- Powerful desktop PCs dont show significant boost
  - Most gain in CPU usage, kernel CPU usage < user space CPU usage

What did we learn?

4. Memory fragmentation is real

- Long-running servers with frequent malloc/free cause extreme memory fragmentation
- Application crashes on Ubuntu laptops with ”Page allocation failure” error
- Especially prominent when using SMB PDU size > 64 KB
- Memory pools save the day
  - Static pools sometimes limit performance
  - Dynamic pools boosted performance
  - CPU usage reduced by about 5-10% when using pools

Tuxera Inc.

Corporate Headquarter

Itämerenkatu 9 00180 Helsinki, Finland

info@tuxera.com