Managarm: Design of a pragmatic fully-asynchronous microkernel

Alexander van der Grinten
avdgrinten@managarm.org

The Managarm Project

FOSDEM'22

Bio

Postdoctoral researcher, CS

- Humboldt-University of Berlin, Germany
- PhD in 2018 from University of Cologne, Germany

Research focus: engineering of parallel algorithms, graph algorithms, hard combinatorial problems

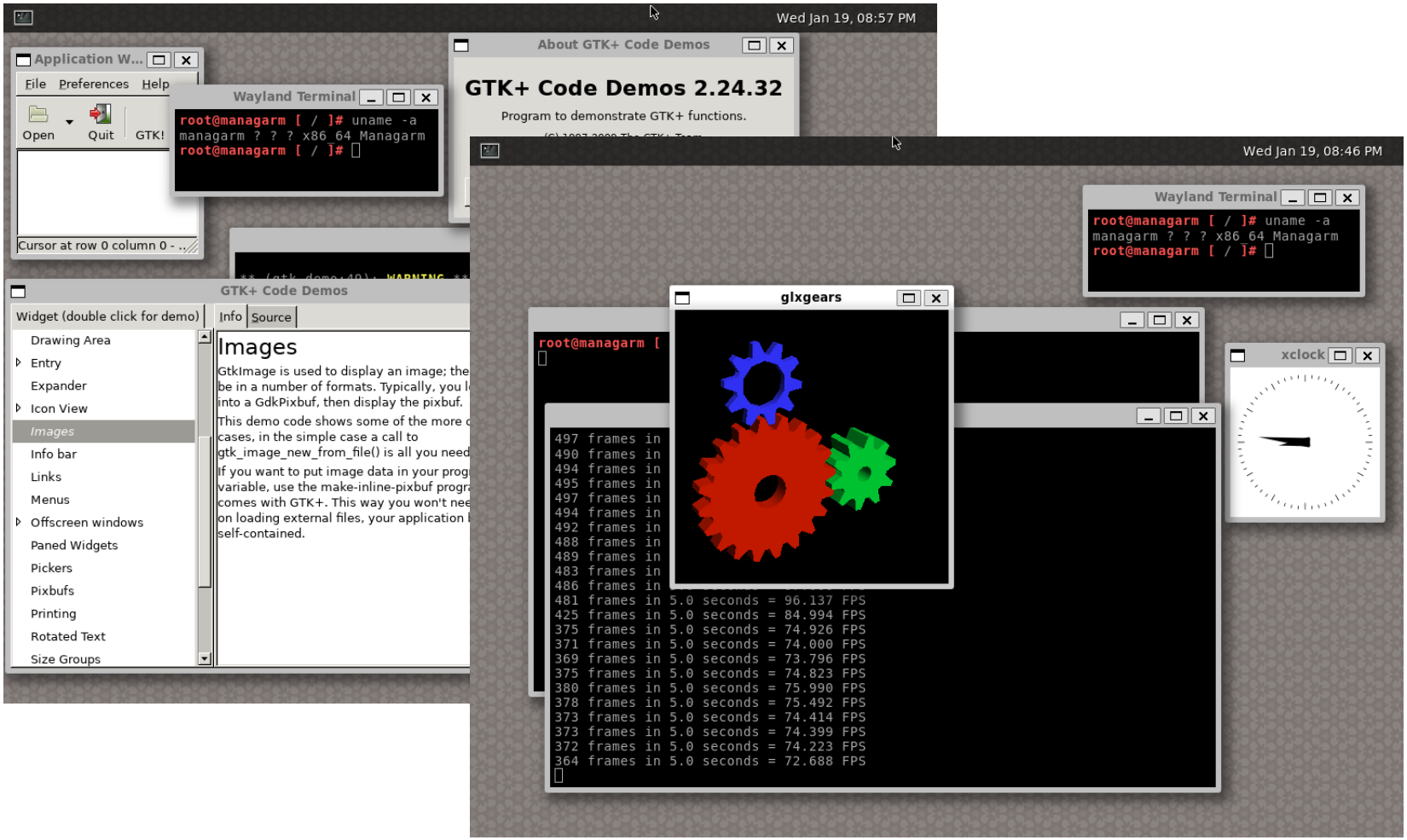
Currently: distributed graph algorithms,

german research foundation (DFG) grant.

This talk: Design of the Managarm OS, particularly its kernel.

- Open-source project, mostly written in modern C++.
- Founded in 2014.
- Many active contributors.

What is Managarm?



What is Managarm?

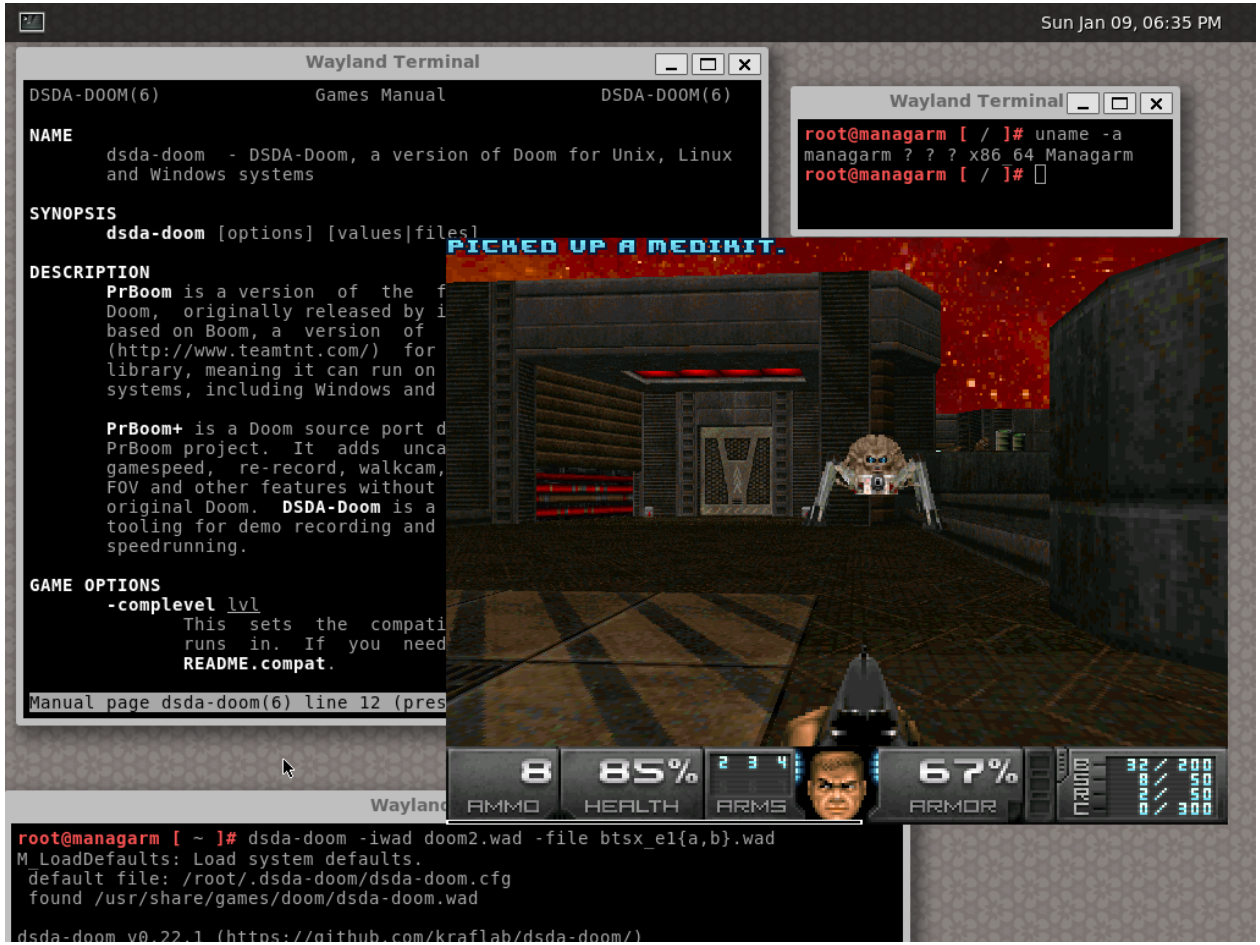
Managarm: microkernel-based **general-purpose** OS with focus on asynchronous I/O.

- Fully asynchronous = (almost) all system calls are asynchronous (same for drivers/servers, ...).
- Reasonably good source-level compatiblity to Linux via user space emulation.

Advantage of asynchronicity: can handle high number of concurrent requests faster and using fewer resources (e.g., fewer threads, less RAM).

Potential **use cases**: servers, desktop, mobile devices.
More generally: systems with a MMU and a few MiB of RAM.

What is Managarm?



. . . and it can also run Doom.

Agenda

- System Architecture
- Inter-Process Communication (IPC)

System Architecture

In what sense is Managarm "pragmatic"?

L4's design follows a minimality principle:

"A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations, would prevent the implementation of the system's required functionality." (Liedtke, 1995)

Managarm also allows concepts inside the kernel if they enable the implementation of user space APIs (e.g., POSIX APIs) in a **fundamentally more efficient** way.

$\rightsquigarrow$ Results in a somewhat bigger kernel than L4, similar in scope to Google's Zircon.

Examples of performance-oriented features

MM Kernel can handle **copy-on-write** (CoW) memory regions.

- Avoids unnecessary context switches compared to a user space implementation.

MM Kernel can handle **page caches** (e.g., of files).

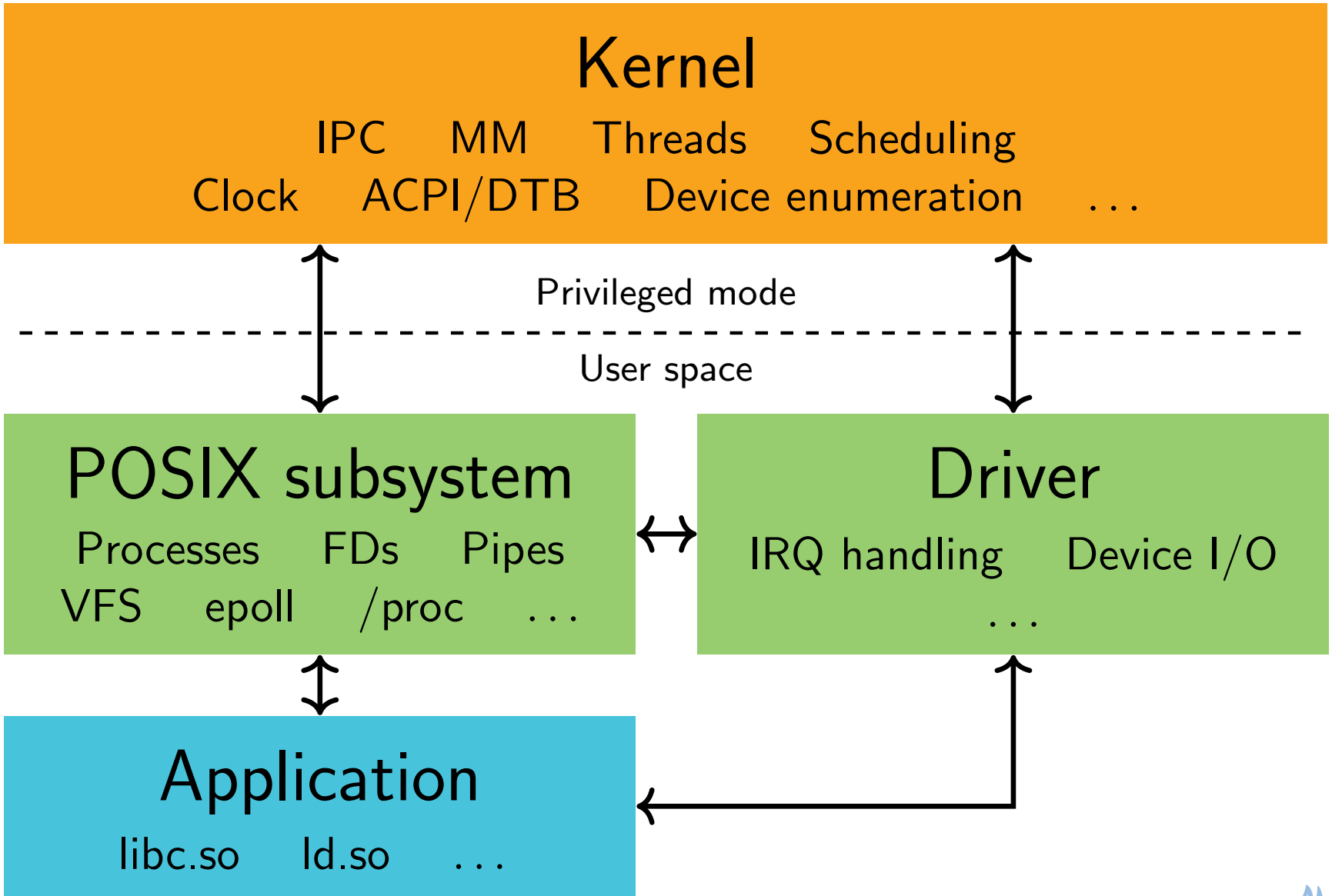
- Kernel has no concept of a file – only of a cache of memory pages.
- Page faults (= loading page contents on demand) and writeback are still handled by user space.

IRQs Kernel allows small snippets of code to run in kernel-level IRQ handlers, before user space handlers are invoked.

- Requires some protection mechanism (e.g., safe programming languages, signing) to enforce security boundaries.

...

Managarm: block diagram



Capability-based design

Like many other OSes: Managarm uses a **capability-based** design.
(Well-established design; nothing new.)

A thread can only access a (kernel) resource
if the thread owns a **capability** that refers to the resource
(and that permits the desired operation).

- In user space capabilities are represented by **integer handles**.
- Handles are only meaningful relative to the thread that operates on them. (Same handle can refer to a different capability from a different thread's perspective.)
- Kernel resources have **no global name** (in contrast to designs that use VFSes or named IPC ports).

Capabilities in Managarm

- IPC streams (details later in this talk).
- Threads.
- (Virtual) address spaces.
- Memory objects: collections of physical pages
  (+ extra features such as CoW), or device memory.
- "Universes": collections of capabilities
  (each thread has an associated universe).
- IRQs.
- Virtualized CPUs.
- ...

Asynchronous system calls

Almost all syscalls are **async** in Managarm.

- Examples: IPC, mapping/unmapping memory, waiting for an IRQ, ...

Exception: system calls that explicitly synchronize threads; these are mostly **futex** operations.

- Required to implement mutexes, condition variables etc. in user space.
- Also **required to block a thread** when there is no work to do.

Asynchronous notifications

In contrast to synchronous syscalls: async syscalls do not complete when control flow returns to user space.

Instead: **lock-free ring buffer** is used to notify user space whenever an async syscall completes.

- Similar to io_uring in Linux, ...
- Retrieving notification requires **zero syscalls** on the fast path.
  Syscall is only required if thread needs to block when no notifications are available.
- User space uses a pointer-sized value to match completion notifications to pending syscalls.

Inter-Process Communication (IPC)

IPC: overview

IPC model $\approx$ async variant of L4 IPC

- IPC is only dispatched when **both** sender/receiver are ready.
- IPC operations (e.g., send/receive) are queued.
- Message contents (= bytes) are **not** queued.

Advantage: can handle efficiently arbitrary number of concurrent requests from a single thread.

Disadvantage: queuing of IPC operations requires memory allocations and bookkeeping within the kernel.

But: in-kernel representation of IPC operations is fixed-size and small.

$\rightsquigarrow$ Fast path still **competitive** with simpler synchronous IPC (e.g., in L4-like kernels).

IPC streams

Managarm uses "**streams**" as IPC primitive.

- Two endpoints per stream.

Threads post "**actions**" (= IPC operations) to each endpoint of the stream.

- Examples: sending/receiving bytes, transfering capabilities.
- Actions on both ends are **matched** against each other and dispatched: first action on first endpoint is matched to first action on second endpoint, ...
- Actions on both ends of the stream must be **compatible**: send/receive ✓, send/send ✗.

A single IPC syscall can submit multiple actions to a stream.

Data transfer

Basic actions to **send/receive bytes**: `SendFromBuffer` / `RecvToBuffer` (size must be known in advance).

Other send/receive actions to avoid copying in certain situations:

- `RecvInline`: receive data to the notification ring buffer (as part of completion notification; bounded size).
- `SendFromBufferSg`: scatter-gather.
- ...

All send* actions are compatible with all recv* actions.

Other actions allow **transfering capabilities** or have specialized purposes (e.g., proving the identity of the thread that operates on the first endpoint to the second endpoint).

Request/response logic

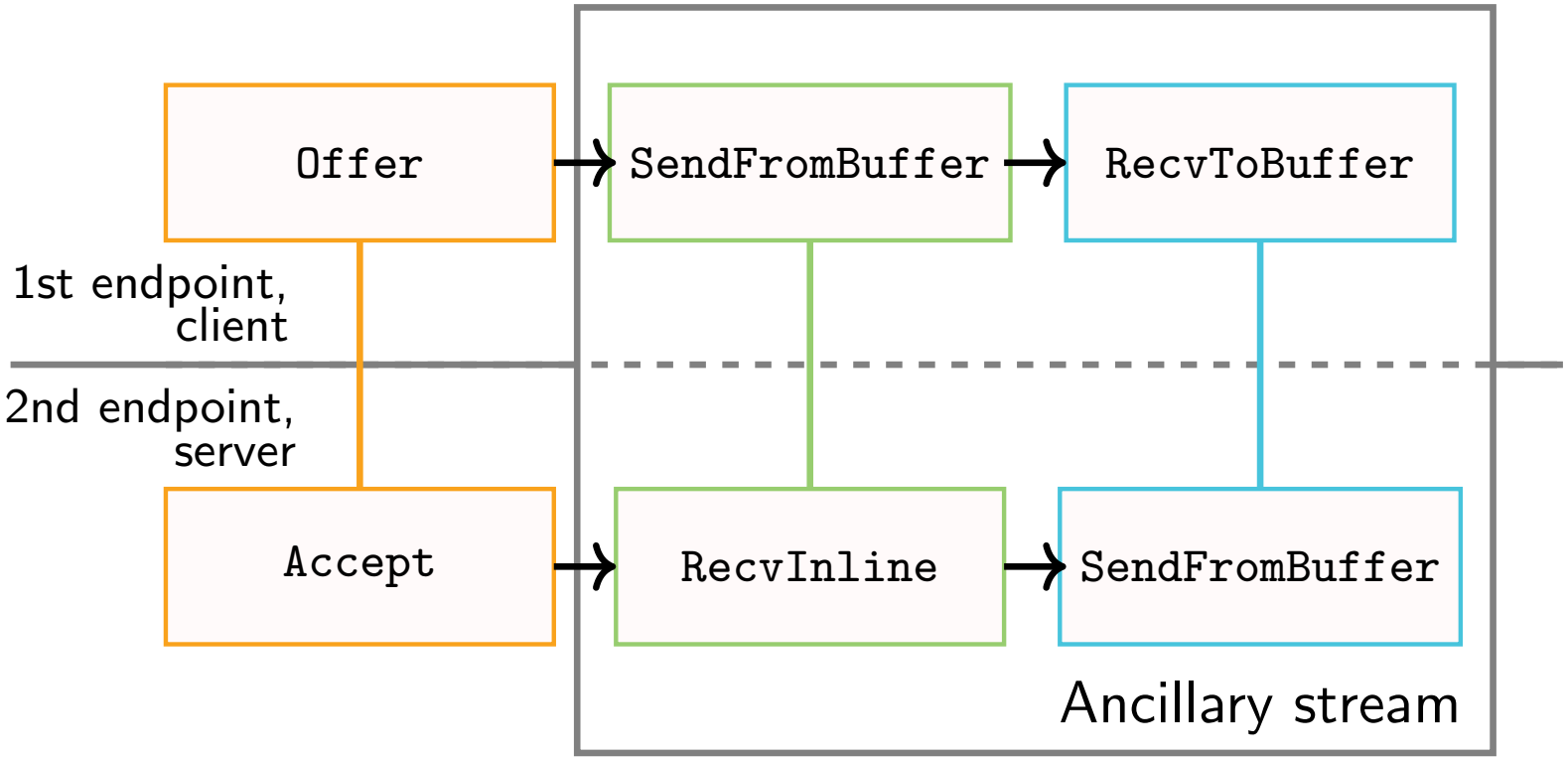
It is often desirable to multiplex multiple concurrent requests/responses over a single stream.

$\rightsquigarrow$ Allows multiple clients to talk to same server.

Managarm uses "`Offer`"/"`Accept`" actions for this purpose.

- `Offer`/`Accept` pair creates a new "**ancillary**" stream (Ancillary stream is usually short-lived and discarded after a single request/response).
- Subsequent actions can be directed to the new stream (without the need to invoke an additional syscall).

Example: client/server IPC



1. Server posts `Accept` → `RecvInline` (to receive a request).
2. Client posts `Offer` → `SendFromBuffer` → `RecvToBuffer` (to send a request and receive a response).
3. Server posts `SendFromBuffer` (to send the response).

Acknowledgements

Check out the project: `github.com/managarm/managarm`

Blog: `managarm.org` Twitter: `@managarm_OS`

Thanks to all contributors!

Co-maintainer of the project: Kacper Słomiński (`@qookei`).

Arsen Arsenović (`@ArsenArsen`), Dennis Bonke (`@Dennisbonke`), Geert Custers (`@Geertiebear`), Matteo Semenzato (`@Matt8898`), Matt Taylor (`@64`), Thomas Woertman (`@thomtl`), `@0xqf`, `@Apache-HB`, `@AtieP`, `@BrainStackOverFlow`, `@Itay2805`, `@JasonBrave`, `@Jimx-`, `@Menotdan`, `@N00byEdge`, `@PositronTheory`, `@RicardoLuis0`, `@Sebastian-byte`, `@alula`, `@aurelian2`, `@austanss`, `@cflaviu`, `@dchapiesky`, `@fentanyl-1`, `@fido2020`, `@ikbenlike`, `@itsmevjnk`, `@kITerE`, `@krkk`, `@mintsuki`, `@msathieu`, `@no92`, `@notYuriy`, `@piotrrak`, `@sappigeninja`, `@streaksu`, `@tearosccebe`, `@toor`, .