Efficient Synchronization of Linux Memory Regions over a Network: A Comparative Study and Implementation

TODO: Add subtitle

Felicitas Pojtinger (Stuttgart Media University)

2023-08-04

Introduction

Introduction

TODO: Add introduction

Technology

The open-source Linux kernel, was created by Linus Torvalds in 1991. Developed primarily in the C programming language, it has recently seen the addition of Rust as an approved language for further expansion and development, esp. of drivers[1]. The powers millions of devices across the globe, including servers, desktop computers, mobile phones, and embedded devices. It serves as an intermediary between hardware and applications, as an abstraction layer that simplifies the interaction between them. It is engineered for compatibility with a wide array of architectures, such as ARM, x86, RISC-V, and others.

The kernel does not function as a standalone operating system. This role is fulfilled by distributions, which build upon the Linux kernel to create fully-fledged operating systems[2]. Distributions supplement the kernel with additional userspace tools, examples being GNU coreutils or BusyBox. Depending on their target audience, they further enhance functionality by integrating desktop environments and other software.

Linux is a extensible, but not a microkernel. Despite it's monolithic nature, it allows for the integration of kernel modules[2]. Kernel modules are small pieces of kernel-level code that can be dynamically incorporated into the kernel, presenting the advantage of extending kernel functionality without necessitating system reboots.

The dynamism of these modules comes from their ability to be loaded and unloaded into the running kernel as per user needs. This functionality aids in keeping the kernel size both manageable and maintainable, thereby promoting efficiency. Kernel modules are traditionally developed using the C programming language, like the kernel itself, ensuring compatibility and consistent performance.

Kernel modules interact with the kernel via APIs (Application Programming Interfaces). Despite their utility, since they run in kernel space, modules do carry a potential risk. If not written with careful attention to detail, UNIX signals are an integral component of UNIX-like systems, including Linux. They function as software interrupts, notifying a process of significant occurrences, such as exceptions. Signals may be generated from various sources, including the kernel, user input, or other processes, making them a versatile tool for inter-process notifications.

Aside from this notification role, signals also serve as an asynchronous communication mechanism between processes or between the kernel and a process. As such, they have an inherent ability to deliver important notifications without requiring the recipient process to be in a specific state of readiness[4]. Each signal has a default action associated with it, the most common of which are terminating the process or simply ignoring the signal.

To customize how a process should react upon receiving a specific signal, handlers can be utilized. Handlers dictate the course of action a process

Principle of Locality

The principle of locality, or locality of reference, refers to the tendency of a processor in a computer system to recurrently access the same set of memory locations within a brief span of time. This principle forms the basis of a predictable pattern of behavior that is evident across computer systems, and can be divided into two distinct types: temporal locality and spatial locality[6].

Temporal locality revolves around the frequent use of particular data within a limited time period. Essentially, if a memory location is accessed once, it is probable that this same location will be accessed again in the near future. To leverage this pattern and improve performance, computer systems are designed to maintain a copy of this frequently accessed data in a faster memory storage, which in turn, significantly reduces the latency in subsequent references.

Spatial locality, on the other hand, refers to the use of data elements that

The memory hierarchy in computers is an organized structure based on factors such as size, speed, cost, and proximity to the Central Processing Unit (CPU). It follows the principle of locality, which suggests that data and instructions that are accessed frequently should be stored as close to the CPU as possible[7]. This principle is crucial primarily due to the limitations of "the speed of the cable", where both throughput and latency decrease as distance increases due to factors like signal dampening and the finite speed of light.

TODO: Add graphic of the memory hierarchy

At the top of the hierarchy are registers, which are closest to the CPU. They offer very high speed, but provide limited storage space, typically accommodating 32-64 bits of data. These registers are used by the CPU to perform operations.

Following registers in the hierarchy is cache memory, typically divided into

Memory management forms a cornerstone of any operating system, serving as a critical buffer between applications and physical memory. Arguably, it can be considered one of the fundamental purposes of an operating system itself. This system helps maintain system stability and provides security guarantees, such as ensuring that only a specific process can access its allocated memory.

Within the context of the Linux operating system, memory management is divided into two major segments: kernel space and user space.

Kernel space is where the kernel itself and kernel modules operate. The kernel memory module is responsible for managing this segment. Slab allocation is a technique employed in kernel space management; this technique groups objects of the same size into caches, enhancing memory allocation speed and reducing fragmentation of memory[10].

User space is the memory segment where applications and certain drivers

Swap space refers to a designated portion of the secondary storage utilized as virtual memory in a computer system[11]. This feature plays a crucial role in systems that run multiple applications simultaneously. When memory resources are strained, swap space comes into play, relocating inactive parts of the RAM to secondary storage. This action frees up space in primary memory for other processes, enabling smoother operation and preventing a potential system crash.

In the case of Linux, swap space implementation aligns with a demand paging system. This means that memory is allocated only when required. The swap space in Linux can be a swap partition, which is a distinct area within the secondary storage, or it can take the form of a swap file, which is a standard file that can be expanded or truncated based on need. The usage of swap partitions and files is transparent to the user.

The Linux kernel employs a Least Recently Used (LRU) algorithm to

Page faults are instances in which a process attempts to access a page that is not currently available in primary memory. This situation triggers the operating system to swap the necessary page from secondary storage into primary memory. These are significant events in memory management, as they determine how efficiently an operating system utilizes its resources.

Page faults can be broadly categorized into two types: minor and major. Minor page faults occur when the desired page resides in memory but isn't linked to the process that requires it. On the other hand, a major page fault takes place when the page has to be loaded from secondary storage, a process that typically takes more time and resources[3].

To minimize the occurrence of page faults, memory management algorithms such as the afore-mentioned Least Recently Used (LRU) and the more straightforward clock algorithm are often employed. These algorithms effectively manage the order and priority of memory pages, mmap is a versatile UNIX system call, used for mapping files or devices into memory, enabling a variety of core tasks like shared memory, file I/O, and fine-grained memory allocation. Due to its powerful nature, it is commonly harnessed in applications like databases.

One standout feature of mmap is its ability to create what is essentially a direct memory mapping between a file and a region of memory[14]. This connection means that read operations performed on the mapped memory region directly correspond to reading the file and vice versa, enhancing efficiency by reducing the overhead as the necessity for context switches (compared to i.e. the read or write system calls) diminishes.

The key advantage that mmap provides is the capacity to facilitate zero-copy operations. In practical terms, this signifies data can be accessed directly as if it were positioned in memory, eliminating the need to copy it from the disk first. This direct memory access saves time and

inotify

The `inotify` is an event-driven notification system of the Linux kernel, designed to monitor the file system for different events, such as modifications and accesses, among others[15]. Its particularly useful because it can be configured to watch only write operations on certain files, i.e. only write operations. This level of control can offer considerable benefits in cases where there is a need to focus system resources on certain file system events, and not on others.

Naturally, `inotify` comes with some recognizable advantages. Significantly, it diminishes overhead and resource use when compared to polling strategies. Polling is an operation-heavy approach as it continuously checks the status of the file system, regardless of whether any changes have occurred. In contrast, `inotify` works in a more event-driven way, where it only takes action when a specific event actually occurs. This is usually more efficient, reducing overhead especially where there are infrequent changes to the file system.

Caching is a key feature of the Linux kernel that work to boost efficiency and performance. Within this framework, there are two broad categories: disk caching and file caching.

Disk caching in Linux is a strategic method that temporarily stores frequently accessed data in RAM. It is implemented through the page cache subsystem, and operates under the assumption that data situated near data that has already been accessed will be needed soon. By retaining data close to the CPU where it may be swiftly accessed without costly disk reads can greatly reduce overall access time. The data within the cache is also managed using the LRU algorithm, which prunes the least recently used items first when space is needed.

Linux also caches file system metadata in specialized structures known as the dentry and inode caches. This metadata encompasses varied information such as file names, attributes, and locations. The key benefit

TCP (Transmission Control Protocol), UDP (User Datagram Protocol), and QUIC (Quick UDP Internet Connections) are three key communication protocols utilized in the internet today.

TCP has long been the reliable backbone for internet communication due to its connection-oriented nature [16]. It ensures the guaranteed delivery of data packets and their correct order, rendering it a highly dependable means for data transmission. Significantly, TCP incorporates error checking, allowing the detection and subsequent retransmission of lost packets. TCP also includes a congestion control mechanism to manage data transmission seamlessly during high traffic. Due to to these features and it's long legacy, TCP is widely used to power the majority of the web where reliable, ordered, and error-checked data transmission is required.

UDP is a connectionless protocol that does not make the same guarantees about the reliability or ordered delivery of data packets [17]. This lends

Delta Synchronization

Delta synchronization is a technique that allows for efficient synchronization of files between hosts, aiming to transfer only those parts of the file that have undergone changes instead of the entire file in order to reduce network and I/O overhead. Perhaps the most recognized tool employing this method of synchronization is rsync, an open-source data synchronization utility in Unix-like operating systems[20].

TODO: Add sequence diagram of the delta sync protocol from https://blog.acolyer.org/2018/03/02/towards-web-based-delta-synchronization-for-cloud-storage-systems/

While there are many applications of such an algorithm, it typically starts on file block division, dissecting the file on the destination side into fixed-size blocks. For each of these blocks, a quick albeit weak checksum calculation is performed, and these checksums are transferred to the source system.

File Systems In Userspace (FUSE)

File Systems in Userspace (FUSE) is a software interface that enables the creation of custom file systems in the userspace, as opposed to developing them as kernel modules. This reduces the need for the low-level kernel development skills that are usually associated with creating new file systems.

The FUSE APIs are available on various platforms; though mostly deployed on Linux, it can also be found on macOS and FreeBSD. In FUSE, a userspace program registers itself with the FUSE kernel module and provides callbacks for the file system operations. A simple read-only FUSE can for example implement the following callbacks:

The getattr function is responsible for getting the attributes of a file. For a real file system, this would include things like the file's size, its permissions, when it was last accessed or modified, and so forth:

```
static int example_getattr(const char *path, struct stat *st
```

Network Block Device (NBD) is a protocol for connecting to a remote Linux block device. It typically works by communicating between a user space-provided server and a Kernel-provided client. Though potentially deployable over Wide Area Networks (WAN), it is primarily designed for Local Area Networks (LAN) or localhost usage. The protocol is divided into two phases: the handshake and the transmission[25].

TODO: Add sequence diagram of the NBD protocol

The NBD protocol involves multiple participants, notably one or several clients, a server, and the concept of an export. It starts with a client establishing a connection with the server. The server reciprocates by delivering a greeting message highlighting various server flags. The client responds by transmitting its own flags along with the name of an export to use; a single NBD server can expose multiple devices.

After receiving this, the server sends the size of the export and other

Virtual Machine Live Migration

Virtual machine live migration involves the shifting of a virtual machine, its state, and its connected devices from one host to another, with the objective to minimize disrupted service by minimizing downtime during data transfer processes.

Algorithms that intent to implement this usecase can be categorized into two broad types: pre-copy migration and post-copy migration.

Pre-Copy

The primary characteristic of pre-copy migration is its “run-while-copy” nature, meaning that the copying of data from the source to the destination occurs concurrently while the VM continues to operate. This method is also applicable in a generic migration context where an application or another data state is being updated.

In the case of a VM, the pre-copy migration procedure starts with transfering the initial state of VM’s memory to the destination host. During this operation, if modifications occur to any chunks of data, they are flagged as “dirty”. These modified or “dirty” chunks of data are then transferred to the destination until only a small number remain - an amount small enough to stay within the allowable maximum downtime criteria.

Following this, the VM is suspended at the source, enabling the synchronization of the remaining chunks of data to the destination

Post-copy migration is an alternative live migration approach. While pre-copy migration operates by copying data before the VM halt, post-copy migration opts for another strategy: it immediately suspends the VM operation on the source and resumes it on the destination – all with only a minimal subset of the VM's data.

During this resumed operation, whenever the VM attempts to access a chunk of data not initially transferred during the move, a page fault arises. A page fault, in this context, is a type of interrupt generated when the VM tries to read or write a chunk that is not currently present on the destination. This triggers the system to retrieve the missing chunk from the source host, enabling the VM to continue its operations[28].

The main advantage of post-copy migration centers around the fact that it eliminates the necessity of re-transmitting chunks of "dirty" or changed data before hitting the maximum tolerable downtime. This process can

Recent studies have explored different strategies to determine the most suitable timing for virtual machine migration. Even though these mostly focus on virtual machines, the methodologies proposed could be adapted for use with various other applications or migration circumstances, too.

One method[29] proposed identifies cyclical workload patterns of VMs and leverages this knowledge to delay migration when it is beneficial. This is achieved by analyzing recurring patterns that may unnecessarily postpone VM migration, and then constructing a model of optimal cycles within which VMs can be migrated. In the context of VM migration, such cycles could for example be triggered by a large application's garbage collector that results in numerous changes to VM memory.

When migration is proposed, the system verifies whether it is in an optimal cycle for migration. If it is, the migration proceeds; if not, the migration is postponed until the next cycle. The proposed process employs a Bayesian

Streams and pipelines are fundamental constructs in computer science, enabling efficient, sequential processing of large datasets without the need for loading an entire dataset into memory. They form the backbone of modular and efficient data processing techniques, with each concept having its unique characteristics and use cases.

A stream represents a continuous sequence of data, serving as a connector between different points in a system. Streams can be either a source or a destination for data. Examples include files, network connections, and standard input/output devices and many others. The power of streams comes from their ability to process data as it becomes available; this aspect allows for minimization of memory consumption, making streams particularly impactful for scenarios involving long-running processes where data is streamed over extended periods of time[30].

Pipelines comprise a series of data processing stages, wherein the output gRPC is an open-source, high-performance remote procedure call (RPC) framework developed by Google in 2015. It is recognized for its cross-platform compatibility, supporting a variety of languages including Go, Rust, JavaScript and more. gRPC is being maintained by the Cloud Native Computing Foundation (CNCF), which ensures vendor neutrality.

One of the notable features of the gRPC is its usage of HTTP/2 as the transport protocol. This allows it to exploit features of HTTP/2 such as header compression, which minimizes bandwidth usage, and request multiplexing, enabling multiple requests to be sent concurrently over a single connection. In addition to HTTP/2, gRPC utilizes Protocol Buffers (protobuf) as the Interface Definition Language (IDL) and wire format. Protobuf is a compact, high-performance, and language-neutral mechanism for data serialization. This makes it preferable over the more dynamic, but more verbose and slower JSON format often used in REST APIs.

Redis

Redis (Remote Dictionary Server) is an in-memory data structure store, primarily utilized as an ephemeral database, cache, and message broker introduced by Salvatore Sanfilippo in 2009. Compared to other key-value stores and NoSQL databases, Redis supports a multitude of data structures, including lists, sets, hashes, and bitmaps, making it a good choice for caching or storing data that does not fit well into a traditional SQL architecture[33].

One of the primary reasons for Redis's speed is its reliance on in-memory data storage rather than on disk, enabling very low-latency reads and writes. While the primary usecase of Redis is in in-memory operations, it also supports persistence by flushing data to disk. This feature broadens the use cases for Redis, allowing it to handle applications that require longer-term data storage in addition to a caching mechanism. In addition to it being mostly in-memory, Redis also supports quick concurrent reads/writes thanks to its non-blocking I/O model, making it a good

S3 and Minio

S3 is a scalable object storage service, especially designed for large-scale applications with frequent reads and writes. It is one of the prominent services offered by Amazon Web Services. S3's design allows for global distribution, which means the data can be stored across multiple geographically diverse servers. This permits fast access times from virtually any location on the globe, crucial for globally distributed services or applications with users spread across different continents.

S3 offers a variety of storage classes for to different needs, i.e. for whether the requirement is for frequent data access, infrequent data retrieval, or long-term archival. This ensures that it can meet a wide array of demands through the same API. S3 also comes equipped with comprehensive security features, including authentication and authorization mechanisms.

Communication with S3 is done through a HTTP API. Users and applications can interact with the stored data - including files and folders

Apache Cassandra is a wide-column NoSQL database tailored for large-scale, distributed data management tasks. It blends the distributed nature of Amazon's Dynamo model with the structure of Google's Bigtable model, leading to a highly available database system. It is known for its scalability, designed to handle vast amounts of data spread across numerous servers. Unique to Cassandra is the absence of a single point of failure, thus ensuring continuous availability and robustness, which is critical for systems requiring high uptime.

Cassandra's consistency model is tunable according to needs, ranging from eventual to strong consistency. It distinguishes itself by not employing master nodes due to its usage of a peer-to-peer protocol and a distributed hash ring design. These design choices eradicate the bottleneck and failure risks associated with master nodes[37].

Despite these robust capabilities, Cassandra does come with certain