The Ares System

Drew DeVault

June 1, 2022

1 Introduction

The Ares Operating System is an operating system under development which aims to provide a robust and reliable general-purpose system. This is achieved with a focus on simplicity and the composition of discrete components, each with a carefully considered scope, many of which have general-purpose applicability beyond the Ares system itself.

The major components of the Ares system are:

- **Helios**: A micro-kernel with capability-based security and minimal services.
- **Mercury**: Low-level userspace service bus and driver framework.
- **Venus**: A collection of hardware drivers for Mercury.
- **Gaia**: The Ares high-level programming environment & API.
- **Luna**: A POSIX compatibility subsystem for Gaia.
- **Ares**: The Ares desktop software distribution and "super-system".

The ultimate goal of the Ares system is to provide a reliable general-purpose operating system for both end-users and service providers. Its novel approach to networking will combine all of the typical user's devices into a single system, secure, distributed, and unconcerned with restrictions such as NAT and network censorship. It will enable users to build a distributed computing system which accommodates their own needs through personal digital sovereignty, as well as to enable small-to-medium service providers to offer highly integrated services to provide CPU, storage, and other resources to users who prefer a more managed experience.

The Ares system is a great undertaking that, at this time, remains largely aspirational. However, over the past three years, the system's programming language, Hare, was developed from scratch and has reached a sufficient level of maturity to facilitate the development of the Ares system. The micro-kernel, Helios, is also under active development. The remainder of the system is in the design and conception phase.

As this document describes higher- and higher-level components, it becomes more vague. Many details of components like Gaia and Ares are less novel and innovative, drawing from existing operating system designs, and these matters are not covered by this document. To avoid over-designing the system, these details will be made more concrete only when the lower-level components which facilitate them are ready to be built upon.

2 Scope and ambitions

This paper lays out a wildly ambitious system which will take many years to complete and is likely to fail. It will be worthwhile nevertheless.

Ares will cultivate a new generation of programmers with a set of waning skills in areas like kernel hacking and driver development. The team leading up this work has an extensive established background in many relevant technologies, from language and compiler design to kernel hacking and graphics.

Ares will drive research and development in many generalizable domains. It will validate and improve the design of the Hare programming language. The Helios and Mercury systems together are small in scope, but useful on their own merits even without the rest of Ares. The debugger is novel and will be portable to other systems and languages, and will drive the development of features like DWARF in Hare. The GUI toolkit designed for Ares will also be portable and inject some much-needed innovation into open source GUI toolkits.

The successful execution of this project will produce a useful system early in its development and set it up so that the innovations justified by this system will build towards a complete solution. The development of this system will produce useful innovations early, even if the complete result will take a long time and a lot of investment.

3 The Helios microkernel

The Helios microkernel is at the heart of the Ares system. The kernel design is inspired by seL4, but with a greater focus on practicality. It relies on capability-based security and provides a small set of services to userspace, including:

- System initialization
- Address space management
- Capability management
- Process scheduling
- Inter-process communication
- Interrupts
- Hardware I/O

These primitives are provided to the initial userspace process, which is then responsible for all policy related to the delegation of any of these rights via the capability system. Similar to seL4, the kernel does not allocate resources for itself, and has highly predictable runtime behavior.

3.1 Security

Capability-based security is based on the use of unforgeable tokens, enforced by the kernel, which entitle the bearer to certain rights. These tokens can be delegated to other processes, or derived from to produce a capability tree. The administrator can take advantage of this, with careful planning, to produce a hierarchy with any desired level of sophistication or isolation, providing any required degree of isolation without resorting to virtual machines or complex containerization systems.

The kernel's capability-based approach to security is significantly more reliable than approaches favored by other systems. Unix-like systems such as Linux rely on Unix file system permissions, groups, and file modes, to limit the creation of file descriptors. Modern Unix-like systems have essentially converted file descriptors into capabilities. However, this system was developed from the top-down, rather than the bottom-up. It is a frequent source of errors and the APIs designed around them are cumbersome and ill-suited to this design.

Capability-based systems such as seL4, as well as higher-level systems, have demonstrated capabilities as the superior approach to system security. Helios will take advantage of this approach accordingly.

3.2 Portability

The kernel currently targets x86_64. A separate prototype kernel for RISC-V, Carrot, was independently developed in Hare and is planned to be merged into Helios. AArch64 support is planned as well. Additional platforms will first require the development of a qbe backend, so that the Hare compiler can target these platforms. The effort required for such a port is smaller than for most programming systems; RISC-V support was completed by one person over the course of a few months.

Porting the kernel itself to a new platform is relatively straightforward. It is a small program whose platform-specific code is smaller still.

4 The Mercury system

The Mercury system sits on top of Helios and provides two major components for Ares:

- A framework for driver development
- Abstractions for low-level system features
- A low-level service bus

Mercury provides a small set of runtime libraries to support a minimal programming environment, with primitives such as string manipulation and hashing functions. In this respect it provides services similar to libc, though much smaller in scope.

Mercury will also provide a service bus, which will establish IPC primitives between various system components, allowing userspace software to take advantage of services such as networking and file systems. In this respect, it holds a view of the structure of the low-level services available for the operating system as a whole, and provides access to these services to high-level userspace processes.

Finally, the Mercury system will provide abstractions which are implemented by the drivers. It will provide an interface for disk drivers to implement that may be leveraged through a common API, and build on top of these with file system abstractions and a virtual file system. It will also provide implementations of some common features, such as TCP and UDP.

4.1 Application-specific usage and Mercury service composition

The service bus will be designed irrespective of the services which run on it, thus the availability of features such as a virtual file system can be included or omitted at the discretion of the administrator, or novel features may be included as appropriate. Thus, it will be possible to build application-specific systems on top of Mercury which omit the remainder of the Ares architecture.

Such systems have a smaller attack surface, and less complexity, and thus poses more reliability. Special-purpose computers, such as network appliances, robotics applications, and embedded use-cases, may be designed on top of this platform. There is little need for a firewall to have user logins, system consoles, or graphics drivers – let alone a graphics subsystem. Likewise, a device for monitoring machine health at a factory might transmit UDP packets and read data from GPIO, but has little need for anything else. The most secure and reliable code is code that is not there.

5 The Venus driver collection

Venus is a collection of free software driver implementations built on top of Mercury. It is technically unbounded in scope; any driver is welcome for any class of hardware, and there is no shortage of hardware designs to accommodate.

However, in practice, we do not expect to have the labor pool necessary to support the development of a large number of drivers. We intend to focus instead on the drivers which are necessary to support our own use-cases, with specific hardware in mind, and anticipate that other groups with an interest in Venus will do the same. It is expected that the following drivers will be developed at a minimum:

- Most hardware emulated by QEMU
- Disk drivers (SATA, NVMe, etc)
- Various Ethernet drivers
- Intel HD graphics, Intel HD audio
- FAT & ext4 file system drivers
- Various simple drivers: clocks, serial, etc

Some generalizable components (e.g. PCIe support) will call for the development of Venusian abstractions which are out of scope for Mercury.

Graphics are an area with a great deal of depth and complexity in drivers. It is likely that, at first, the goal of graphics drivers in Venus will be to provide simple frame-buffers, omitting hardware accelerated or 3D graphics. Within these constraints, it may be feasible to develop additional graphics drivers with a small team, such as for AMD GPUs. It may also be possible to port some drivers from other systems, independently from Venus, such as Linux's KMS/DRM stack and Mesa's OpenGL and Vulkan implementations. The development of a more sophisticated graphics stack in Venus would also allow us to build drivers for features such as webcams or hardware-accelerated video encoding and decoding.

Wireless networking drivers and printer drivers (the latter likely permanently limited to printers supporting the Internet Printing Protocol) are additional "stretch" goals for Venus. Many of these drivers will require non-free firmware components to work; replacing these is out-of-scope for Venus.

Importantly, though Venus is presented as a single collection of drivers, the microkernel design allows for the user to pick and choose only the drivers which are applicable to their hardware. Compare this with Linux, where most kernel configurations are shipped with thousands of drivers which are loaded as modules in ring 0. Venusian drivers run in userspace and derive their rights from narrow capabilities, preventing poorly scrutinized drivers or obscure code-paths from leading to a larger compromised system. The worst thing that a floppy disc driver with a vulnerability could do is erase your floppy discs; on Linux it could compromise your entire system.

6 The Gaia programming environment

Gaia will provide a general purpose computing environment for the Ares system. It will include the following components:

- A port of the Hare standard library
- A virtual file system for accessing device drivers
- System utilities (shell, core utilities, etc)
- A multi-user environment; logins and session management
- A high-level service manager (like OpenRC or systemd)
- Programming tools (compiler, debugger, etc)

Gaia will draw much of its inspiration from Plan 9. Many of the design details of Gaia are subject to change: it is the primary means by which most programmers will interact with Ares, and as such this component will be the focus of a great deal of research and development.

6.1 The virtual file system

Gaia will introduce concepts similar to Unix users and file ownership semantics, and will provide access to lower-level system components through virtual file systems analogous to /dev, /proc, and /sys on Linux. Unlike Linux and other modern Unix-likes such as *BSD, the implementation of these files will draw more inspiration from Plan 9, facilitating their features through text-oriented APIs rather than ioctls. Also similar to Plan 9, Gaia will provide per-process file system namespaces and bind & union mounts to create a view of the system which includes only what is appropriate to each process. This approach provides strong isolation without requiring containerization, and offers design improvements for features like VPNs and display servers.

6.2 The Gaia shell & system tools

Gaia will also provide a scriptable interactive command shell, drawing design inspiration from Plan 9's rc. This design is much simpler than POSIX shell or its derivatives (such as bash), and also allows the user to write more robust scripts without concerning themselves with matters such as proper use of quoting.

The system utilities will diverge somewhat from Plan 9. The suite of POSIX utilities such as "ls" are reasonably well-designed, but exceed the appropriate scope. Most Gaia commands will be compatible with a subset of the POSIX specification. However, no deference to the standards will be made for the sake of compatibility alone; for example we will ship "tar" rather than "pax".

6.3 Multi-user environment

Gaia will provide a multi-user environment for the Ares system. It is responsible for user management in a manner similar to Unix, as well as handling user log-ins and session management. It will enforce isolation between users by providing different views of the file system, each user's personal data being encrypted at rest and isolated at runtime with capabilities.

These systems will be designed to be extended upon by higher-level components. Gaia will provide a simple login system analogous to "getty" on Unix, but will also provide a means for Ares components to offer different approaches to session instantiation, such as a remote access system or a graphical login manager.

6.4 Service management

The Gaia service manager will work similarly to service managers on other operating systems. It will run a set of services on start-up or on-demand, on a system-wide and per-user basis. It will offer the ability to configure the capability tree and resource allocations for these services. The service manager will also act as a process supervisor, automatically restarting faulting services according to system policy. It will also provide a centralized logging system, which will be available over the network and can aggregate logs over a larger network.

The service manager may be redundant with the Mercury service bus; these components could ultimately be merged.

6.5 Programming tools

Gaia will provide tools to help programmers develop software for the system, as well as for planning and characterizing the behavior of their software at runtime. One approach will take the form of observability tools, which will allow the programmer to characterize their software's resource utilization and other metrics of interest. This system will also allow for remote monitoring for service providers. In essence, it will provide the functionality offered by Unix tools such as Prometheus, Alertmanager, Jaeger, and DTrace.

The Gaia system will also provide a debugger. The design of this debugger will be inspired mainly by that of Plan 9's acid. This design calls for the development of a domain-specific debugging language, whose primitives can be used to build higher-level components such as an interactive debugger. We intend to generalize this approach further than Plan 9 to facilitate the other features mentioned above, such as DTrace-like functionality, as components built on top of the debugging system.

6.6 Luna: POSIX support

POSIX was initially published in 1988 and has maintained backwards compatibility ever since; a laudable result when considered with its goals. However, many of its drawbacks remain as persistent sore points in Unix environments, and POSIX does not provide a framework for overcoming these issues. We wish to move on from the legacy design of POSIX with Gaia.

This said, we must acknowledge that the vast body of software written today is written for POSIX systems. To provide users a means of using this body of software, the optional Luna system will offer an alternative interface to Gaia which is, for the most part, POSIX compatible. The Luna system will be developed with a focus on facilitating the use of existing software, but not from a desire to achieve perfect POSIX compatibility. We will not shy away from patching existing software to work around the limitations of Luna.

7 The Ares system

8 Comparison with other Operating Systems

A few words regarding Ares in comparison to other operating systems. Note that proprietary opreating systems are not regarded as worthy of comment.

seL4: Helios takes the bulk of its inspiration from seL4. The Helios API and kernel design is very similar to seL4. However, in contrast to seL4, Helios takes less of an academic focus and establishes more practical priorities. This is facilitated in part by the fact that Helios development is grounded in the context of the Ares system as a whole. Its purpose is to facilitate the development of the Mercury system, to support a driver ecosystem for Venus, and so on. These goals will re-enforce the necessity for a generally and practically useful system to a degree which exceeds the scope of seL4. And, though we hold seL4 in high regard and owe much of our inspiration to it, we respectfully believe that we can produce a better implementation.

Minix: Even moreso than seL4, Minix has priorities which place academic research ahead of practical utility. Additionally, Minix is designed to be a Unix-like system, and its security model reflects this. We believe that there is a future beyond POSIX, and Minix is not working towards that future. Additionally, again largely due to its academic nature, there is little momentum to advance Minix's implementation in areas such as USB support, graphics support, or additional ports such as 64-bit targets. However, we are looking forward to Minix 4.

Linux: Linux is the most sophisticated operating system available today. We do not anticipate supplanting it, and we cannot muster anywhere near the resources Linux has at its disposal. A consequence is that we will prefer to choose hardware which is suitable for our needs, rather than trying to support as much hardware as possible. However, though Linux enjoys a rich labor pool, it is difficult for all of the interests to communicate effectively across the whole project, causing it to lack a cohesive system-wide design. Linux also does not have aspirations in userspace, and thus many competing userspace implementations exist, leading to market fragmentation and a division of labor among redundant projects. This is particularly inefficient in areas where there is no commercial entity underwriting development, as volunteer resources are spread thin on Linux.