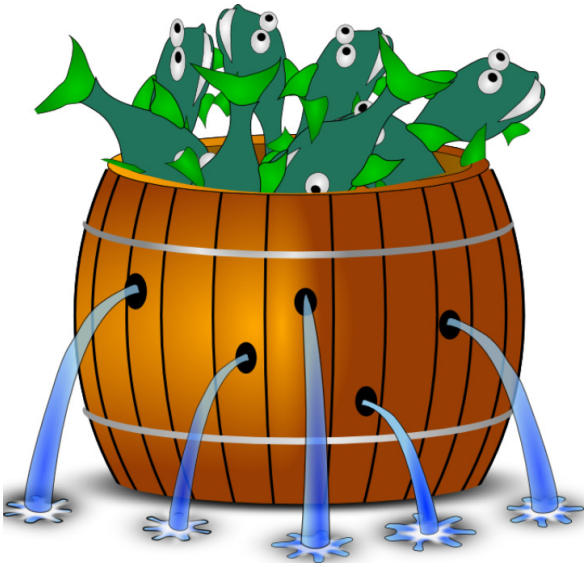
Barrelfish Project
ETH Zurich



Barrelfish OS Services

Barrelfish Technical Note 012

Team Barrelfish

20.08.2010

Systems Group
Department of Computer Science
ETH Zurich
CAB F.79, Universitätstrasse 6, Zurich 8092, Switzerland
`http://www.barrelfish.org/`

Revision History

Revision	Date	Author(s)	Description
1.0	20.08.2010	kuzi	Initial version

Chapter 1

Barrelfish OS Services

1.1 Introduction

Barrelfish is a multi-server OS, which means that any OS based on Barrelfish will consist of a collection of mutually dependent services. Each of these services provides some functionality required for the OS.

In Barrelfish, a service can be implemented as a single-dispatcher domain, a multi-dispatcher domain, multiple domains, one or more libraries, or some combination of these.

In this document we review the services that will make up the base building blocks of a Barrelfish OS. After introducing the services, we discuss dependencies between the services, and then the current implementation status of each service.

1.2 Services

The specific services we've identified are (presented in alphabetical order):

- Binding
- Capability Management
- Debugging
- Device Driver
- Device Management
- Environment
- Group Communication
- Locking
- Memory Allocation
- Naming
- Network Stack
- Power Management
- Principals
- Process Management
- Resource Accounting and Management

- Routing
- Shell
- SKB
- Storage Management
- Terminal
- Threading
- Tracing
- Virtual Machine Monitor
- Virtual Memory

1.2.1 Binding

The binding service facilitates communication in the system, allowing dispatchers to export interfaces through endpoints, and enabling other threads of execution to connect to dispatchers through endpoints. The binding service also implements the inter-dispatcher communication mechanisms allowing threads to send messages over connections.

1.2.2 Capability Management

The capability management service provides the functionality to manage capabilities outside the kernel. It provides the ability to allocate space for and create new capabilities, as well as to manipulate them by moving, copying, and destroying them. It also implements the functionality necessary to transfer capabilities between cores, and to manage and invoke remote capabilities.

1.2.3 Debugging

The debugging service provides the functionality to access and manipulate system memory, and control execution of the system. It may provide these services at a minimal level, or provide higher level services that allow control of system abstractions such as capabilities, dipatchers, domains, and threads (for example by attaching to and controlling specific threads).

1.2.4 Device Driver

A device driver service provides access and control over a specific device. A device driver manages access to device hardware as well as handling interrupts and other device events. The devices managed by device drivers inlcude IO devices, as well as internal devices such as busses, timers, etc.

1.2.5 Device Management

The device management service stores and manages information about devices in the system. Its responsibilities include collecting such information, making it available to others, and implementing policy relating to device initialisation, startup, shutdown, power management, etc.

1.2.6 Environment

The environment service provides an execution environment to threads of execution. Examples of this are environment variables.

1.2.7 Group Communication

The group communication service provides facilities for multicast commuincation within groups. It will provide group management, as well as management of delivery trees, and the mechanisms for message delivery and reception (including impementation of necessary consensus protocols).

1.2.8 Locking

The locking service provides facilities and primitives for mutual exclusion and synchronisation. This includes implementation of locks, condition variables, semaphores, etc. It also involves managing excution threads to block and wake them as necessary.

1.2.9 Memory Allocation

The memory allocation service manages physical memory (using capabilities). It processes requests for memory, allocating appropriate memory, and freeing it when necessary. It tracks which memory is in use, and which memory is free and can be reused.

1.2.10 Naming

The naming service provides a name space and a name/value mapping database that allows entries to be added or retreived. It is largely used in communication for finding registered services. It also supports notification of new entries and changed entries, and a blocking lookup.

1.2.11 Network Stack

The network stack implements network protocols. It works together with a networking device. It may provide further surfaces such as packet filtering, packet instpection, etc.

1.2.12 Power Management

The power management services monitors power usage in the system, reacts to power-related events, and can manipulate system resources to control power usage. It also manages system sleep and wakeup, and can manipulate subsystems and hardware according to its policy.

1.2.13 Principals

The principals service provides the concept of principals in the system (e.g., accounts, users, process ids, etc.). The principals are used for access control, that is, access control policies can be defined in terms of principals' access to objects. This service can also provide authentication functionalities, to authenticate principals' identities, as well as management of principles (e.g., account control).

1.2.14 Process Management

The process management service provides functionality to create, manipulate, and manage dispatchers and domains. It may provide a conventional process abstraction based on the Barrelfish primitives, or deal with the primitives themselves. The functionality provided includes creating, starting, stopping, and pausing processes, as well as mechanisms for loading processs, and migrating processes. Part of process creation and management involves setting up and managing address spaces.

1.2.15 Resource Accounting and Management

The resource accounting and management service provides mechanisms to monitor resource usage, keep accounting, and manipulate resource usage. Resources include basic resources such as CPU, memory, and IRQs, as well as OS resources such as devices, files, network bandwidth, etc. Manipulating resource usage includes scheduling processes, scheduling IO operations, throttling network traffic, etc.

1.2.16 Routing

The routing service supports communication between entities that are not directly connected. The service is responsible for determining (intra-machine) routes between the entities, as well as sending the messages over the communication network, and reserving sufficient resources over the routes to be able to provide performance and reliability guarantees.

1.2.17 Shell

The shell service provides a command interpreter and task launcher. It provides an interface for dynamically starting, stopping, and manipulating system tasks. Note that this is not limited to a traditional command-line console.

1.2.18 SKB

The system knowledge base service provides a database for storing information about the system (including available hardware, properties of the hardware, configuration, running services, etc.). It also provides an interface for querying the database, as well as for running complex analysis (e.g., constraint solving) of the data stored within.

1.2.19 Storage Management

The storage management service manages the system's storage infrastructure and provides abstractions for persistent storage. This could be in the form of a file system, a database, or something completely different.

1.2.20 Terminal

The terminal service provides a terminal abstraction that provides programs with input and output. A terminal typically multiplexes output from different programs to a single output destination and demultiplexes a single source of input among multiple programs. The terminal service implements the terminal abstraction, and provides mechanisms for multiplexing multiple terminals.

1.2.21 Threading

The threading service provides a thread model and thread manipulation functionality. This allows thread manipulation (creation, destruction, suspend, wakeup, etc.), thread scheduling and thread local storage.

1.2.22 Tracing

The tracing service collects tracing information and makes it available to analysis and display applications. The service allows running entities in the system to log traces with it as well as to query it about these traces. Such a service can be used for logging system events, debugging information, error events, etc. The tracing service can also actively monitor the system, and dynamically query system entities based on queries. It can also provide notification functionality to notify entities when given events occur.

1.2.23 Virtual Machine Monitor

The virtualisation service provides a virtual machine monitor that enables guest software (typically an OS) to be run on virtual hardware. The service manages the loading and unloading of the guest software, manages emulation of sensitive and privileged instructions and hardware, and manages the virtualised memory.

1.2.24 Virtual Memory

The virtual memory management service manages the system's virtual memory (by manipulating Barrelfish vnodes and memory caps). It provides functionality to map and unmap pages, manage mapping of device memory, and the creation and manipulation of address spaces. It is also responsible for paging and swapping.

1.3 Service Categories

The above services can be grouped into two categories, depending on what they are used for in the system. Roughly, we distinguish between services that provide key Barrelfish functionality, and services that provide more general OS functionality.

1.3.1 Fundamental Barrelfish Services

- Binding
- Capability Management
- Group Communication
- Naming
- Process Management
- Routing
- SKB

1.3.2 General OS Services

- Debugging
- Device Driver
- Device Management
- Environment
- Locking
- Memory Allocation
- Network Stack
- Power Management
- Principals
- Resource Accounting and Management
- Shell
- Storage Management
- Terminal
- Threading
- Tracing
- Virtual Machine Monitor
- Virtual Memory

Chapter 2

Service Dependencies

The above services are system building blocks and do not operate in isolation but make use of each other's services. This leads to dependencies between the various services.

We distinguish between several types of dependencies.

- Fundamental dependencies: dependencies that exist due to the service's required functionality (e.g., network stack relies on the ethernet device driver).
- Implementation dependencies: dependencies that exist due to implementation details. This can be further subdivided into:
  - dependencies due to being implemented as communicating Barrelfish dispatchers (e.g., network stack relies on binding because it needs to communicate with the ethernet device driver).
  - dependencies due to specific implementation decisions (e.g., network stack using the locking service because it is multithreaded).

2.1 Dependencies

All the dependencies between the specific services are shown in Figure 2.1.

The subset of fundamental dependencies are shown in Figure 2.2. Likewise the subset of implementation dependencies are shown in Figures 2.3 and 2.4.

Discussion of these dependencies is presented below.

2.1.1 Binding

All services have a Barrelfish-specific dependency on the binding service, since, if they are implemented as Barrelfish dispatchers, they will require the binding service to make their interfaces available to others, as well as to connect to and communicate with other services.

The binding service itself requires endpoint capabilities for the communication and therefore has a dependency on the capability management service. It also needs to know about the cores and hardware support for communication available in the system, and therefore has a dependency on the SKB, which provides such information.

The binding service, besides providing poin-to-point connections, also provides access to the group communication service and the routing service for more complex communication, and therefore has a

dependency on both these services. Communication also involves notifying and scheduling dispatchers when messages arrive, so the binding service must also make use of the process management service.

Finally, the implmentation of the binding service needs to set up shared memory areas for the implementation of UMP, so needs to use the virtual memory service. Its impementation may also make use of locking and synchronisation primitives, leading to a potential dependency on the locking service.

2.1.2 Capability Management

Managing capabilities requires access to basic memory capabilities, which are provided through the memory allocation service, leading to a dependency on that service. This service will also depend on the virtual memory management service, since it will require control of memory for management of metadata.

2.1.3 Debugging

The debugging service needs access to lower level services in order to inspect and modify the system. This includes: capability management for access to system capabilities, threading to attach to and control threads, process management to attach to and control processes (dispatchers and domains), virtual memory service to inspect and modify address spaces, memory mappings, etc. The debugging service will also use the SKB service to provide further information about the system and its resources.

2.1.4 Device Drivers

Device drivers depend on the capability management service since they need to manage and use capabilities to the hardware devices. Drivers also use the memory allocation service since they need to manipulate memory, for example, to use DMA and access device memory. This also requires the use of the virtual memory service.

Since drivers must react to system power events (e.g., to turn devices on and off when the system suspends) they may also depend on the power management service.

2.1.5 Device Management

The device management service works closely with the SKB since it needs to find out (and register) information about devices in the system. It may also depend on the power management service to determine current power state and possibly inform drivers of changes in the state.

2.1.6 Environment

The environment service may use the principals service if provides per-principal environments, and the process management service if it provides per-process environements.

2.1.7 Group Communication

The group communication service depends on the binding service to provide a frontend interface, as well as to send individual messages. The group communication service will also depend on the SKB to find out information about the system (e.g., interconnect toplogy) to help implement efficient message delivery (e.g., multicast trees).

Group communication is typically associated with consensus protocols, and may therefore rely on a locking services. Alternatively, the group communication service may be used by the locking service to implement some synchronisation.

The service may require endpoints to implement the underlying communication, and will then be dependent on the capability management service. The service will typically also make use of the routing service to send messages to group members.

2.1.8 Locking

The locking service interacts closely with the process management and threading services since it must be able to suspend and resume execution to correspond with blocking and signalling.

The locking service may depend on the group communication service to implement consensus and other forms of synchronisation.

2.1.9 Memory Allocation

The memory allocation service requires capabilities to do memory management in Barrelfish, so has a dependency on the capabiity management service.

It may also depend on the SKB to find out about all the memory resources in the system, and it may also act as a client of the resource accounting and management service to provide it with information about memory usage.

2.1.10 Naming

All services have a Barrelfish-specific dependency on the name service, since, if they are implemented as Barrelfish dispatchers then they will either need to register with it, or use it to find other services that they depend on.

Depending on its implementation the naming service may use the group communication service.

2.1.11 Network Stack

The network stack service requires access to NIC drivers to access the network hardware. It may also need to use the SKB to determine what network hardware and drivers are available in the system.

Depending on implementation details, the network stack may also need to use the virtual memory service to implement zero-copy mechanisms, etc.

2.1.12 Power Management

The power management service requires access to device drivers and the device management service to monitor and control the power states of hardware. It will also need to use the SKB to determine what devices to manage in the system and what power management hardware is available.

The service may also need access to the the resource management service to help make power management decisions. The power management service may also need to manipulate capabilities for power management hardware, in which case it will need access to the capability management service.

Depending on its implementation it may also use the group communication and locking services.

2.1.13 Principals

In Barrelfish the principals service will make use of capabilities to implement access control, and will therefore require access to the capability management service.

2.1.14 Process Management

The process management service will need access to the capability management and virtual memory services in order to prepare and manipulate process address spaces for execution. It will also work closely with the threading service, to manage execution, and the principals service, to provide processes with the appropriate caps for access control.

The service may use the group communication service depending on how it is implemented.

2.1.15 Resource Accounting and Management

The resource accounting and management service will use the various other management and accounting services (power management, process management, bus management, drivers, storage management) to determine the state of the system and resource usage in the system. It will also use these services to change resource usage according to policy.

It will also both read from and write to the SKB.

In order to manage resource usage the resource management service may need to manipulate caps, in which case it will use the capability management service.

Depending on implementation it may also use the group communication and locking services.

2.1.16 Routing

The routing service will require endpoint capabilities for communication and therefore has a dependency on the capability management service. It also needs to know about the cores and hardware support for communication available in the system, and therefore has a dependency on the SKB, which provides such information.

The routing service will both make use of the binding service for its point-to-point connections, as well as be accessed through the binding service, and thus has a strong dependency to it.

The routing service will use the SKB to determine the system topology.

It will likely also be implemented using locking and group communication.

2.1.17 Shell

The shell service requires access to process management and threading to start and control programs, it also requires access to the environment service to provide programs with appropriate execution environments, and allow manipulation of those environments. It uses the principals service to determine and set up access control for the programs it starts. The storage management service is used to fetch the code to execute, and the terminal service is used for input and output.

2.1.18 SKB

The SKB relies on the storage management service to read its startup files, and possibly store its knowledge base.

2.1.19 Storage Management

The storage management service needs access to appropriate storage hardware (e.g., disk).

It will also use the virtual memory service if it implements zero-copy features and needs to manipulate shared memory buffers. It may need to use the SKB to determine what devices are available.

The service may also require access to the principals service for access control, however this depends on the chosen design.

Depending on implementation it may also use the using locking and group communication services.

2.1.20 Terminal

The terminal service will need to use drivers to access the input and output hardware.

Depending on implementation it may use the virtual memory and capability management services to provide access through a framebuffer.

The implementation will likely use locking, and may require the SKB to determine what interaction devices are available.

2.1.21 Threading

The threading service closely cooperates with the process management service. It also needs to manage the memory accessible by threads and uses the capability management and virtual memory management services for this.

It may also cooperate closely with the locking service to suspend and resume threads as part of the locking mechanism implemententations.

2.1.22 Tracing

The tracing service is not directly dependent on other services (other than binding and naming).

Any service may use the tracing service to log trace points. We have not represented these dependencies explicitly.

2.1.23 Virtual Machine Monitor

The virtual machine monitor service uses the capability management, memory allocation, virtual memory management, and process management services to provide a virtual machine environment to a guest OS. It also uses the driver and device menagement services to provide access to hardware devices.

It may also use the storage managment and network stack services to virtualise disk and network traffic. The power management service may be used to allow power management of and by the virtual machine. The SKB may be used to determine which system resources to provide to the virtual machine.

Its implementation is likely to use locking.

2.1.24 Virtual Memory

The virtual memory service uses the capability management and memory allocation services to help manage the virtual memory mappings and address spaces. The SKB is used to determine the available memory resources in the system.

Paging will use an appropriate disk driver, and may require access to a storage management service.

The implementation of virtual memory management is likely to use locking and group communication.

2.2 Status

Some of the services discussed above have already been implemented. In many cases the implementation has been a simple prototype implementation, where the basic functionality has been implemented, but without regard to issues such as scalability, and also, without attempting to dynamically configure or alter functionality based on information about the system gathered at runtime (e.g., from the SKB).

In this section we provide an overview of the current status of each service. A service's status can be:

N : Does not exist

P : Exists as a prototype

E : Fully featured implementation

D : Scalable, distributed, implementation

We also provide an overview of the development priority of services. The priorities relate to the goal of developing a complete OS and an environment and tools for developing complete OSes. The priorities are:

High : This service must exist before more of the system can be built, or it is a bottleneck and must be implemented in a scalable way. It is a core building block.

Medium : Some services rely on this, so it has some priority. Availability of the service may also help the usability of the system. It is a useful building block.

Low : Hardly any services depend on this, so lack of it will not stand in the way of developing a system.

The status of the services is as follows:

Service	Status	Priority	Comment
Binding	E	High	implemented by combination of monitor, barrelfish library, and flounder stubs.
Capability Management	P	High	implemented by combination of monitor and barrelfish library. The revoke fnctionality is not implemented.
Debugging	N	Low	
Device Driver	E	Medium	existing drivers: serial, pci, e1000
Device Management	P	Medium	implemented by pci driver and skb
Environment	P	Low	implemented by barrelfish library and spawnd dispatcher
Group Communication	N	Medium	
Locking	P	High	??? how is locking/synchronisation currently done???
Memory Allocation	P	High	implemented by memory server domain. No ability to free or reuse memory.
Naming	E	High	implemented by chips domain.
Network Stack	P	Medium	implemented as library. can be used by only 1 dispatcher at a time. It is based on LWIP.
Power Management	N	Low	
Principals	N	Medium	
Process Management	P	High	implemented by combination of spawnd domain, monitor, and barrelfish library.
Resource Accounting and Management	N	Low	
Routing	N	Medium	
Shell	P	Low	implemented by fish domain.
SKB	E	Medium	implemented by skb domain.
Storage Management	P	Medium	implemented by combination of ramfsd domain, and vfs library.
Terminal	N	Medium	there is some code to handle this in lib barrelfish.
Threading	E	High	implemented by barrelfish library.
Tracing	E	Medium	implemented by tracing library.
Virtual Machine Monitor	P	Medium	implemented by vmmkit domain.
Virtual Memory	P	High	implemented by monitor and barrelfish library (???). No support for paging, unmap, or fault handling.

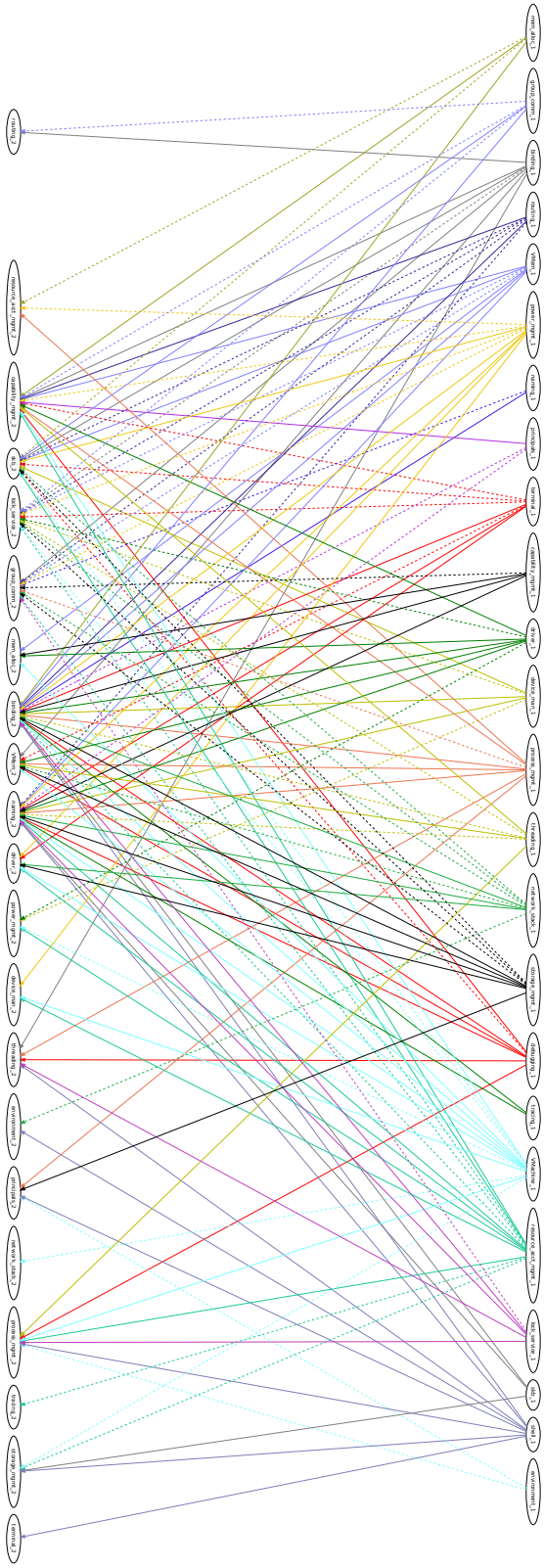


Figure 2.1: Use dependencies between OS services. Dashed lines indicate dependencies we are uncertain about.

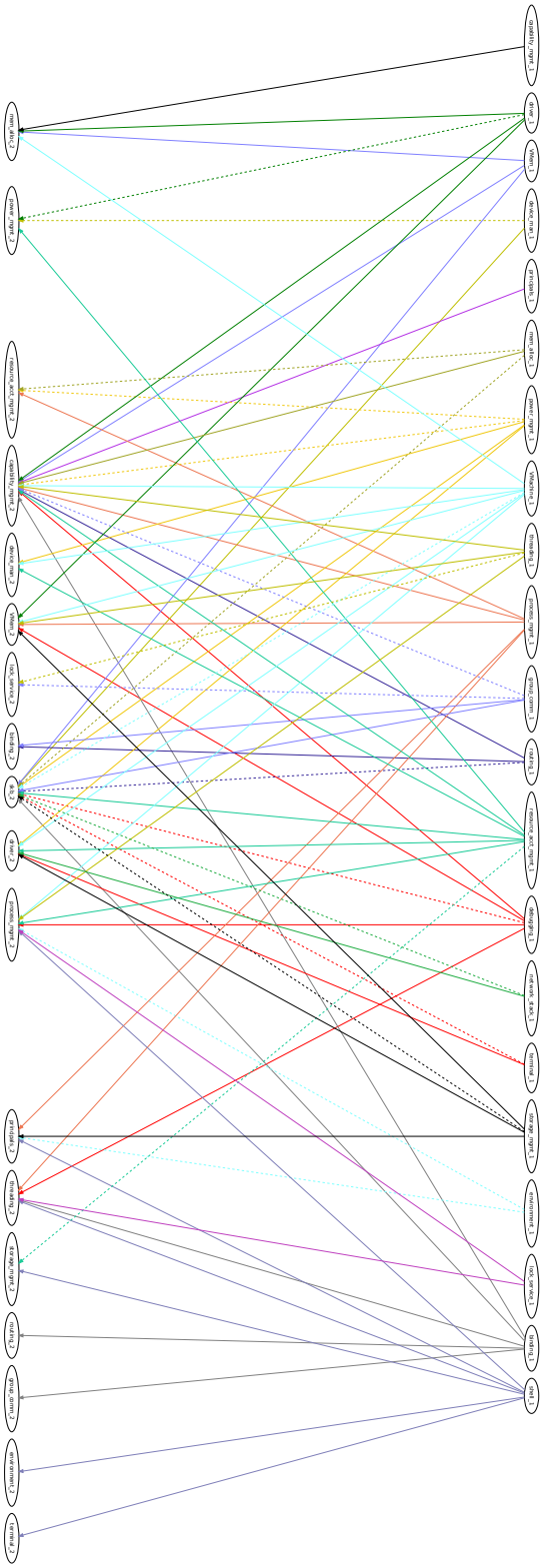


Figure 2.2: Subset of fundamental use dependencies between OS services.

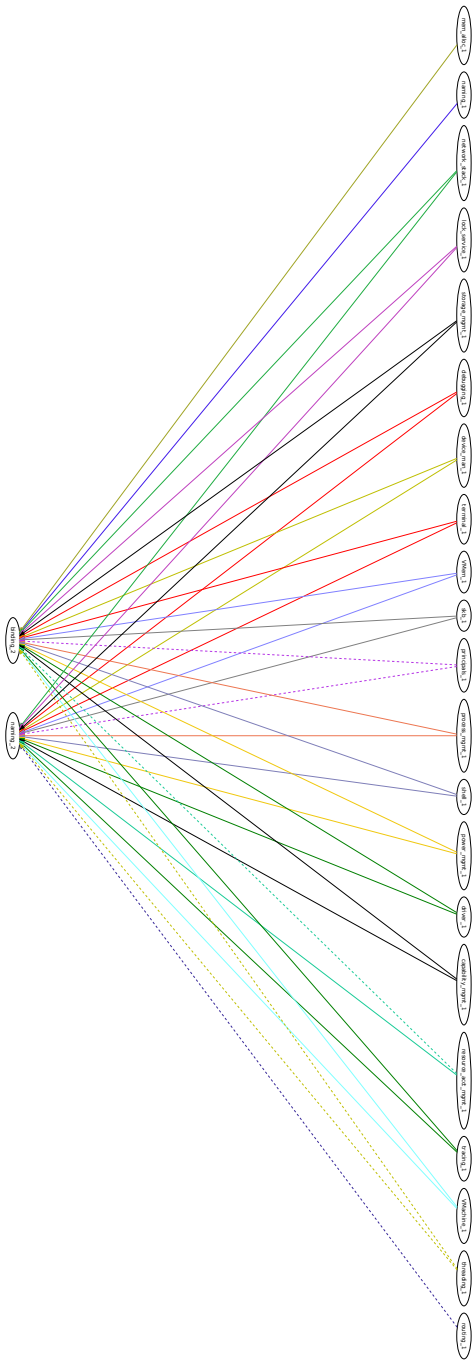


Figure 2.3: Subset of barrelfish-implementation specific use dependencies between OS services.

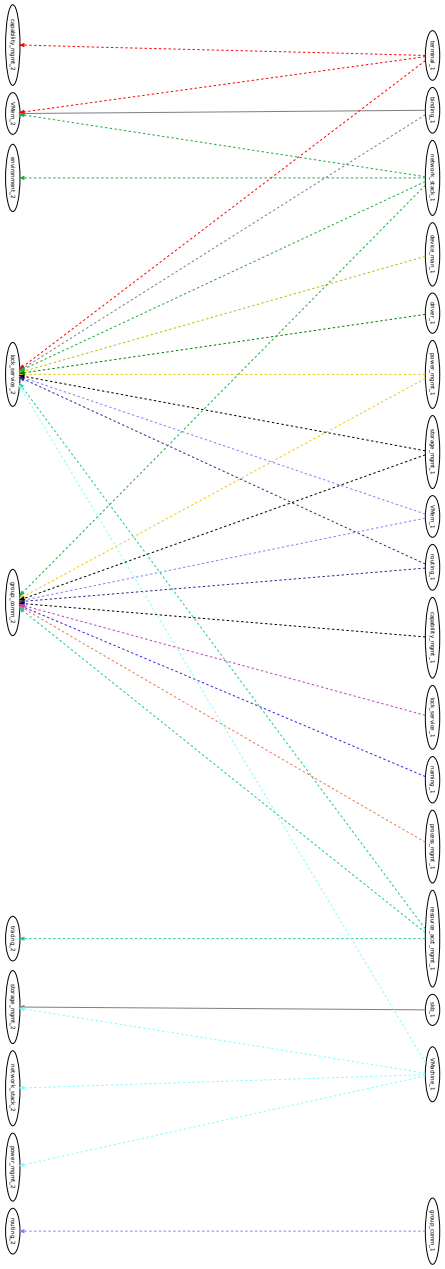


Figure 2.4: Subset of non-barrelfish implementation dependencies between OS services