FAQ for Adaptive Partitioning - Memory

Aka APM

Memory Partitioning and the related documentation, including this FAQ, is based on a prototype of the feature by engineering. It is not currently available for release. If you have an interest in this feature and would like to see it as part of a future product offering, please contact your sales representative.

Please note that additional detail on most, if not all of the following FAQ's can be answered by reading Memory Partitioning - Design and Architecture Overview.

What is it?#

It's a flexible, per memory class partitioning mechanism that allows for the independent specification of both memory resource guarantees (reservations) and restrictions for each process in a running system.

Memory Partitioning is part of an overall resource partitioning strategy. See Adaptive Partitioning.

What do you mean by a memory class?#

In the context of memory partitioning, a memory class refers specifically to an independently partition-able segment of non-overlapping memory. The segments are established at startup and refer to the syspage asinfo entries (visible with *pidin syspage=asinfo*). The system architect is free to ascribe any attributes or purpose to the memory segments (with the exception of sysram which is the class used for general allocations) as memory partitioning does not impose any such attributes. Once defined in startup, and subsequently added to the partitioning name space, each of these segments is referred to as a class (or type if you prefer) for the purpose of partitioning and allocation/deallocation. The sysram memory class is special, in that it always exists and is used for internal (kernel) allocations as well as unspecified (un-typed) allocations.

Example:

Let's suppose a custom board had some SRAM implemented within an custom FPGA (which likely also had additional functionality which made use of the SRAM). This segment of memory would reside at a particular physical address and would be considered a different class of memory, not only because it truly has different physical attributes (faster, maybe persistent and perhaps smaller than the DRAM used for general storage) but also because it serves a different purpose within the system. Also suppose that there are several independent processes which all wished to allocate portions of the memory class but that some of these processes were more important than others. Memory partitioning could be used to partition this memory class into several partitions, perhaps of different size and type and allow the processes requiring this memory class to be independently associated with specific partitions. The more important processes could be guaranteed to have access to configured amounts of this memory class independent of any other memory classes it required or the demands of other processes. All of the processes could be "coded" the same way ... to simply allocate memory of the specified class without any of the interprocess coordination that would otherwise be required to ensure the priority of allocation to more important processes.

How do I allocate memory of a different class?#

We (currently) rely on the POSIX typed memory interfaces for this.

How can I use this?#

See the Example above under *What do you mean by a memory class?*

What is the sysram memory class?#

The sysram (system ram) memory class is a permanent memory class which is used for all un-specified (un-typed) memory allocations. When your process does *malloc()*, you are implicitly requesting an allocation of the system ram memory class. See also *What do you mean by a memory class?*

What is the system partition and what is it used for?#

The system partition is a permanent partition of the sysram memory class. Unless otherwise specified, all process will be associated with the system partition. That is, without specifically associating with a different sysram memory class partition, all processes will be associated with the system partition for their sysram memory class (ie. malloc and friends) allocations. Note that although the system partition is only visible when the memory partitioning module is installed, conceptually it always exists. In fact without memory partitioning, you effectively have the equivalent of a single class, single partition system with a *minimum* size of 0 and a *maximum* size of infinity.

What is the difference between the minimum size attribute and the maximum size attribute?#

The *minimum* size attribute establishes a reservation or guarantee for allocations made by processes associated with the partition. The *maximum* size attribute establishes a restriction on allocations made by processes associated with the partition.

What is the minimum size attribute used for?#

When a partition is created/modified, a non-zero *minimum* size attribute indicates that the specified amount of memory for the class should be set aside (reserved) for future allocations by processes associated with the partition. See also *What is reserved memory?*

What is the maximum size attribute used for?#

When a partition is created/modified, the *maximum* size attribute will specify a limit on the amount of memory that can be allocated by all processes associated with the partition. Don't worry, you can specify infinity for this attribute ... but you'll still only get as much memory as is physically available ... but 'ya know what's cool ... if you dynamically expand your physical memory, you'll be able to just use it. See also *What is discretionary memory?*

Why do I need to have both minimum and maximum size attribute?#

Well, although both attributes exist, you can set them to the same value and get the same behaviour as you do with other partitioning schemes ... fixed partition size of no more and no less. That's like having just one attribute. We think our scheme provides more flexibility and is more adaptable to variable processing conditions, like transient memory allocations, and so we allow these attributes to be independently specified.

What is reserved memory?#

Reserved memory is memory of a specific class that has been set aside (in the allocator for that class) during partition creation/modification. It represents a guarantee. In other words, processes associated with a partition with a non-zero *minimum* size attribute are guaranteed successful allocation of *minimum* bytes of memory of that class. It is important to note that this IS NOT a preallocation and therefore does not suffer from any of the side effects of that technique (like potential fragmentation). See also Memory Partitioning - Design and Architecture Overview

What is discretionary memory?#

Discretionary memory is memory that you may or may not have access to, depending on your system wide partition configuration and the behaviour of other processes in the system. It's what you have when you don't use memory partitioning or don't use Neutrino (shame). From a partitioning perspective, it's the numerical difference between the *maximum* size attribute and the *minimum* size attribute. Discretionary memory represents memory that exists beyond the reservations established by all partitions in a system (and it may be configured to be zero) that is available for allocation to any process in any partition that has not yet exceed its *maximum* size limit. See Memory Partitioning - Design and Architecture Overview for a pretty good description of the types of partitions that can be configured and how they might be used.

What about internal allocations like page tables and timers? How do you handle that smart guys?#

Ok, this is a tough one, but we do have an answer. In fact all of the internal (kernel) allocations in the system are subject to the same partitioning rules as for user space allocations. Since most of the internal allocations are being performed on behalf of some process (yes, there are a few which exist that are not accountable to any one process), we simply figure out what partition that the requesting process is associated with, apply the partitioning rules, and either fail or pass the request. In fact procnto (a process) is subject to the same partition rules as any other process for its sysram allocations (which are governed by the system partition).

One more thing. Currently, all internal (kernel object) allocations use the sysram memory class by default so the sysram partition for each process will govern the allocation of the kernel objects it needs (like page tables) ... but in the future, we might even allow you to specify that yourself ;)

It sounds easy but its actually a bit tricky to implement.

... and that's how we help stop you from fork()'ing yourself!

So how do I know which partitions my process will be associated with?#

An POSIX interface is defined which allows a user written application to get all kinds of interesting information on partitions and memory classes, the details of which will be in a forthcoming user document (still being written). Suffice it to say that it won't be much more complicated than open()/devctl()/close(). You could also just do a recursive listing of */proc/self/partition* (ie. *ls -1R /proc/self/partition*) from a shell.

How do I specify which partitions my process will be associated with?#

The *posix_spawn()* interface will be the primary means to control partition association. It has been extended to add this behaviour. Of course, you could also just rely on the inheritance rules, but that's not very interesting. The details will be in a forthcoming user document (still being written) but for now, there is a pretty good description in Memory Partitioning - Design and Architecture Overview. Search for ***posix_spawn()*** and ***inheritance***

Does Memory Partitioning implement inheritance like scheduling partitioning?#

Yes and no. Yes, memory partitioning implements inheritance, but it is a bit different.

In order for a process to execute, it MUST be associated with at least one partition, a partition of the sysram memory class. With no partition associations, it would be like trying to execute software on a computer with no memory installed. So in order to ensure that all programs run, we must ensure that they all have a sysram memory class partition association. This is accomplished through inheritance. The very first process in the system, procnto, will be associated with the system partition. As it creates other processes, these processes will inherit their partition associations from their parent (ie. procnto) and as these processes fork()/spawn(), the situation continues, each child inheriting all of its parent partition associations.

By default, a child process inherits ALL of the partition associations of its parent but because, procnto is (currently) only associated with the system partition, this is the only partition that will have any associations regardless of whether you have configured other partitions of other classes. This is what allows a program to continue to execute unchanged in the presence of memory partitioning. Fortunately, we have provided API's and utilities that allow you to control partition association right from after procnto starts up, right from the build file. You can free the system partition for only procnto/kernel allocations and have every other process in the system associated with a different partition. All of the details will be found in a forthcoming user document (still being written) but for now, there is a pretty good description in Memory Partitioning - Design and Architecture Overview. Search for ***inheritance***

Can memory partitioning and scheduling partitioning be used together?#

Yes!

Can memory partitioning and scheduling partitioning be used separately?#

Yes!

Can I build a kernel without memory partitioning?#

Yes, partitioning software is an optional module.