HugeTLB Pages

Overview

The intent of this file is to give a brief summary of hugetlbpage support in the Linux kernel. This support is built on top of multiple page size support that is provided by most modern architectures. For example, x86 CPUs normally support 4K and 2M (1G if architecturally supported) page sizes, ia64 architecture supports multiple page sizes 4K, 8K, 64K, 256K, 1M, 4M, 16M, 256M and ppc64 supports 4K and 16M. A TLB is a cache of virtual-to-physical translations. Typically this is a very scarce resource on processor. Operating systems try to make best use of limited number of TLB resources. This optimization is more critical now as bigger and bigger physical memories (several GBs) are more readily available.

Users can use the huge page support in Linux kernel by either using the mmap system call or standard SYSV shared memory system calls (shmget, shmat).

First the Linux kernel needs to be built with the CONFIG_HUGETLBFS (present under "File systems") and CONFIG_HUGETLB_PAGE (selected automatically when CONFIG_HUGETLBFS is selected) configuration options.

The `/proc/meminfo` file provides information about the total number of persistent hugetlb pages in the kernel's huge page pool. It also displays default huge page size and information about the number of free, reserved and surplus huge pages in the pool of huge pages of default size. The huge page size is needed for generating the proper alignment and size of the arguments to system calls that map huge page regions.

The output of `cat /proc/meminfo` will include lines like:

```
HugePages_Total: uuu
HugePages_Free:  vvv
HugePages_Rsvd:  www
HugePages_Surp:  xxx
Hugepagesize:    yyy kB
Hugetlb:         zzz kB
```

where:

HugePages_Total
: is the size of the pool of huge pages.

HugePages_Free
: is the number of huge pages in the pool that are not yet allocated.

HugePages_Rsvd
: is short for "reserved," and is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made. Reserved huge pages guarantee that an application will be able to allocate a huge page from the pool of huge pages at fault time.

HugePages_Surp
: is short for "surplus," and is the number of huge pages in the pool above the value in `/proc/sys/vm/nr_hugepages`. The maximum number of surplus huge pages is controlled by `/proc/sys/vm/nr_overcommit_hugepages`. Note: When the feature of freeing unused vmemmap pages associated with each hugetlb page is enabled, the number of surplus huge pages may be temporarily larger than the maximum number of surplus huge pages when the system is under memory pressure.

Hugepagesize
: is the default hugepage size (in Kb).

Hugetlb
: is the total amount of memory (in kB), consumed by huge pages of all sizes. If huge pages of different sizes are in use, this number will exceed HugePages_Total * Hugepagesize. To get more detailed information, please, refer to `/sys/kernel/mm/hugepages` (described below).

`/proc/filesystems` should also show a filesystem of type "hugetlbfs" configured in the kernel.

`/proc/sys/vm/nr_hugepages` indicates the current number of "persistent" huge pages in the kernel's huge page pool. "Persistent" huge pages will be returned to the huge page pool when freed by a task. A user with root privileges can dynamically allocate more or free some persistent huge pages by increasing or decreasing the value of `nr_hugepages`.

Note: When the feature of freeing unused vmemmap pages associated with each hugetlb page is enabled, we can fail to free the huge pages triggered by the user when ths system is under memory pressure. Please try again later.

Pages that are used as huge pages are reserved inside the kernel and cannot be used for other purposes. Huge pages cannot be swapped out under memory pressure.

Once a number of huge pages have been pre-allocated to the kernel huge page pool, a user with appropriate privilege can use either the mmap system call or shared memory system calls to use the huge pages. See the discussion of :ref:`Using Huge Pages <using_huge_pages>`, below.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-`

master\Documentation\admin-guide\mm\(linux-master)(Documentation)(admin-guide)(mm)hugetlbpage.rst, line 95); *backlink*

Unknown interpreted text role "ref".

The administrator can allocate persistent huge pages on the kernel boot command line by specifying the "hugepages=N" parameter, where 'N' = the number of huge pages requested. This is the most reliable method of allocating huge pages as memory has not yet become fragmented.

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, one must precede the huge pages boot command parameters with a huge page size selection parameter "hugepagesz=<size>". <size> must be specified in bytes with optional scale suffix [kKmMgG]. The default huge page size may be selected with the "default_hugepagesz=<size>" boot parameter.

Hugetlb boot command line parameter semantics

hugepagesz

Specify a huge page size. Used in conjunction with hugepages parameter to preallocate a number of huge pages of the specified size. Hence, hugepagesz and hugepages are typically specified in pairs such as:

```
hugepagesz=2M hugepages=512
```

hugepagesz can only be specified once on the command line for a specific huge page size. Valid huge page sizes are architecture dependent.

hugepages

Specify the number of huge pages to preallocate. This typically follows a valid hugepagesz or default_hugepagesz parameter. However, if hugepages is the first or only hugetlb command line parameter it implicitly specifies the number of huge pages of default size to allocate. If the number of huge pages of default size is implicitly specified, it can not be overwritten by a hugepagesz,hugepages parameter pair for the default size. This parameter also has a node format. The node format specifies the number of huge pages to allocate on specific nodes.

For example, on an architecture with 2M default huge page size:

```
hugepages=256 hugepagesz=2M hugepages=512
```

will result in 256 2M huge pages being allocated and a warning message indicating that the hugepages=512 parameter is ignored. If a hugepages parameter is preceded by an invalid hugepagesz parameter, it will be ignored.

Node format example:

```
hugepagesz=2M hugepages=0:1,1:2
```

It will allocate 1 2M hugepage on node0 and 2 2M hugepages on node1. If the node number is invalid, the parameter will be ignored.

default_hugepagesz

Specify the default huge page size. This parameter can only be specified once on the command line. default_hugepagesz can optionally be followed by the hugepages parameter to preallocate a specific number of huge pages of default size. The number of default sized huge pages to preallocate can also be implicitly specified as mentioned in the hugepages section above. Therefore, on an architecture with 2M default huge page size:

```
hugepages=256
default_hugepagesz=2M hugepages=256
hugepages=256 default_hugepagesz=2M
```

will all result in 256 2M huge pages being allocated. Valid default huge page size is architecture dependent.

hugetlb_free_vmemmap

When CONFIG_HUGETLB_PAGE_FREE_VMEMMAP is set, this enables freeing unused vmemmap pages associated with each HugeTLB page.

When multiple huge page sizes are supported, `/proc/sys/vm/nr_hugepages` indicates the current number of pre-allocated huge pages of the default size. Thus, one can use the following command to dynamically allocate/deallocate default sized persistent huge pages:

```
echo 20 > /proc/sys/vm/nr_hugepages
```

This command will try to adjust the number of default sized huge pages in the huge page pool to 20, allocating or freeing huge pages, as required.

On a NUMA platform, the kernel will attempt to distribute the huge page pool over all the set of allowed nodes specified by the NUMA memory policy of the task that modifies `nr_hugepages`. The default for the allowed nodes--when the task has default memory policy--is all on-line nodes with memory. Allowed nodes with insufficient available, contiguous memory for a huge page will be silently skipped when allocating persistent huge pages. See the :ref:`discussion below <mem_policy_and_hp_alloc>` of the interaction of task memory policy, cpusets and per node attributes with the allocation and freeing of persistent huge pages.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master)(Documentation)(admin-guide)(mm)hugetlbpage.rst, line 180); *backlink*

Unknown interpreted text role "ref".

The success or failure of huge page allocation depends on the amount of physically contiguous memory that is present in system at the time of the allocation attempt. If the kernel is unable to allocate huge pages from some nodes in a NUMA system, it will attempt to make up the difference by allocating extra pages on other nodes with sufficient available contiguous memory, if any.

System administrators may want to put this command in one of the local rc init files. This will enable the kernel to allocate huge pages early in the boot process when the possibility of getting physical contiguous pages is still very high. Administrators can verify the number of huge pages actually allocated by checking the sysctl or meminfo. To check the per node distribution of huge pages in a NUMA system, use:

```
cat /sys/devices/system/node/node*/meminfo | fgrep Huge
```

`/proc/sys/vm/nr_overcommit_hugepages` specifies how large the pool of huge pages can grow, if more huge pages than `/proc/sys/vm/nr_hugepages` are requested by applications. Writing any non-zero value into this file indicates that the hugetlb subsystem is allowed to try to obtain that number of "surplus" huge pages from the kernel's normal page pool, when the persistent huge page pool is exhausted. As these surplus huge pages become unused, they are freed back to the kernel's normal page pool.

When increasing the huge page pool size via `nr_hugepages`, any existing surplus pages will first be promoted to persistent huge pages. Then, additional huge pages will be allocated, if necessary and if possible, to fulfill the new persistent huge page pool size.

The administrator may shrink the pool of persistent huge pages for the default huge page size by setting the `nr_hugepages` sysctl to a smaller value. The kernel will attempt to balance the freeing of huge pages across all nodes in the memory policy of the task modifying `nr_hugepages`. Any free huge pages on the selected nodes will be freed back to the kernel's normal page pool.

Caveat: Shrinking the persistent huge page pool via `nr_hugepages` such that it becomes less than the number of huge pages in use will convert the balance of the in-use huge pages to surplus huge pages. This will occur even if the number of surplus pages would exceed the overcommit value. As long as this condition holds--that is, until `nr_hugepages+nr_overcommit_hugepages` is increased sufficiently, or the surplus huge pages go out of use and are freed-- no more surplus huge pages will be allowed to be allocated.

With support for multiple huge page pools at run-time available, much of the huge page userspace interface in `/proc/sys/vm` has been duplicated in sysfs. The `/proc` interfaces discussed above have been retained for backwards compatibility. The root huge page control directory in sysfs is:

```
/sys/kernel/mm/hugepages
```

For each huge page size supported by the running kernel, a subdirectory will exist, of the form:

```
hugepages-${size}kB
```

Inside each of these directories, the set of files contained in `/proc` will exist. In addition, two additional interfaces for demoting huge pages may exist:

```
demote
demote_size
nr_hugepages
nr_hugepages_mempolicy
nr_overcommit_hugepages
free_hugepages
resv_hugepages
surplus_hugepages
```

The demote interfaces provide the ability to split a huge page into smaller huge pages. For example, the x86 architecture supports both 1GB and 2MB huge pages sizes. A 1GB huge page can be split into 512 2MB huge pages. Demote interfaces are not available for the smallest huge page size. The demote interfaces are:

demote_size
: is the size of demoted pages. When a page is demoted a corresponding number of huge pages of demote_size will be created. By default, demote_size is set to the next smaller huge page size. If there are multiple smaller huge page sizes, demote_size can be set to any of these smaller sizes. Only huge page sizes less than the current huge pages size are allowed.

demote
: is used to demote a number of huge pages. A user with root privileges can write to this file. It may not be possible to demote the requested number of huge pages. To determine how many pages were actually demoted, compare the value of nr_hugepages before and after writing to the demote interface. demote is a write only interface.

The interfaces which are the same as in `/proc` (all except demote and demote_size) function as described above for the default huge page-sized case.

Interaction of Task Memory Policy with Huge Page Allocation/Freeing

Whether huge pages are allocated and freed via the `/proc` interface or the `/sysfs` interface using the `nr_hugepages_mempolicy` attribute, the NUMA nodes from which huge pages are allocated or freed are controlled by the NUMA memory policy of the task that modifies the `nr_hugepages_mempolicy` sysctl or attribute. When the `nr_hugepages` attribute is used, mempolicy is ignored.

The recommended method to allocate or free huge pages to/from the kernel huge page pool, using the `nr_hugepages` example above, is:

```
numactl --interleave <node-list> echo 20 \
                            >/proc/sys/vm/nr_hugepages_mempolicy
```

or, more succinctly:

```
numactl -m <node-list> echo 20 >/proc/sys/vm/nr_hugepages_mempolicy
```

This will allocate or free `abs(20 - nr_hugepages)` to or from the nodes specified in <node-list>, depending on whether number of persistent huge pages is initially less than or greater than 20, respectively. No huge pages will be allocated nor freed on any node not included in the specified <node-list>.

When adjusting the persistent hugepage count via `nr_hugepages_mempolicy`, any memory policy mode--bind, preferred, local or interleave--may be used. The resulting effect on persistent huge page allocation is as follows:

1. Regardless of mempolicy mode [see :ref:`Documentation/admin-guide/mm/numa_memory_policy.rst <numa_memory_policy>`], persistent huge pages will be distributed across the node or nodes specified in the mempolicy as if "interleave" had been specified. However, if a node in the policy does not contain sufficient contiguous memory for a huge page, the allocation will not "fallback" to the nearest neighbor node with sufficient contiguous memory. To do this would cause undesirable imbalance in the distribution of the huge page pool, or possibly, allocation of persistent huge pages on nodes not allowed by the task's memory policy.

   > **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master)(Documentation)(admin-guide)(mm)hugetlbpage.rst`, line 315); *backlink***
   >
   > Unknown interpreted text role "ref".

2. One or more nodes may be specified with the bind or interleave policy. If more than one node is specified with the preferred policy, only the lowest numeric id will be used. Local policy will select the node where the task is running at the time the nodes_allowed mask is constructed. For local policy to be deterministic, the task must be bound to a cpu or cpus in a single node. Otherwise, the task could be migrated to some other node at any time after launch and the resulting node will be indeterminate. Thus, local policy is not very useful for this purpose. Any of the other mempolicy modes may be used to specify a single node.
3. The nodes allowed mask will be derived from any non-default task mempolicy, whether this policy was set explicitly by the task itself or one of its ancestors, such as numactl. This means that if the task is invoked from a shell with non-default policy, that policy will be used. One can specify a node list of "all" with numactl --interleave or --membind [-m] to achieve interleaving over all nodes in the system or cpuset.
4. Any task mempolicy specified--e.g., using numactl--will be constrained by the resource limits of any cpuset in which the task runs. Thus, there will be no way for a task with non-default policy running in a cpuset with a subset of the system nodes to allocate huge pages outside the cpuset without first moving to a cpuset that contains all of the desired nodes.
5. Boot-time huge page allocation attempts to distribute the requested number of huge pages over all on-lines nodes with memory.

Per Node Hugepages Attributes

A subset of the contents of the root huge page control directory in sysfs, described above, will be replicated under each the system device of each NUMA node with memory in:

```
/sys/devices/system/node/node[0-9]*/hugepages/
```

Under this directory, the subdirectory for each supported huge page size contains the following attribute files:

```
nr_hugepages
free_hugepages
surplus_hugepages
```

The free_' and surplus_' attribute files are read-only. They return the number of free and surplus [overcommitted] huge pages, respectively, on the parent node.

The `nr_hugepages` attribute returns the total number of huge pages on the specified node. When this attribute is written, the number of persistent huge pages on the parent node will be adjusted to the specified value, if sufficient resources exist, regardless of the task's mempolicy or cpuset constraints.

Note that the number of overcommit and reserve pages remain global quantities, as we don't know until fault time, when the faulting

task's mempolicy is applied, from which node the huge page allocation will be attempted.

Using Huge Pages

If the user applications are going to request huge pages using mmap system call, then it is required that system administrator mount a file system of type hugetlbfs:

```
mount -t hugetlbfs \
      -o uid=<value>,gid=<value>,mode=<value>,pagesize=<value>,size=<value>,\
      min_size=<value>,nr_inodes=<value> none /mnt/huge
```

This command mounts a (pseudo) filesystem of type hugetlbfs on the directory `/mnt/huge`. Any file created on `/mnt/huge` uses huge pages.

The `uid` and `gid` options sets the owner and group of the root of the file system. By default the `uid` and `gid` of the current process are taken.

The `mode` option sets the mode of root of file system to value & 01777. This value is given in octal. By default the value 0755 is picked.

If the platform supports multiple huge page sizes, the `pagesize` option can be used to specify the huge page size and associated pool. `pagesize` is specified in bytes. If `pagesize` is not specified the platform's default huge page size and associated pool will be used.

The `size` option sets the maximum value of memory (huge pages) allowed for that filesystem (`/mnt/huge`). The `size` option can be specified in bytes, or as a percentage of the specified huge page pool (`nr_hugepages`). The size is rounded down to HPAGE_SIZE boundary.

The `min_size` option sets the minimum value of memory (huge pages) allowed for the filesystem. `min_size` can be specified in the same way as `size`, either bytes or a percentage of the huge page pool. At mount time, the number of huge pages specified by `min_size` are reserved for use by the filesystem. If there are not enough free huge pages available, the mount will fail. As huge pages are allocated to the filesystem and freed, the reserve count is adjusted so that the sum of allocated and reserved huge pages is always at least `min_size`.

The option `nr_inodes` sets the maximum number of inodes that `/mnt/huge` can use.

If the `size`, `min_size` or `nr_inodes` option is not provided on command line then no limits are set.

For `pagesize`, `size`, `min_size` and `nr_inodes` options, you can use [G|g]/[M|m]/[K|k] to represent giga/mega/kilo. For example, size=2K has the same meaning as size=2048.

While read system calls are supported on files that reside on hugetlb file systems, write system calls are not.

Regular chown, chgrp, and chmod commands (with right permissions) could be used to change the file attributes on hugetlbfs.

Also, it is important to note that no such mount command is required if applications are going to use only shmat/shmget system calls or mmap with MAP_HUGETLB. For an example of how to use mmap with MAP_HUGETLB see :ref:`map_hugetlb <map_hugetlb>` below.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master)(Documentation)(admin-guide)(mm)hugetlbpage.rst, line 440); *backlink***
>
> Unknown interpreted text role "ref".

Users who wish to use hugetlb memory via shared memory segment should be members of a supplementary group and system admin needs to configure that gid into `/proc/sys/vm/hugetlb_shm_group`. It is possible for same or different applications to use any combination of mmaps and shm* calls, though the mount of filesystem will be required for using mmap calls without MAP_HUGETLB.

Syscalls that operate on memory backed by hugetlb pages only have their lengths aligned to the native page size of the processor; they will normally fail with errno set to EINVAL or exclude hugetlb pages that extend beyond the length if not hugepage aligned. For example, munmap(2) will fail if memory is backed by a hugetlb page and the length is smaller than the hugepage size.

Examples

`map_hugetlb`
: see tools/testing/selftests/vm/map_hugetlb.c

`hugepage-shm`
: see tools/testing/selftests/vm/hugepage-shm.c

`hugepage-mmap`
: see tools/testing/selftests/vm/hugepage-mmap.c

The libhugetlbfs library provides a wide range of userspace tools to help with huge page usability, environment setup, and control.