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, i386 architecture supports 4K and 4M (2M in PAE mode) 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 information about the number of free, reserved and surplus huge pages and the default huge page 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: vvv
HugePages_Free:  www
HugePages_Rsvd:  xxx
HugePages_Surp:  yyy
Hugepagesize:    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.

/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'.

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 Using Huge Pages, below.

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 preceed 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.

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 discussion below of the interaction of task memory policy, cpusets and per node attributes with the allocation and freeing of persistent huge pages.

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 it 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 same set of files will exist:

```
nr_hugepages
nr_hugepages_mempolicy
nr_overcommit_hugepages
free_hugepages
resv_hugepages
surplus_hugepages
```

which 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 Documentation/vm/numa_memory_policy.txt], 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.

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 specifed--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>,size=<value>,nr_inodes=<value> \
        none /mnt/huge

This command mounts a (pseudo) filesystem of type hugetlbfs on the directory
/mnt/huge. Any files 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 & 0777. This value is given in octal.
By default the value 0755 is picked. The size option sets the maximum value of
memory (huge pages) allowed for that filesystem (/mnt/huge). The size is
rounded down to HPAGE_SIZE. The option nr_inodes sets the maximum number of
inodes that /mnt/huge can use. If the size or nr_inodes option is not
provided on command line then no limits are set. For 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 the
applications are going to use only shmat/shmget system calls or mmap with
MAP_HUGETLB. Users who wish to use hugetlb page via shared memory segment
should be a member 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. For an example of how to use mmap with MAP_HUGETLB see
map_hugetlb.c.

****

```
/*
 * hugepage-shm:  see Documentation/vm/hugepage-shm.c
 */
```

****

```
/*
 * hugepage-mmap:  see Documentation/vm/hugepage-mmap.c
 */
```