Tmpfs

Tmpfs is a file system which keeps all of its files in virtual memory.

Everything in tmpfs is temporary in the sense that no files will be created on your hard drive. If you unmount a tmpfs instance, everything stored therein is lost.

tmpfs puts everything into the kernel internal caches and grows and shrinks to accommodate the files it contains and is able to swap unneeded pages out to swap space. It has maximum size limits which can be adjusted on the fly via 'mount -o remount ...'

If you compare it to ramfs (which was the template to create tmpfs) you gain swapping and limit checking. Another similar thing is the RAM disk (/dev/ram*), which simulates a fixed size hard disk in physical RAM, where you have to create an ordinary filesystem on top. Ramdisks cannot swap and you do not have the possibility to resize them.

Since tmpfs lives completely in the page cache and on swap, all tmpfs pages will be shown as "Shmem" in /proc/meminfo and "Shared" in free(1). Notice that these counters also include shared memory (shmem, see ipcs(1)). The most reliable way to get the count is using df(1) and du(1).

tmpfs has the following uses:

1. There is always a kernel internal mount which you will not see at all. This is used for shared anonymous mappings and SYSV shared memory.

   This mount does not depend on CONFIG_TMPFS. If CONFIG_TMPFS is not set, the user visible part of tmpfs is not built. But the internal mechanisms are always present.

2. glibc 2.2 and above expects tmpfs to be mounted at /dev/shm for POSIX shared memory (shm_open, shm_unlink). Adding the following line to /etc/fstab should take care of this:

   ```
   tmpfs   /dev/shm        tmpfs   defaults        0 0
   ```

   Remember to create the directory that you intend to mount tmpfs on if necessary.

   This mount is _not_ needed for SYSV shared memory. The internal mount is used for that. (In the 2.3 kernel versions it was necessary to mount the predecessor of tmpfs (shm fs) to use SYSV shared memory.)

3. Some people (including me) find it very convenient to mount it e.g. on /tmp and /var/tmp and have a big swap partition. And now loop mounts of tmpfs files do work, so mkinitrd shipped by most distributions should succeed with a tmpfs /tmp.

4. And probably a lot more I do not know about :-)

tmpfs has three mount options for sizing:

size	The limit of allocated bytes for this tmpfs instance. The default is half of your physical RAM without swap. If you oversize your tmpfs instances the machine will deadlock since the OOM handler will not be able to free that memory.
nr_blocks	The same as size, but in blocks of PAGE_SIZE.
nr_inodes	The maximum number of inodes for this instance. The default is half of the number of your physical RAM pages, or (on a machine with highmem) the number of lowmem RAM pages, whichever is the lower.

These parameters accept a suffix k, m or g for kilo, mega and giga and can be changed on remount. The size parameter also accepts a suffix % to limit this tmpfs instance to that percentage of your physical RAM: the default, when neither size nor nr_blocks is specified, is size=50%

If nr_blocks=0 (or size=0), blocks will not be limited in that instance; if nr_inodes=0, inodes will not be limited. It is generally unwise to mount with such options, since it allows any user with write access to use up all the memory on the machine; but enhances the scalability of that instance in a system with many CPUs making intensive use of it.

tmpfs has a mount option to set the NUMA memory allocation policy for all files in that instance (if CONFIG_NUMA is enabled) - which can be adjusted on the fly via 'mount -o remount ...'

mpol=default	use the process allocation policy (see set_mempolicy(2))
mpol=prefer:Node	prefers to allocate memory from the given Node
mpol=bind:NodeList	allocates memory only from nodes in NodeList
mpol=interleave	prefers to allocate from each node in turn
mpol=interleave:NodeList	allocates from each node of NodeList in turn
mpol=local	prefers to allocate memory from the local node

NodeList format is a comma-separated list of decimal numbers and ranges, a range being two hyphen-separated decimal numbers, the smallest and largest node numbers in the range. For example, mpol=bind:0-3,5,7,9-15

A memory policy with a valid NodeList will be saved, as specified, for use at file creation time. When a task allocates a file in the file system, the mount option memory policy will be applied with a NodeList, if any, modified by the calling task's cpuset constraints [See Documentation/admin-guide/cgroup-v1/cpusets.rst] and any optional flags, listed below. If the resulting NodeLists is the empty set,

the effective memory policy for the file will revert to "default" policy.

NUMA memory allocation policies have optional flags that can be used in conjunction with their modes. These optional flags can be specified when tmpfs is mounted by appending them to the mode before the NodeList. See Documentation/admin-guide/mm/numa_memory_policy.rst for a list of all available memory allocation policy mode flags and their effect on memory policy.

```
=static         is equivalent to        MPOL_F_STATIC_NODES
=relative       is equivalent to        MPOL_F_RELATIVE_NODES
```

For example, mpol=bind=static:NodeList, is the equivalent of an allocation policy of MPOL_BIND | MPOL_F_STATIC_NODES.

Note that trying to mount a tmpfs with an mpol option will fail if the running kernel does not support NUMA; and will fail if its nodelist specifies a node which is not online. If your system relies on that tmpfs being mounted, but from time to time runs a kernel built without NUMA capability (perhaps a safe recovery kernel), or with fewer nodes online, then it is advisable to omit the mpol option from automatic mount options. It can be added later, when the tmpfs is already mounted on MountPoint, by 'mount -o remount,mpol=Policy:NodeList MountPoint'.

To specify the initial root directory you can use the following mount options:

mode	The permissions as an octal number
uid	The user id
gid	The group id

These options do not have any effect on remount. You can change these parameters with chmod(1), chown(1) and chgrp(1) on a mounted filesystem.

tmpfs has a mount option to select whether it will wrap at 32- or 64-bit inode numbers:

inode64	Use 64-bit inode numbers
inode32	Use 32-bit inode numbers

On a 32-bit kernel, inode32 is implicit, and inode64 is refused at mount time. On a 64-bit kernel, CONFIG_TMPFS_INODE64 sets the default. inode64 avoids the possibility of multiple files with the same inode number on a single device; but risks glibc failing with EOVERFLOW once 33-bit inode numbers are reached - if a long-lived tmpfs is accessed by 32-bit applications so ancient that opening a file larger than 2GiB fails with EINVAL.

So 'mount -t tmpfs -o size=10G,nr_inodes=10k,mode=700 tmpfs /mytmpfs' will give you tmpfs instance on /mytmpfs which can allocate 10GB RAM/SWAP in 10240 inodes and it is only accessible by root.

Author: Christoph Rohland <cr@sap.com>, 1.12.01
Updated: Hugh Dickins, 4 June 2007
Updated: KOSAKI Motohiro, 16 Mar 2010
Updated: Chris Down, 13 July 2020