Overcommit Accounting

The Linux kernel supports the following overcommit handling modes

0

> Heuristic overcommit handling. Obvious overcommits of address space are refused. Used for a typical system. It ensures a seriously wild allocation fails while allowing overcommit to reduce swap usage. root is allowed to allocate slightly more memory in this mode. This is the default.

1

> Always overcommit. Appropriate for some scientific applications. Classic example is code using sparse arrays and just relying on the virtual memory consisting almost entirely of zero pages.

2

> Don't overcommit. The total address space commit for the system is not permitted to exceed swap + a configurable amount (default is 50%) of physical RAM. Depending on the amount you use, in most situations this means a process will not be killed while accessing pages but will receive errors on memory allocation as appropriate.
>
> Useful for applications that want to guarantee their memory allocations will be available in the future without having to initialize every page.

The overcommit policy is set via the sysctl `vm.overcommit_memory`.

The overcommit amount can be set via `vm.overcommit_ratio` (percentage) or `vm.overcommit_kbytes` (absolute value). These only have an effect when `vm.overcommit_memory` is set to 2.

The current overcommit limit and amount committed are viewable in `/proc/meminfo` as CommitLimit and Committed_AS respectively.

Gotchas

The C language stack growth does an implicit mremap. If you want absolute guarantees and run close to the edge you MUST mmap your stack for the largest size you think you will need. For typical stack usage this does not matter much but it's a corner case if you really really care

In mode 2 the MAP_NORESERVE flag is ignored.

How It Works

The overcommit is based on the following rules

For a file backed map

> SHARED or READ-only - 0 cost (the file is the map not swap)
> PRIVATE WRITABLE - size of mapping per instance

For an anonymous or `/dev/zero` map

> SHARED - size of mapping
> PRIVATE READ-only - 0 cost (but of little use)
> PRIVATE WRITABLE - size of mapping per instance

Additional accounting

> Pages made writable copies by mmap
> shmfs memory drawn from the same pool

Status

- We account mmap memory mappings
- We account mprotect changes in commit
- We account mremap changes in size
- We account brk
- We account munmap
- We report the commit status in /proc
- Account and check on fork
- Review stack handling/building on exec
- SHMfs accounting
- Implement actual limit enforcement

To Do

- Account ptrace pages (this is hard)