GFP masks used from FS/IO context

Date: May, 2018
Author: Michal Hocko <mhocko@kernel.org>

Introduction

Code paths in the filesystem and IO stacks must be careful when allocating memory to prevent recursion deadlocks caused by direct memory reclaim calling back into the FS or IO paths and blocking on already held resources (e.g. locks - most commonly those used for the transaction context).

The traditional way to avoid this deadlock problem is to clear __GFP_FS respectively __GFP_IO (note the latter implies clearing the first as well) in the gfp mask when calling an allocator. GFP_NOFS respectively GFP_NOIO can be used as shortcut. It turned out though that above approach has led to abuses when the restricted gfp mask is used "just in case" without a deeper consideration which leads to problems because an excessive use of GFP_NOFS/GFP_NOIO can lead to memory over-reclaim or other memory reclaim issues.

New API

Since 4.12 we do have a generic scope API for both NOFS and NOIO context `memalloc_nofs_save`, `memalloc_nofs_restore` respectively `memalloc_noio_save`, `memalloc_noio_restore` which allow to mark a scope to be a critical section from a filesystem or I/O point of view. Any allocation from that scope will inherently drop __GFP_FS respectively __GFP_IO from the given mask so no memory allocation can recurse back in the FS/IO.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)gfp_mask-from-fs-io.rst, line 38)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/sched/mm.h
   :functions: memalloc_nofs_save memalloc_nofs_restore
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)gfp_mask-from-fs-io.rst, line 40)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/sched/mm.h
   :functions: memalloc_noio_save memalloc_noio_restore
```

FS/IO code then simply calls the appropriate save function before any critical section with respect to the reclaim is started - e.g. lock shared with the reclaim context or when a transaction context nesting would be possible via reclaim. The restore function should be called when the critical section ends. All that ideally along with an explanation what is the reclaim context for easier maintenance.

Please note that the proper pairing of save/restore functions allows nesting so it is safe to call `memalloc_noio_save` or `memalloc_noio_restore` respectively from an existing NOIO or NOFS scope.

What about __vmalloc(GFP_NOFS)

vmalloc doesn't support GFP_NOFS semantic because there are hardcoded GFP_KERNEL allocations deep inside the allocator which are quite non-trivial to fix up. That means that calling `vmalloc` with GFP_NOFS/GFP_NOIO is almost always a bug. The good news is that the NOFS/NOIO semantic can be achieved by the scope API.

In the ideal world, upper layers should already mark dangerous contexts and so no special care is required and vmalloc should be called without any problems. Sometimes if the context is not really clear or there are layering violations then the recommended way around that is to wrap `vmalloc` by the scope API with a comment explaining the problem.