GFP Flags

2016-11-05 (http://jake.dothome.co.kr/gfp-flag/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

3 underscore ___GFP flag

No direct use (3 underscores)

- It is only used inside include/linux/gfp.h, and not directly from other sources.
- New additions up to kernel v4.4
  - ___GFP_ATOMIC
  - ___GFP_ACCOUNT
  - ___GFP_DIRECT_RECLAIM
  - ___GFP_KSWAPD_RECLAIM
- Items deleted up to kernel v4.4
  - ___GFP_WAIT
  - ___GFP_NOACCOUNT
  - ___GFP_NOKSWAPD

```
01 /* Plain integer GFP bitmasks. Do not use this directly. */
02 #define ___GFP_DMA              0x01u
03 #define ___GFP_HIGHMEM          0x02u
04 #define ___GFP_DMA32            0x04u
05 #define ___GFP_MOVABLE          0x08u
06 #define ___GFP_WAIT             0x10u
07 #define ___GFP_HIGH             0x20u
08 #define ___GFP_IO               0x40u
09 #define ___GFP_FS               0x80u
10 #define ___GFP_COLD             0x100u
11 #define ___GFP_NOWARN           0x200u
12 #define ___GFP_REPEAT           0x400u
13 #define ___GFP_NOFAIL           0x800u
14 #define ___GFP_NORETRY          0x1000u
15 #define ___GFP_MEMALLOC         0x2000u
16 #define ___GFP_COMP             0x4000u
17 #define ___GFP_ZERO             0x8000u
18 #define ___GFP_NOMEMALLOC       0x10000u
19 #define ___GFP_HARDWALL         0x20000u
20 #define ___GFP_THISNODE         0x40000u
21 #define ___GFP_RECLAIMABLE      0x80000u
22 #define ___GFP_NOACCOUNT        0x100000u
23 #define ___GFP_NOTRACK          0x200000u
24 #define ___GFP_NO_KSWAPD        0x400000u
25 #define ___GFP_OTHER_NODE       0x800000u
26 #define ___GFP_WRITE            0x1000000u
```

- ___GFP_DMA
  - ZONE_DMA requests to be assigned to an area.
- ___GFP_HIGHMEM
  - ZONE_HIGHMEM requests to be assigned to zones.

- ___GFP_DMA32
  - ZONE_DMA32 requests to be assigned to zones.
- ___GFP_MOVABLE
  - Use for two purposes
    - Request assignment in this area when ZONE_MOVABLE are available.
    - Request an allocation so that the page can be migrated.
- ___GFP_RECLAIMABLE
  - Request allocation to a page that can be retrieved.
- ___GFP_WAIT
  - It asks you to allow sleep while allocating memory.
- ___GFP_HIGH
  - Request that it be handled in a high priority manner.
- ___GFP_IO
  - Request that any I/O processing be possible while allocating memory.
- ___GFP_FS
  - Request that File System calls be made available while allocating memory.
- ___GFP_COLD
  - Ask for management on cold pages instead of warm (hot) pages to have less impact on memory fragmentation
- ___GFP_NOWARN
  - Request that no warning be processed when memory allocation fails.
- ___GFP_REPEAT
  - If the memory allocation fails the first time, ask for one more attempt.
- ___GFP_NOFAIL
  - It does not allow failures, and asks that memory allocation requests be processed until they succeed.
- ___GFP_NORETRY
  - Request that memory allocation requests not be retried on failure.
- ___GFP_MEMALLOC
  - Request that the emergency area be used for memory allocation.
- ___GFP_COMP
  - Asks you to construct metadata or a series of composite pages.
- ___GFP_ZERO
  - Request that the allocated area be initialized to zero.
- ___GFP_NOMEMALLOC
  - Request that you do not use the emergency area for memory allocation.
- ___GFP_HARDWALL
  - Requests to use the cpuset memory allocation policy specified in the current task.
- ___GFP_THISNODE
  - Assignments are allowed only on specified nodes.
- ___GFP_NOACCOUNT
  - Request that you not be subject to usage controls by the KMEMCG (Memory Control Group).
- ___GFP_NOTRACK

    - Request not to allow debug tracking using kmemcheck.
- ___GFP_NO_KSWAPD
    - Requests that the assigned page cannot be moved to the Swap file.
- ___GFP_OTHERNODE
    - Request to make an assignment on the remote node.
- ___GFP_WRITE
    - Request a dirty (cache of writable files) page.

2 underscores __GFP flag

ZONE-related (2 underscores)

- Use the bottom four bits.

```
/*
 * Physical address zone modifiers (see linux/mmzone.h - low four bits)
 *
 * Do not put any conditional on these. If necessary modify the definitions
 * without the underscores and use them consistently. The definitions here may
 * be used in bit comparisons.
 */
#define __GFP_DMA       ((__force gfp_t)___GFP_DMA)
#define __GFP_HIGHMEM   ((__force gfp_t)___GFP_HIGHMEM)
#define __GFP_DMA32     ((__force gfp_t)___GFP_DMA32)
#define __GFP_MOVABLE   ((__force gfp_t)___GFP_MOVABLE)  /* Page is movable */
#define GFP_ZONEMASK    (__GFP_DMA|__GFP_HIGHMEM|__GFP_DMA32|__GFP_MOVABLE)
```

- __GFP_DMA
    - Requesting assignment to ZONE_DMA zones
- __GFP_HIGHMEM
    - Requesting an assignment to a ZONE_HIGHMEM zone
- __GFP_DMA32
    - Requesting an assignment to a ZONE_DMA32 zone
- __GFP_MOVABLE
    - If ZONE_MOVABLE is allowed, request assignment to this zone
- GFP_ZONEMASK
    - Includes the above 4 zones
    - When the GFP flag is not used, it usually means ZONE_NORMAL.

Page Mobility and Place hints (2 underscores)

```
/*
 * Page mobility and placement hints
 *
 * These flags provide hints about how mobile the page is. Pages with similar
```

```
 * mobility are placed within the same pageblocks to minimise problems d
ue
 * to external fragmentation.
 *
 * __GFP_MOVABLE (also a zone modifier) indicates that the page can be
 *   moved by page migration during memory compaction or can be reclaime
d.
 *
 * __GFP_RECLAIMABLE is used for slab allocations that specify
 *   SLAB_RECLAIM_ACCOUNT and whose pages can be freed via shrinkers.
 *
 * __GFP_WRITE indicates the caller intends to dirty the page. Where pos
sible,
 *   these pages will be spread between local zones to avoid all the dir
ty
 *   pages being in one zone (fair zone allocation policy).
 *
 * __GFP_HARDWALL enforces the cpuset memory allocation policy.
 *
 * __GFP_THISNODE forces the allocation to be satisified from the reques
ted
 *   node with no fallbacks or placement policy enforcements.
 *
 * __GFP_ACCOUNT causes the allocation to be accounted to kmemcg (only r
elevant
 *   to kmem allocations).
 */
#define __GFP_RECLAIMABLE ((__force gfp_t)___GFP_RECLAIMABLE)
#define __GFP_WRITE     ((__force gfp_t)___GFP_WRITE)
#define __GFP_HARDWALL   ((__force gfp_t)___GFP_HARDWALL)
#define __GFP_THISNODE  ((__force gfp_t)___GFP_THISNODE)
#define __GFP_ACCOUNT   ((__force gfp_t)___GFP_ACCOUNT)
```

- __GFP_RECLAIMABLE
  - Assign it to a page that can be retrieved.
- __GFP_WRITE
  - Request a dirty (cache of writable files) page.
- __GFP_HARDWALL
  - Requests to use the cpuset memory allocation policy specified in the current task.
    - Use this option when calling the get_page_from_freelist() function when assigning a fastpath, see the note below for an exact explanation.
    - Note: Zonned Allocator -1- (Page Assignment-Fastpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-fastpath/) | Qc
- __GFP_THISNODE
  - Assignments are allowed only on specified nodes.
- __GFP_ACCOUNT
  - Request that you not be subject to usage controls by the KMEMCG (Memory Control Group).

Watermark (2 underscores)

```
/*
 * Watermark modifiers -- controls access to emergency reserves
 *
 * __GFP_HIGH indicates that the caller is high-priority and that granti
ng
```

```
 *   the request is necessary before the system can make forward progres
s.
 *   For example, creating an IO context to clean pages.
 *
 * __GFP_ATOMIC indicates that the caller cannot reclaim or sleep and is
 *   high priority. Users are typically interrupt handlers. This may be
 *   used in conjunction with __GFP_HIGH
 *
 * __GFP_MEMALLOC allows access to all memory. This should only be used
when
 *   the caller guarantees the allocation will allow more memory to be f
reed
 *   very shortly e.g. process exiting or swapping. Users either should
 *   be the MM or co-ordinating closely with the VM (e.g. swap over NF
S).
 *
 * __GFP_NOMEMALLOC is used to explicitly forbid access to emergency res
erves.
 *   This takes precedence over the __GFP_MEMALLOC flag if both are set.
 */
#define __GFP_ATOMIC    ((__force gfp_t)___GFP_ATOMIC)
#define __GFP_HIGH      ((__force gfp_t)___GFP_HIGH)
#define __GFP_MEMALLOC  ((__force gfp_t)___GFP_MEMALLOC)
#define __GFP_NOMEMALLOC ((__force gfp_t)___GFP_NOMEMALLOC)
```

- __GFP_ATOMIC
  - We ask that page recalls or slips are not allowed and that they be treated with high priority.
  - It is commonly used by interrupt handlers and can also be used in conjunction with __GFP_HIGH.
- __GFP_HIGH
  - Request that they be treated with high priority. \
- __GFP_MEMALLOC
  - Request permission to access all memory.
  - This is necessary when memory allocation is required in a very short period of time, such as process termination or the use of swapping.
- __GFP_NOMEMALLOC
  - We request that you strictly prohibit the use of the emergency reserves area.

Page recall (2 underscores)

- In kernel v4.4-rc1, __GFP_WAIT was renamed to __GFP_RECLAIM.
  - 참고: mm, page_alloc: rename __GFP_WAIT to __GFP_RECLAIM (https://github.com/torvalds/linux/commit/71baba4b92dc1fa1bc461742c6ab1942ec6034e9)
- __GFP_NOACCOUNT was added in kernel v4.1-rc4.
  - Note: gfp: add __GFP_NOACCOUNT (https://github.com/torvalds/linux/commit/8f4fc071b1926d0b20336e2b3f8ab85c94c734c5)
- In kernel v4.1-rc4 it has been clarified that the __GFP_NOFAIL is in the deprecate state.
  - 참고: mm: clarify __GFP_NOFAIL deprecation status (https://github.com/torvalds/linux/commit/647757197cd34fae041e21af39ded00f5c346fc4)

```
/*
```

```
 * Reclaim modifiers
 *
 * __GFP_IO can start physical IO.
 *
 * __GFP_FS can call down to the low-level FS. Clearing the flag avoids the
 *   allocator recursing into the filesystem which might already be holding
 *   locks.
 *
 * __GFP_DIRECT_RECLAIM indicates that the caller may enter direct reclaim.
 *   This flag can be cleared to avoid unnecessary delays when a fallback
 *   option is available.
 *
 * __GFP_KSWAPD_RECLAIM indicates that the caller wants to wake kswapd when
 *   the low watermark is reached and have it reclaim pages until the high
 *   watermark is reached. A caller may wish to clear this flag when fallback
 *   options are available and the reclaim is likely to disrupt the system. The
 *   canonical example is THP allocation where a fallback is cheap but
 *   reclaim/compaction may cause indirect stalls.
 *
 * __GFP_RECLAIM is shorthand to allow/forbid both direct and kswapd reclaim.
 *
 * __GFP_REPEAT: Try hard to allocate the memory, but the allocation attempt
 *   _might_ fail.  This depends upon the particular VM implementation.
 *
 * __GFP_NOFAIL: The VM implementation _must_ retry infinitely: the caller
 *   cannot handle allocation failures. New users should be evaluated carefully
 *   (and the flag should be used only when there is no reasonable failure
 *   policy) but it is definitely preferable to use the flag rather than
 *   opencode endless loop around allocator.
 *
 * __GFP_NORETRY: The VM implementation must not retry indefinitely and will
 *   return NULL when direct reclaim and memory compaction have failed to allow
 *   the allocation to succeed.  The OOM killer is not called with the current
 *   implementation.
 */
#define __GFP_IO        ((__force gfp_t)___GFP_IO)
#define __GFP_FS        ((__force gfp_t)___GFP_FS)
#define __GFP_DIRECT_RECLAIM    ((__force gfp_t)___GFP_DIRECT_RECLAIM) /* Caller can reclaim */
#define __GFP_KSWAPD_RECLAIM    ((__force gfp_t)___GFP_KSWAPD_RECLAIM) /* kswapd can wake */
#define __GFP_RECLAIM ((__force gfp_t)(___GFP_DIRECT_RECLAIM|___GFP_KSWAPD_RECLAIM))
#define __GFP_REPEAT    ((__force gfp_t)___GFP_REPEAT)
#define __GFP_NOFAIL    ((__force gfp_t)___GFP_NOFAIL)
#define __GFP_NORETRY   ((__force gfp_t)___GFP_NORETRY)
```

- __GFP_IO
  - Request that any I/O processing be possible while allocating memory.
- __GFP_FS

    - Request that File System calls be made available while allocating memory.
- __GFP_DIRECT_RECLAIM
    - If there are not enough free pages when requesting page allocation, request that a direct reclaim be entered.
    - If there is a fallback prepared for memory allocation, the fallback routine will explicitly request to remove this flag to eliminate unnecessary delays.
- __GFP_KSWAPD_RECLAIM
    - If you approach a low watermark, it will wake up kswapd and ask it to reclaim the page until it is on a high watermark.
- __GFP_RECLAIM
    - It requests a recall using the above two flags, i.e., direct reclaim and kswapd.
- __GFP_REPEAT
    - If the memory allocation fails the first time, ask for one more attempt.
- __GFP_NOFAIL
    - It does not allow failures, and asks that memory allocation requests be processed until they succeed.
- __GFP_NORETRY
    - Request that memory allocation requests not be retried on failure.

Action (2 underscores)

```
01 /*
02  * Action modifiers
03  *
04  * __GFP_COLD indicates that the caller does not expect to be used in th
   e near
05  *   future. Where possible, a cache-cold page will be returned.
06  *
07  * __GFP_NOWARN suppresses allocation failure reports.
08  *
09  * __GFP_COMP address compound page metadata.
10  *
11  * __GFP_ZERO returns a zeroed page on success.
12  *
13  * __GFP_NOTRACK avoids tracking with kmemcheck.
14  *
15  * __GFP_NOTRACK_FALSE_POSITIVE is an alias of __GFP_NOTRACK. It's a mea
   ns of
16  *   distinguishing in the source between false positives and allocation
   s that
17  *   cannot be supported (e.g. page tables).
18  *
19  * __GFP_OTHER_NODE is for allocations that are on a remote node but tha
   t
20  *   should not be accounted for as a remote allocation in vmstat. A
21  *   typical user would be khugepaged collapsing a huge page on a remote
22  *   node.
23  */
24 #define __GFP_COLD      ((__force gfp_t)___GFP_COLD)
25 #define __GFP_NOWARN    ((__force gfp_t)___GFP_NOWARN)
26 #define __GFP_COMP      ((__force gfp_t)___GFP_COMP)
27 #define __GFP_ZERO      ((__force gfp_t)___GFP_ZERO)
28 #define __GFP_NOTRACK   ((__force gfp_t)___GFP_NOTRACK)
29 #define __GFP_NOTRACK_FALSE_POSITIVE (__GFP_NOTRACK)
30 #define __GFP_OTHER_NODE ((__force gfp_t)___GFP_OTHER_NODE)
```

- __GFP_COLD
  - Ask for management on cold pages instead of warm (hot) pages to have less impact on memory fragmentation
- __GFP_NOWARN
  - Request that no warning be processed when memory allocation fails.
- __GFP_COMP
  - Asks you to construct metadata or a series of composite pages.
- __GFP_ZERO
  - Request that the allocated area be initialized to zero.
- __GFP_NOTRACK
  - Request not to allow debug tracking using kmemcheck.
- __GFP_NOTRACK_FALSE_POSITIVE
  - Request not to allow false positive debug tracking using kmemcheck.
- __GFP_OTHERNODE
  - Request to make an assignment on the remote node.

GFP flags without underscores

- GFP_THISNODE removed from kernel v4.1-rc4.
  - mm: remove GFP_THISNODE (https://github.com/torvalds/linux/commit/4167e9b2cf10f8a4bcda0c713ddc8bb0a18e8187)
- __GFP_IOFS removed from kernel v4.4-rc1.
  - Note: WAIT mm: page_alloc: remove GFP_IOFS (https://github.com/torvalds/linux/commit/40113370836e8e79befa585277296ed42781ef31)

```
01 /*
02  * Useful GFP flag combinations that are commonly used. It is recommende
   d
03  * that subsystems start with one of these combinations and then set/cle
   ar
04  * __GFP_FOO flags as necessary.
05  *
06  * GFP_ATOMIC users can not sleep and need the allocation to succeed. A
   lower
07  *   watermark is applied to allow access to "atomic reserves"
08  *
09  * GFP_KERNEL is typical for kernel-internal allocations. The caller req
   uires
10  *   ZONE_NORMAL or a lower zone for direct access but can direct reclai
   m.
11  *
12  * GFP_KERNEL_ACCOUNT is the same as GFP_KERNEL, except the allocation i
   s
13  *   accounted to kmemcg.
14  *
15  * GFP_NOWAIT is for kernel allocations that should not stall for direct
16  *   reclaim, start physical IO or use any filesystem callback.
17  *
18  * GFP_NOIO will use direct reclaim to discard clean pages or slab pages
19  *   that do not require the starting of any physical IO.
20  *
```

```
 * GFP_NOFS will use direct reclaim but will not use any filesystem interfaces.
 *
 * GFP_USER is for userspace allocations that also need to be directly
 *   accessibly by the kernel or hardware. It is typically used by hardware
 *   for buffers that are mapped to userspace (e.g. graphics) that hardware
 *   still must DMA to. cpuset limits are enforced for these allocations.
 *
 * GFP_DMA exists for historical reasons and should be avoided where possible.
 *   The flags indicates that the caller requires that the lowest zone be
 *   used (ZONE_DMA or 16M on x86-64). Ideally, this would be removed but
 *   it would require careful auditing as some users really require it and
 *   others use the flag to avoid lowmem reserves in ZONE_DMA and treat the
 *   lowest zone as a type of emergency reserve.
 *
 * GFP_DMA32 is similar to GFP_DMA except that the caller requires a 32-bit
 *   address.
 *
 * GFP_DMA32 is similar to GFP_DMA except that the caller requires a 32-bit
 *   address.
 *
 * GFP_HIGHUSER is for userspace allocations that may be mapped to userspace,
 *   do not need to be directly accessible by the kernel but that cannot
 *   move once in use. An example may be a hardware allocation that maps
 *   data directly into userspace but has no addressing limitations.
 *
 * GFP_HIGHUSER_MOVABLE is for userspace allocations that the kernel does not
 *   need direct access to but can use kmap() when access is required. They
 *   are expected to be movable via page reclaim or page migration. Typically,
 *   pages on the LRU would also be allocated with GFP_HIGHUSER_MOVABLE.
 *
 * GFP_TRANSHUGE is used for THP allocations. They are compound allocations
 *   that will fail quickly if memory is not available and will not wake
 *   kswapd on failure.
 */
#define GFP_ATOMIC      (__GFP_HIGH|__GFP_ATOMIC|__GFP_KSWAPD_RECLAIM)
#define GFP_KERNEL      (__GFP_RECLAIM | __GFP_IO | __GFP_FS)
#define GFP_KERNEL_ACCOUNT (GFP_KERNEL | __GFP_ACCOUNT)
#define GFP_NOWAIT      (__GFP_KSWAPD_RECLAIM)
#define GFP_NOIO        (__GFP_RECLAIM)
#define GFP_NOFS        (__GFP_RECLAIM | __GFP_IO)
#define GFP_TEMPORARY   (__GFP_RECLAIM | __GFP_IO | __GFP_FS | \
                         __GFP_RECLAIMABLE)
#define GFP_USER        (__GFP_RECLAIM | __GFP_IO | __GFP_FS | __GFP_HARDWALL)
#define GFP_DMA         __GFP_DMA
#define GFP_DMA32       __GFP_DMA32
#define GFP_HIGHUSER    (GFP_USER | __GFP_HIGHMEM)
#define GFP_HIGHUSER_MOVABLE    (GFP_HIGHUSER | __GFP_MOVABLE)
#define GFP_TRANSHUGE   ((GFP_HIGHUSER_MOVABLE | __GFP_COMP | \
                          __GFP_NOMEMALLOC | __GFP_NORETRY | __GFP_NOWARN) & \
                          ~__GFP_RECLAIM)
```

- GFP_ATOMIC
  - It should not be slipped and requested to be applied with a low watermark that allows access to "atomic reserves".
- GFP_KERNEL
  - It is used for allocation using the kernel's internal algorithms, which can be reclaimed via direct-reclaim or kswapd, and requests the use of ZONE_NORMAL or lower zones with IO and FS available.
- GFP_KERNEL_ACCOUNT
  - It is the same as GFP_KERNEL, except that it is subject to usage control by KMEMCG (Memory Control Group).
- GFP_NOWAIT
  - Request that reclaim using kswapd be enabled for kernel allocation.
- GFP_NOIO
  - When using Direct Reclaim, you can't throw away clean or slab pages.
- GFP_NOFS
  - When using Direct Reclaim, IO processing is possible, but the File System interface cannot be used.
- GFP_USER
  - Request direct access by the kernel and hardware for userspace allocation.
  - Requests to use the cpuset memory allocation policy specified in the current task.
- GFP_DMA
  - Request the lowest zone (ZONE_DMA).
- GFP_DMA32
  - Ask for ZONE_DMA.
- GFP_HIGHUSER
  - Request the use of highmem in the GFP_USER for userspace allocation.
- GFP_HIGHUSER_MOVABLE
  - Request the use of highmem and movable migrate types in the GFP_USER for userspace allocation.
- GFP_TRANSHUGE
  - It is used for Transparent Huge Page (THP) allocation and will fail quickly if there is not enough memory.
  - If it fails, don't wake kswapd.

---

4 thoughts to "GFP Flag"

SAMURO
2020-07-07 11:21 (http://jake.dothome.co.kr/gfp-flag/#comment-260659)

I have a question about __GFP_DIRECT_RECLAIM.
You explained that "if fallback is valid, it should be specified clearly to eliminate unnecessary delays", which seems to imply that clear can be removed. In fact, if you

look inside the bvec_alloc() code in block/bio.c,

```
206 if (*idx == BVEC_POOL_MAX) {
207 fallback:
208 bvl = mempool_alloc(pool, gfp_mask);
209 } else {
210 struct biovec_slab *bvs = bvec_slabs + *idx;
211 gfp_t __gfp_mask = gfp_mask & ~(__GFP_DIRECT_RECLAIM | __GFP_IO);
212
213 /*
214 * Make this allocation restricted and don't dump info on
215 * allocation failures, since we'll fallback to the mempool
216 * in case of failure.
217 */
218 __gfp_mask |= __GFP_NOMEMALLOC | __GFP_NORETRY | __GFP_NOWARN;
219
220 /*
221 * Try a slab allocation. If this fails and __GFP_DIRECT_RECLAIM
222 * is set, retry with the 1-entry mempool
223 */
224 bvl = kmem_cache_alloc(bvs->slab, __gfp_mask);
```

There is something like this. If there is a mempool to be used as a fallback, I understand this to mean that the __GFP_DIRECT_RECLAIM flag will be removed.
I wonder if I'm misunderstood, please comment!

RESPONSE (/GFP-FLAG/?REPLYTOCOM=260659#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2020-07-07 13:50 (http://jake.dothome.co.kr/gfp-flag/#comment-260661)

Hello?

Regarding __GFP_DIRECT_RECLAIM, there is a part of my article that has been explained in the opposite way, so I have corrected it.

If the user's fallback code is prepared for the assignment, the fallback code can be used to clear the __GFP_DIRECT_RECLAIM flag and reduce latency.

I'm sure you understand and ask, but just in case, if you fail with the direct-reclaim option on the first memory allocation,
then you have very little chance of reclaiming the memory if you try to do it again on the second memory allocation. It's just a waste of time.

Therefore, if you have fallback code ready for memory allocation, we recommend that you remove the direct-reclaim flag and proceed.

I appreciate it.

RESPONSE (/GFP-FLAG/?REPLYTOCOM=260661#RESPOND)

POPPING
2021-04-01 22:15 (http://jake.dothome.co.kr/gfp-flag/#comment-305023)

You're the author of a mosquito repellent book! Now I'm looking at this gem of a site and studying this and that. I appreciate it.

I think you have a misunderstanding about GFP_DIRECT_RECLAIM, so I want to correct you. Unless you have a very large memory allocation, especially in a short period of time, this doesn't add to the delay. However, there are times when this flag should not be given.
– If you can't fall asleep. The reason why memory allocation falls asleep is because direct reclaim works. GFP_ATOMIC is what it is used for, but ATOMIC has some additional rewards, such as allowing a little more min wmark, and if you don't need that, you can just delete the DIRECT_RECLAIM. (By the way, if that's the case, you can use ATOMIC.)

If you give something like a GFP_KERNEL that contains GFP_DIRECT_RECLAIM and the memory allocation fails.. That actually means that there is a serious problem with the system's memory operation, or that the system memory is low compared to the required working set, so you need to look into it.
If there's a case to clear and assign this flag, as you said above, it's if you want to quickly move on to a fallback. There are many reasons for this, such as if you have an emergency fund, if you take out your emergency fund first rather than waiting for pocket money, or if you want to get a compound page but don't want to wait, and if you can't, you get a small page.

RESPONSE (/GFP-FLAG/?REPLYTOCOM=305023#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2021-04-03 14:56 (http://jake.dothome.co.kr/gfp-flag/#comment-305049)

Hello?

You explained it correctly.

To reinforce this a bit, a flag like __GFP_DIRECT_RECLAIM is not a flag that users would normally use in a normal situation.
Users typically only use GFP_ATOMIC flags when using GFP_KERNEL or wanting to allocate memory within an interrupt context.

The difference between using a two-line __GFP_DIRECT_RECLAIM inside the kernel and not using it is as follows:
– If there is a low memory situation without the above flag, it will just return an error.
– When a memory shortage occurs with the above flag, try to free up memory by trying to direct reclaim.

If you use GFP_KERNEL to work through the code inside the kernel, you can see that if the allocation fails due to low memory, it will repeat with a slight change, but at certain times it will try without the __GFP_DIRECT_RECLAIM flag. This has already been flagged and failed, and the situation has changed slightly and the retry should be done without the __GFP_DIRECT_RECLAIM flag to reduce the time wasted due to reclaiming, so that it doesn't go back into unnecessary reclaim. For this reason, even if the user tries to allocate a page using GFP_KERNEL, the above flag is removed in some situations from the kernel to reduce the latency caused by duplicate reclaims.

I appreciate it.

RESPONSE (/GFP-FLAG/?REPLYTOCOM=305049#RESPOND)

LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

‹ early_irq_init() (http://jake.dothome.co.kr/early_irq_init/)

kmalloc vs vmalloc › (http://jake.dothome.co.kr/kmalloc-vs-vmalloc/)

Munc Blog (2015 ~ 2023)