Examining Process Page Tables

pagemap is a new (as of 2.6.25) set of interfaces in the kernel that allow userspace programs to examine the page tables and related information by reading files in `/proc`.

There are four components to pagemap:

- `/proc/pid/pagemap`. This file lets a userspace process find out which physical frame each virtual page is mapped to. It contains one 64-bit value for each virtual page, containing the following data (from `fs/proc/task_mmu.c`, above pagemap_read):

    - Bits 0-54 page frame number (PFN) if present
    - Bits 0-4 swap type if swapped
    - Bits 5-54 swap offset if swapped
    - Bit 55 pte is soft-dirty (see :ref:`Documentation/admin-guide/mm/soft-dirty.rst <soft_dirty>`)

        > **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master) (Documentation) (admin-guide) (mm)pagemap.rst`, line 21); *backlink***
        >
        > Unknown interpreted text role "ref".

    - Bit 56 page exclusively mapped (since 4.2)
    - Bit 57 pte is uffd-wp write-protected (since 5.13) (see :ref:`Documentation/admin-guide/mm/userfaultfd.rst <userfaultfd>`)

        > **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master) (Documentation) (admin-guide) (mm)pagemap.rst`, line 24); *backlink***
        >
        > Unknown interpreted text role "ref".

    - Bits 58-60 zero
    - Bit 61 page is file-page or shared-anon (since 3.5)
    - Bit 62 page swapped
    - Bit 63 page present

    Since Linux 4.0 only users with the CAP_SYS_ADMIN capability can get PFNs. In 4.0 and 4.1 opens by unprivileged fail with -EPERM. Starting from 4.2 the PFN field is zeroed if the user does not have CAP_SYS_ADMIN. Reason: information about PFNs helps in exploiting Rowhammer vulnerability.

    If the page is not present but in swap, then the PFN contains an encoding of the swap file number and the page's offset into the swap. Unmapped pages return a null PFN. This allows determining precisely which pages are mapped (or in swap) and comparing mapped pages between processes.

    Efficient users of this interface will use `/proc/pid/maps` to determine which areas of memory are actually mapped and llseek to skip over unmapped regions.

- `/proc/kpagecount`. This file contains a 64-bit count of the number of times each page is mapped, indexed by PFN.

The page-types tool in the tools/vm directory can be used to query the number of times a page is mapped.

- `/proc/kpageflags`. This file contains a 64-bit set of flags for each page, indexed by PFN.

    The flags are (from `fs/proc/page.c`, above kpageflags_read):

    0. LOCKED
    1. ERROR
    2. REFERENCED
    3. UPTODATE
    4. DIRTY
    5. LRU

6. ACTIVE
7. SLAB
8. WRITEBACK
9. RECLAIM
10. BUDDY
11. MMAP
12. ANON
13. SWAPCACHE
14. SWAPBACKED
15. COMPOUND_HEAD
16. COMPOUND_TAIL
17. HUGE
18. UNEVICTABLE
19. HWPOISON
20. NOPAGE
21. KSM
22. THP
23. OFFLINE
24. ZERO_PAGE
25. IDLE
26. PGTABLE

- `/proc/kpagecgroup`. This file contains a 64-bit inode number of the memory cgroup each page is charged to, indexed by PFN. Only available when CONFIG_MEMCG is set.

Short descriptions to the page flags

0 - LOCKED

The page is being locked for exclusive access, e.g. by undergoing read/write IO.

7 - SLAB

The page is managed by the SLAB/SLOB/SLUB/SLQB kernel memory allocator. When compound page is used, SLUB/SLQB will only set this flag on the head page; SLOB will not flag it at all.

10 - BUDDY

A free memory block managed by the buddy system allocator. The buddy system organizes free memory in blocks of various orders. An order N block has 2^N physically contiguous pages, with the BUDDY flag set for and _only_ for the first page.

15 - COMPOUND_HEAD

A compound page with order N consists of 2^N physically contiguous pages. A compound page with order 2 takes the form of "HTTT", where H donates its head page and T donates its tail page(s). The major consumers of compound pages are hugeTLB pages (:ref:`Documentation/admin-guide/mm/hugetlbpage.rst <hugetlbpage>`), the SLUB etc. memory allocators and various device drivers. However in this interface, only huge/giga pages are made visible to end users.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master) (Documentation) (admin-guide) (mm)pagemap.rst`, line 105); *backlink*

Unknown interpreted text role "ref".

16 - COMPOUND_TAIL

A compound page tail (see description above).

17 - HUGE

This is an integral part of a HugeTLB page.

19 - HWPOISON

Hardware detected memory corruption on this page: don't touch the data!

20 - NOPAGE

No page frame exists at the requested address.

21 - KSM

Identical memory pages dynamically shared between one or more processes.

22 - THP

Contiguous pages which construct transparent hugepages.

23 - OFFLINE

The page is logically offline.

24 - ZERO_PAGE

Zero page for pfn_zero or huge_zero page.

25 - IDLE

The page has not been accessed since it was marked idle (see :ref:`Documentation/admin-guide/mm/idle_page_tracking.rst <idle_page_tracking>`). Note that this flag may be stale in case the page was accessed via a PTE. To make sure the flag is up-to-date one has to read `/sys/kernel/mm/page_idle/bitmap` first.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\mm\(linux-master)(Documentation)(admin-guide)(mm)pagemap.rst`, line 130); *backlink***
>
> Unknown interpreted text role "ref".

26 - PGTABLE

The page is in use as a page table.

IO related page flags

1 - ERROR
IO error occurred.

3 - UPTODATE
The page has up-to-date data. ie. for file backed page: (in-memory data revision >= on-disk one)

4 - DIRTY
The page has been written to, hence contains new data. i.e. for file backed page: (in-memory data revision > on-disk one)

8 - WRITEBACK
The page is being synced to disk.

LRU related page flags

5 - LRU
The page is in one of the LRU lists.

6 - ACTIVE
The page is in the active LRU list.

18 - UNEVICTABLE
The page is in the unevictable (non-)LRU list It is somehow pinned and not a candidate for LRU page reclaims, e.g. ramfs pages, shmctl(SHM_LOCK) and mlock() memory segments.

2 - REFERENCED
The page has been referenced since last LRU list enqueue/requeue.

9 - RECLAIM
The page will be reclaimed soon after its pageout IO completed.

11 - MMAP
A memory mapped page.

12 - ANON
A memory mapped page that is not part of a file.

13 - SWAPCACHE
The page is mapped to swap space, i.e. has an associated swap entry.

14 - SWAPBACKED
The page is backed by swap/RAM.

The page-types tool in the tools/vm directory can be used to query the above flags.

Using pagemap to do something useful

The general procedure for using pagemap to find out about a process' memory usage goes like this:

1. Read `/proc/pid/maps` to determine which parts of the memory space are mapped to what.
2. Select the maps you are interested in -- all of them, or a particular library, or the stack or the heap, etc.
3. Open `/proc/pid/pagemap` and seek to the pages you would like to examine.
4. Read a u64 for each page from pagemap.

5. Open `/proc/kpagecount` and/or `/proc/kpageflags`. For each PFN you just read, seek to that entry in the file, and read the data you want.

For example, to find the "unique set size" (USS), which is the amount of memory that a process is using that is not shared with any other process, you can go through every map in the process, find the PFNs, look those up in kpagecount, and tally up the number of pages that are only referenced once.

Exceptions for Shared Memory

Page table entries for shared pages are cleared when the pages are zapped or swapped out. This makes swapped out pages indistinguishable from never-allocated ones.

In kernel space, the swap location can still be retrieved from the page cache. However, values stored only on the normal PTE get lost irretrievably when the page is swapped out (i.e. SOFT_DIRTY).

In user space, whether the page is present, swapped or none can be deduced with the help of lseek and/or mincore system calls.

lseek() can differentiate between accessed pages (present or swapped out) and holes (none/non-allocated) by specifying the SEEK_DATA flag on the file where the pages are backed. For anonymous shared pages, the file can be found in `/proc/pid/map_files/`.

mincore() can differentiate between pages in memory (present, including swap cache) and out of memory (swapped out or none/non-allocated).

Other notes

Reading from any of the files will return -EINVAL if you are not starting the read on an 8-byte boundary (e.g., if you sought an odd number of bytes into the file), or if the size of the read is not a multiple of 8 bytes.

Before Linux 3.11 pagemap bits 55-60 were used for "page-shift" (which is always 12 at most architectures). Since Linux 3.11 their meaning changes after first clear of soft-dirty bits. Since Linux 4.2 they are used for flags unconditionally.