page owner: Tracking about who allocated each page

Introduction

page owner is for the tracking about who allocated each page. It can be used to debug memory leak or to find a memory hogger. When allocation happens, information about allocation such as call stack and order of pages is stored into certain storage for each page. When we need to know about status of all pages, we can get and analyze this information.

Although we already have tracepoint for tracing page allocation/free, using it for analyzing who allocate each page is rather complex. We need to enlarge the trace buffer for preventing overlapping until userspace program launched. And, launched program continually dump out the trace buffer for later analysis and it would change system behaviour with more possibility rather than just keeping it in memory, so bad for debugging.

page owner can also be used for various purposes. For example, accurate fragmentation statistics can be obtained through gfp flag information of each page. It is already implemented and activated if page owner is enabled. Other usages are more than welcome.

page owner is disabled by default. So, if you'd like to use it, you need to add "page_owner=on" to your boot cmdline. If the kernel is built with page owner and page owner is disabled in runtime due to not enabling boot option, runtime overhead is marginal. If disabled in runtime, it doesn't require memory to store owner information, so there is no runtime memory overhead. And, page owner inserts just two unlikely branches into the page allocator hotpath and if not enabled, then allocation is done like as the kernel without page owner. These two unlikely branches should not affect to allocation performance, especially if the static keys jump label patching functionality is available. Following is the kernel's code size change due to this facility.

- Without page owner:

  ```
  text    data     bss     dec     hex filename
  48392   2333     644   51369    c8a9 mm/page_alloc.o
  ```

- With page owner:

  ```
  text    data     bss     dec     hex filename
  48800   2445     644   51889    cab1 mm/page_alloc.o
  6662     108      29    6799    1a8f mm/page_owner.o
  1025       8       8    1041     411 mm/page_ext.o
  ```

Although, roughly, 8 KB code is added in total, page_alloc.o increase by 520 bytes and less than half of it is in hotpath. Building the kernel with page owner and turning it on if needed would be great option to debug kernel memory problem.

There is one notice that is caused by implementation detail. page owner stores information into the memory from struct page extension. This memory is initialized some time later than that page allocator starts in sparse memory system, so, until initialization, many pages can be allocated and they would have no owner information. To fix it up, these early allocated pages are investigated and marked as allocated in initialization phase. Although it doesn't mean that they have the right owner information, at least, we can tell whether the page is allocated or not, more accurately. On 2GB memory x86-64 VM box, 13343 early allocated pages are catched and marked, although they are mostly allocated from struct page extension feature. Anyway, after that, no page is left in un-tracking state.

Usage

1. Build user-space helper:

   ```
   cd tools/vm
   make page_owner_sort
   ```

2. Enable page owner: add "page_owner=on" to boot cmdline.
3. Do the job that you want to debug.
4. Analyze information from page owner:

   ```
   cat /sys/kernel/debug/page_owner > page_owner_full.txt
   ./page_owner_sort page_owner_full.txt sorted_page_owner.txt
   ```

   The general output of `page_owner_full.txt` is as follows:

   ```
   Page allocated via order XXX, ...
   PFN XXX ...
   // Detailed stack

   Page allocated via order XXX, ...
   PFN XXX ...
   // Detailed stack
   ```

   The `page_owner_sort` tool ignores `PFN` rows, puts the remaining rows in buf, uses regexp to extract the page order value, counts the times and pages of buf, and finally sorts them according to the parameter(s).

See the result about who allocated each page in the `sorted_page_owner.txt`. General output:

```
XXX times, XXX pages:
Page allocated via order XXX, ...
// Detailed stack
```

By default, `page_owner_sort` is sorted according to the times of buf. If you want to sort by the page nums of buf, use the `-m` parameter. The detailed parameters are:

fundamental function:

Sort:

`-a`	Sort by memory allocation time.
`-m`	Sort by total memory.
`-p`	Sort by pid.
`-P`	Sort by tgid.
`-n`	Sort by task command name.
`-r`	Sort by memory release time.
`-s`	Sort by stack trace.
`-t`	Sort by times (default).

additional function:

Cull:

`--cull` *<rules>*	Specify culling rules.Culling syntax is key[,key[,...]].Choose a multi-letter key from the **STANDARD FORMAT SPECIFIERS** section.

<rules> is a single argument in the form of a comma-separated list, which offers a way to specify individual culling rules. The recognized keywords are described in the **STANDARD FORMAT SPECIFIERS** section below. <rules> can be specified by the sequence of keys k1,k2, ..., as described in the STANDARD SORT KEYS section below. Mixed use of abbreviated and complete-form of keys is allowed.

Examples:
./page_owner_sort <input> <output> **--**cull=stacktrace ./page_owner_sort <input> <output> **--**cull=st,pid,name ./page_owner_sort <input> <output> **--**cull=n,f

Filter:

`-f`	Filter out the information of blocks whose memory has been released.

Select:

`--pid` *<PID>*	Select by pid.
`--tgid` *<TGID>*	Select by tgid.
`--name` *<command>*	Select by task command name.

STANDARD FORMAT SPECIFIERS

KEY LONG DESCRIPTION p pid process ID tg tgid thread group ID n name task command name f free whether the page has been released or not st stacktrace stace trace of the page allocation