Slub_leak_debug

Some platforms maybe too old or too new, we cannot use slab trace.
Or sometimes the page_owner is not avaible. Here I get a method to debug slub memleak.

1: first you need to confirm the slub is keep leaking.

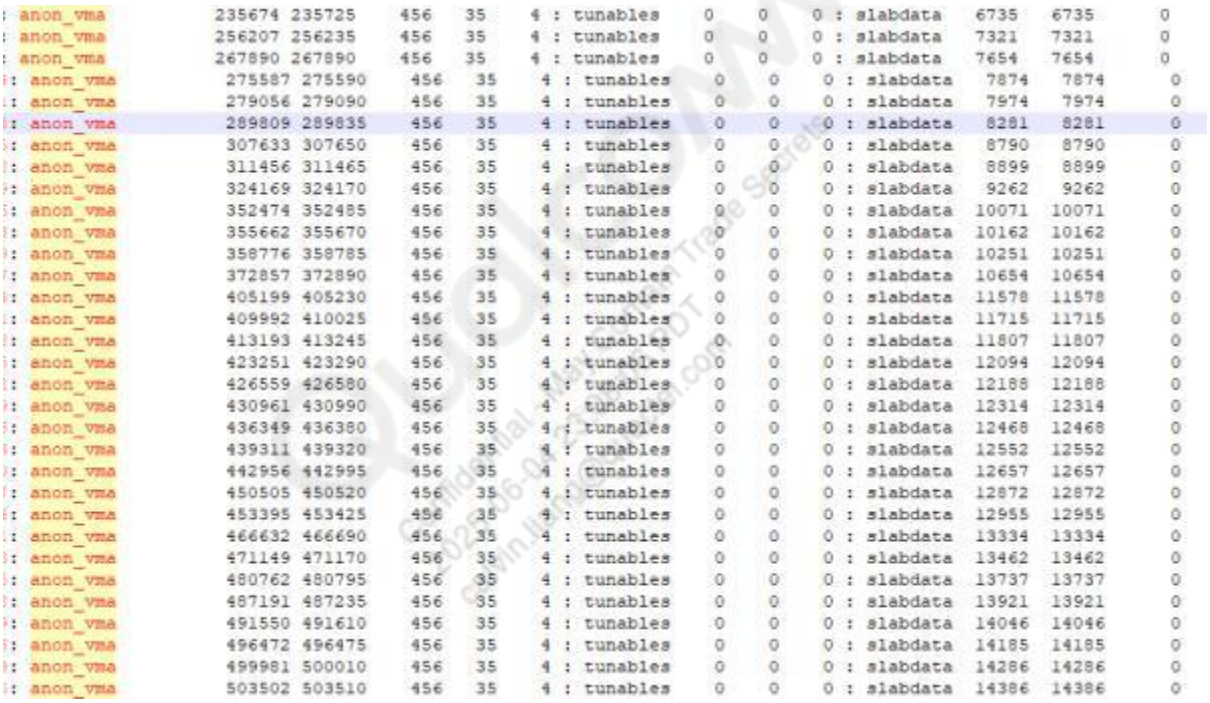
Cat /proc/meminfo
Then check the SUnreclaim for exp:
Here is we cat /proc/meminfo every 2-3 hours, then we found the SUnreclaim is keep growing

行 26: SUnreclaim: 781988 kB
行 71: SUnreclaim: 806644 kB
行 116: SUnreclaim: 815900 kB
行 161: SUnreclaim: 829012 kB
行 206: SUnreclaim: 838196 kB
行 251: SUnreclaim: 851020 kB
行 296: SUnreclaim: 873748 kB
行 341: SUnreclaim: 873476 kB
行 386: SUnreclaim: 889788 kB
行 431: SUnreclaim: 904212 kB
行 476: SUnreclaim: 906100 kB
行 520: SUnreclaim: 907468 kB
行 566: SUnreclaim: 917092 kB
行 611: SUnreclaim: 947828 kB
行 656: SUnreclaim: 949708 kB
行 701: SUnreclaim: 951100 kB
行 746: SUnreclaim: 954340 kB
行 791: SUnreclaim: 955300 kB
行 836: SUnreclaim: 956996 kB
行 881: SUnreclaim: 958532 kB
行 926: SUnreclaim: 960140 kB
行 971: SUnreclaim: 962076 kB
行 1016: SUnreclaim: 968716 kB
行 1061: SUnreclaim: 968276 kB
行 1106: SUnreclaim: 973860 kB
行 1151: SUnreclaim: 977180 kB
行 1196: SUnreclaim: 982332 kB
行 1241: SUnreclaim: 986188 kB
行 1286: SUnreclaim: 989196 kB
行 1331: SUnreclaim: 990236 kB

行 1376: SUnreclaim: 992356 kB
行 1421: SUnreclaim: 994628 kB
行 1466: SUnreclaim: 994820 kB
行 1510: SUnreclaim: 1006092 kB

2: meanwhile we need also check the slab info in /proc/slabinfo every 2-3 hours,

Then we can find below slab is keep growing



anon_vma	235674	235725	456	35	4	tunables	0	0	0	slabdata	6735	6735	0
anon_vma	256207	256235	456	35	4	tunables	0	0	0	slabdata	7321	7321	0
anon_vma	267890	267890	456	35	4	tunables	0	0	0	slabdata	7654	7654	0
anon_vma	275587	275590	456	35	4	tunables	0	0	0	slabdata	7874	7874	0
anon_vma	279056	279090	456	35	4	tunables	0	0	0	slabdata	7974	7974	0
anon_vma	289809	289835	456	35	4	tunables	0	0	0	slabdata	8281	8281	0
anon_vma	307633	307650	456	35	4	tunables	0	0	0	slabdata	8790	8790	0
anon_vma	311456	311465	456	35	4	tunables	0	0	0	slabdata	8899	8899	0
anon_vma	324169	324170	456	35	4	tunables	0	0	0	slabdata	9262	9262	0
anon_vma	352474	352485	456	35	4	tunables	0	0	0	slabdata	10071	10071	0
anon_vma	355662	355670	456	35	4	tunables	0	0	0	slabdata	10162	10162	0
anon_vma	358776	358785	456	35	4	tunables	0	0	0	slabdata	10251	10251	0
anon_vma	372857	372890	456	35	4	tunables	0	0	0	slabdata	10654	10654	0
anon_vma	405199	405230	456	35	4	tunables	0	0	0	slabdata	11578	11578	0
anon_vma	409992	410025	456	35	4	tunables	0	0	0	slabdata	11715	11715	0
anon_vma	413193	413245	456	35	4	tunables	0	0	0	slabdata	11807	11807	0
anon_vma	423251	423290	456	35	4	tunables	0	0	0	slabdata	12094	12094	0
anon_vma	426559	426580	456	35	4	tunables	0	0	0	slabdata	12188	12188	0
anon_vma	430961	430990	456	35	4	tunables	0	0	0	slabdata	12314	12314	0
anon_vma	436349	436380	456	35	4	tunables	0	0	0	slabdata	12468	12468	0
anon_vma	439311	439320	456	35	4	tunables	0	0	0	slabdata	12552	12552	0
anon_vma	442956	442995	456	35	4	tunables	0	0	0	slabdata	12657	12657	0
anon_vma	450505	450520	456	35	4	tunables	0	0	0	slabdata	12872	12872	0
anon_vma	453395	453425	456	35	4	tunables	0	0	0	slabdata	12955	12955	0
anon_vma	466632	466690	456	35	4	tunables	0	0	0	slabdata	13334	13334	0
anon_vma	471149	471170	456	35	4	tunables	0	0	0	slabdata	13462	13462	0
anon_vma	480762	480795	456	35	4	tunables	0	0	0	slabdata	13737	13737	0
anon_vma	487191	487235	456	35	4	tunables	0	0	0	slabdata	13921	13921	0
anon_vma	491550	491610	456	35	4	tunables	0	0	0	slabdata	14046	14046	0
anon_vma	496472	496475	456	35	4	tunables	0	0	0	slabdata	14185	14185	0
anon_vma	499981	500010	456	35	4	tunables	0	0	0	slabdata	14286	14286	0
anon_vma	503502	503510	456	35	4	tunables	0	0	0	slabdata	14386	14386	0

3: next we can add trace on the slab alloc function.

diff --git a/mm/slub.c b/mm/slub.c
index 229136a8ef70..cb05adc1ca54 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -229,6 +229,27 @@ static inline void stat(const struct kmem_cache *s, enum stat_item si)
#endif
}

+void trace_my_debug_slub_alloc(struct kmem_cache *s)
+{

```
+if((0 == strcmp("anon_vma",s->name))){       // change this to trace your leak slab
+ trace_printk(
+ "alloc slub %s = Callers:(%ps<-%ps<-%ps<-%ps<-%ps<-%ps)\n",s->name,
+ (void *)CALLER_ADDR0, (void *)CALLER_ADDR1, (void *) CALLER_ADDR2,
+ (void *)CALLER_ADDR3, (void *)CALLER_ADDR4, (void *)CALLER_ADDR5,
+ (void *)CALLER_ADDR6);
+}
+}
+
+void trace_my_debug_slub_free(struct kmem_cache *s)
+{
+if((0 == strcmp("anon_vma",s->name))){
+ trace_printk(
+ "free slub %s = Callers:(%ps<-%ps<-%ps<-%ps<-%ps<-%ps)\n",s->name,
+ (void *)CALLER_ADDR0, (void *)CALLER_ADDR1, (void *) CALLER_ADDR2,
+ (void *)CALLER_ADDR3, (void *)CALLER_ADDR4, (void *)CALLER_ADDR5,
+ (void *)CALLER_ADDR6);
+}
+}
/*****************************************************************
* Core slab cache functions
*****************************************************************/
@@ -2969,6 +2990,7 @@ void *kmem_cache_alloc(struct kmem_cache *s, gfp_t gfpflags)
trace_kmem_cache_alloc(_RET_IP_, ret, s->object_size,
s->size, gfpflags);

+ trace_my_debug_slub_alloc(s);
return ret;
}
EXPORT_SYMBOL(kmem_cache_alloc);
@@ -3225,6 +3247,7 @@ void kmem_cache_free(struct kmem_cache *s, void *x)
return;
slab_free(s, virt_to_head_page(x), x, NULL, 1, _RET_IP_);
trace_kmem_cache_free(_RET_IP_, x);
+ trace_my_debug_slub_free(s);
}
EXPORT_SYMBOL(kmem_cache_free);
```

4: then we can find below info from the trace log,

```
echo 0 > /sys/kernel/debug/tracing/tracing_on
echo 1024 > /sys/kernel/debug/tracing/buffer_size_kb
```

```
echo "" >> /sys/kernel/debug/tracing/set_event
echo 1 > /sys/kernel/debug/tracing/tracing_on
cat /sys/kernel/debug/tracing/trace_pipe > /sdcard/ftrace.txt &     // get trace log
```

dumpsys-17178 [005] ..... 61327.754025: trace_my_debug_slub_alloc: alloc slub anon_vma = Callers:(kmem_cache_alloc<-do_anonymous_page<-handle_pte_fault<-do_handle_mm_fault<-do_page_fault<-do_translation_fault<-do_mem_abort)

5: then you can use the sort2.c I provide in this KAB to sort the potential leak tasks.

```
gcc sort2.c  -o sort_slab
./sort_slab    ftrace_15.txt      //  最新版本已做数据排序
```

```
       procrank-7328     leak 79 times
       procrank-7201     leak 79 times
       procrank-6959     leak 79 times
       procrank-6666     leak 79 times
       procrank-5889     leak 79 times
       procrank-583      leak 80 times
       procrank-10892    leak 80 times
    tloc_daemon-1689     leak 83 times
            sh-5240      leak 89 times
            sh-5244      leak 92 times
           cat-5309      leak 94 times
           cat-5243      leak 94 times
           cat-5195      leak 99 times
           cat-5301      leak 102 times
           cat-5189      leak 107 times
           cat-5305      leak 109 times
           cat-5241      leak 110 times
            sh-5297      leak 113 times
            sh-21281     leak 75 times
            sh-13040     leak 79 times
            sh-13053     leak 192 times
       procrank-22948    leak 428 times
       procrank-15484    leak 435 times
       procrank-5772     leak 445 times
       procrank-27076    leak 448 times
           adbd-4630     leak 450 times
       procrank-6484     leak 451 times
            sh-21355     leak 465 times
       procrank-11770    leak 466 times
  binder:2464_2-3268     leak 656 times
ThreadPoolManag-2847     leak 9812 times
          <...>-5163     leak 22797 times
            sh-5163      leak 705272 times

==============
----total alloc 1457855----------
----total free 88----------
----total leak 1457767----------
```

```
/*
* 使用方法，编译好后 如 gcc sort2.c   -o debug_slab_sort
*
* ./debug_slab_sort   ftrace.txt
*
*   其中可自行调整 printList 函数来决定输出的数据，目前已做数据排序
*
*/



#include  <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct Node {
        char name[100];
        unsigned long int alloc_count;
        unsigned long int free_count;
        struct Node* next;
} Node;
#define LEAK_TIMES 0
Node* head = NULL;
Node* phead = NULL;
int check_alloc_or_free(char * strings) {
        char str1[] = "trace_my_debug_slub_alloc";
        char str2[] = "trace_my_debug_slub_free";
        char *ptr = strstr(strings, str1);
        if (ptr) {
                return 1;
        }
        ptr = strstr(strings, str2);
        if (ptr) {
                return 2;
        }
        return 0;
}
char* get_task(char * strings) {
        const char s[2] = "[";
```

```
    char *token;
    token = strtok(strings, s);
    return token;
}
Node* createNode(char* name) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Error! Unable to allocate memory.\n");
        return NULL;
    }
    strcpy(newNode->name,name);
    newNode->alloc_count=0;
    newNode->free_count=0;
    newNode->next = NULL;
    return newNode;
}
void appendNode( char*name, int state) {
    Node* newNode = createNode(name);
    Node* temp = NULL;
    if (head == NULL) {
        head = newNode;
        phead = newNode;
        if(state == 1)
            head->alloc_count++;
        if(state == 2)
        {
            head->free_count++;
        }
        return;
    } else {
        temp = phead;
        while (temp != NULL) {
            if((0 == strcmp (temp->name , newNode->name )))
            {
                if(state == 1)
                {
                    temp->alloc_count++;
                }
                if(state == 2)
                {
                    temp->free_count++;
                }
                free(newNode);
                return;
```

```
                }
                temp = temp->next;
            }
            temp = phead;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
            if(state == 1)
            {
                newNode->alloc_count++;
            }
            if(state == 2)
            {
                newNode->free_count++;
            }
        }
}
//=========================================
void insertionSort(Node** head_ref) {
    Node* sorted = NULL;
    Node* current = *head_ref;

    while (current != NULL) {
        Node* next = current->next;
        if (sorted == NULL || sorted->alloc_count >= current->alloc_count) {
            current->next = sorted;
            sorted = current;
        } else {
            Node* temp = sorted;
            while (temp->next != NULL && temp->next->alloc_count < current->alloc_count) {
                temp = temp->next;
            }
            current->next = temp->next;
            temp->next = current;
        }
        current = next;
    }
    *head_ref = sorted;
}


//=====================================
void printList(Node* head) {
```

```
unsigned long int total_alloc =0;
unsigned long int total_free =0;

	while (phead != NULL) {
//		printf("================\n");
//		 printf("%s \n", phead->name);	// 打印对应 task name
//		 printf("allos %lu times\n", phead->alloc_count);	// 打印对应 task alloc 的次数
//		 printf("free %lu times\n", phead->free_count);	// 打印对应 task free 的次数
				total_alloc = total_alloc + phead->alloc_count; // 计算所有 task 加起来 alloc 的次数
				total_free = total_free + phead->free_count;// 计算所有 task 加起来 free 的次数
		if (phead->alloc_count > phead->free_count)
		{
			if ( (phead->alloc_count - phead->free_count) > LEAK_TIMES )	// 大量数据的 ftrace log 可筛选泄漏次数大于 LEAK_TIMES 次的 task，可自行调整打印输出
				printf("%s leak %lu times\n", phead->name,(phead->alloc_count - phead->free_count));

		}
		phead = phead->next;
	}
	printf("\n");
		printf("================\n");
		printf("====total alloc %lu===========\n",total_alloc);
		printf("====total free %lu===========\n",total_free);
		printf("====total leak %lu===========\n",(total_alloc-total_free));
}
int add_to_list(char * name,int state) {
	if (state==1 || state==2)
	{
		appendNode(name , state);
	}
	return 0;
}
void free_Node()
{
	printList(phead);
	Node* current = head;
	while (current != NULL) {
		Node* next = current->next;
		free(current);
		current = next;
```

```
        }
}
int main(int argc, char **argv) {
        char * debug_task_name;
        int state=0;
        FILE *file = fopen(argv[1], "r");
        if (file == NULL) {
                perror("Error opening file");
                return 1;
        }
        char line[1024];
        while (fgets(line, sizeof(line), file)) {
                state = check_alloc_or_free(line);
                debug_task_name = (void *) get_task(line);
                if((debug_task_name != NULL )&& (state ==1 || state ==2))
                {
                        add_to_list(debug_task_name,state);
                }
        }
printf("===============\n");
                insertionSort(&phead);
  printf("===============\n");
        free_Node();
        fclose(file);
        return 0;
}
```