

佳 > 正文

记一个程序oom的排查过程

原创leejia19892017-07-31 20:45:32评论(0)

1058人阅读

一，背景

收到应用服务报警，然后登录上服务器查看原因，发现进程不再了。

二，问题分析

1，那么判断进程被干掉的原因如下：

（1），机器重启了

通过uptime看机器并未重启

（2），程序有bug自动退出了

通过查询程序的error log，并未发现异常

（3），被别人干掉了

由于程序比较消耗内存，故猜想是不是oom了，被系统给干掉了。所以查messages日志，发现的确是oom了：

Jul 27 13:29:54 kernel: Out of memory: Kill process 17982 (java) score 77 or sacrifice child

2，通过oom详细信息输出分析被干掉的具体原因

```
1 [511250.458988] mysqld invoked oom-killer: gfp_mask=0x201da, order=0, oom_score_adj=0
2 [511250.458993] mysqld cpuset=/ mems_allowed=0
3 [511250.458996] CPU: 7 PID: 30063 Comm: mysqld Not tainted 3.10.0-514.21.2.el7.x86_64 #
4 [511250.458997] Hardware name: Alibaba Cloud Alibaba Cloud ECS, BIOS rel-1.7.5-0-ge5148
5 [511250.458999] ffff88056236bec0 00000000040f4df68 ffff88044b76b910 ffffffff81687073
6 [511250.459002] ffff88044b76b9a0 ffffffff8168201e ffffffff810eb0dc ffff88081ae80c20
7 [511250.459004] ffff88081ae80c38 ffff88044b76b9f8 ffff88056236bec0 0000000000000000
8 [511250.459007] Call Trace:
9 [511250.459015] [<ffffffff81687073>] dump_stack+0x19/0x1b
10 [511250.459020] [<ffffffff8168201e>] dump_header+0x8e/0x225
11 [511250.459026] [<ffffffff810eb0dc>] ? ktime_get_ts64+0x4c/0xf0
12 [511250.459033] [<ffffffff81184cfe>] oom_kill_process+0x24e/0x3c0
13 [511250.459035] [<ffffffff8118479d>] ? oom_unkillable_task+0xcd/0x120
14 [511250.459038] [<ffffffff81184846>] ? find_lock_task_mm+0x56/0xc0
15 [511250.459042] [<ffffffff81093c0e>] ? has_capability_noaudit+0x1e/0x30
16 [511250.459045] [<ffffffff81185536>] out_of_memory+0x4b6/0x4f0
17 [511250.459047] [<ffffffff81682b27>] __alloc_pages_slowpath+0x5d7/0x725
18 [511250.459051] [<ffffffff8118b645>] __alloc_pages_nodemask+0x405/0x420
19 [511250.459055] [<ffffffff811cf94a>] alloc_pages_current+0xaa/0x170
20 [511250.459058] [<ffffffff81180bd7>] __page_cache_alloc+0x97/0xb0
21 [511250.459060] [<ffffffff81183750>] filemap_fault+0x170/0x410
22 [511250.459078] [<ffffffffa01b5016>] ext4_filemap_fault+0x36/0x50 [ext4]
23 [511250.459082] [<ffffffff811ac84c>] __do_fault+0x4c/0xc0
24 [511250.459084] [<ffffffff811acce3>] do_read_fault.isra.42+0x43/0x130
25 [511250.459087] [<ffffffff811b1471>] handle_mm_fault+0x6b1/0x1040
26 [511250.459091] [<ffffffff810f55c0>] ? futex_wake+0x80/0x160
27 [511250.459096] [<ffffffff81692c04>] __do_page_fault+0x154/0x450
28 [511250.459098] [<ffffffff81692fe6>] trace_do_page_fault+0x56/0x150
29 [511250.459101] [<ffffffff8169268b>] do_async_page_fault+0x1b/0xd0
30 [511250.459103] [<ffffffff8168f178>] async_page_fault+0x28/0x30
31 [511250.459104] Mem-Info:
32 [511250.459109] active_anon:7922627 inactive_anon:1653 isolated_anon:0
33 [511250.459112] active_file:1675 inactive_file:2820 isolated_file:0
34 [511250.459117] unevictable:0 dirty:11 writeback:2 unstable:0
35 [511250.459120] slab_reclaimable:61817 slab_unreclaimable:25990
36 [511250.459123] mapped:3607 shmem:4602 pagetables:42625 bounce:0
37 [511250.459126] free:50021 free_pcp:149 free_cma:0
38 [511250.459127] Node 0 DMA free:15892kB min:32kB low:40kB high:48kB active_anon:0kB ina
39 [511250.459129] lowmem_reserve[]: 0 2814 31994 31994
40 [511250.459131] Node 0 DMA32 free:119704kB min:5940kB low:7424kB high:8908kB active_anc
41 [511250.459133] lowmem_reserve[]: 0 0 29180 29180
42 [511250.459135] Node 0 Normal free:63896kB min:61608kB low:77008kB high:92412kB active_
43 [511250.459137] lowmem_reserve[]: 0 0 0 0
44 [511250.459139] Node 0 DMA: 1*4kB (U) 0*8kB 1*16kB (U) 0*32kB 2*64kB (U) 1*128kB (U) 1*
45 [511250.459141] Node 0 DMA32: 9372*4kB (UEM) 2427*8kB (UEM) 1179*16kB (UEM) 369*32kB (U
```

佳

88	28W+	37
文章	人气	评论



leejia1989

关注私信

好的文章，和好友一起

TA的其他文章

- centos7 systemctl status sen
- linux设置http/https proxy及忽
- linux kernel crash问题分析解
- linux系统收到SYN但不回SYN
- lua table引用问题

七日热门

- 大数据采集、清洗、处理：使
- No.5 Nginx急速入门
- 掌握MySQL数据库这些优化技
- Kubernetes 1.10中部署dashb
- Python重启深信服设备

```

46 [511250.459154] Node 0 Normal: 1540*4kB (UE) 6148*8kB (UE) 503*16kB (UE) 0*32kB 0*64kB
47 [511250.459162] Node 0 hugepages_total=0 hugepages_free=0 hugepages_surp=0 hugepages_si
48 [511250.459163] Node 0 hugepages_total=0 hugepages_free=0 hugepages_surp=0 hugepages_si
49 [511250.459164] 9275 total pagecache pages
50 [511250.459166] 0 pages in swap cache
51 [511250.459167] Swap cache stats: add 0, delete 0, find 0/0
52 [511250.459168] Free swap = 0kB
53 [511250.459168] Total swap = 0kB
54 [511250.459169] 8388478 pages RAM
55 [511250.459170] 0 pages HighMem/MovableOnly
56 [511250.459171] 193375 pages reserved
57 [511250.459172] [ pid ] uid tgid total_vm rss nr_ptes swapents oom_score_adj na
58 [511250.459178] [ 444 ] 0 444 30482 118 63 0 0 sy
59 [511250.459180] [ 476 ] 0 476 14365 114 28 0 -1000 au
60 [511250.459182] [ 508 ] 0 508 5315 75 14 0 0 ir
61 [511250.459184] [ 509 ] 998 509 132421 1908 50 0 0 pc
62 [511250.459186] [ 510 ] 0 510 6686 196 17 0 0 sy
63 [511250.459188] [ 514 ] 81 514 6672 148 16 0 -900 db
64 [511250.459189] [ 592 ] 0 592 6972 52 18 0 0 at
65 [511250.459191] [ 595 ] 0 595 31969 188 17 0 0 cr
66 [511250.459193] [ 607 ] 0 607 28020 44 11 0 0 ag
67 [511250.459195] [ 1036 ] 0 1036 138798 3179 89 0 0 tu
68 [511250.459197] [ 1037 ] 0 1037 174118 357 182 0 0 rs
69 [511250.459198] [ 1089 ] 38 1089 7865 174 19 0 0 nt
70 [511250.459200] [ 4714 ] 0 4714 26866 243 54 0 -1000 ss
71 [511250.459202] [ 6624 ] 0 6624 920 100 4 0 0 al
72 [511250.459204] [19284] 0 19284 8386 171 21 0 0 Al
73 [511250.459206] [19335] 0 19335 34887 1367 64 0 0 Al
74 [511250.459208] [21657] 26 21657 59097 1539 52 0 -1000 pc
75 [511250.459210] [21658] 26 21658 48503 264 43 0 0 pc
76 [511250.459212] [21660] 26 21660 59124 2338 52 0 0 pc
77 [511250.459213] [21661] 26 21661 59097 332 48 0 0 pc
78 [511250.459215] [21662] 26 21662 59097 537 47 0 0 pc
79 [511250.459217] [21663] 26 21663 59328 513 50 0 0 pc
80 [511250.459218] [21664] 26 21664 49067 317 44 0 0 pc
81 [511250.459220] [ 7276 ] 0 7276 32471 164 16 0 0 sc
82 [511250.459222] [ 7277 ] 0 7277 29357 123 13 0 0 ba
83 [511250.459223] [ 7388 ] 0 7388 4303 1880 12 0 0 sa
84 [511250.459225] [ 7747 ] 0 7747 32504 200 16 0 0 sc
85 [511250.459226] [ 7748 ] 0 7748 29357 122 14 0 0 ba
86 [511250.459228] [ 7781 ] 0 7781 8051 4108 20 0 0 ta
87 [511250.459230] [ 9897 ] 0 9897 3062553 270245 774 0 0 ja
88 [511250.459231] [ 9937 ] 26 9937 59406 657 53 0 0 pc
89 [511250.459233] [ 9940 ] 26 9940 60212 2570 57 0 0 pc
90 [511250.459235] [ 9997 ] 26 9997 60098 2346 56 0 0 pc
91 [511250.459236] [10076] 26 10076 59574 964 54 0 0 pc
92 [511250.459238] [10077] 26 10077 59618 1006 54 0 0 pc
93 [511250.459239] [10078] 26 10078 59617 1005 54 0 0 pc
94 [511250.459241] [11611] 0 11611 60826 4190 73 0 0 py
95 [511250.459243] [11619] 0 11619 348938 6222 118 0 0 py
96 [511250.459245] [12396] 26 12396 60086 2078 56 0 0 pc
97 [511250.459246] [12499] 1001 12499 1448783 99046 328 0 0 ja
98 [511250.459248] [12600] 1003 12600 2226317 312995 847 0 0 ja
99 [511250.459249] [29241] 0 29241 78180 1320 101 0 0 ph
100 [511250.459251] [29242] 1004 29242 135239 2687 108 0 0 ph
101 [511250.459253] [29243] 1004 29243 134924 2408 108 0 0 ph
102 [511250.459255] [29244] 1004 29244 135371 2707 108 0 0 ph
103 [511250.459256] [29245] 1004 29245 143755 11294 125 0 0 ph
104 [511250.459258] [29246] 1004 29246 135367 2706 108 0 0 ph
105 [511250.459260] [29826] 27 29826 28792 86 13 0 0 my
106 [511250.459261] [30051] 27 30051 322930 39761 133 0 0 my
107 [511250.459263] [30234] 0 30234 11365 125 22 0 -1000 sy
108 [511250.459264] [11182] 0 11182 82780 5702 114 0 0 sa
109 [511250.459266] [11193] 0 11193 171406 8289 144 0 0 sa
110 [511250.459268] [11195] 0 11195 101432 5712 110 0 0 sa
111 [511250.459269] [29678] 1004 29678 140301 7833 118 0 0 ph
112 [511250.459271] [29998] 1004 29998 134983 2404 108 0 0 ph
113 [511250.459273] [11833] 0 11833 69721 2098 58 0 0 py
114 [511250.459275] [32113] 26 32113 60131 2012 56 0 0 pc
115 [511250.459276] [ 1017 ] 1004 1017 135410 2748 108 0 0 ph
116 [511250.459278] [11915] 1004 11915 144263 11778 126 0 0 ph
117 [511250.459280] [ 5999 ] 0 5999 8115 3139 20 0 0 ta
118 [511250.459281] [21572] 1004 21572 134919 2379 108 0 0 ph
119 [511250.459283] [21752] 1004 21752 143751 11276 125 0 0 ph
120 [511250.459285] [ 2977 ] 1004 2977 134920 2406 107 0 0 ph
121 [511250.459286] [ 9217 ] 0 9217 330989 183882 550 0 0 py
122 [511250.459288] [ 2008 ] 1004 2008 135816 3328 109 0 0 ph
123 [511250.459290] [25089] 1000 25089 2800777 187701 710 0 0 ja
124 [511250.459291] [25405] 1000 25405 1335611 105668 366 0 0 ja
125 [511250.459293] [26033] 1000 26033 1680746 96082 367 0 0 ja
126 [511250.459295] [26112] 1000 26112 1148121 61227 230 0 0 ja
127 [511250.459296] [14446] 0 14446 31082 540 56 0 0 ng
128 [511250.459298] [14447] 1004 14447 31278 739 58 0 0 ng
129 [511250.459299] [14448] 1004 14448 31278 725 58 0 0 ng
130 [511250.459301] [14449] 1004 14449 31278 714 58 0 0 ng
131 [511250.459303] [14450] 1004 14450 31278 715 58 0 0 ng

```

132	[511250.459304]	[14451]	1004	14451	31245	705	58	0	0	ng
133	[511250.459306]	[14452]	1004	14452	31245	696	58	0	0	ng
134	[511250.459307]	[14453]	1004	14453	31278	712	58	0	0	ng
135	[511250.459309]	[14454]	1004	14454	31245	728	58	0	0	ng
136	[511250.459310]	[14455]	1004	14455	31278	730	58	0	0	ng
137	[511250.459312]	[14456]	1004	14456	31278	718	58	0	0	ng
138	[511250.459314]	[14457]	1004	14457	31245	707	58	0	0	ng
139	[511250.459315]	[14458]	1004	14458	31278	722	58	0	0	ng
140	[511250.459317]	[14459]	1004	14459	31278	717	58	0	0	ng
141	[511250.459318]	[14460]	1004	14460	31245	688	58	0	0	ng
142	[511250.459320]	[14462]	1004	14462	31278	712	58	0	0	ng
143	[511250.459321]	[14463]	1004	14463	31278	736	58	0	0	ng
144	[511250.459323]	[14571]	0	14571	3222105	119555	906	0	0	py
145	[511250.459325]	[13969]	0	13969	134928	8719	143	0	0	sa
146	[511250.459326]	[13982]	0	13982	78554	5647	100	0	0	sa
147	[511250.459328]	[13985]	0	13985	116150	8034	134	0	0	sa
148	[511250.459330]	[13989]	0	13989	151040	38826	238	0	0	sa
149	[511250.459331]	[13990]	0	13990	103527	12904	148	0	0	sa
150	[511250.459333]	[14067]	0	14067	280592	9651	151	0	0	sa
151	[511250.459334]	[14072]	0	14072	135099	9889	141	0	0	sa
152	[511250.459336]	[14220]	0	14220	134928	8828	135	0	0	sa
153	[511250.459338]	[14221]	0	14221	1941362	9675	332	0	0	sa
154	[511250.459339]	[14228]	0	14228	175360	9657	148	0	0	sa
155	[511250.459341]	[14268]	0	14268	175362	9655	148	0	0	sa
156	[511250.459343]	[14314]	0	14314	175361	9662	148	0	0	sa
157	[511250.459344]	[14327]	0	14327	175363	9663	148	0	0	sa
158	[511250.459346]	[14329]	0	14329	175363	9666	148	0	0	sa
159	[511250.459347]	[14330]	0	14330	175364	9666	148	0	0	sa
160	[511250.459349]	[14331]	0	14331	175365	9666	148	0	0	sa
161	[511250.459350]	[14334]	0	14334	175366	9670	148	0	0	sa
162	[511250.459352]	[14338]	0	14338	175366	9669	148	0	0	sa
163	[511250.459354]	[14340]	0	14340	175366	9674	148	0	0	sa
164	[511250.459355]	[14345]	0	14345	175367	9679	148	0	0	sa
165	[511250.459357]	[14349]	0	14349	175367	9675	148	0	0	sa
166	[511250.459358]	[14350]	0	14350	175367	9671	148	0	0	sa
167	[511250.459360]	[14354]	0	14354	175368	9672	148	0	0	sa
168	[511250.459362]	[14357]	0	14357	175369	9678	148	0	0	sa
169	[511250.459363]	[14358]	0	14358	175369	9673	148	0	0	sa
170	[511250.459365]	[14362]	0	14362	175369	9677	148	0	0	sa
171	[511250.459366]	[14364]	0	14364	175370	9680	148	0	0	sa
172	[511250.459368]	[14365]	0	14365	175371	9681	148	0	0	sa
173	[511250.459369]	[14368]	0	14368	175371	9676	148	0	0	sa
174	[511250.459371]	[14370]	0	14370	175371	9674	148	0	0	sa
175	[511250.459372]	[14372]	0	14372	175372	9682	148	0	0	sa
176	[511250.459374]	[14376]	0	14376	175373	9682	148	0	0	sa
177	[511250.459375]	[14377]	0	14377	175374	9676	148	0	0	sa
178	[511250.459377]	[14378]	0	14378	175374	9689	148	0	0	sa
179	[511250.459379]	[14380]	0	14380	175650	9716	149	0	0	sa
180	[511250.459381]	[14384]	0	14384	175375	9690	148	0	0	sa
181	[511250.459382]	[14385]	0	14385	175375	9685	148	0	0	sa
182	[511250.459384]	[14401]	0	14401	175376	9687	148	0	0	sa
183	[511250.459385]	[14404]	0	14404	175377	9685	148	0	0	sa
184	[511250.459387]	[14413]	0	14413	175377	9685	148	0	0	sa
185	[511250.459388]	[14420]	0	14420	175377	9687	148	0	0	sa
186	[511250.459390]	[14421]	0	14421	175378	9686	148	0	0	sa
187	[511250.459392]	[14424]	0	14424	175380	9693	148	0	0	sa
188	[511250.459393]	[14428]	0	14428	175380	9689	148	0	0	sa
189	[511250.459395]	[14435]	0	14435	175382	9698	148	0	0	sa
190	[511250.459396]	[14437]	0	14437	175382	9694	148	0	0	sa
191	[511250.459398]	[14439]	0	14439	175383	9692	148	0	0	sa
192	[511250.459399]	[14442]	0	14442	175384	9694	148	0	0	sa
193	[511250.459401]	[14445]	0	14445	175385	9692	148	0	0	sa
194	[511250.459403]	[14465]	0	14465	175385	9695	148	0	0	sa
195	[511250.459404]	[14473]	0	14473	175385	9695	148	0	0	sa
196	[511250.459406]	[14486]	0	14486	175386	9697	148	0	0	sa
197	[511250.459407]	[14489]	0	14489	175386	9699	148	0	0	sa
198	[511250.459409]	[14503]	0	14503	175386	9699	148	0	0	sa
199	[511250.459410]	[14513]	0	14513	175387	9700	148	0	0	sa
200	[511250.459412]	[14520]	0	14520	175388	9704	148	0	0	sa
201	[511250.459414]	[14523]	0	14523	175389	9700	148	0	0	sa
202	[511250.459415]	[14525]	0	14525	175389	9703	148	0	0	sa
203	[511250.459417]	[14527]	0	14527	175390	9710	148	0	0	sa
204	[511250.459419]	[14533]	0	14533	175390	9705	148	0	0	sa
205	[511250.459420]	[14539]	0	14539	175390	9709	148	0	0	sa
206	[511250.459422]	[14590]	0	14590	175391	9713	148	0	0	sa
207	[511250.459423]	[14598]	0	14598	175390	9705	148	0	0	sa
208	[511250.459425]	[14613]	0	14613	175391	9705	148	0	0	sa
209	[511250.459426]	[14624]	0	14624	175392	9713	148	0	0	sa
210	[511250.459428]	[14630]	0	14630	175392	9707	148	0	0	sa
211	[511250.459429]	[14634]	0	14634	175393	9707	148	0	0	sa
212	[511250.459431]	[14652]	0	14652	175393	9709	148	0	0	sa
213	[511250.459433]	[14677]	0	14677	175394	9708	148	0	0	sa
214	[511250.459434]	[14679]	0	14679	175394	9711	148	0	0	sa
215	[511250.459436]	[14709]	0	14709	175395	9713	148	0	0	sa
216	[511250.459438]	[14718]	0	14718	175396	9710	148	0	0	sa
217	[511250.459439]	[14723]	0	14723	175396	9710	148	0	0	sa

218	[511250.459441]	[14746]	0	14746	175396	9716	148	0	0	sa
219	[511250.459443]	[14752]	0	14752	175461	9717	148	0	0	sa
220	[511250.459444]	[14791]	0	14791	175398	9715	148	0	0	sa
221	[511250.459446]	[14799]	0	14799	175397	9720	148	0	0	sa
222	[511250.459447]	[14804]	0	14804	175472	9721	148	0	0	sa
223	[511250.459449]	[14835]	0	14835	175462	9729	148	0	0	sa
224	[511250.459450]	[14840]	0	14840	175463	9735	148	0	0	sa
225	[511250.459452]	[14864]	0	14864	175463	9727	148	0	0	sa
226	[511250.459453]	[14882]	0	14882	175464	9731	148	0	0	sa
227	[511250.459455]	[14893]	0	14893	175465	9731	148	0	0	sa
228	[511250.459456]	[14899]	0	14899	175465	9720	148	0	0	sa
229	[511250.459458]	[14906]	0	14906	175466	9721	148	0	0	sa
230	[511250.459460]	[14910]	0	14910	175402	9723	148	0	0	sa
231	[511250.459461]	[14984]	0	14984	175466	9725	148	0	0	sa
232	[511250.459463]	[14988]	0	14988	175467	9735	148	0	0	sa
233	[511250.459464]	[14992]	0	14992	175468	9734	148	0	0	sa
234	[511250.459466]	[15072]	0	15072	175468	9735	148	0	0	sa
235	[511250.459467]	[15101]	0	15101	175468	9731	148	0	0	sa
236	[511250.459469]	[15129]	0	15129	175469	9733	148	0	0	sa
237	[511250.459470]	[15143]	0	15143	175469	9737	148	0	0	sa
238	[511250.459472]	[15168]	0	15168	175470	9740	148	0	0	sa
239	[511250.459474]	[15181]	0	15181	175474	9744	148	0	0	sa
240	[511250.459475]	[15219]	0	15219	175474	9734	148	0	0	sa
241	[511250.459477]	[15223]	0	15223	175477	9753	148	0	0	sa
242	[511250.459479]	[15259]	0	15259	175475	9734	148	0	0	sa
243	[511250.459481]	[15266]	0	15266	175476	9735	148	0	0	sa
244	[511250.459482]	[15322]	0	15322	175476	9736	148	0	0	sa
245	[511250.459493]	[15350]	0	15350	175476	9745	148	0	0	sa
246	[511250.459495]	[15366]	0	15366	175477	9743	148	0	0	sa
247	[511250.459497]	[15380]	0	15380	175506	9745	148	0	0	sa
248	[511250.459498]	[15399]	0	15399	175754	9769	149	0	0	sa
249	[511250.459500]	[15407]	0	15407	175479	9747	148	0	0	sa
250	[511250.459501]	[15447]	0	15447	175479	9742	148	0	0	sa
251	[511250.459503]	[15450]	0	15450	175479	9751	148	0	0	sa
252	[511250.459504]	[15454]	0	15454	175481	9747	148	0	0	sa
253	[511250.459506]	[15462]	0	15462	175480	9748	148	0	0	sa
254	[511250.459508]	[23316]	1000	23316	3085650	27853	144	0	0	ja
255	[511250.459509]	[23319]	1000	23319	3085650	27289	144	0	0	ja
256	[511250.459511]	[23348]	1000	23348	3085650	27778	142	0	0	ja
257	[511250.459512]	[23351]	1000	23351	3085650	26840	141	0	0	ja
258	[511250.459514]	[23373]	1000	23373	3085650	27380	143	0	0	ja
259	[511250.459515]	[23406]	1000	23406	3085650	26933	143	0	0	ja
260	[511250.459517]	[23425]	1000	23425	3085650	27371	142	0	0	ja
261	[511250.459518]	[23445]	1000	23445	3085650	27861	141	0	0	ja
262	[511250.459520]	[23476]	1000	23476	3085650	27716	143	0	0	ja
263	[511250.459522]	[23497]	1000	23497	3085650	27902	144	0	0	ja
264	[511250.459523]	[23690]	1000	23690	2049475	328916	865	0	0	ja
265	[511250.459525]	[23691]	1000	23691	2082756	356868	894	0	0	ja
266	[511250.459527]	[23693]	1000	23693	2027460	612751	1357	0	0	ja
267	[511250.459528]	[23712]	1000	23712	2027460	610571	1348	0	0	ja
268	[511250.459529]	[23754]	1000	23754	2049474	337457	886	0	0	ja
269	[511250.459531]	[23785]	1000	23785	2049474	330831	864	0	0	ja
270	[511250.459533]	[23805]	1000	23805	2027460	615907	1366	0	0	ja
271	[511250.459534]	[23828]	1000	23828	2027460	610191	1346	0	0	ja
272	[511250.459536]	[23855]	1000	23855	2629446	589971	1351	0	0	ja
273	[511250.459537]	[23860]	1000	23860	2328022	144465	519	0	0	ja
274	[511250.459539]	[13536]	1004	13536	134981	2523	108	0	0	ph
275	[511250.459540]	[1813]	0	1813	1481817	46140	246	0	0	ja
276	[511250.459542]	[3187]	0	3187	1481817	53461	253	0	0	ja
277	[511250.459544]	[2993]	26	2993	59779	1712	55	0	0	pc
278	[511250.459546]	[3059]	1000	3059	3085528	16411	141	0	0	ja
279	[511250.459547]	[3146]	1000	3146	2027460	211779	628	0	0	ja
280	[511250.459549]	[17982]	996	17982	4950828	635077	1629	0	0	ja
281	[511250.459551]	[16433]	0	16433	37607	360	74	0	0	ss
282	[511250.459553]	[16436]	0	16436	29390	141	13	0	0	ba
283	[511250.459554]	[16466]	0	16466	29390	136	14	0	0	ba
284	[511250.459556]	[22511]	0	22511	36968	433	72	0	0	ss
285	[511250.459558]	[22515]	0	22515	19016	257	40	0	0	ss
286	[511250.459560]	[22519]	0	22519	19107	350	39	0	0	ss
287	[511250.459562]	[22522]	0	22522	19016	259	38	0	0	ss
288	[511250.459563]	[24770]	0	24770	38342	657	30	0	0	vi
289	[511250.459565]	[24781]	0	24781	45009	303	41	0	0	cr
290	[511250.459566]	[24784]	0	24784	91360	8641	134	0	0	py
291	[511250.459568]	[24932]	0	24932	28791	45	13	0	0	sh
292	[511250.459570]	[24933]	0	24933	93538	7284	104	0	0	an
293	[511250.459571]	[24942]	0	24942	94424	7584	103	0	0	an
294	[511250.459573]	[24943]	0	24943	96455	9707	107	0	0	an
295	[511250.459574]	[24944]	0	24944	94436	7599	103	0	0	an
296	[511250.459576]	[24945]	0	24945	16336	70	33	0	0	ss
297	[511250.459578]	[24946]	0	24946	16336	71	33	0	0	ss
298	[511250.459579]	[24947]	0	24947	16336	69	30	0	0	ss
299	[511250.459581]	Out of memory: Kill process 17982 (java) score 77 or sacrifice child								
300	[511250.459642]	Killed process 17982 (java) total-vm:19803312kB, anon-rss:2540308kB, fi								

(1) mysqld触发了oom killer，既mysqld要申请的内存大于了系统可用的物理内存大小。/proc/sys/vm/min_free_kbyte s参数来控制，当系统可用内存（不包含buffer和cache）小于这个值的时候，系统会启动内核线程kswapd来对内存进行回收。而还是触发了oom killer，则表明内存真的不够用了或者在内存回收前或者回收中直接触发了oom killer。

(2) 如下的输出表明了申请了3次内存都没有成功

```
1 [511250.459112] Node 0 DMA free:15892kB min:32kB low:40kB high:48kB active_anon:0kB inac
2 [511250.459117] lowmem_reserve[]: 0 2814 31994 31994
3 [511250.459120] Node 0 DMA32 free:119704kB min:5940kB low:7424kB high:8908kB active_anon
4 [511250.459124] lowmem_reserve[]: 0 0 29180 29180
5 [511250.459127] Node 0 Normal free:63896kB min:61608kB low:77008kB high:92412kB active_a
6 [511250.459131] lowmem_reserve[]: 0 0 0 0
7 [511250.459134] Node 0 DMA: 1*4kB (U) 0*8kB 1*16kB (U) 0*32kB 2*64kB (U) 1*128kB (U) 1*2
8 [511250.459144] Node 0 DMA32: 9372*4kB (UEM) 2427*8kB (UEM) 1179*16kB (UEM) 369*32kB (UE
9 [511250.459154] Node 0 Normal: 1540*4kB (UE) 6148*8kB (UE) 503*16kB (UE) 0*32kB 0*64kB 0
10 [511250.459162] Node 0 hugepages_total=0 hugepages_free=0 hugepages_surp=0 hugepages_siz
11 [511250.459163] Node 0 hugepages_total=0 hugepages_free=0 hugepages_surp=0 hugepages_siz
12 [511250.459164] 9275 total pagecache pages
```

(3) 被干掉进程信息

如需输出确认了被kill的进程为17982

```
1 [511250.459581] Out of memory: Kill process 17982 (java) score 77 or sacrifice child
2 [511250.459642] Killed process 17982 (java) total-vm:19803312kB, anon-rss:2540308kB, file
```

如下为17982进程占用的内存页数量635077，换算为内存占用量是635077*4096=2GB

```
1 [511250.459172] [ pid ] uid tgid total_vm rss nr_ptes swapents oom_score_adj name
2 [511250.459549] [17982] 996 17982 4950828 635077 1629 0
```

每列的含义为：

- pid进程ID。
- uid用户ID。
- tgid线程组ID。
- total_vm虚拟内存使用(单位为4 kB内存页)
- rss居民 memory 使用(单位4 kB内存页)
- nr_ptes页表项
- swapents交换条目
- oom_score_adj通常为0;较低的数字表示当调用OOM杀手时，进程将不太可能死亡。

(4) 分析系统所有进程rss内存（rss为程序实际使用物理内存，单位为4kB内存页）

把oom输出中进程的rss内存相加，发现已经使用了32g，那就说明系统是内存耗尽了才触发的oom killer。而通过分析，发现java程序占用的内存总量为26g，是最大头。

三，解决

使用的解决办法：

- 1，限制java进程的max heap，并且降低java程序的worker数量，从而降低内存使用
- 2，发现系统没有开启swap，给系统加了8G的swap空间

其它解决办法（不推荐），不允许内存申请过量：

```
# echo "2" > /proc/sys/vm/overcommit_memory
# echo "80" > /proc/sys/vm/overcommit_ratio
```

四，扩展

1，overcommit_memory(/proc/sys/vm/overcommit_memory)

Linux是允许memory overcommit的，只要你来申请内存我就给你，寄希望于进程实际上用不到那么多内存，但万一用到那么多了呢？那就会发生类似“银行挤兑”的危机，现金(内存)不足了。Linux设计了一个OOM killer机制(OOM = out-of-memory)来处理这种危机：挑选一个进程出来杀死，以腾出部分内存，如果还不够就继续杀...也可通过设置内核参数 vm.panic_on_oom 使得发生OOM时自动重启系统。这都是有风险的机制，重启有可能造成业务中断，杀死

进程也有可能导致业务中断。所以Linux 2.6之后允许通过内核参数 `vm.overcommit_memory` 禁止memory overcommit。

(1) 内核参数 `vm.overcommit_memory` 接受三种取值:

0 – Heuristic overcommit handling. 这是缺省值, 它允许overcommit, 但过于明目张胆的overcommit会被拒绝, 比如m alloc一次性申请的内存大小就超过了系统总内存。Heuristic的意思是“试探式的”, 内核利用某种算法猜测你的内存申请是否合理, 它认为不合理就会拒绝overcommit。

1 – Always overcommit. 允许overcommit, 对内存申请者不拒。内核执行无内存过量使用处理。使用这个设置会增大内存超载的可能性, 但也可以增强大量使用内存任务的性能。

2 – Don't overcommit. 禁止overcommit。内存拒绝等于或者大于总可用 swap 大小以及 `overcommit_ratio` 指定的物理 RAM 比例的内存请求。如果您希望减小内存过度使用的风险, 这个设置就是最好的。

(2) Heuristic overcommit算法在以下函数中实现, 基本上可以这么理解:

单次申请的内存大小不能超过【free memory + free swap + pagecache的大小 + SLAB中可回收的部分】, 否则本次申请就会失败。

(3) 关于禁止overcommit (`vm.overcommit_memory=2`), 需要知道的是, 怎样才算是overcommit呢? kernel设有一个阈值, 申请的内存总数超过这个阈值就算overcommit, 在`/proc/meminfo`中可以看到这个阈值的大小:

```
1 # grep -i commit /proc/meminfo
2 CommitLimit:      5967744 kB
3 Committed_AS:    5363236 kB
```

CommitLimit 就是overcommit的阈值, 申请的内存总数超过CommitLimit的话就算是overcommit。

这个阈值是如何计算出来的呢? 它既不是物理内存的大小, 也不是free memory的大小, 它是通过内核参数`vm.overcommit_ratio`或`vm.overcommit_kbytes`间接设置的, 公式如下:

【CommitLimit = (Physical RAM * `vm.overcommit_ratio` / 100) + Swap】

注:

`vm.overcommit_ratio` 是内核参数, 缺省值是50, 表示物理内存的50%。如果你不想使用比率, 也可以直接指定内存的字节数大小, 通过另一个内核参数 `vm.overcommit_kbytes` 即可;

如果使用了huge pages, 那么需要从物理内存中减去, 公式变成:

$\text{CommitLimit} = ([\text{total RAM}] - [\text{total huge TLB RAM}]) * \text{vm.overcommit_ratio} / 100 + \text{swap}$

参见<https://access.redhat.com/solutions/665023>

`/proc/meminfo`中的 `Committed_AS` 表示所有进程已经申请的内存总大小, (注意是已经申请的, 不是已经分配的), 如果 `Committed_AS` 超过 `CommitLimit` 就表示发生了 overcommit, 超出越多表示 overcommit 越严重。`Committed_AS` 的含义换一种说法就是, 如果要绝对保证不发生OOM (out of memory) 需要多少物理内存。

(4) “sar -r” 是查看内存使用状况的常用工具, 它的输出结果中有两个与overcommit有关, `kbcommit` 和 `%commit`:

`kbcommit`对应`/proc/meminfo`中的 `Committed_AS`;

`%commit`的计算公式并没有采用 `CommitLimit`作分母, 而是 $\text{Committed_AS} / (\text{MemTotal} + \text{SwapTotal})$, 意思是内存申请占物理内存与交换区之和的百分比。

```
1 # sar -r
2 05:00:01 PM kbmemfree kbmemused %memused kbbuffers kbcached kbcommit %commit kbacti
3 05:10:01 PM 160576 3648460 95.78 0 1846212 4939368 62.74 13902
```

2, panic_on_oom(/proc/sys/vm/panic_on_oom)

决定系统出现oom的时候, 要做的操作。接受的三种取值如下:

0 - 默认值, 当出现oom的时候, 触发oom killer

1 - 程序在有cpuset、memory policy、memcg的约束情况下的OOM，可以考虑不panic，而是启动OOM killer。其它情况触发 kernel panic，即系统直接重启

2 - 当出现oom，直接触发kernel panic，即系统直接重启

3, oom_adj、oom_score_adj和oom_score

准确的说这几个参数都是和具体进程相关的，因此它们位于/proc/xxx/目录下（xxx是进程ID）。假设我们选择在出现OOM状况的时候杀死进程，那么一个很自然的问题就浮现出来：到底干掉哪一个呢？内核的算法倒是非常简单，那就是打分（oom_score，注意，该参数是read only的），找到分数最高的就OK了。那么怎么来算分数呢？可以参考内核中的oom_badness函数：

```
1 unsigned long oom_badness(struct task_struct *p, struct mem_cgroup *memcg,
2                          const nodemask_t *nodemask, unsigned long totalpages)
3 {
4     adj = (long)p->signal->oom_score_adj;
5     if (adj == OOM_SCORE_ADJ_MIN) {----- (1)
6         task_unlock(p);
7         return 0;----- (2)
8     }
9     points = get_mm_rss(p->mm) + get_mm_counter(p->mm, MM_SWAPENTS) +
10             atomic_long_read(&p->mm->nr_ptes) + mm_nr_pmds(p->mm);----- (3)
11     task_unlock(p);
12
13     if (has_capability_noaudit(p, CAP_SYS_ADMIN))----- (4)
14         points -= (points * 3) / 100;
15     adj *= totalpages / 1000;----- (5)
16     points += adj;
17
18     return points > 0 ? points : 1;
19 }
```

(1) 对某一个task进行打分（oom_score）主要有两部分组成，一部分是系统打分，主要是根据该task的内存使用情况。另外一部分是用户打分，也就是oom_score_adj了，该task的实际得分需要综合考虑两方面的打分。如果用户将该task的 oom_score_adj设定成OOM_SCORE_ADJ_MIN（-1000）的话，那么实际上就是禁止了OOM killer杀死该进程。

(2) 这里返回了0也就是告知OOM killer，该进程是“good process”，不要干掉它。后面我们可以看到，实际计算分数的时候最低分是1分。

(3) 前面说过了，系统打分就是看物理内存消耗量，主要是三部分，RSS部分，swap file或者swap device上占用的内存情况以及页表占用的内存情况。

(4) root进程有3%的内存使用特权，因此这里要减去那些内存使用量。

(5) 用户可以调整oom_score，具体如何操作呢？oom_score_adj的取值范围是-1000 ~ 1000，0表示用户不调整oom_score，负值表示要在实际打分值上减去一个折扣，正值表示要惩罚该task，也就是增加该进程的oom_score。在实际操作中，需要根据本次内存分配时候可分配内存来计算（如果没有内存分配约束，那么就是系统中的所有可用内存，如果系统支持cpuset，那么这里的可分配内存就是该cpuset的实际额度值）。oom_badness函数有一个传入参数totalpages，该参数就是当时的可分配的内存上限值。实际的分数值（points）要根据oom_score_adj进行调整，例如如果oom_score_adj设定-500，那么表示实际分数要打五折（基数是totalpages），也就是说该任务实际使用的内存要减去可分配的内存上限值的一半。

了解了oom_score_adj和oom_score之后，应该是尘埃落定了，oom_adj是一个旧的接口参数，其功能类似oom_score_adj，为了兼容，目前仍然保留这个参数，当操作这个参数的时候，kernel实际上是会换算成oom_score_adj，有兴趣的同学可以自行了解，这里不再细述了。

plus：

由任意调整的进程衍生的任意进程将继承该进程的 oom_score。例如：如果 sshd 进程不受 oom_killer功能影响，所有由 SSH 会话产生的进程都将不受其影响。这可在出现 OOM 时影响 oom_killer 功能救援系统的能力。

4, min_free_kbytes(/proc/sys/vm/min_free_kbytes)

先看官方解释：

This is used to force the Linux VM to keep a minimum number of kilobytes free. The VM uses this number to compute a watermark[WMARK_MIN] value for each lowmem zone in the system. Each lowmem zone gets a number of reserved free pages based proportionally on its size.

Some minimal amount of memory is needed to satisfy PF_MEMALLOC allocations; if you set this to lower than 1024 KB, your system will become subtly broken, and prone to deadlock under high loads.

Setting this too high will OOM your machine instantly.

解释已经很清楚了，主要有以下几个关键点：

(1). 代表系统所保留空闲内存的最低限。

在系统初始化时会根据内存大小计算一个默认值，计算规则是：

```
1 min_free_kbytes = sqrt(lowmem_kbytes * 16) = 4 * sqrt(lowmem_kbytes) (注: lowmem_kbytes即
```

另外，计算出来的值有最小最大限制，最小为128K，最大为64M。

可以看出，min_free_kbytes随着内存的增大不是线性增长，comments里提到了原因“because network bandwidth does not increase linearly with machine size”。随着内存的增大，没有必要也线性的预留出过多的内存，能保证紧急时刻的使用量便足矣。

(2).min_free_kbytes的主要用途是计算影响内存回收的三个参数 watermark[min/low/high]

1) watermark[high] > watermark[low] > watermark[min]，各个zone各一套

2)在系统空闲内存低于 watermark[low]时，开始启动内核线程kswapd进行内存回收（每个zone一个），直到该zone的空闲内存数量达到watermark[high]后停止回收。如果上层申请内存的速度太快，导致空闲内存降至watermark[min]后，内核就会进行direct reclaim（直接回收），即直接在应用程序的进程上下文中进行回收，再用回收上来的空闲页满足内存申请，因此实际会阻塞应用程序，带来一定的响应延迟，而且可能会触发系统OOM。这是因为watermark[min]以下的内存属于系统的自留内存，用以满足特殊使用，所以不会给用户态的普通申请来用。

3) 三个watermark的计算方法：

```
1 watermark[min] = min_free_kbytes换算为page单位即可，假设为min_free_pages。（因为是每个zone一个）
2 watermark[low] = watermark[min] * 5 / 4
3 watermark[high] = watermark[min] * 3 / 2
```

所以中间的buffer量为 high - low = low - min = per_zone_min_free_pages * 1/4。因为min_free_kbytes = 4 * sqrt(lowmem_kbytes)，也可以看出中间的buffer量也是跟内存的增长速度成开方关系。

4) 可以通过/proc/zoneinfo查看每个zone的watermark

例如：

```
1 Node 0, zone      DMA
2 pages free      3960
3 min             65
4 low             81
5 high           97
```

(3).min_free_kbytes大小的影响

min_free_kbytes设的越大，watermark的线越高，同时三个线之间的buffer量也相应会增加。这意味着会较早的启动kswapd进行回收，且会回收上来较多的内存（直至watermark[high]才会停止），这会使得系统预留过多的空闲内存，从而在一定程度上降低了应用程序可使用的内存量。极端情况下设置min_free_kbytes接近内存大小时，留给应用程序的内存就会太少而可能会频繁地导致OOM的发生。

min_free_kbytes设的过小，则会导致系统预留内存过小。kswapd回收的过程中也会有少量的内存分配行为（会设上PF_MEMALLOC）标志，这个标志会允许kswapd使用预留内存；另外一种情况是被OOM选中杀死的进程在退出过程中，如果需要申请内存也可以使用预留部分。这两种情况下让他们使用预留内存可以避免系统进入deadlock状态。

5, lowmem与highmem

关于lowmem和highmem的定义在这里就不详细展开了，推荐两篇文章：

<http://linuxkernel.com/?p=1013>

链接内讲的比较清楚，这里只说结论：

(1) 当系统的物理内存 > 内核的地址空间范围时，才需要引入highmem概念。

x86架构下，linux默认会把进程的虚拟地址空间（4G）按3:1拆分，0~3G user space通过页表映射，3G~4G kernel space线性映射到进程高地址。就是说，x86机器的物理内存超过1G时，需要引入highmem概念。

(2) 内核不能直接访问1G以上的物理内存（因为这部分内存没法映射到内核的地址空间），当内核需要访问1G以上的物理内存时，需要通过临时映射的方式，把高地址的物理内存映射到内核可以访问的地址空间里。

(3) 当lowmem被占满之后，就算剩余的物理内存很大，还是会出现oom的情况。对于linux2.6来说，oom之后会根据score杀掉一个进程（oom的话题这里不展开了）。

(4) x86_64架构下，内核可用的地址空间远大于实际物理内存空间，所以目前没有上面讨论的highmem的问题，可以认为系统内存等于lowmem。

6 , lowmem_reserve_ratio(/proc/sys/vm/lowmem_reserve_ratio)

官方解释：

For some specialised workloads on highmem machines it is dangerous for the kernel to allow process memory to be allocated from the "lowmem" zone. This is because that memory could then be pinned via the `mlock()` system call, or by unavailability of swap space.

And on large highmem machines this lack of reclaimable lowmem memory can be fatal.

So the Linux page allocator has a mechanism which prevents allocations which `_could_` use highmem from using too much lowmem. This means that a certain amount of lowmem is defended from the possibility of being captured into pinned user memory.

The `'lowmem_reserve_ratio'` tunable determines how aggressive the kernel is in defending these lower zones.

If you have a machine which uses highmem or ISA DMA and your applications are using `mlock()`, or if you are running with no swap then you probably should change the `lowmem_reserve_ratio` setting.

(1).作用

除了min_free_kbytes会在每个zone上预留一部分内存外，lowmem_reserve_ratio是在各个zone之间进行一定的防卫预留，主要是防止高端zone在没内存的情况下过度使用低端zone的内存资源。

例如现在常见的一个node的机器有三个zone: DMA, DMA32和NORMAL。DMA和DMA32属于低端zone, 内存也较小, 如96G内存的机器两个zone总和才1G左右, NORMAL就相对于属于高端内存 (现在一般没有HIGH zone), 而且数量较大 (>90G)。低端内存有一定的特殊作用比如发生DMA时只能分配DMA zone的低端内存, 因此需要在 尽量可以使用高端内存时 而不使用低端内存, 同时防止高端内存分配不足的时候抢占稀有的低端内存。

(2). 计算方法

```
# cat /proc/sys/vm/lowmem_reserve_ratio
```

256 256 32

内核利用上述的protection数组计算每个zone的预留page量，计算出来也是数组形式，从/proc/zoneinfo里可以查看：

```

1 Node 0, zone          DMA
2   pages free         1355
3       min            3
4       low             3
5       high           4
6       :
7       :
8   numa_other        0
9   protection: (0, 2004, 2004, 2004)
10 ~~~~~~
11 pagesets
12   cpu: 0 pcp: 0
13   :
```

在进行内存分配时，这些预留页数值和watermark相加来一起决定现在是满足分配请求，还是认为空闲内存量过低需要启动回收。

例如，如果一个normal区(index = 2)的页申请来试图分配DMA区的内存，且现在使用的判断标准是watermark[low]时，内核计算出 page_free = 1355，而watermark + protection[2] = 3 + 2004 = 2007 > page_free，则认为空闲内存太少而不予以分配。如果分配请求本就来自DMA zone，则 protection[0] = 0会被使用，而满足分配申请。

zone[i] 的 protection[i] 计算规则如下：

```

1  (i < j):
2      zone[i]->protection[j]
3      = (total sums of present_pages from zone[i+1] to zone[j] on the node)
4        / lowmem_reserve_ratio[i];
5  (i = j):
6      (should not be protected. = 0;
7  (i > j):
8      (not necessary, but looks 0)

```

默认的 lowmem reserve ratio[i] 值是：

256 (if zone[i] means DMA or DMA32 zone)

32 (others).

从上面的计算规则可以看出，预留内存值是ratio的倒数关系，如果是256则代表 1/256，即为 0.39% 的高端zone内存大小。如果想要预留更多页，应该设更小一点的值，最小值是1（1/1 -> 100%）。

(3). 和min_free_kbytes (watermark) 的配合示例

下面是一段某线上服务器 (96G) 内存申请失败时打印出的log :

```

1 [38905.295014] java: page allocation failure. order:1, mode:0x20, zone 2
2 [38905.295020] Pid: 25174, comm: java Not tainted 2.6.32-220.23.1.tb750.el5.x86_64 #1
3 ...
4 [38905.295348] active_anon:5730961 inactive_anon:216708 isolated_anon:0
5 [38905.295349] active_file:2251981 inactive_file:15562505 isolated_file:0
6 [38905.295350] unevictable:1256 dirty:790255 writeback:0 unstable:0
7 [38905.295351] free:113095 slab_reclaimable:577285 slab_unreclaimable:31941
8 [38905.295352] mapped:7816 shmem:4 pagetables:13911 bounce:0
9 [38905.295355] Node 0 DMA free:15796kB min:4kB low:4kB high:4kB active_anon:0kB inactive
10 [38905.295365] lowmem_reserve[]: 0 1951 96891 96891
11 [38905.295369] Node 0 DMA32 free:380032kB min:800kB low:1000kB high:1200kB active_anon:4
12 [38905.295379] lowmem_reserve[]: 0 0 94940 94940
13 [38905.295383] Node 0 Normal free:56552kB min:39032kB low:48788kB high:58548kB active_an
14 [38905.295393] lowmem_reserve[]: 0 0 0 0
15 [38905.295396] Node 0 DMA: 1*4kB 2*8kB 0*16kB 1*32kB 2*64kB 0*128kB 1*256kB 0*512kB 1*1
16 [38905.295405] Node 0 DMA32: 130*4kB 65*8kB 75*16kB 72*32kB 95*64kB 22*128kB 10*256kB 7*
17 [38905.295414] Node 0 Normal: 12544*4kB 68*8kB 0*16kB 0*32kB 0*64kB 0*128kB 0*256kB 0*51
18 [38905.295423] 17816926 total pagecache pages

```

1) 从第一行log"order:1, mode:0x20"可以看出是GFP_ATOMIC类型的申请, 且order = 1(page = 2)

2) 第一次内存申请尝试

在__alloc_pages_nodemask()里, 首先调用 get_page_from_freelist() 尝试第一次申请, 使用的标志位是 ALLOC_WMARK_LOW|ALLOC_CPUSET, 它会对每个zone都做 zone_watermark_ok()的检查, 使用的就是传进的watermark[low] 阈值。

在zone_watermark_ok()里会考虑z->lowmem_reserve[], 导致在normal上的申请不会落到低端zone。比如对于DMA32:

free pages = 380032KB = 95008 pages < low(1000KB = 250 pages) + lowmem_reserve[normal](94940) = 95190

所以就认为DMA32也不平不ok, 同理更用不了DMA的内存。

而对于normal自己内存来说, free pages = 56552 KB = 14138 pages, 也不用考虑lowmem_reserve(0), 但这时还会考虑申请order(1), 减去order 0的12544个page后只剩 14138 - 12544 = 1594, 也小于 low / 2 = (48788KB=12197pages) / 2 = 6098 pages。

所以初次申请尝试失败, 进入__alloc_pages_slowpath() 尝试进行更为积极一些的申请。

3)第二次内存申请尝试

__alloc_pages_slowpath()首先是通过 gfp_to_alloc_flags() 修改alloc_pages, 设上更为强硬的标志位。这块根据原来的GFP_ATOMIC会设上 ALLOC_WMARK_MIN | ALLOC_HARDER | ALLOC_HIGH。但注意的是不会设上 ALLOC_NO_WATERMARKS 标志位。这个标志位不再判断zone的水位限制, 属于优先级最高的申请, 可以动用所有的reserve 内存, 但条件是 (!in_interrupt() && ((p->flags & PF_MEMALLOC) || unlikely(test_thread_flag(TIF_MEMDIE)))) , 即要求不能在中断上下文, 且是正在进行回收 (例如kswapd) 或者正在退出的进程。

之后进入拿着新的alloc_pages重新进入get_page_from_freelist() 尝试第二次申请, 虽然有了 ALLOC_HARDER和ALLOC_HIGH, 但是不幸的是在3个zone的zone_watermark_ok检查中还是都无法通过, 例如对于DMA32:

free pages = 380032KB = 95008 pages

因为设上了ALLOC_HIGH 所以会将得到的watermark[min]减半, 即min = min/2 = 800K / 2 = 400K = 100pages

而又因为设上了ALLOC_HARDER, 会再将min砍去1/4, 即min = 3 * min / 4 = 100 pages * 3 / 4 = 75 pages

即便如此, min(75 pages) + lowmem_reserve[normal](94940) = 95015, 仍大于free pages, 仍认为无法分配内存, 同理DMA也不成功, 而normal中 free pages里连续8K的页太少也无法满足分配

第二次失败后, 由于没有ALLOC_NO_WATERMARK也不会进入__alloc_pages_high_priority 进行最高优先级的申请, 同时由于是GFP_ATOMIC类型的分配不能阻塞回收或者进入OOM, 因此就以申请失败告终。

遇到此种情况可以适当调高 min_free_kbytes 使kswapd较早启动回收, 使系统一直留有较多的空闲内存, 同时可以适当降低 lowmem_reserve_ratio (可选), 使得内存不足的情况下 (主要是normal zone) 可以借用DMA32/DMA的内存救急 (注意不能也不能过低)。

参考:

http://kernel.taobao.org/index.php?title=Kernel_Documents/mm_sysctl

<http://linuxkernel.com/?p=1013>

<https://www.kernel.org/doc/Documentation/sysctl/vm.txt>

版权声明：原创作品，如需转载，请注明出处。否则将追究法律责任

内存 Out oom

2

收藏 分享

上一篇：lua table引用问题 下一篇：linux系统收到SYN但不回SYN+ACK问题排查

猜你喜欢

Linux学习之路-Nginx（3）模块简要介绍篇【26...

LNMP环境相关配置Nginx

openstack queens版本安装-搭建你自己的企业...

使用nmcli 实现 bond0 网络组 网桥三种模式


linux集群搭建之nfs服务的搭建

详解CentOS6.9源码Openvpn2.4.1安装

通过zabbix 3.2监控nginx活动状态

Centos 7 磁盘阵列配置介绍

发表评论



用心的评论会被更多人看到和认可

Ctrl+Enter 发布 取消 发布