Linux memory management summary

Memory accounting

- Charging memory to processes
- total_vm

Information sources

- /proc/<PID>/…
- /proc/…

What all these memory types are

- Clean vs dirty
- Shared vs private
- Named vs anonymous
- Virtual vs resident

What to expect

- of heap usage
- of file mapping
- of anonymous mapping
- of stack
- of child processes/threads

OOM killing

When this happens

Who gets killed

Kernel threads or Init process never get killed by this mechanism.

For other processes we count their “score” and kill one that have maximal score.

```
 * The formula used is relatively simple and documented inline in the
 * function. The main rationale is that we want to select a good task
 * to kill when we run out of memory.
 *
 * Good in this context means that:
 * 1) we lose the minimum amount of work done
 * 2) we recover a large amount of memory
 * 3) we don't kill anything innocent of eating tons of memory
 * 4) we want to kill the minimum amount of processes (one)
 * 5) we try to kill the process the user expects us to kill, this
 *    algorithm has been meticulously tuned to meet the principle
 *    of least surprise ... (be careful when you change it)
```

Process that currently executes swapoff system call is always the first candidate to be oom-killed with score of ULONG_MAX.

In other cases process score is counted as folows:

1. The memory size of the process is the basis for the badness;
   - points = total_vm
2. Take child processes into an account. Processes which fork a lot of child processes are likely a good choice. We add half the vmsize of the children if they have an own mm. This prevents forking servers to flood the machine with an endless amount of children. In case a single child is eating the vast majority of memory, adding only half to the parents will make the child our kill candidate of choice;
   - for each child process with own address space: points += (1 + child→total_vm/2)
3. Take process lifetime into an account. *(CPU time is in tens of seconds and run time is in thousands of seconds)*;
   - cpu_time = (user_time + system_time) / 8; *(that is, consumed cpu time in user and kernel mode, as reported by e.g. time)*
   - run_time = (real time elapsed since process start) / 1024;
   - if (cpu_time > 0) points /= int_sqrt(cpu_time);
   - if (run_time > 0) points /= int_sqrt(int_sqrt(run_time));
4. Rise score for niced processes. *(Niced processes are most likely less important, so double their badness points)*;
   - if (task_nice > 0) points *= 2;
5. Lower score for superuser processes. *(Superuser processes are usually more important, so we make it less likely that we kill those)*;
   - if (has_capability_noaudit(p, CAP_SYS_ADMIN) || has_capability_noaudit(p, CAP_SYS_RESOURCE)) points /= 4;
6. Lower score for a process that have direct hardware access. *(We don't want to kill a process with direct hardware access. Not only could that mess up the hardware, but usually users tend to only have this flag set on applications they think of as important)*;
   - if (has_capability_noaudit(p, CAP_SYS_RAWIO)) points /= 4;
7. Finally adjust the score by oom_adj;
   - if (oom_adj > 0) points «= oom_adj; *(if points == 0 before shift, points = 1)*
   - if (oom_adj < 0) points »= -oom_adj;

How to control OOM-killer

- oom_adjust
- vm.panic_on_oom
- vm.oom_kill_allocating_task
- vm.oom_dump_tasks

- vm.would_have_oomkilled

Memleak detection

Direct memleak evidences