MCv0: State Transition, Purging and Tab Discarding

TL; DR
Update 2017-04-11: Obsoleted. This has been merged into the v0 design doc.

This doc describes work-in-progress design changes in memory coordinator v0. See the original design doc for the motivation, success criteria and testing plan. I will update the original design doc when we finalize the internal design.

Summary

Proposal for introducing internal state for memory coordinator and replace global state with it so that we can build strategies for when/how to change memory state, purge memory, and request tab discarding.

Motivation

The current heuristic of memory coordinator was chosen when we didn't have enough insights on how to become a good memory citizen. Since then we've got feedback and made some design changes. The heuristics also needs to be updated to follow up recent changes/findings:

- As of crrev.com/2671303004 all purging logic moved from OnMemoryStateChange() to OnPurgeMemory(). State transitions no longer free up memory so we need to think about when/how to purge memory, in addition to when/how to change memory state.
- Tab discarding is going to become a service and memory coordinator needs to trigger tab discarding as the replacement of MemoryPressureListener. We need to think about when to trigger tab discarding.
- Tab suspending still requires a lot of work/verifications and we wouldn't be able to use it for now.

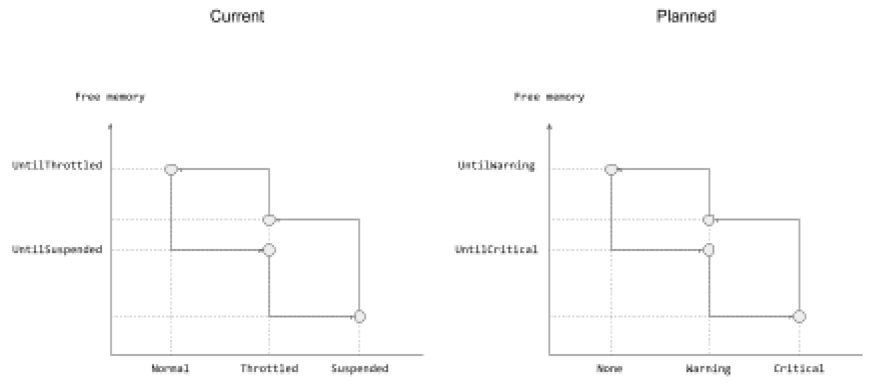
In this document, I propose some changes and strategies for state transition, purging and tab discarding.

Generalizing global state calculation

The current heuristic only focus on global memory state transition. It was reasonable decision as state transition was the key and the only task MCv0 should handle. However, calculating global state is no longer the only task MC should handle. It also needs to determine when to purge memory and when to request tab discarding. This means that the current notion of “global memory state” should be generalized. In other words, it would be better to have a notion/value that indicates how close the system is going to start swapping/compressing. We can use the value to determine memory state for each process, when to purge memory, when to trigger tab discarding, etc.

That said, **there won’t be significant changes from an implementation point of view**. What I want to propose is rephasing existing values and thresholds. Specifically,

- Global memory state -> Memory condition
- THROTTLED global memory state -> WARNING memory condition
- SUSPENDED global memory state -> CRITICAL memory condition



Note that *memory condition* is an **internal state** of memory coordinator. I don’t intend to expose this internal state to clients/child processes at this point. base::MemoryState won’t be changed. Clients/child processes receive OnStateChange(state) where state = {NORMAL, THROTTLED}.

Available memory update scheduling

Memory condition will be calculated by using GetFreeMemoryUntilCriticalMB(). How frequently we should update available free memory then? This depends on how memory budget API is designed, and I don’t have specific plan at this point. I think we can start with every 5 seconds. This is almost the same as the current state update scheduling interval.

State transition

Memory state of a process is determined by memory pressure level and it’s visibility.

Visibility \ Condition	NONE	WARNING	CRITICAL
Foreground	NORMAL	NORMAL	THROTTLED
Background	NORMAL	THROTTLED	THROTTLED

Purging

We can think of a lot of options when to purge memory but I’d like to start with a very simple and conservative (from performance POV) one.

- When memory condition becomes WARNING request purging memory to background processes.

- When a foreground process goes background in WARNING memory condition, request purging memory to the process.
- Don't request purging on CRITICAL memory condition. Memory coordinator already requested purging for background processes at WARNING memory condition.

The purpose of purging is to free up memory before the system starts swapping/compressing pages and I think above strategy aligns the purpose.

If above isn't enough, we can try other things like:

- Request purging when memory condition becomes WARNING to all processes.
- Schedule a task which keeps purging while memory condition remains WARNING/CRITICAL, with exponential backoff.

Tab discarding

Tab discarding will become a service and the service decides which tab should be discarded. Memory coordinator's job is to trigger a tab discarding via the service.

When memory condition becomes CRITICAL, memory coordinator will start a task which requests one tab discarding to the service. Memory coordinator keeps scheduling the task with some interval (e.g. 5 seconds) until memory condition becomes NORMAL/WARNING.