User Space Cache Management By Kernel

Introduction

Currently applications find it difficult to free memory on system global memory pressure. This project intends to solve this basic problem. The kernel internal caches use "shrinker callbacks" and then free some objects when the system runs low on memory. Currently this is being designed for interaction between hypervisors and kernels where there is a similar problem, the hypervisor might know about memory pressure, which the kernel doesn't know about. We are trying to achieve an analogous scheme for User-space<->Kernel interaction.

Project Goals

- Minimum kernel modifications.
- Minimum performance impact on users not interested in UCMK functionality.
- Overall increase in performance if this functionality is used.

Project Status

Design Phase: Project started on 21th Aug 2007

Authors

Soumendu Sekhar Satapathy : Soumendu.Satapathy@in.unisys.com

Design and Implementation

(1) Using the Signal Handling Mechanism of the Kernel.
The PFRA (Page Frame Reclaiming Algorithm) has various entry points. The Page Frame reclaiming is performed on mainly three occasions.

(a) Low on Memory reclaiming :- The Kernel detects a "low on memory" condition. to perform.

(b) Hibernation reclaiming :- The Kernel must free memory because it is entering into suspend-to-disk state.

(c) Periodic reclaiming :- A Kernel thread is activated periodically to perform memory reclaim if necessary.

Among the above three types (a), (b) and (c), what we will be interested in is,
Low on memory reclaiming. The “out of memory” killer gets invoked by __alloc_pages() when the free memory is very low and the PFRA has not succeeded in reclaiming any

page frames. This guy basically selects a process in the system and abruptly kills it to free its page frames.

We don't want to go as far as OOM killer. We will hook our functionality in the try_to_free_pages() kernel routine to implement what we intend to do. We will define a new signal type lets say , SIGSRKCH (signal for shrinking cache) in the kernel.
And this signal can be sent to the interested process in a heavy pressure condition before a "out of memory" condition even takes place so that it just releases some of its cached memory in a signal handler in the user space application process. One more issue is how to know which all processes are interested to be shrinked, probably the kernel can know that, if that process has registered a signal handler for this particular signal i.e SIGSRKCH , and it can be known from the task_struct. We have to tell the application how much to shrink.

(2) Using the Kernel Event Layer Mechanism.
We can define an event in the Kernel for heavy memory pressure condition. Using the kobject_uevent we can send the signal to the user space when the memory pressure is more. We can use a threshold i.e a low/high watermark to detect a heavy memory pressure condition. For all this we can construct a kernel module. In the use space we can use netlink to read events.

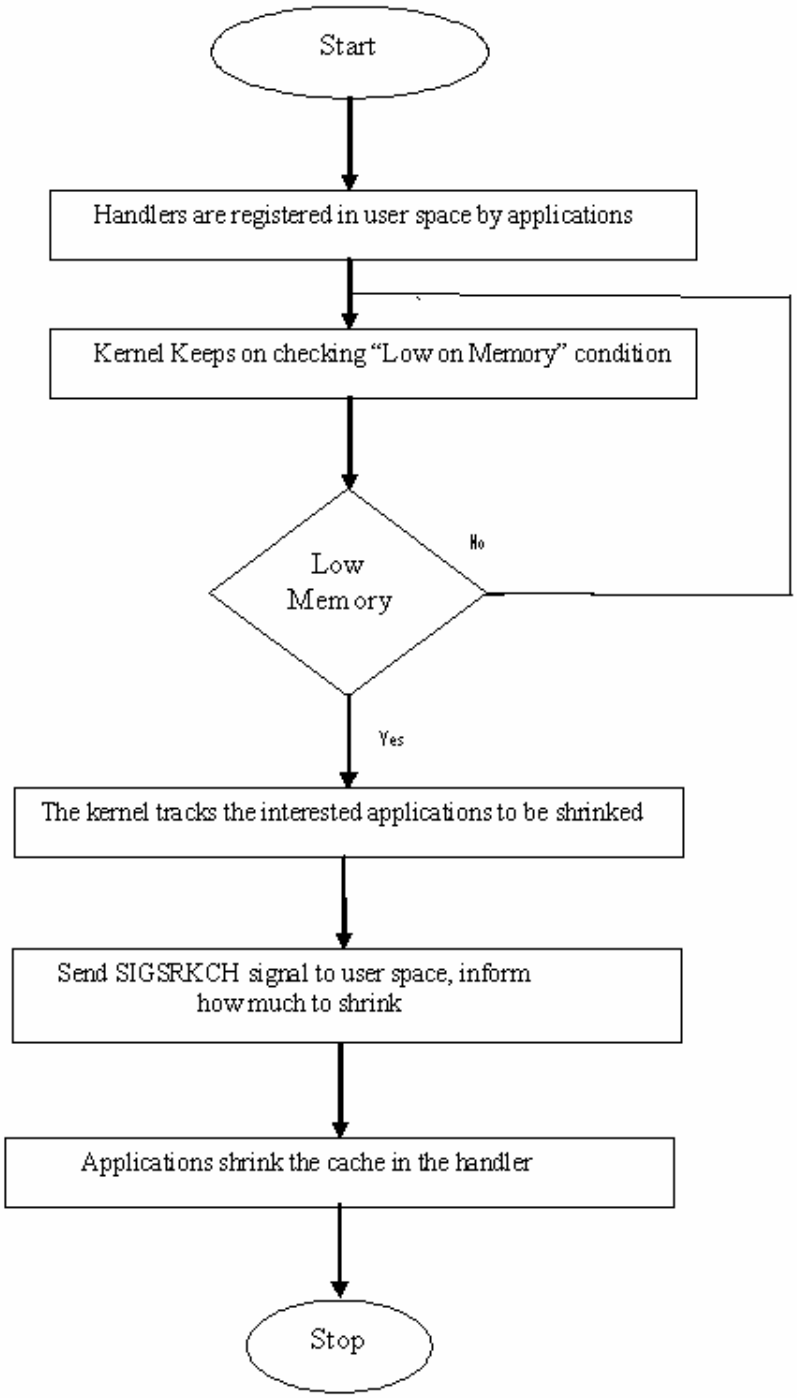
(3) sys_madvise() system call implementation
Using a reclaimable page type in the madvise. We can add an extra behaviour lets say MADV_RECLAIM and have a reclaimable page which the Kernel can take away.

Downloads

Currently not available.

Algorithms and High Level Design

Algorithm -1
Using Signal Handling Mechanism of Kernel


Start
Handlers are registered in user space by applications
Kernel Keeps on checking "Low on Memory" condition
Low Memory
No
Yes
The kernel tracks the interested applications to be shrinked
Send SIGSRKCH signal to user space, inform how much to shrink
Applications shrink the cache in the handler
Stop