

# Memory Thrashing Protection in Multi-Programming Environment

Xiaodong Zhang

Ohio State University

In collaborations with

Song Jiang (Wayne State University)

# Memory Management for Multiprogramming

- Space sharing among **interactive programs** in virtual memory is managed by page replacement.
- Commonly used policy is the **global LRU** replacement in the **entire user memory space**.
- **Thrashing**: accumulated memory requests of multiple programs exceed available user space,
  - No program is able to establish its working set;
  - causing large page faults;
  - low CPU utilization; and
  - execution of each program practically stops.

# Past and Existing Thrashing Protection Methods

## ■ Local page replacement:

- Each program is statically allocated a fixed size.
- DEC VAX machine had this in its VMS in early 1980's.
- Memory underutilization: not adapting dynamics.
- It is no longer used in any systems.

## ■ Load control:

- While thrashing, some job(s) is/are **suspended/swapped**.
- Open BSD operating systems, IBM RS/6000, HP9000.
- HP-UX has a ``serialize()'' command for thrashing.
- Linux makes load controls based on RSS (resident set size) reporting the total number of occupied pages.

# Limits and Problems of Load Controls

- A thrashing is often triggered by a brief **spike of memory demand**, a load control can over-react.
- Suspending a job **causes other related jobs to quit.**
- When a job is suspended, its working set can be replaced quickly by other running programs, very **expensive to rebuild the working set.**
- A lightweight and dynamic protection is much more desirable than a brute-force action.

# Some Insights into Thrashing

- The global LRU replacement generates two types of LRU pages for replacement:
  - **True LRU pages:** to which programs do not need to access.
  - **False LRU pages:** to which programs have not been able to access due to required working set is not set up yet, or page faults are being conducted.
- A system **cannot distinguish true or false LRU pages**, but selects both for replacement.
- The amount false LRU pages is a status indicator: **no, marginally, or seriously thrashing.**

# Token-based Thrashing Protection Facility

- Jiang/Zhang, *Performance Evaluation*, 05, ([Ohio State](#)).
- Conducted intensive experiments at the kernel level along with analysis on memory thrashing:
  - **A sudden spike of memory demand** from one can generate many false LRU pages in others, particularly in an less demanding one.
  - As **false pages** reach to a certain amount, the system becomes little productive even when physical memory is not too small.
- Basic idea of the token mechanism:
  - As the system enters a pre-thrashing stage (low RSS, and high idle CPU), a token is issued to a process so that it can quickly form its working set and proceed.
  - This approach can effectively and timely avoid thrashing.

# Some Alternatives in Its Implementation

- Which process to issue the token?
  - A less memory demanding process.
- How long does a process hold the token?
  - It is adjustable and proportional to the thrashing degree.
- What happens if thrashing is too serious?
  - It becomes a polite load control mechanism by setting a long token time so that each program has to be executed one by one.
- Multi-tokens can be effective for light thrashing.
- The token and its variations were implemented and tested in **Linux kernel 2.2.**

# Outcome and Impact of This Work

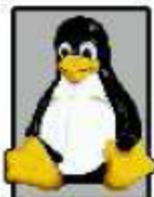
- A paper entitled “Token-ordered LRU: ...” has been rejected by several top system conferences. (Main reason: this is not a hot OS topic anymore).
- **A successful technology transfer based on it!**
  - A group of independent Linux kernel developers organized by **Rik van Reil** of RedHat started a project to include the token into the Linux kernel in July 2004.
  - The **implementation insights and detailed technical discussions** are well documented in the Internet.
- **Token-ordered LRU, renamed as *Swap token*, was formally adopted in Linux kernel 2.6.9, 12/04, serving millions of users world wide.**

## Impact of This Work (continued)

- *Swap token* is introduced in book *Understanding Linux Kernel* (3rd edition), (Bovet and Casati)
- *Swap token* is a section in Book *Professional Linux Kernel Architecture*
- **False LRU page** concept is quoted in OS wiki.
- Continued efforts on adaptive swap token in kernel:
  - Switch on/off the token adaptive to VM load changes.
  - Other alternative proposed in the paper.

# The Evolution of Swap Token in Linux

- **First version:** token is randomly given to a process
  - A time stamp is used to handover the token one by one
  - **Limit 1:** the token may not hit to the most desirable one
  - **Limit 2:** a constant time stamp may not address urgency
- **preempt swap token** (current version)
  - A “priority counter” is set for each process to record the number of swap-out pages.
  - The counter is incremented for a unit of swap-out pages
  - The token is always to the process with high “priority”
  - The length of time stamp varies by the priority degree



# Cross-Referencing Linux

## Linux/mm/thrash.c

[ [source navigation](#) ]  
[ [diff markup](#) ]  
[ [identifier search](#) ]  
[ [freetext search](#) ]  
[ [file search](#) ]

Version: [ [1.0.9](#) ] [ [1.2.13](#) ] [ [2.0.40](#) ] [ [2.2.26](#) ] [ [2.4.18](#) ] [ [2.4.20](#) ] [ [2.4.28](#) ] [ [2.6.10](#) ]  
[ [2.6.11](#) ]

Architecture: [ [i386](#) ] [ [alpha](#) ] [ [arm](#) ] [ [ia64](#) ] [ [m68k](#) ] [ [mips](#) ] [ [mips64](#) ] [ [ppc](#) ] [ [s390](#) ] [ [sh](#) ]  
[ [sparc](#) ] [ [sparc64](#) ] [ [x86\\_64](#) ]

```
1 /*
2  * mm/thrash.c
3  *
4  * Copyright (C) 2004, Red Hat, Inc.
5  * Copyright (C) 2004, Rik van Riel <riel@redhat.com>
6  * Released under the GPL, see the file COPYING for details.
7  *
8  * Simple token based thrashing protection, using the algorithm
9  * described in: http://www.cs.wm.edu/~sjiang/token.pdf
10 */
11 #include linux/jiffies.h
12 #include linux/mm.h
13 #include linux/sched.h
14 #include linux/swap.h
15
16 static DEFINE\_SPINLOCK(swap_token_lock);
17 static unsigned long swap\_token\_timeout
18 unsigned long swap\_token\_check
19 struct mm\_struct * swap\_token\_mm = &init_mm;
20
21 #define SWAP\_TOKEN\_CHECK\_INTERVAL(HZ * 2)
22 #define SWAP\_TOKEN\_TIMEOUT 0
23 /*
24  * Currently disabled; Needs further code to work at HZ * 300.
25  */
26 unsigned long swap\_token\_default\_timeout = SWAP\_TOKEN\_TIMEOUT;
27
28 /*
29  * Take the token away if the process had no page faults
30  * in the last interval, or if it has held the token for
31  * too long.
32  */
33 #define SWAP\_TOKEN\_ENOUGH\_RSS
34 #define SWAP\_TOKEN\_TIMED\_OUT
35 static int should\_release\_swap\_token(struct mm\_struct *mm)
36 /
```