SREcon20 Americas

Capacity Planning and Performance Enhancement with Page Reference Sampling

Danny Chen
Trading Solutions SRE Team, Bloomberg, LP

TechAtBloomberg.com
Outline

- Page reference sampling - some personal history
- The Solaris page reference sampling interface (an open secret?)
- Brief review of paging architectures
- Utilities I’ve written and examples
- Summary - why we need this and what else do we need
RUNTIME ENVIRONMENTS

Thursday (4:00-5:30)  
Chair: Jay Lepreau

A Memory Allocation Profiler for C and Lisp Programs .......................... 223  
Benjamin Zorn & Paul Hilfinger, University of California, Berkeley

A RISC Approach to Runtime Exceptions ............................................. 239  
Mark Hinekstein, MIPS Computer Systems, Inc.; Steven Correll, Key Computer Laboratories; Kevin Enderby, NeXT Inc.

CASPER the Friendly Daemon ............................................................. 251  
Ronald E. Bailey, AT&T Information Systems; Danny Chen, AT&T Bell Laboratories
# Software & Systems: Then vs Now

<table>
<thead>
<tr>
<th>Then</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit</td>
<td>32 and 64 bit</td>
</tr>
<tr>
<td>• Small address space programs</td>
<td>• Large, sparse address space programs</td>
</tr>
<tr>
<td>Uni-processor</td>
<td>Multi-processor, multi-threads</td>
</tr>
<tr>
<td>Speed in 1-10 MIPS</td>
<td>Speed in $10^5$-$10^6$ MIPS</td>
</tr>
<tr>
<td>Small processor caches</td>
<td>Large, multi-level processor caches</td>
</tr>
<tr>
<td>Main memory in Megabytes</td>
<td>Main memory in GB and TB</td>
</tr>
<tr>
<td>Static linking</td>
<td>Dynamic linking and loading</td>
</tr>
<tr>
<td>Mainly “utilities” (short lived)</td>
<td>Lots of “servers” (long lived)</td>
</tr>
<tr>
<td>Compiled</td>
<td>Interpreted languages with GC</td>
</tr>
<tr>
<td>Memory locking and advising</td>
<td>Memory locking and advising</td>
</tr>
<tr>
<td>Optional large page sizes</td>
<td>Optional large page sizes</td>
</tr>
</tbody>
</table>
Opening the page data file enables tracking of address space references and modifications on a per page basis.
Available on Solaris via /proc/pid/pagedata

Opening the page data file enables tracking of address space references and modifications on a per-page basis.

A `read()` of the page data file descriptor returns structured page data and atomically clears the page data maintained for the file by the system. That is to say, each read returns data collected since the last read; the first read returns data collected since the file was opened. When the call completes, the read buffer contains the following structure as its header and thereafter contains a number of section header structures and associated data:

```c
typedef struct prpagehdr {
    timespec_t pr_tstamp; /* real time stamp, time of read() */
    ulong_t pr_nmap; /* number of address space mappings */
    ulong_t pr_npage; /* total number of pages */
} prpagehdr_t;
```

The header is followed by `pr_nmap` prssmap structures and associated data arrays. The `prssmap` structure contains the following elements:

```c
typedef struct prssmap {
    uintptr_t pr_vaddr; /* virtual address of mapping */
    ulong_t pr_npage; /* number of pages in mapping */
    char pr_pemapsz[PRMAPSZ]; /* name in /proc/pid/object */
    offset_t pr_offset; /* offset into mapped object, if any */
    int pr_flags; /* protection and attribute flags */
    int pr_pagesize; /* pagesize for this mapping in bytes */
    int pr_shdsize; /* SysV shared memory identifier */
} prssmap_t;
```

Each section header is followed by `pr_npage` bytes, one byte for each page in the mapping, plus 0.7 null bytes at the end so that the new structure begins on an eight-byte aligned boundary. Each data byte may contain these flags:

- `PG_REFERENCED` page has been referenced.
- `PG_MODIFIED` page has been modified.

If the read buffer is not large enough to contain all of the page data, the read fails with E22250 and the page data is not cleared. The required size of the read buffer can be determined through `fstat()`. Application of `lseek()` to the page data file descriptor is ineffective; every read starts from the beginning of the file. Closing the page data file descriptor terminates the system overhead associated with collecting the data.

More than one page data file descriptor for the same process can be opened, up to a system-imposed limit per traced process. A read of one does not affect the data being collected by the system for the others. An open of the page data file will fail with errno if the system-imposed limit would be exceeded.
How is this done?

Access to the state for a particular MMU context is controlled by two registers:
1. Context #, used by TLB
2. PDIR, used by multi-level page map

MMU translation graphic from https://computationstructures.org/lectures/virtual/virtual.html#2
Page table entry graphic from: https://users.soe.ucsc.edu/~scott/courses/Fall05/111/Slides/chapter4.pdf
Counting page references with wssts

Usage: wssts -c <command string> [-i data_interval] [-l metadata_interval]

Working Set Size Time Series (wssts)
● -c command string: monitor processes that match this command string
● -l metadata_interval: the interval used to look for processes that match
  the command string
● -i data_interval: interval to do page reference polling
Example: looking at crwrkr with wssts

```
ps -opid,rss,rssprivate,comm,args | grep
```
Example: looking at crwrkr with wssts (cont’d)

- RSS doesn’t factor in (or factor out) shared pages
- In the absence of page stealing, RSS is monotonically increasing
- RSSprivate factors out shared pages
- Dirty pages have to be written before they are stolen

In this particular case, it looks like there MAY be a large memory startup cost. E.g. initialization code and data.

<table>
<thead>
<tr>
<th>pid</th>
<th>rss</th>
<th>rssprivate</th>
<th>wss (K)</th>
<th>rocnt (K)</th>
<th>wrcnt (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3280</td>
<td>1761584</td>
<td>623080</td>
<td>634896</td>
<td>189008</td>
<td>445888</td>
</tr>
<tr>
<td>16474</td>
<td>1684672</td>
<td>542375</td>
<td>548728</td>
<td>177464</td>
<td>371264</td>
</tr>
<tr>
<td>39246</td>
<td>1609976</td>
<td>515640</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39248</td>
<td>1646464</td>
<td>528416</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39247</td>
<td>1616752</td>
<td>517160</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39249</td>
<td>1637928</td>
<td>538424</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39250</td>
<td>1500048</td>
<td>530144</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39252</td>
<td>1509948</td>
<td>538144</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39253</td>
<td>1589894</td>
<td>751656</td>
<td>759600</td>
<td>193952</td>
<td>565696</td>
</tr>
<tr>
<td>39254</td>
<td>1505904</td>
<td>497840</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39255</td>
<td>1668040</td>
<td>617768</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39256</td>
<td>1596624</td>
<td>506288</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39257</td>
<td>1606296</td>
<td>509360</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39258</td>
<td>1581760</td>
<td>513632</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39259</td>
<td>1806162</td>
<td>695072</td>
<td>8544</td>
<td>384</td>
<td>8160</td>
</tr>
<tr>
<td>39260</td>
<td>1861544</td>
<td>710136</td>
<td>740912</td>
<td>193176</td>
<td>547736</td>
</tr>
<tr>
<td>39261</td>
<td>1536608</td>
<td>506240</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39262</td>
<td>1516624</td>
<td>503840</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39263</td>
<td>1617768</td>
<td>507696</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39264</td>
<td>1613192</td>
<td>519312</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39265</td>
<td>1588776</td>
<td>512064</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>39266</td>
<td>1832872</td>
<td>605912</td>
<td>411860</td>
<td>164024</td>
<td>217856</td>
</tr>
<tr>
<td>39267</td>
<td>1758440</td>
<td>625496</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>40181</td>
<td>1672640</td>
<td>549480</td>
<td>4512</td>
<td>384</td>
<td>4128</td>
</tr>
<tr>
<td>40182</td>
<td>1404232</td>
<td>400136</td>
<td>4362</td>
<td>224</td>
<td>4128</td>
</tr>
<tr>
<td>40183</td>
<td>1427424</td>
<td>410736</td>
<td>336</td>
<td>296</td>
<td>40</td>
</tr>
<tr>
<td>40184</td>
<td>1412592</td>
<td>404464</td>
<td>280</td>
<td>240</td>
<td>40</td>
</tr>
<tr>
<td>41197</td>
<td>1542876</td>
<td>471048</td>
<td>4362</td>
<td>224</td>
<td>4128</td>
</tr>
<tr>
<td>42349</td>
<td>1770784</td>
<td>628536</td>
<td>434224</td>
<td>172800</td>
<td>311432</td>
</tr>
<tr>
<td>57096</td>
<td>1786480</td>
<td>634944</td>
<td>484128</td>
<td>173808</td>
<td>318520</td>
</tr>
<tr>
<td>60466</td>
<td>1710192</td>
<td>571520</td>
<td>491856</td>
<td>173384</td>
<td>228472</td>
</tr>
</tbody>
</table>

* all memory values normalized to kilobytes
Capturing Page References (prsample)

Usage: prsample [-o outputfile] -p pid | command args
Example: prsample

- Wall time (seconds)
- User time (second, nanoseconds)
- System time (seconds, nanoseconds)
- Mapping name
- Base virtual address
- Page offset
- R (read) or M (modified)
Plotting Page References (plotpr.tk)
Counting Page References Over Time

- Knowing which pages are referenced the most (from the sampling perspective) can provide insights into which pages should be kept in memory
  - Leverage OS memory locking facilities
  - External utilities can sometimes help with locking
  - On systems that support pre-paging, knowing which pages to pre-load can help with start up performance
- Caveat: program behavior is often workload dependent
Predicting Page Fragmentation

[dchen294@nylxdev3 wsts (master)]$ ./pagefrag.py crwrkr.refcnts 8
'/bb/sys/package/c/crwrkr/0.000000.1053959-20200301T025606-1053959/bin/crwrkr.tsk',
'38234000': 6/1
'/bb/sys/package/c/crwrkr/0.000000.1053959-20200301T025606-1053959/bin/crwrkr.tsk',
'38240000': 4/4763
'/lib/ld.so.1', 'fb9a0000': 0/31
'/lib/libc.so.1', 'fa2c0000': 0/183
'/lib/libc.so.1', 'fa448000': 2/5
'anon', '3a7a2000': 3/4
'anon', '3a800000': 2/95741
'anon', 'a2400000': 0/511
'anon', 'a2800000': 4/155
'anon', 'a2c00000': 0/511
'anon', 'c0c00000': 0/15
'anon', 'c7c00000': 0/2559
'anon', 'cbc00000': 4/515
'anon', 'cc800000': 0/3583
'anon', 'd3000000': 0/1023

Bloomberg
Engineering
What about Linux?

• Can estimate wss (see http://www.brendangregg.com/wss.html)

• /proc/pid/smaps - # of pages referenced in a segment

• /proc/pid/clear_refs - clears the reference bits
  — A separate operation from smaps
  — Not coordinated with the page replacement system
In Closing...

• A different view into process behavior
  • Roughly what the page replacement system sees
  • Memory capacity planning
• Selecting page sizes
  • Pro: Larger page sizes -> smaller page tables -> faster address translation -> faster code execution
  • Con: Larger memory footprint
  • Solaris 11 will grow pages in mappings automatically
    — Would be good to be able to disable so that more granular page reference data can be collected if needed.
• Selecting memory locking strategies and other “advice” to the OS
In Closing...

- Memory layout for programs/processes
  - Feedback for memory layout strategies to reduce memory footprint
- Text vs data
  - Something like valgrind for data “profiling”?
- Inline vs no-inline?
- What about interpreted languages and JIT compilers?
  - Interpreters have visibility into data access patterns
References

• Solaris /proc: https://docs.oracle.com/cd/E19253-01/816-5174/proc-4/index.html
• Linux /proc: http://man7.org/linux/man-pages/man5/proc.5.html
• Brendan Gregg’s wss estimation: http://www.brendangregg.com/wss.html
• MMU diagram: https://computationstructures.org/lectures/virtual/virtual.html#2
• Page table entry diagram: https://users.soe.ucsc.edu/~scott/courses/Fall05/111/Slides/chapter4.pdf