Virtual Memory Primitives for User Programs

Andrew W. Appel and Kai Li
© 1991

Presented by: Nish Aravamudan
May 17, 2006
Overview

• Background
• Necessary Primitives
• Performance Considerations
• System Design
• Examples
Background

- Virtual memory “traditionally” increases size of process' address space
- Can also perform “tricks” via protection
  - Including userspace handlers for protection violations
  - If certain primitives are available to userspace ...
Necessary Primitives

- **TRAP** – Userspace page-fault handler
- **PROT1/PROTN** - decrease accessibility of page(s)
- **UNPROT** – increase accessibility of page
- **DIRTY** – list of dirtied pages since last call
- **MAP2** – map same physical page to two different virtual addresses, with different protections, in same address space
Performance Comparison

- The better the primitives perform, the better the algorithms perform

<table>
<thead>
<tr>
<th>Machine</th>
<th>OS</th>
<th>ADD</th>
<th>TRAP</th>
<th>TRAP PROT 1+ UNPROT</th>
<th>TRAP PROT N+ UNPROT</th>
<th>MAP 2</th>
<th>PAGESIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun 3/60</td>
<td>SunOS 4.0</td>
<td>0.12</td>
<td>760</td>
<td>1238</td>
<td>1016</td>
<td>yes</td>
<td>8192</td>
</tr>
<tr>
<td>Sun 3/60</td>
<td>SunOS 4.1</td>
<td>0.12</td>
<td>2080</td>
<td>1800</td>
<td>yes</td>
<td>8192</td>
<td></td>
</tr>
<tr>
<td>Sun 3/60</td>
<td>Mach 2.5(xp)</td>
<td>0.12</td>
<td>3280</td>
<td>2540</td>
<td>yes</td>
<td>8192</td>
<td></td>
</tr>
<tr>
<td>Sun 3/60</td>
<td>Mach 2.5(exc)</td>
<td>0.12</td>
<td>3380</td>
<td>2880</td>
<td>yes</td>
<td>8192</td>
<td></td>
</tr>
<tr>
<td>SparcStn 1</td>
<td>SunOS 4.0.3c</td>
<td>0.05</td>
<td>†230</td>
<td>*919</td>
<td>*839</td>
<td>yes</td>
<td>4096</td>
</tr>
<tr>
<td>SparcStn 1</td>
<td>SunOS 4.1</td>
<td>0.05</td>
<td>1008</td>
<td>909</td>
<td>yes</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>SparcStn 1</td>
<td>Mach 2.5(xp)</td>
<td>0.05</td>
<td>1550</td>
<td>1230</td>
<td>yes</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>SparcStn 1</td>
<td>Mach 2.5(exc)</td>
<td>0.05</td>
<td>1770</td>
<td>1470</td>
<td>yes</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>DEC 3100</td>
<td>Ultrix 4.1</td>
<td>0.062</td>
<td>210</td>
<td>393</td>
<td>344</td>
<td>no</td>
<td>4096</td>
</tr>
<tr>
<td>DEC 3100</td>
<td>Mach 2.5 (xp)</td>
<td>0.062</td>
<td>937</td>
<td>766</td>
<td>no</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>DEC 3100</td>
<td>Mach 2.5 (exc)</td>
<td>0.062</td>
<td>1203</td>
<td>1063</td>
<td>no</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>µVax 3</td>
<td>Ultrix 2.3</td>
<td>0.21</td>
<td>314</td>
<td>612</td>
<td>486</td>
<td>no</td>
<td>1024</td>
</tr>
<tr>
<td>i386 on iPSC/2</td>
<td>NX/2</td>
<td>0.15</td>
<td>172</td>
<td>302</td>
<td>252</td>
<td>yes</td>
<td>4096</td>
</tr>
</tbody>
</table>
Graphical Performance Comparison

- Sun 3/60+SunOS4.0
- Sun 3/60+SunOS4.1
- Sun 3/60+Mach2.5(xp)
- Sun 3/60+Mach2.5(exc)
- SparcStn1+SunOS4.0.3c
- SparcStn1+SunOS4.1
- SparcStn1+Mach2.5(xp)
- SparcStn1+Mach2.5(exc)
- DEC3100+Ulrix4.1
- DEC3100+Mach2.5(xp)
- DEC3100+Mach2.5(exc)
- μVax3+Ulrix4.1
- i386+NX/2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10,000</td>
<td>20,000</td>
</tr>
</tbody>
</table>
## Examples' Primitive Usage

<table>
<thead>
<tr>
<th>Methods</th>
<th>TRAP</th>
<th>PROT1</th>
<th>PROT₂</th>
<th>UNPROT</th>
<th>MAP₂</th>
<th>DIRTY</th>
<th>PAGESIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent GC</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>SVM</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concurrent checkpoint</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
<td>‡</td>
<td>√</td>
</tr>
<tr>
<td>Generational GC</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
<td>‡</td>
<td>√</td>
</tr>
<tr>
<td>Persistent store</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Extending addressability</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Data-compression paging</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap overflow</td>
<td>√</td>
<td></td>
<td>†</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concurrent Garbage Collection

• Synchronize collector and mutator
• Based on Baker's algorithm
  - At beginning of collection, all objects in from-space, to-space is empty
  - Collector traces graph of reachable objects, copying each into to-space
  - Every time mutator allocates new object, invoke collector
Concurrent Garbage Collection

• Baker's algorithm's invariants
  - Mutator only sees to-space pointers in registers
  - Objects in new and scanned areas only contain to-space pointers
  - Objects in unscanned area contains both to- and from-space pointers
Concurrent Garbage Collection

- VM protections detect from-space references and synchronize mutator/collector
  - Unscanned area set to no-access
  - Mutator will get page-access trap on unscanned objects
  - Collector handles trap, copying objects and forwarding pointers
  - Collector unprotects page and restarts mutator
  - Collector also scans unscanned area concurrently with mutator execution
Concurrent Garbage Collection

Collector
- Copies from-space objects to unscanned area
- Scans objects for pointers to from-space objects
- Copies object to unscanned area, if necessary
- Forwards pointers to to-space
- Once scanned == unscanned, all of from-space is garbage

Mutator
- Allocates at new
- In Baker's version, blocks while Collector runs

From reference [4]
Concurrent Garbage Collection

R/W

A
B
-\rightarrow C
-\rightarrow D
...
E
-\rightarrow F
G
-\rightarrow A

N/A

From-Space

To-Space

Registers

R_1 \rightarrow A
R_2 \rightarrow B

Time 0
Concurrent Garbage Collection

Collector:
- Copies reachable objects (from registers) to to-space
- Forwards pointers stored in registers

From-Space

A
B
→ C
→ D
...
E
→ F
G
→ A

R/W

To-Space

A₁
B₁
→ GC₁
→ DD₁

R/W

Registers

Time 1

R₁ → A₁

R₂ → B₁
Concurrent Garbage Collection

Collector:
- Protects unscanned area
- Starts mutator

From-Space

R/W

A
B
-> C
-> D
...
E
-> F
G
-> A

To-Space

R/W

A₁
B₁
-> C₁
-> D₁

Registers

R₁ --&gt; A₁
R₂ --&gt; B₁

Time 2
Concurrent Garbage Collection

Mutator:
Uses $R_1$, $R_2$

Collector:
Concurrently scans next page, copies reachable objects and forwards pointers

Registers

$R_1 \rightarrow A_1$

$R_2 \rightarrow B_1$

Time 3
Concurrent Garbage Collection

Mutator:
- Uses object E
- Page fault
- Assume G also reachable

Collector:
- Handles page fault
- Remap page
- Copy / forward
- Unprotect

Registers:
- \( R_1 \rightarrow E \rightarrow E_1 \)
- \( R_2 \rightarrow B \)

Later
Shared Virtual Memory

- Multiple processor/memory nodes, connected by fast message-passing network
- SVM presents large coherent shared memory address space
- Maintain single-writer and multiple-reader coherence at page level
  - Many copies of read-only pages, one copy of read-write pages
Concurrent Checkpointing

- Overlap time-consuming checkpoint operations to overlap with program execution

1) Stop all threads in program
2) Heap, globals, stacks saved
   - Set address space to read-only
   - Restart threads
   - Copying thread sequentially scans address space, copying pages to new addresses, then sets access to read-write
3) If restarted thread writes page, gets trap
Concurrent Checkpointing

1) R/W --> R/O
2) Restart application threads
3) Copier thread copies pages to separate space
4) Thread accesses uncopied page
5) Copier handles page fault
6) Copier restarts thread
Generational Garbage Collection

- Two properties of dynamically allocated records
  1) Younger records more likely to die soon
  2) Younger records tend to point to older records
- All records in $G_i$ older than $G_{i+1}$
- 2\textsuperscript{nd} property means few or no pointers from $G_i$ to $G_j$, $i < j$
- Collect in youngest generation
- Detect assignments to old objects
Persistent Stores

- Dynamic allocation heap persisting from one invocation to the next
  - On stable storage
- Memory-map persistent store disk file
- Application can modify store, but must commit modifications (dirty pages)
Extending Addressability

- Persistent store can exceed $2^{32}$ objects
  - In any one run, not likely
- Use 64-bit addresses in disk objects and 32-bit addresses in core objects
- Translation table stores 64-bit pointer -> 32-bit pointer translations on pagein
  - Accessing 32-bit pointer may result in page fault
Data-Compression Paging

- Pages of 32-bit words can be compressed to $\frac{1}{4}$ of page
  - Compress instead of paging out
  - Uncompress instead of paging in
- Can be done in OS or in garbage collector
Heap Overflow Detection

• Must protect against overflow of stack
  − Mark pages above stack no-access
  − Kernel usually handles page faults by allocating physical pages for stack transparently

• Heap overflow in garbage-collected system
  − Ordinarily done with compare and conditional-branch
  − Instead, terminate heap with guard page
    • On fault, collect garbage
Primitive Performance Matters

- Algorithms share several common traits
  - Memory made less-accessible in large batches and more-accessible page by page
  - Almost all fault-handling done by CPU and takes time proportional to page size
  - Every page fault makes page more accessible
  - Frequency of faults inversely related to locality
  - User-mode service routines need access to pages protected from user-mode client routines
  - Service routines don't care about client's CPU state
Relevance to System Design

• TLB Consistency with less-accessible pages
  - Batch shootdown of TLB entries
  - Same for paging out

• Optimal page size
  - Fault-handling by CPU in $\Theta(\text{PAGE\_SIZE})$

• Access to protected pages
  - Allow one thread access while preventing others
  - Multiple mapping, kernel copy, shared pages or in-kernel thread
Conclusions

- VM design has significant impact on userspace performance and program design
- If you provide good services, applications will run better
  - Perhaps more important to provide the right services
    - Other primitives as well, e.g. pin pages (mlock)
- Also must ensure correctness
  - e.g., mprotect() not flushing TLB