Virtual Memory Primitives for User Programs

Andrew W. Appel & Kai Li

Department of Computer Science
Princeton University

Presented By: Anirban Sinha (aka Ani), anirbans@cs.ubc.ca

About the Authors

- **Andrew W. Appel** (http://www.cs.princeton.edu/~appel/)
  "**Research Interests.** I do research in computer security, compilers, programming languages, type theory, and functional programming. "

- **Kai Li** (http://www.cs.princeton.edu/~li/)
  - Supervisor of Dr. Mark Greenstreet (UBC)
  - Mainly into Parallel & Distributed Systems & Computer Architectures.

Outline

- The Main Objectives of the Paper.
- The User Level Algorithms.
- Performance Analysis of VM Primitives.
- Conclusion drawn from the analysis.
- Lessons Learnt - Design Issues.
- Analysis of the paper.

Objectives

- This is I guess the first survey paper we have got so far.
- In the words of the authors:
  - "*What virtual memory primitives should the operating system provide to user processes & how well do today's operating systems provide them*"

Virtual Machine Terminologies

- TRAP: Trap to kernel to handle page faults.
- PROT1: Write Protect 1 page.
- PROTN: Write protect N Pages.
- UNPROT: Unprotect a single page.
- DIRTY: Return list of dirty pages.
- MAP: Map save physical page at two different virtual pages with different levels of protection at same address space.

VM Applications

- Concurrent Garbage Collection.✓
- Shared virtual memory between processors. ✓
- Concurrent Checkpointing. ✓
- Generational Garbage collection. ✓
- Persistent stores. ✓
- Extending addressability.
- Data Compression Paging.
- Heap Overflow Detection. ✓

Application 1: Concurrent Garbage Collection

- Garbage collector threads can not run concurrently with mutator threads on which garbage collection is to be performed.
- Garbage collection is normally run after the end of the execution of the threads.
- It poses a problem to run garbage collector on kernel threads of OSes because Operating System threads keep running, they never really STOP.
- We need to perform garbage collection concurrently.
- This algorithm tries to address this issue.

Baker's sequential real time copying collector algorithm

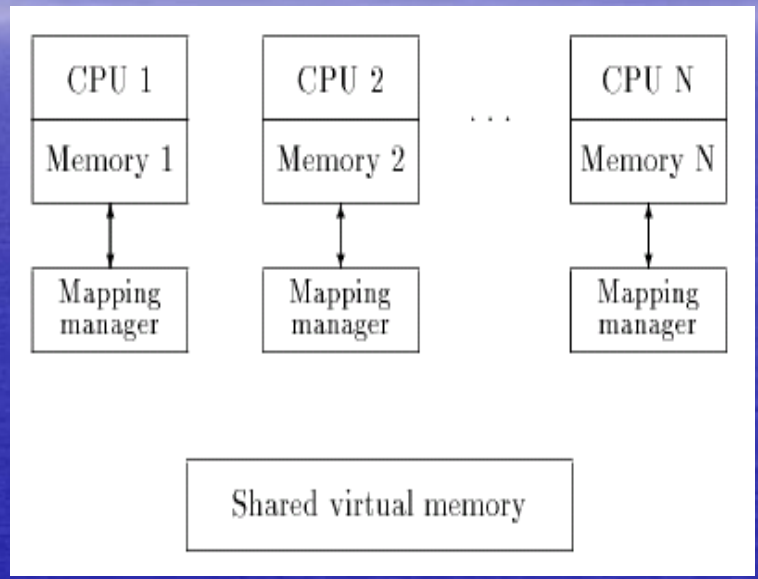
- Two classes of objects, "from" & "to". Those reachable by current thread from different paths, starting from registers & pointers for example, are moved from "from" space to "to" space incrementally.
- Mutator sees objects in "to" space only.
- When mutator allocates space for new objects using "new", it causes the collector to copy a few more objects from "from" space to "to" space.
- New objects contain "to" space pointers only.
- All objects in scanned area contain "to" space objects only.
- Unscanned area contain objects that have not been classified as "garbage" or "not garbage". They contain both "to" space & "from" space pointers.

Use Of Protected VM

- Set unscanned area pages as “no access”.
- When mutator thread tries to access an object in the unscanned area, it causes a page fault.
- Collector handles the TRAP, scans the objects on that page, removes them to “to” space, updated pointers & unlocks the page.
- Mutator does not notice that the page was originally in “from” space.
- The collector runs concurrently in the unscanned area of the code, unprotects the pages as they are being scanned & removes them to “to” space.
- If collector threads runs faster than mutator, page faults will be less.
- Uses TRAP, PROTN, UNPROT & MAP2.

Application 2: Shared VM

- Multiple Processors, some common shared VM area.
- SVM address space partitioned into pages.
- “read only” pages are shared.
- Writable pages reside in physical memory of a SINGLE processor.
- If a processor wants to write to a page other than its own, it must obtain a copy of it & invalidate other copies.
- A page fault occurs when a processor tries to write to a page other than its own.
- Uses Trap, PROT1 & UNPROT.

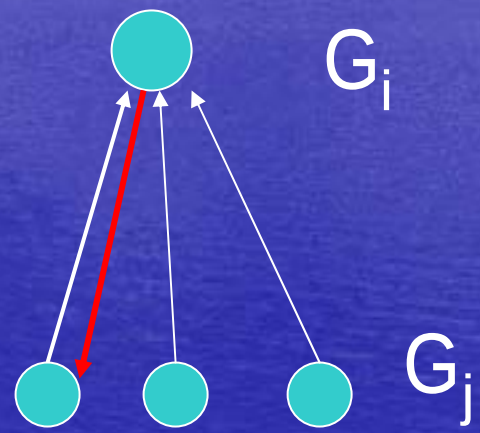


Application 3: Concurrent Checkpointing

- Originally, in order to have checkpoints, one had to stop the running thread.
- It is possible to have concurrent real time checkpoints.
- Set accessibility of entire address space pages to read only.
- Start copying the pages to a separate virtual address space.
- When copying of a page is done set it to “read-write” mode.
- When the check pointing thread access a page in “read” mode, no fault occurs.
- When it tries to write, write memory access fault occurs, the page is quickly copied to separate address space & status of the current page set to “read-write”.
- Restart faulting thread.
- Incremental checkpoints can also be applied at ease with this algorithm.
- Uses PROT, UNPROT, PROT1, PROTN, DIRTY.

Application 4: Generational Garbage collection

- Dynamically allocated records in programming languages has two important properties:
  - Younger records are more likely to terminate soon than older records.
  - More pointers go from younger records to older records than the reverse.
  - If i<j, very few pointers from G_i to G_j.
- Garbage collection efforts thus should be concentrated on younger records.
- Inspect DIRTY pages from Gi's. Else:-
  - Write protect pages of all G_i's.
  - When G_j calls G_i, trap is generated.
  - Save the trapping address in a list.
  - Unprotect page.
  - Find the possible pages from the list that are candidates for garbage in G_j.
- Uses PROTN, UNPROT, TRAP or simply DIRTY.



Application 5: Persistent Stores

- A data structure, namely a heap that is saved into the disk.
- Kept on a stable storage device.
- Implemented in UNIX using:-
  - void *mmap(void **addr*, size_t *len*, int *prot*, int *flags*, int *fildes*, off_t *off*);
  - Check http://www.opengroup.org/onlinepubs/009695399/functions/mmap.html
- Pointer traversal in persistent store is same as in incore traversal.
- Page fault can occur if the currently accessed page is not actually in memory.
- Permanent image of the file in the disk should not be modified until “commit”. During “commit”, all dirty pages moved to disk.
- Use garbage collection at commit time.

Application 6: Heap Overflow Detection

- Ordinarily, heap overflow is detected by compare & conditional branch during each memory allocation.
- Can be done more efficiently if we have a guard page which is protected at the end of the memory allocated for heap.
- Page fault occurs when program tries to access a page beyond the Heap area.
- Garbage collector is invoked which may free some unused heap pages or a rearrangement may be required.
- Restart faulting threads.
- PROT1 & TRAP used.

Outline

- The Main Objectives of the Paper. ✓
- The User Level Algorithms. ✓
- Performance Analysis of VM Primitives.
- Conclusion drawn from the analysis.
- Lessons Learnt - Design Issues.
- Analysis of the paper.

Performance Analysis

- Four stages:

1. Sum of PROT1, TRAP & UNPROT by 100 repetitions of the following steps
   - Access a random protected page.
   - In TRAP, unprotect this page & protect some other page.
2. Sum of PROTN, TRAP & UNPROT.
   - Protect 100 pages.
   - Access each page in random sequence.
   - In TRAP handler, unprotect faulting page.
3. Measure the time for a TRAP operation which does not change protection of a page – Useful for heap-overflow detections.
4. Time for a single ADD instructions my measuring time taken in a loop that iterates 20 times with each loop having 18 ADD instructions.

Results

Machine	OS	ADD	TRAP	TRAP +PROT1 +UNPROT	TRAP +PROTN +UNPROT	MAP2	PAGESIZE
Sun 3/60	SunOS 4.0	0.12	760	1238	1016	yes	8192
Sun 3/60	SunOS 4.1	0.12		2080	1800	yes	8192
Sun 3/60	Mach 2.5(xp)	0.12		3300	2540	yes	8192
Sun 3/60	Mach 2.5(exc)	0.12		3380	2880	yes	8192
SparcStn 1	SunOS 4.0.3c	0.05		*919	*839	yes	4096
SparcStn 1	SunOS 4.1	0.05	†230	1008	909	yes	4096
SparcStn 1	Mach 2.5(xp)	0.05		1550	1230	yes	4096
SparcStn 1	Mach 2.5(exc)	0.05		1770	1470	yes	4096
DEC 3100	Ultrix 4.1	0.062	210	393	344	no	4096
DEC 3100	Mach 2.5 (xp)	0.062		937	766	no	4096
DEC 3100	Mach 2.5 (exc)	0.062		1203	1063	no	4096
μVax 3	Ultrix 2.3	0.21	314	612	486	no	1024
i386 on iPSC/2	NX/2	0.15	172	302	252	yes	4096

An interesting statistic will be to normalize these results with CPU speed to get the actual picture.

Normalized statistic

- The figure shows the number of ADD instructions that can be executed by the CPU during the time it executes PROT+TRAP + UNPROT.
- Shows clearly that memory protection mechanisms & TRAP are not optimized in most operating systems.

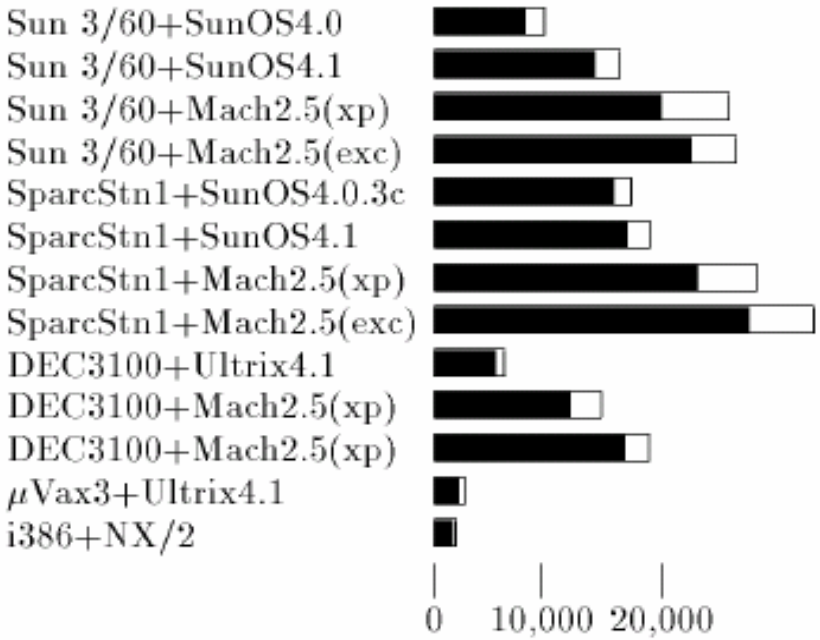


Figure 2: Instructions per PROT + TRAP + UNPROT.

The black bars show the results when pages are protected in large batches (PROTN), and the white bars are the additional time taken when pages are protected one at a time (PROT1).

Conclusions

- Wide variations between the performance of the operating systems on the same hardware.
  - Indicative of the fact that there may be scopes of improvement.
- Since all the applications discussed involve CPU only & not disk (which has very high seek latency times), it is important that these operations are optimized otherwise this becomes a bottleneck.
- The situation becomes more acute in real time systems where time constraints are very rigid.

Outline

- The Main Objectives of the Paper. ✓
- The User Level Algorithms. ✓
- Performance Analysis of VM Primitives. ✓
- Conclusion drawn from the analysis. ✓
- Lessons Learnt - Design Issues.
- Analysis of the paper.

Lessons Learnt from Analysis – The Design Issues

- Make pages less accessible in batches & more accessible in singles as & when required by the process.
  - When a page is made less accessible, outdated information in TLB can cause illegal access violation.
  - Can be prevented by flushing the TLB pages in batches because batch operation reduces the cost per page.

Lessons Learnt from Analysis – The Design Issues

- If page faults are handled by CPU, handling time directly depends on the page size.
  - It is important to make page sized optimal.
  - Traditionally though page sizes has been larger because when page faults occurs between physical memory & disk, large disk access time called for having large page size to reduce disk IO time.
  - Good idea is to small pages for PROT & UNPROT operations & for disk page faults, contiguous multi-page blocks can be used (Used in VAX).

Lessons Learnt from Analysis – The Design Issues

- When User mode service routines need to access a protected page, allow them by:-
  - Either have multiple mapping of the same page at different addresses with different protection levels in the same address page.
  - Use a separate system call.
  - other alternates exists (did not really understand fully) ...

Is it too tough to implement the applications in OSes?

- Hmm, synchronous behavior in user programs create a lot of trouble in pipelined architectures.
- But all except heap overflow detection use asynchronous behavior.
- Heap overflow detection using VM page faults in VAX or Motorola is thus ugly & is not reliable.
- Thus its not really a big ask!

Analysis of the Paper

- Strengths:-
  - Discusses a whole lot of applications of protected VM.
  - Brings out the weaknesses of OS's in implementations of this algos.
  - Proposes some design optimization issues that must be looked into.

Analysis of the Paper

- Weaknesses:-
  - I did not find the addressability extension & data compression applications too compelling.
  - Analysis of VM primitive performance in modern OS's not very exhaustive.
  - It's an old paper, ACM 1991; it would be interesting to discuss how far these ideas are still relevant I the context of modern systems having large physical memory.
  - Any other comments ? ...

Questions, Comments & Discussions ...