

# Instructor Notes

## Virtual Memory Management in the VAX/VMS Operating System

### Background

1. What challenges did VAX/VMS need to contend with?

- Wide range of workloads -- Intended to provide a single environment for all VAX-based applications (timesharing, realtime, batch)
  - o Adjust to changing demands of timesharing jobs while giving predictable performance to realtime and batch jobs
- Had to operate on a broad class of VAX-11 machines
  - o Range of configurations (250KB – 8MB of memory)
  - o Small inexpensive CPUs
  - o Slow moving head disks

(Side: what is a fixed-head paging disk)?

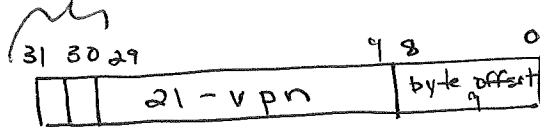
2. What were that VAX/VMS designers particularly concerned with for a paging system? Why were these issues a particular concern in their environment?
- a. One heavily paging program could impact others (worse with little memory; timesharing vs. realtime)
  - b. High startup (and restart) time of demand paging (worse with small page size of 512 bytes)
  - c. Disk traffic caused by paging (paging hurts file system activity; no separate disk)
  - d. CPU takes time dealing with page lists (slow CPUs)

3. What were their solutions?
  - a. Use process-local page replacement
  - b. Cluster multiple reads together
  - c. Employ a swapper to bring in entire resident set of process
  - d. Use simple FIFO replacement with global free and modified lists

4. HARDWARE: With the VAX-11 hardware, how is the 32-bit virtual address allocated? (DRAW.)  
 With a 512 byte page, how many bits are part of the offset? How many bits for the virtual page number (vpn)?

a. 2 high-order bits for segment or region (00, 01 – user; 10 – system; 11 – reserved)

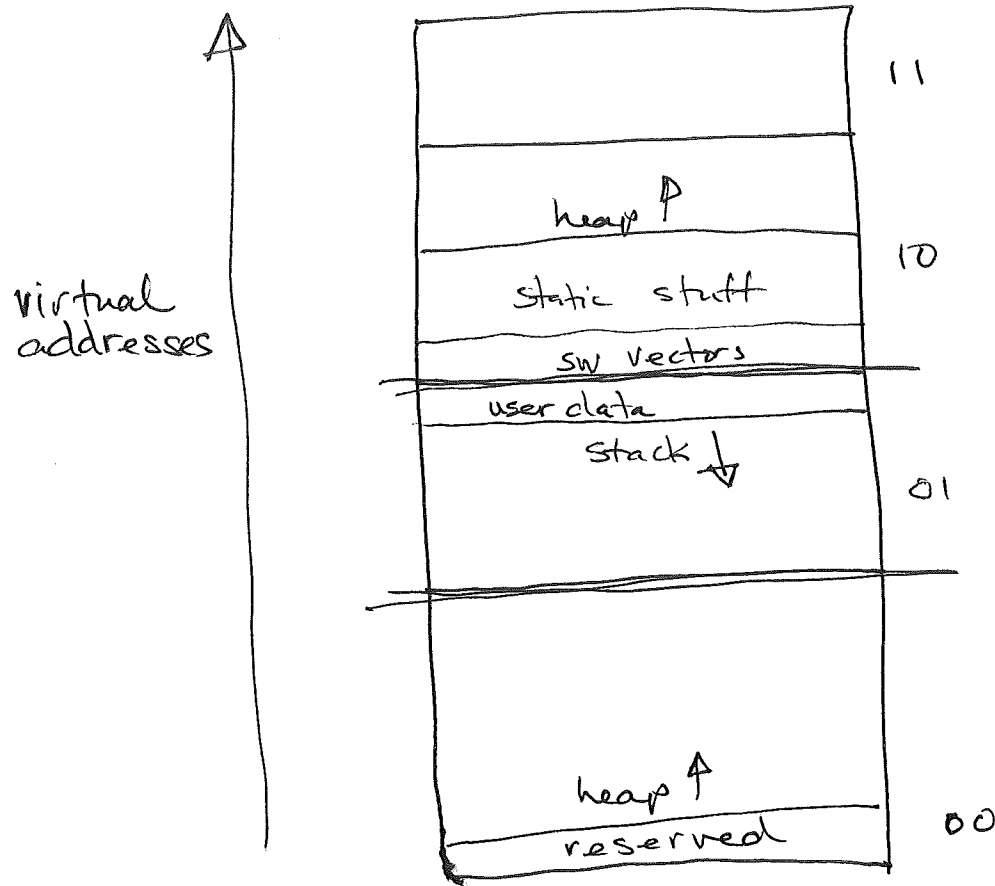
Segment



user p0 – 00      32 bits  
 user p1 – 01  
 system – 10  
 unused – 11

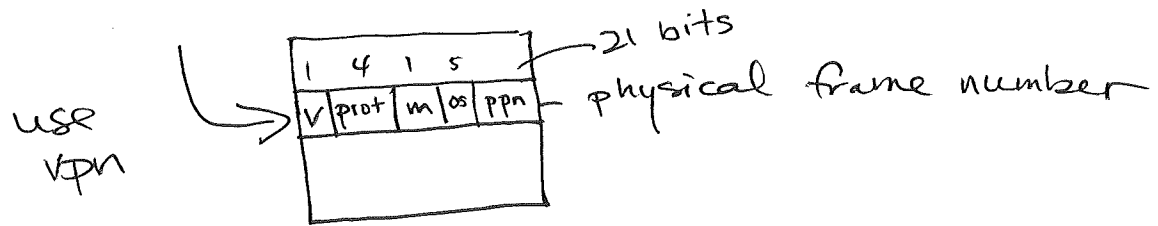
$$2^9 = 512$$

5. What does the virtual address space look like? Why is the first page of P0 region reserved?  
a. Filled with zeros for null pointer references



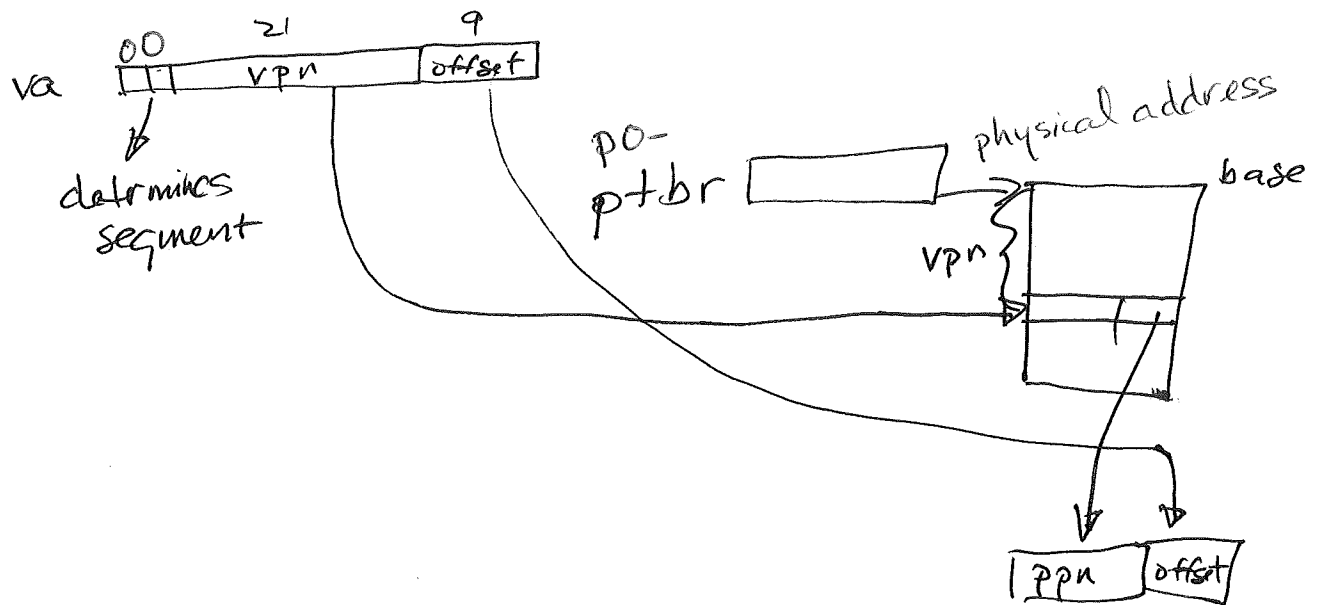
don't need to  
allocate pages or  
page table  
entries for  
parts of stack +  
heap that  
haven't been  
allocated yet

6. What does a page-table-entry (PTE) look like?



If

7. Imagine that P0 and P1 PTBR (page table base register) pointed to a physical address. How would you do an address translation? What are the drawbacks of keeping the user page tables in physical memory?



Problem: Each segment page table must be contiguous in memory - very large

Soln: Put page tables in system virtual space  
(page the page tables)

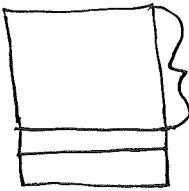
8. With the VAX/VMS approach, the user page tables are kept in the systems virtual address space. Thus, the P0 and P1 PTBR contain a virtual addresses in the system space. And the system PTBR (SPTBR) contains a physical address. What are the steps for VAX/VMS to translate a virtual address in user space to its physical address?

① Calculate v.a. of pte

vpn	offset
-----	--------

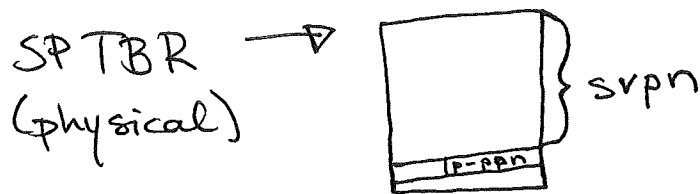
virtual address  
 $P_i \text{ PTBR} + (\text{vpn} \times \text{sizeof}(\text{entry})) \rightarrow \text{SVA}$

svpn	offset'
------	---------

base  $\rightarrow$   } vpn

have virtual address of this!  
 - read that virtual address  
 to get ppn

② <sup>Read</sup> Lookup ~~in~~ in system page table to get physical ppn of pte



$\text{sptbr} + (\text{svpn} \times \text{sizeof}(\text{entry}))$   
 tells us ppn of page table!

③ Read 

p-ppn	offset'
-------	---------

 to ~~get~~ read from user's page table  
 obtain ppn

Final: Read 

ppn : offset
--------------



9. How many extra memory accesses are required for each memory access performed by an application? How is this overhead reduced? Does anything special need to happen on a context switch?

— 2 - 1 system page table to find user page table

1 user page table to do  $vpn \rightarrow ppn$

- Use TLB

$vpn \rightarrow ppn$  mappings

- flush user portion on context switch

**Solution: Use process-local page replacement**

**Solution: Use simple FIFO replacement with global free and modified lists**

10. How does per-process local replacement of pages work?

Each process has a resident set list (current set of pages in physical memory)

Divide physical memory by number of processes (with some minimum allowable amount of memory)

Replace pages in FIFO order from resident set list

11. FIFO is not a good replacement policy because it does not handle locality in access patterns. Why didn't VAX/VMS use LRU? How does VAX/VMS get around FIFO's limitations? What happens on a page fault?

LRU requires a reference or use bit, which hardware did not provide

2<sup>nd</sup>-level global cache (victim cache); move to tail of either global FREE PAGE LIST or global MODIFIED PAGE LIST. Put on free list if modify bit is 0 (grab from head of list when need). Put on modified page list if modify bit is 1 (dirty and needs to be written out to disk); When reach high limit, write out pages (up to low limit)

On page fault, check both global lists and move back to process RESIDENT SET list if present. Approximates LRU.

12. What are the advantages of using modified page list?

- Acts as a cache

- Can write many pages at once, rearranging for better layout

- Lazy - Write avoided if page faults or process terminates

**Solution: Cluster multiple reads together**

13. What are the requirements to perform a cluster read?

Group multiple 512-byte reads into a cluster; must be contiguous on disk and in virtual memory space

14. When is clustering most effective? Why is clustering not always useful?

From initial program execution; text file of program is contiguous on disk and in VA. Clustering not as useful when reading from paging file (shared across multiple processes) if wasn't able to write out pages contiguously.

15. Can you create a synthetic workload that could perform worse with clustering?

Read just one byte from each cluster; VAX/VMS is performing "prefetching", so if don't use those pages, this causes extra disk traffic and could replace more useful pages in memory.

**Solution: Employ a swapper to bring in entire resident set of process**

16. What is the purpose of the swapper? How does the swapper work?

Keep high-priority jobs resident; avoid thrashing

Invoked when system is low on space and must swap out a process. Writes out entire process resident set list to swap file (and also handles modified page list).

Invoked when process should be swapped back in. Must find space by writing out modified page list or swapping out another process. Updates process's page table entries to reflect the new virtual-to-physical mappings.



## Conclusion

VAX/VMS: Made lots of reasonable decisions

Many good ideas

- 2<sup>nd</sup> chance global list (lazy writes)

- clustering-prefetching

- (also, demand-zero, copy-on-write);