

COS214 Tutorial 4

Roberto Togneri, 2000

SOLUTIONS

1. Explain how the following algorithms work in allocating memory:
- First fit.
 - Best fit.
 - Worst fit.

Given free memory blocks of 100K, 500K, 200K, 300K and 600K (in this order on a linked list), how would each of the above algorithms place requests for 212K, 417K, 112K and 426K (requested in that order)? Which algorithm makes best use of memory?

Answer

Initially:

BLOCK:	a	b	c	d	e
FREE:	100	500	200	300	600

Memory Allocation:

First Fit	Blocks	Free	Best Fit	Blocks	Free	Worst Fit	Blocks	Hole
212	b	288	212	d	88	212	e	388
417	e	183	417	b	83	417	b	83
112	b	176	112	c	88	112	e	276
426	Wait		426	e	174	426	Wait	

Best fit is best!

2. (a) What is the difference between internal and external fragmentation? In these days of large RAM memories is this any longer important?
- (b) What is the difference between physical and virtual addresses?
- (c) Is it reasonable to have the following situations:
- physical address space \gg virtual address space
 - virtual address space \gg physical address space
 - virtual address space $=$ physical address space
- (d) In the lecture the MMU is shown as translating every virtual address to a physical address. Explain what is required to make this an efficient process given that even a simple instruction will need at least one translation and some may require several.

Answer

- (a) internal fragmentation: unused space within allocated segment. Common in systems where allocated space is greater than required.
external fragmentation: unused space between allocated segments.
With large RAM memories the proportion of memory wasted due to fragmentation is usually less. The cost in time and performance of de-fragmenting memory is not usually worth it.
- (b) physical address: real RAM locations needed at micro level.
virtual address: pseudo addresses used by processes which must be mapped to physical addresses by MMU.

- (c) (i) PA \gg VA: Looks silly, but useful for preventing any one process hogging all the memory, that is each process has a $VA \ll PA$.
(ii) VA \gg PA: Normal use of VA to allow processes to “use” more memory than is physically available, especially separating text/data and stack by placing them at opposite ends of the VA range.
(iii) VA $=$ PA: May eliminate some overheads in the VA to PA mapping or result in fast mapping implementations since the size of $f = \text{size of } p$
- (d) Use a *fast associative memory (translation lookaside buffer)*.

3. (a) With the ever increasing number of bits per memory chip (e.g. 1M, 4M, 16M, and 64M), will there still be the need to consider virtual memory techniques?
- (b) The Intel 8086 processor did not support virtual memory. Nevertheless, some companies sold systems that contained an unmodified 8086 CPU and did paging. Make an educated guess as to how they did it.

Answer

- (a) Yes probably. So far the demand for more memory (graphics, images, ANN etc) has outstripped technology.
- (b) They built an MMU and inserted it between the CPU and the bus. Thus all 8086 physical addresses went into the MMU as virtual addresses. The MMU then mapped them onto physical addresses, which went to the bus.

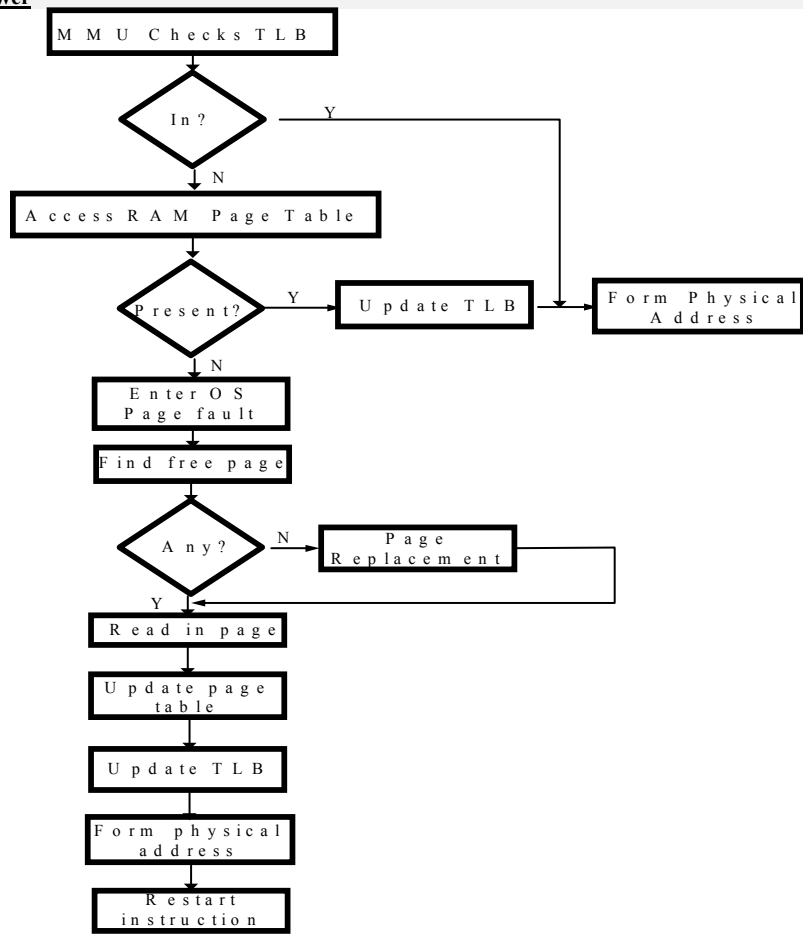
4. (a) A machine has a CPU with a 32 bit address space and uses 8K pages. The page table is entirely in hardware, with one 32 bit word per entry. When a process starts the page table is copied to the hardware from memory, at one word every 100nsec. If each process runs for 100msec (including time to load the page table), what fraction of the CPU time is devoted to loading the page tables?
- (b) A computer with a 32 bit address space uses a two-level page table. Virtual addresses are split into a 9 bit top-level page table field, an 11 bit second level page table field, and an offset. How large are the pages and how many are there in the virtual address space?

Answer

- (a) The page table contains $2^{32}/2^{13}$ entries, which is 524,288. Loading the page table takes 52.4 msec. If a process gets 100msec, this consists of 52.4 msec for loading the page table and 48 msec for running with it. Thus 52 percent of the time is spent loading page tables.
- (b) Twenty bits are used for the virtual page numbers, leaving 12 over for the offset. This yields a 4K page. Twenty bits for the virtual page implies 2^{20} pages.

5. Draw up a flow-chart to cover all the situations that arise in the operation of virtual memory paging with a TLB. Ignore the detail of running another waiting process whilst waiting for a missing page to be read in from the swap/paging disk area.

Answer



6. (a) A computer whose processes have 1024 pages in their address spaces keeps its page tables in memory. The overhead required for reading a word from the page table is 500 nsec. To reduce this overhead, the computer has an associative memory, which holds 32 (virtual page, physical page frame) pairs, and can do a look up in 100 nsec. What hit rate is needed to reduce the mean overhead to 200 nsec?
- (b) A group of operating system designers for the Frugal Computer Company are thinking about ways of reducing the amount of backing store needed in their new OS. The head guru has just suggested not bothering to save the program text in the swap area at all, but just page it in directly from the binary file whenever it is needed. Are there any problems with this approach?
- (c) Consider a paging system with the page table stored in memory. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take? If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

Answer

- (a) The effective instruction time is $100h + 500(1-h)$, where h is the hit rate. If we equate this formula with 200 and solve for h we find that h must be at least 0.75.
- (b) What happens if the executable binary file is modified while the program is running? Using a mixture of pages from the old and new binaries is sure to crash it. While this event is unlikely, it is a calculated risk. Alternatively, the binary could be locked against modification while it was being used as a backing store.
- (c) 400 nanoseconds: 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory. With a TLB the effective access time = $0.75 \times (200 \text{ nanoseconds}) + 0.25 \times (400 \text{ nanoseconds}) = 250 \text{ nanoseconds}$.

7. (a) For a 2-level page table system with pure paging detail the exact software and hardware implementations.
- (b) A 32-bit OS uses a 2-level page table scheme with 4K pages. The page table structures are stored in main memory (this is the hint!). Explain why $p = 10$ (10-bits to index top-level page table) and $q = 10$ (10-bits to index the 2nd level page table) is the only solution of assigning p and q (such that $p + q = 20$) to minimise internal fragmentation?
- (c) The OS in (b) gets a VA = 00D4D578H. Detail which page table structures and entries are accessed (in the event of a TLB miss) and how the PA is formed.

Answer

- (a) *Software*: Main memory used to store the top-level and 2nd level page tables. Kernel code to handle memory management.
Hardware: TLB cache, Top-level page table pointer Register
- (b) With $p = 10$ and $q = 10$ there are $2^{10} = 1024$ entries in each page table. The top-level page table will require 32-bit entries to store the pointer to the 2nd level page table. The 2nd level page tables will also have 32-bit entries to conform to the 32-bit word boundary operation of the OS. Hence each page table is 1024×4 bytes = 4096 bytes = 4K in size which fits neatly in one page frame without any internal fragmentation. IT Engineers will ensure that is a deliberate design outcome not luck!
- (c) VA = 00D4D578
→ $p = 00[11]H$, $q = [01]4DH$, $d = 578H$
→ $p = 3$, $q = 333$, $d = 1400$
Entry #3 in the top-level page table is indexed and the pointer to 2nd level page table #3 is retrieved. Entry #333 in 2nd level page table #3 yields the page frame number, f and the PA = [f | 578H], that is offset 1400 in page frame # f .