

Section 8: Performance and Address Translation

July 20, 2020

Contents

1	Vocabulary	2
2	Queuing Theory	3
3	Paging and Address Translation	5
3.1	Conceptual Questions	5
3.2	Page Allocation	6
3.3	Address Translation	7
3.4	Page Fault Handling for Pages Only On Disk	8

1 Vocabulary

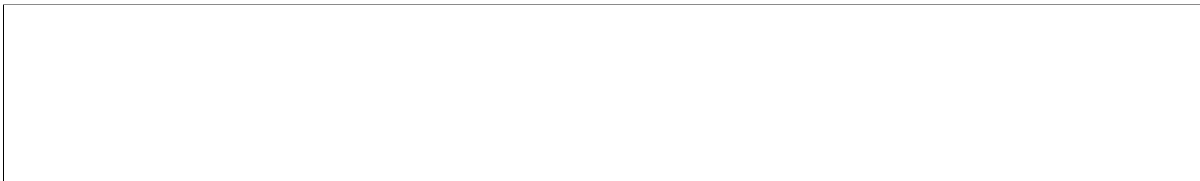
- **Queuing Theory** Here are some useful symbols: (both the symbols used in lecture and in the book are listed)
 - μ is the average service rate (jobs per second)
 - T_{ser} or S is the average service time, so $T_{ser} = \frac{1}{\mu}$
 - λ is the average arrival rate (jobs per second)
 - U or u or ρ is the utilization (fraction from 0 to 1), so $U = \frac{\lambda}{\mu} = \lambda S$
 - T_q or W is the average queuing time (aka waiting time) which is how much time a task needs to wait before getting serviced (it does not include the time needed to actually perform the task)
 - L_q or Q is the average length of the queue, and it's equal to λT_q (this is Little's law)
- **Virtual Memory** - Virtual Memory is a memory management technique in which every process operates in its own address space, under the assumption that it has the entire address space to itself. A virtual address requires translation into a physical address to actually access the system's memory.
- **Memory Management Unit** - The memory management unit (MMU) is responsible for translating a process' virtual addresses into the corresponding physical address for accessing physical memory. It does all the calculation associating with mapping virtual address to physical addresses, and then populates the address translation structures.
- **Address Translation Structures** - There are two kinds you learned about in lecture: segmentation and page tables. Segments are linearly addressed chunks of memory that typically contain logically-related information, such as program code, data, stack of a single process. They are of the form (s,i) where memory addresses must be within an offset of i from base segment s. A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. Virtual addresses are used by the accessing process, while physical addresses are used by the hardware or more specifically to the RAM.
- **Translation Lookaside Buffer (TLB)** - A translation lookaside buffer (TLB) is a cache that memory management hardware uses to improve virtual address translation speed. It stores virtual address to physical address mappings, so that the MMU can store recently used address mappings instead of having to retrieve them multiple times through page table accesses.

2 Queuing Theory

Explain intuitively why response time is nonlinear with utilization. Draw a plot of utilization (x axis) vs response time (y axis) and label the endpoints on the x axis.



If 50 jobs arrive at a system every second and the average response time for any particular job is 100ms, how many jobs are in the system (either queued or being serviced) on average at a particular moment? Which law describes this relationship?



Is it better to have N queues, each of which is serviced at the rate of 1 job per second, or 1 queue that is serviced at the rate of N jobs per second? Give reasons to justify your answer.



What is the average queuing time for a work queue with 1 server, average arrival rate of λ , average service time S , and squared coefficient of variation of service time C ?

What does it mean if $C = 0$? What does it mean if $C = 1$?

3 Paging and Address Translation

3.1 Conceptual Questions

If the physical memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

If the physical memory size (in bytes) is doubled, how does the number of entries in the page table change?

If the virtual memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

If the virtual memory size (in bytes) is doubled, how does the number of entries in the page map change?

If the page size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

If the page size (in bytes) is doubled, how does the number of entries in the page table change?

The following table shows the first 8 entries in the page table. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

Valid Bit	Physical Page
0	7
1	9
0	3
1	2
1	5
0	5
0	4
1	1

If there are 1024 bytes per page, what is the physical address corresponding to the hexadecimal virtual address 0xF74?

3.2 Page Allocation

Suppose that you have a system with 8-bit virtual memory addresses, 8 pages of virtual memory, and 4 pages of physical memory.

How large is each page? Assume memory is byte addressed.

Suppose that a program has the following memory allocation and page table.

Memory Segment	Virtual Page Number	Physical Page Number
N/A	000	NULL
Code Segment	001	10
Heap	010	11
N/A	011	NULL
N/A	100	NULL
N/A	101	NULL
N/A	110	NULL
Stack	111	01

What will the page table look like if the program runs the following function? Page out the least recently used page of memory if a page needs to be allocated when physical memory is full. Assume that the stack will never exceed one page of memory.

```
#define PAGE_SIZE 1024;
```

```
void helper(void) {
    char *args[5];
```

```

int i;
for (i = 0; i < 5; i++) {
    // Assume malloc allocates an entire page every time
    args[i] = (char*) malloc(PAGE_SIZE);
}
printf("%s", args[0]);
}

```

What happens when the system runs out of physical memory? What if the program tries to access an address that isn't in physical memory? Describe what happens in the user program, the operating system, and the hardware in these situations.

3.3 Address Translation

Consider a machine with a physical memory of 8 GB, a page size of 8 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page?

List the fields of a Page Table Entry (PTE) in your scheme.

Without a cache or TLB, how many memory operations are required to read or write a single 32-bit word?

With a TLB, how many memory operations can this be reduced to? Best-case scenario? Worst-case scenario?

The pagemap is moved to main memory and accessed via a TLB. Each main memory access takes 50 ns and each TLB access takes 10 ns. Each virtual memory access involves:

- mapping VPN to PPN using TLB (10 ns)
- if TLB miss: mapping VPN to PPN using page map in main memory (50 ns)
- accessing main memory at appropriate physical address (50 ns)

Assuming no page faults (i.e. all virtual memory is resident) what TLB hit rate is required for an average virtual memory access time of 61ns?

Assuming a TLB hit rate of .50, how does the average virtual memory access time of this scenario compare to no TLB?

3.4 Page Fault Handling for Pages Only On Disk

The page table maps VPN to PPN, but what if the page is not in main memory and only on disk? Think about structures/bits you might need to add to the page table/OS to account for this. Write pseudocode for a page fault handler to handle this.

