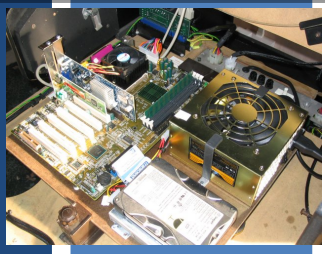
TCSS 422: OPERATING SYSTEMS



Practice Final Exam Questions

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (28 of 46 respondents (10 online) – 60.9%):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.61 (- previous 6.45)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.14 (- previous 5.36)**

FEEDBACK FROM 3/12

- ***What provides the worst-case (hit ratio) when it comes to cache replacement policies?***
- We reviewed: Optimal, FIFO, Random, and (L)east-(R)ecently-(U)sed
- Worst-case hit ratios result from a combination of: (1) replacement policy, (2) cache size, and (3) the workload
- **Random-Access (No Locality) workload**: LRU, FIFO, RAND perform equally poorly until cache size scales up to match the number of memory pages accessed. Optimal is Best
- **80/20 workload:** 80% accesses → 20% hot pages, 20% accesses → 80% cold pages
  RAND/FIFO are worst-case. OPT is Best, LRU is good. Hit rate improves with larger cache size
- **Looping-sequential workload:** LRU/FIFO are worst case until cache size >= #-of-pages, RAND is better, OPT is best

QUESTION 1 – BASE AND BOUNDS

- A computer system uses a simple base/bounds register pair to virtualize address spaces. For each traces fill in the missing values of virtual addresses, physical addresses, base, and/or bounds registers. In some cases, it is not possible to provide an exact value. If so, specify a range (e.g. greater than 100), or value that is not a single number.

Scenario 1

Virtual Address	Physical Address		
100	600		
300	800	Base?	500
699	1199		
700	[fault]	Bounds?	700

Q1 - 2

1801
+1200
3001

Scenario 2

Virtual Address	Physical Address		
300	1500	Base?	1200
1600	2800		
1801	3001 ?	Bounds?	> 2801
2801	4001		

Scenario 3

Virtual Address	Physical Address		
0	1000	Base?	1000
100	1100		
1999	2999	Bounds?	2000
2000	[fault]		

QUESTION 2 – SINGLE-LEVEL PAGE TABLE

- Consider a computer with 4 GB (2^{32}) of physical memory, where the page size is 4 KB (2^{12}). For simplicity assume than 1GB=1000MB, 1MB=1000KB, 1KB=1000 bytes
- (a) How many pages must be tracked by a single-level page table if the computer has 4GB (2^{32}) of physical memory and the page table size is 4 KB (2^{12})? $2^{32} \div 2^{12} = 2^{20}$ ≈1 million PFs
- (b) How many bits are required for the virtual page number (VPN) to address any page within this 4GB (2^{32}) memory space? 20 BITS
- (c) Assuming that the smallest addressable unit of memory within a page is a byte (8-bits), how many bits are required for the offset to refer to any byte in the 4 KB page? 12 BITS
- (d) Assuming each page table entry (PTE) requires 4 bytes of memory, how much memory is required to store the page table for one process (in MB)? $2^{20} * 2^{2} = 2^{22} =$ 4MB

Q2 - 2

- (e) Using this memory requirement, how many processes would fill the memory with page table data on a 4GB computer? $2^{32} \div 2^{22} = 2^{10} = 1,024 \sim 1000$

QUESTION 3 - TWO-LEVEL PAGE TABLE

- Consider a computer with 1 GB (2^{30}) of physical memory, where the page size is 1024 bytes (1KB) (2^{10}). We would like to index memory pages using a two level page table consisting of a page directory which refers to page tables which are created on demand to index the entire memory space. $2^{30} \div 2^{10} = 2^{20}$ CUT IN HALF
- For simplicity assume than 1GB=1000MB, 1MB=1000KB, 1KB=1000 bytes
- (a) For a two-level page table, divide the VPN in half. How many bits are required for the page directory index (PDI) in a two-level scheme? 10 BITS
- (b) How many bits are required for the page table index (PTI)? 10 BITS
- (c) How many bits are required for an offset to address any byte in the 1 KB page? 10 BITS

Q3 - 2

32 bits

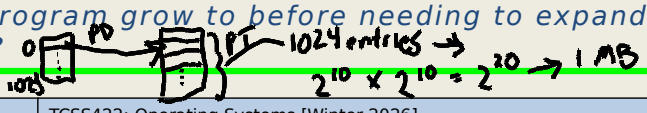
- (d) Assuming each page table entry (PTE) requires 4 bytes of memory, how many extra bits are available for status bits?

10 bits for the PTI, 32 - 10 = 22 → 12 bits

- (e) HelloWorld.c consists of 4 memory pages. One code page, one heap page, one data segment page, and one stack segment page. How large is the two-level page table in bytes with the structure described above that could index the all 4 memory pages of HelloWorld.c?
  Hint: There should be 2 tables, a page directory, and a page table. PD $2^{10} \times 2^2 = 2^{12} = 4KB$ PT $= 2^{12} = 4KB$ PD+PT $= 8KB$

8,192 bytes

- *(f) Assuming the same page table as for HelloWorld.c, using the exact same two-level page table, how large in bytes could the program grow to before needing to expand the page table?*



QUESTION 4 – CACHE TRACING

- Consider a 3-element cache with the cache arrival sequences below.
- Determine the number of cache hits and cache misses using each of the following cache replacement policies:

A. Optimal policy

Arrival sequence:

5 3 7 5 3 1 0 7 1 6 4 3 2 1 3

m m m H H m m m H H m m m m m H H

Working Cache

Cache 1: 5 1

Cache 2: 3 0 6 3

Cache 3: 7 4 2

Hits: 6

Misses: 9

Q4 - 2

B. FIFO policy

Arrival sequence: 5 3 7 5 3 1 0 7 1 6 4 3 2 1 3

Working Cache
Cache 1:
Cache 2:
Cache 3:

Hits: 5
Misses: 10

C. LRU policy

Arrival sequence: 5 3 7 5 3 1 0 7 1 6 4 3 2 1 3

Working Cache
Cache 1:
Cache 2:
Cache 3:

Hits: 4
Misses: 11

QUESTION 5 – FREE SPACE MANAGEMENT

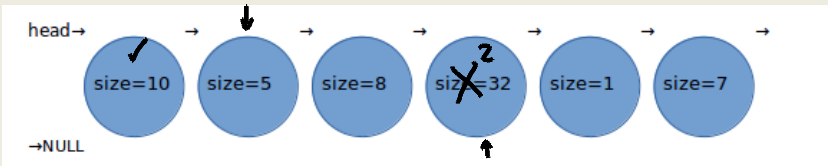
- Free space management involves capturing a description of the computer's free memory using a data structure, storing this data structure in memory, and OS support to rapidly use this structure to determine an appropriate location for new memory allocations. An efficient implementation is very important when scaling up the number of operations the OS is required to perform.
- Consider the use of a linked list for a free space list where each node is represented by placing the following structure in the header of the memory chunk:

```
typedef struct __node_t
{
    int size;
    struct __node_t *next;
} node_t;
```

sizeof(node_t);
4 bytes
4 + 8 = 12 bytes

Q5 - 2

- Consider the following free space list:



- (a) Consider the **next fit** allocation strategy. For this free list above, how many comparison operations must be performed to identify a free chunk of **30-bytes** ? 4
- (b) After the last free space identification, the chunk is split and the remaining free space is returned to the free space list. Now, consider the **next fit** allocation strategy. After finding a free space for the previous request, how many comparisons are required to identify a free chunk of 10-bytes? 4

Q5 - 3

- Now, after the last free space identification the chunk is split and the remaining free space is returned to the free space list. Now consider each of the following free space allocation strategies. How many comparisons are required on the updated free space list to find a free chunk of 2 bytes using:
- (c) best fit? 5 (3)
- (d) worst fit? 5
- (e) first fit? 1

Q6 - TLB CACHING QUESTION

- Consider a computer that indexes memory using 4 KB pages
- The sizeof(long) in 64-bit Linux is 8 bytes
- Consider an array of 10,000 8-byte long ints (80,000 bytes)

```
long big_array[10000];
```

- Assume static allocation on stack pages
- Assuming big_array[0] is allocated on the start of a 4K memory page, and array allocation is contiguous, how many TLB hits and misses will occur with the following code ?

```
for (int i=0;i<10000;i++)
    big_array[i]=0;
```

80000 / 4096 = 19.5 PGS → 20

- long array 80,000 bytes storage, spans 20 memory pages
- 20 misses, 9,980 hits
- 998 hits to 1 miss ratio = 99.8% cache hit rate

Q7 – CONCURRENCY

- (A) In the context of concurrent data structures, describe the broad concept of the lock granularity trade-off. Examples of concurrent data structures include: concurrent linked-list, concurrent queue, concurrent hash table, etc.

1 LOCK TO n nodes → n locks for n nodes

Simple Design — Low Parallelism - concurrency — Low lock API overhead — 1 lock - coarse lock

HAND-OVER-HAND LOCKING — HIGH PARALLELISM - HIGH concurrency — HIGH LOCK API OVERHEAD — n-locks - FINE GRAINED

- (B) For the synchronized array in Quiz 3, what is the issue with adding locks inside the worker? Why should locks be added to get()? Why should the sum() method not call get()?

- ADDING LOCKS TO THE WORKER DOES NOT MAKE THE SYNC ARRAY THREAD-SAFE
- MAKES get() ATOMIC - NO ONE CAN CHANGE THE ARRAY WHILE IT IS BEING READ
- LOCK API OVERHEAD - SUM SHOULD ACQUIRE LOCK FOR LIST TRAVERSAL

Q8 – MEMORY APIS

- (A) For allocating dynamic memory on the heap, which API helps programmers to resize memory allocations while also copying any existing data to the newly resized structure to ensure that the original data remains available ?

realloc ()

- (B) A programmer calls malloc() to allocate an array of characters, and then uses printf to display the contents to the screen. What issue can occur with printing the array of characters? And what memory API can help address this issue ?

- UNINITIALIZED MEMORY CAN CONTAIN RANDOM DATA - OLD DATA
- calloc () ← clear CLEARS OLD DATA

QUESTIONS