CS 351 Spring 2017

Final Exam

May 1^{st}, 2017

Instructions:

- This exam is closed-book, closed-notes. Calculators are not permitted.
- For numbered, multiple-choice questions, fill your answer in the corresponding row on the "bubble" sheet.
- For problems that require a written solution (labeled with the prefix "WP"), write your answer in the space provided on the written solution sheet. Please write legibly and clearly indicate your final answer.
- Turn in the exam question packet, bubble sheet, and written solution sheet separately.
- Good luck!

Concepts. (30 points):

Please choose the *single best* answer to each of the following questions.

1. Which scenario best takes advantage of spatial locality?

   (a) randomly accessing elements within a single large array
   (b) iterating over an array with a stride of 1
   (c) repeatedly accessing a loop counter
   (d) storing a frequently used value in a register

2. What best motivates higher cache *associativity*?

   (a) capacity misses due to large working set sizes
   (b) good spatial locality
   (c) thrashing due to collisions
   (d) small-stride array accesses

3. Which aspect of memory is directly managed by user-level code, and so is best optimized by the compiler?

   (a) mapping variables to registers
   (b) mapping virtual to physical pages
   (c) translating virtual to physical addresses
   (d) deciding what values to store in the cache

4. What is the rationale for using physical addresses to perform cache lookups?

   (a) physical addresses can be sent directly to the cache from the CPU
   (b) it allows cached data to be shared between different processes without ambiguity
   (c) virtual addresses do not fit in registers
   (d) virtual addresses are not large enough to derive the necessary fields (offeset, index, etc.) used in cache lookups

5. What action(s) must be taken by the operating system during each context switch when paging is used to implement virtual memory?

   (a) updating the page table base register
   (b) evicting pages used by the outgoing process
   (c) compacting pages of memory currently in use
   (d) flushing the cache to allow data for the incoming process to be cached

6. What entity is responsible for evicting/reading pages from/into memory and updating corresponding page table entries?

   (a) the memory management unit (MMU)
   (b) the operating system kernel
   (c) the dynamic memory allocator (DMA)
   (d) the user program

7. Which is most likely to result in a decreased TLB hit rate?

   (a) increasing the page size
   (b) decreasing the page size
   (c) increasing the data cache block size
   (d) decreasing the amount of physical memory available

8. Which entity is responsible for carrying out the *mark* phase of the mark-and-sweep garbage collection algorithm?

   (a) the memory management unit (MMU)
   (b) the operating system kernel
   (c) the dynamic memory allocator (DMA)
   (d) the user program

9. Which best explains why external fragmentation of heap memory is difficult to estimate from within an explicit DMA?

   (a) it only occurs as a result of programmer (user) error
   (b) it only results from calls to `realloc`, which occur infrequently
   (c) it depends on the pattern of future allocation requests
   (d) it is proportional to the amount of internal fragmentation

10. The primary benefit to using an *explicit free list* over an *implicit list* is that:

    (a) only free blocks need be searched to find a fit
    (b) it becomes possible to perform `free` in O(1) time
    (c) the entire list need not be traversed to locate the prior block
    (d) best fit becomes a O(lg(N)) operation, where N is the # of free blocks

11. Which action is `malloc` *not* permitted to carry out?

    (a) splitting a free block
    (b) requesting more space using `sbrk`
    (c) removing a node from an explicit linked list of free blocks
    (d) moving a currently allocated block to make room for the new payload

12. Which is most likely to result in a substantial increase in DMA *utilization*?

    (a) splitting when allocating (when possible)
    (b) switching from an implicit to an explicit list mechanism for tracking free blocks
    (c) switching to a "next-fit" from a "first-fit" policy when searching for free blocks
    (d) preferring to call `sbrk` over searching the free list

13. Which is most directly responsible for a C tracing garbage collector being *conservative*?

    (a) the lack of run-time type information
    (b) the restriction on moving allocated blocks in the heap
    (c) the allocation of static variables in heap space
    (d) the possibility of circular references between dynamically allocated structures

14. What advantage does a tracing garbage collector have over a reference counting g.c.?

    (a) there is less overhead when garbage collection takes place
    (b) circular heap references do not need to be given special consideration
    (c) precise garbage collection is possible
    (d) objects are deallocated immediately once they become unreachable

15. What advantage does a reference counting garbage collector have over a tracing g.c.?

    (a) garbage collection can be limited to only recently allocated objects
    (b) circular heap references do not need to be given special consideration
    (c) searching for free blocks in the heap is significantly more efficient
    (d) objects are deallocated immediately once they become unreachable

Cache Hit/Miss Rates (12 points):

Consider the following function, which takes the dimension `D` of a square array to process, and a pointer to the array:

```
void procArr(int D, int A[D][D]) {
  int i, j, tmp;
  for (i=0; i<D; i++) {
    for (j=0; j<D; j++) {
      tmp = A[i][j];
      A[i][j] = A[D-(i+1)][D-(j+1)];
      A[D-(i+1)][D-(j+1)] = tmp;
    }
  }
}
```

For the following questions, only consider memory accesses to the array `A` in the function `procArr` — i.e., assume that all other variable are mapped to registers — and that `int`s are 4 bytes wide.

WP1 (a). Given an argument array with dimension `D=4`, what is the total number of memory accesses — broken down into separate read and write counts — performed by the function `procArr`?

WP1 (b). Given a direct-mapped data cache with 16-byte blocks and 4 total lines, what is the total number of misses incurred while running `procArr` on an array with `D=4`?

Assume the beginning of the array is aligned to the first byte in the cache, and the cache starts out empty.

WP1 (c). Taking the diagram provided on the worksheet to represent the array `A`, shade in the specific cells where *misses* occur in your solution to part (b).

WP1 (d). Using the same cache described in part (b), but this time with an array of dimension `D=8`, what is the total number of misses incurred by `procArr`?

Again, assume the beginning of the array is aligned to the first byte in the cache, and the cache starts out empty.

VM Structures (8 points):

Consider a system that uses 48-bit virtual addresses and has 8GB (2^{33} bytes) of physical DRAM installed. Paging is used as to implement virtual memory, and the page size is 1MB (2^{20} bytes). Memory is byte-addressed, and the word size is 8 bytes.

16. What is the total number of pages in a process's virtual address space?

    (a) 2^{20} pages
    (b) 2^{23} pages
    (c) 2^{28} pages
    (d) 2^{33} pages

17. Given single-level page tables with word-sized page table entries, how large is the size of each per-process page table?

    (a) 2^{23} bytes (8MB)
    (b) 2^{28} bytes (256MB)
    (c) 2^{31} bytes (2GB)
    (d) 2^{40} bytes (1TB)

18. Given a two-level page table setup, where the first-level page directory contains 16K (2^{14}) word-sized pointers to second-level page tables, what is the minimum amount of physical memory required for a process's paging structures?

    (a) 2^{17} bytes (128KB)
    (b) 2^{20} bytes (1MB)
    (c) 2^{24} bytes (16MB)
    (d) 2^{28} bytes (256MB)

19. Consider a process with 10MB of data stored in its heap region. If each page in said region contains, on average, 10KB of data (payload), approximately how much heap memory is being wasted to internal fragmentation?

    (a) 9KB
    (b) 9MB
    (c) 90MB
    (d) 990MB

Address Translation (10 points):

The questions on the next page are based on the cache and memory setup described below:

- Memory is byte addressable
- Virtual addresses are 20 bits wide
- Physical addresses are 15 bits wide
- The page size is 256 bytes (2^8 bytes)
- The TLB is fully-associative, with 8 total lines
- The cache is 4-way set associative, with 4-byte blocks and 8 total lines

The current TLB, cache, and partial page table contents are given here:

TLB			
Index	Tag	Valid	PPN
0	DEF	1	56
	080	1	10
	EEC	1	29
	374	1	35
	34D	0	3B
	2DD	0	26
	DEB	0	5C
	7DD	0	3E

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	04	1	08	10	0
01	44	1	09	15	0
02	2D	0	0A	28	1
03	56	0	0B	11	1
04	23	0	0C	03	0
05	3C	0	0D	6E	1
06	29	0	0E	41	1
07	39	1	0F	33	0

Cache						
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	936	0	BE	11	2B	E7
	735	1	B5	88	F1	58
	F7B	0	40	B6	3D	BA
	0E8	0	91	D5	7E	ED
1	280	1	A7	2D	04	4E
	DA8	0	24	C0	5A	36
	54B	0	86	4D	EC	8E
	6A9	1	41	D8	5C	46

And for your convenience, here's a decimal ↔ hex ↔ binary lookup table:

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

WP2 (a). Using the diagrams on the worksheet, indicate where the following fields are to be found in the virtual and physical addresses (if they exist).

- Virtual address fields:

  VPO : The virtual page offset
  VPN : The virtual page number
  TLBI : The TLB index
  TLBT : The TLB tag

- Physical address fields:

  PPO : The physical page offset
  PPN : The physical page number
  CO : The cache block offset
  CI : The cache set index
  CT : The cache tag

WP2 (b). Translate the virtual address **0x3744D**.

WP2 (c). Translate the virtual address **0x007AA**.

For parts **(b)** and **(c)**, fill in all possible fields in the provided diagrams and tables. If a page fault occurs, leave the physical-address related fields empty.

DMA Metrics (8 points):

Given the following sequence of calls to `malloc` and `free`:

```
void *p1, *p2, *p3, *p4, *p5, *p6;
p1 = malloc(512);
p2 = malloc(512);
free(p1);
p3 = malloc(512);
p4 = malloc(512);
p5 = malloc(512);
free(p2);
free(p4);
p6 = malloc(512);
```

20. What is the maximum aggregate payload?

    (a) 512 bytes
    (b) 1024 bytes
    (c) 2048 bytes
    (d) 3072 bytes

21. Given a final heap size of 4096 bytes, what is the peak memory utilization?

    (a) 100%
    (b) 50%
    (c) 25%
    (d) 15%

Next, consider an explicit-list based dynamic memory allocator with the following properties:

- blocks and payloads are word-aligned, with a word size of 8 bytes
- all blocks have a header and footer word used to store the block size & allocated bit
- free blocks have a next and prior pointer (one word each) to maintain the explicit list

22. What is the minimum block size?

    (a) 16 bytes
    (b) 24 bytes
    (c) 32 bytes
    (d) 48 bytes

23. Given a payload request of 50 bytes, what is the minimum size of a block that can satisfy the request *and* be split into an allocated and free block?

    (a) 88 bytes
    (b) 96 bytes
    (c) 104 bytes
    (d) 112 bytes

Implicit List DMA (18 points):

Consider an implicit-list based allocator and memory architecture with the following properties:

- words are 4-bytes in width, and blocks and payload are word-aligned
- every block has word-sized boundary tags (each corresponding to a `size_t`) that store the size of the block and an allocated bit
- immediate coalescing is performed, and blocks are split during allocation whenever doing so results in two blocks with non-zero payload
- when splitting, the "lower" part of a block is allocated
- a first-fit search is used during allocation

WP3. Starting with the leftmost heap depicted on the worksheet, and moving from left to right, show the updated header/footer values on the heap after each successive call to `malloc`, `free`, and `realloc`.

WP4. Implement the function `void coalesce_both(void *bp);` which is passed a pointer to the start of an allocated block to be freed, that is known to be surrounded on both sides by free blocks. It should coalesce the three blocks, updating header(s) and footer(s) appropriately.

The following definitions are given to help work with aligned values:

```
#define ALIGNMENT 4
#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~3L)
#define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
```

Your implementation should not make use of any external macros nor functions.