Final Exam
ECE 454F 2009: Computer Systems Programming
Date: Thursday, Dec 17, 2009 9:30 a.m. - 12:00 a.m.

Instructor: Cristiana Amza
Department of Electrical and Computer Engineering
University of Toronto

Problem number	Maximum Score	Your Score
1	6	
2	10	
3	20	
4	24	
5	10	
6	30	
total	100	

This exam is open textbook and open lecture notes. You have two hours and a half to complete the exam. Use of computing and/or communicating devices is NOT permitted. You should not need any such devices. You can use a basic calculator if you feel it is absolutely necessary.
Work independently. Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Scratch space is available at the end of the exam.
Write your name and student number in the space below. Do the same on the top of each sheet of this exam book.

Your Student Number	
Your First Name	
Your Last Name	

Student Number: Name:

Problem 1. Basic Facts. (6 Points)

a. Explain in what cases you would recommend the use of ticket locks as opposed to MCS locks.

b. Is the MCS locks implementation more space efficient than Anderson's array-based locks ? Explain why, or why not.

c. Is flag-based synchronization, where busy waiting on flag variables is used better than pthread synchronization based on signal-wait on condition variables for producer-consumer problems ? Explain any trade-offs or cases that may exist.

Student Number: Name:

Problem 2. Performance Optimization. (10 Points)

Consider the following function for computing the dot product of two arrays of n integers each. We have unrolled the loop by a factor of 4.

```
int dotprod(int a[], int b[], int n)
{
  int i, x1, y1, x2, y2, x3, y3, x4, y4;
  int r = 0;
  for (i = 0; i < n-3; i += 4) {
    x1 = a[i]; x2 = a[i+1]; x3 = a[i+2]; x4 = a[i+3]
    y1 = b[i]; y2 = b[i+1]; y3 = b[i+2]; y4 = b[i+3]
    r = r + x1 * y1 + x2 * y2 + x3 * y3 + x4 * y4; // Core computation
  }
  for (; i < n; i++)
     r += a[i] * b[i];
  return r;
}
```

Assume that we run this code on a machine in which multiplication requires 7 cycles, while addition requires 5. Further, assume that these latencies are the only factors constraining the performance of the program. Don't worry about the cost of memory references or integer operations, resource limitations, etc.

Please compute the ideal cycles per element (CPE) for each of the following associations for the core computation of this function and explain its performance. *Note: 1. If your answer is a fraction, leave it as such, you don't need to compute the result as a decimal number. 2. You need to explicitly state your assumptions, if any, and/or explicitly show your work.*

a. ((r + x1 * y1) + x2 * y2) + (x3 * y3 + x4*y4)

b. (r + ((x1 * y1 + x2 * y2) + x3 * y3)) + x4*y4

Student Number: Name:

Problem 3. Cache Miss Rates. (20 Points)

After watching the presidential election you decide to start a business in developing software for electronic voting. The software will run on a machine with a 1024-byte direct-mapped data cache with 64 byte blocks.

You are implementing a prototype of your software that assumes that there are 7 candidates. The C-structures you are using are:

```
struct vote {
    int candidates[7];
    char valid;
};

struct vote vote_array[16][16];
register int i, j, k;
```

You have to decide between two alternative implementations of the routine that initializes the array `vote_array`. You want to choose the one with the better cache performance.

You can assume:

- `sizeof(int) = 4`
- `sizeof(char) = 1`
- `vote_array` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `vote_array`. Variables `i`, `j` and `k` are stored in registers.

A. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++)
    for (j=0; j<16; j++)
        vote_array[i][j].valid=0;

for (i=0; i<16; i++)
    for (j=0; j<16; j++)
        for (k=0; k<7; k++)
            vote_array[i][j].candidates[k] = 0;
```

Student Number: Name:

Miss rate in the first loop: _____ %

Miss rate in the second loop: _____ %

Overall miss rate for writes to `vote_array`: _____ %

B. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        for (k=0; k<7; k++) {
            vote_array[i][j].candidates[k] = 0;
        }
        vote_array[i][j].valid=0;
    }
}
```

Miss rate for writes to `vote_array`: _____ %

Student Number: Name:

Problem 4. Dynamic Memory Allocation (24 Points)

Part A. (9 points)

(a) Suppose that you have been asked to implement a dynamic memory allocator for a real-time system with strict bounds on the amount of time for an operation (a malloc, a free, etc). In order to combat false fragmentation, you have decided to use some sort of coalescing. Which type of coalescing (immediate or deferred) should you use, and why? Use at most two sentences for your answer.

(b) Assume a minimum block size of 32 bytes for this part of the problem. Do implicit free lists or explicit free lists use more memory ? That is, will one of these strategies necessarily use more space than the other, assuming that everything else in the allocators are identical, and why? Use at most two sentences for your answer.

(c) Assume you are designing a special purpose dynamic memory allocator. One of the interesting properties of this system is that only 8 different sizes, all of which you know in advance and all of which are small relative to the page size of the system, will be requested by the user (though there are no guarantees how many times each of these will be requested). What are the most important ways in which your design for this allocator would differ from your design of a general purpose allocator? Use at most two sentences for your answer.

Student Number: Name:

Part B. (15 points)

Consider the following four different designs for a memory allocator:

- *IL (Implicit List).* The free list is represented implicitly. Each block has a header, but no footer. Coalescing of the blocks is deferred to occur during block allocation.
- *ILBT (Implicit List with Boundary Tags).* The free list is represented implicitly. Each block has a header and an identical boundary-tag footer. Coalescing of the blocks occurs as part of the free operation.
- *E1LBT (Explicit List with Boundary Tags, Singly-Linked).* The free list is represented explicitly as a singly-linked list. Each block has a header and an identical boundary-tag footer. Coalescing of the blocks occurs as part of the free operation. A newly freed block is placed at the head of the free list.
- *E2LBT (Explicit List with Boundary Tags, Doubly-Linked).* The free list is represented explicitly as a doubly-linked list. Each block has a header and an identical boundary-tag footer. Coalescing of the blocks occurs as part of the free operation. A newly freed block is placed at the head of the free list.

We define the following operations. Here `bp` denotes some pointer to the beginning of the payload section of a block, while `p` denotes an arbitrary pointer.

`malloc(cnt)` Allocate a block of `cnt` bytes. You may assume that sufficient space is available.

`free(bp)` Free an allocated block

`isRightFree(bp)` Determine whether the block with the next higher address is free.

`isLeftFree(bp)` Determine whether the block with the next lower address is free.

`isValidAddress(p)` Determine whether `p` points to a word within an allocated block.

Assume at some point in the program execution that there are `m` allocated blocks and `n` free blocks.

Fill in the table showing the worst-case asymptotic performance (big O notation) for the optimum implementation of each of the following operations. Example entries are `O(1)` (constant time), `O(m)`, `O(m+n)`, etc.

Operation	IL	ILBT	E1LBT	E2LBT
malloc(cnt)				
free(bp)				
isRightFree(bp)				
isLeftFree(bp)				
isValidAddress(p)				

Student Number: Name:

Problem 5. Parallel Programming for Games (10 points)

You are asked to design a fine-grained locking implementation for a 2D Quake-like game which has an Area Node tree as its basic data structure. The Area Node tree is a spatial data structure in which the root of the tree corresponds to the whole game world, then the left and right children of the root correspond to the left and right halves of the game world, and so on, every time splitting the world either vertically or horizontally, until the leaves of the tree correspond to the finest granularity grid units of the game world, as shown in the lecture notes.

The Area Node tree stores objects in the game world for easy collision detection during processing of player actions. Game objects fully contained in a leaf node are stored as a list in the corresponding leaf of the Area Node tree. Objects that span a leaf boundary are stored as a list in the lowest common ancestor of the respective leaves.

For processing a player action, the game server needs to perform collision detection against all game objects intersecting the player's trajectory. Since the avatar's direction can be altered by collision with entities situated in its path, the avatar's trajectory and its final position are impossible to predict from the beginning of the action. However, the whole player action and its effects on the game world need to appear as a consistent atomic i.e., indivisible unit to the players. In order to make this feasible, we assume that we know beforehand the bounding box of any player action, and how to compute what Area Node leaves the action intersects on the game map. Let's say we can do this through a procedure called *compute_leaves*. Assume we can also compute the common parents of any leaves with a procedure called *compute_parents*. Please describe in words your method for synchronizing accesses for each player action on the Area Node Tree. Make sure to explain why each synchronization you put in is necessary in order to get any credit.

Student Number: Name:

Problem 6. Parallel Programming (30 Points)

Part A. (4 points)

Consider the following loops. Please identify the dependencies in each.

```
for( I = 1; I <= 100; I++){
    S1:  A [I + 1][I + 2] = A [I][I] + C
}
```

Please identify the dependences in the loop above.

```
FOR I = 1, 100
    S1:  A (I + 1, I + 2) = A (I, I) + C
ENDFOR
```

Please identify the dependences in the loop above.

Student Number: Name:

Part B. (6 points)

```
DO I = 2, N
    A(I) = B(I) + C(I)
    D(I) = A(I-1) * 2.0
ENDDO
```

Please identify the dependences in this piece of code. Can the code be executed in parallel ? If not, please apply whatever transformations to the loop that you deem necessary in order to allow parallel execution.

Part C. (20 points)

Consider the following code.

(a). (4 points) Explain what dependencies there are in the following code:

```
for (i = 0; i < 100000; i++)
   a[i + 1000] = a[i] + 1;
```

(b). (6 points) Rewrite the loop above to allow parallel execution and point out which part of the code can be parallelized.

```
Use the "#pragma omp parallel for" notation.
```

(c). (10 points) Rewrite your parallel code to optimize both its cache performance and parallelism.