Csci 201: Sample Final Exam

This is a sample final exam. The actual exam may/will differ from this one in the number of questions and in topics covered by each question.

Name:____

NetID:____

Student to your left: ____________________________________

Student to your right: ____________________________________

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED

Instructions:

- **Write your full name and your NetID on the front of this exam.**
- Make sure that your exam is not missing any sheets. There should be seven (6) double sided pages in the exam.
- Write your answers in the space provided below each problem. If you make a mess, clearly indicate your final answer.
- **Answer all questions in this exam.**
- If you have any questions during the exam, raise your hand and we will try to get to you.
- At the end of the exam, there are blank pages. Use them as your scrap paper. If you need additional scrap paper, raise your hand and we will get it for you.
- This exam is closed books, closed notes, closed computers. You are allowed to have a single double sided sheet of paper with anything you wish on it. That sheet should have your name on front and back.
- **You need to stay in your seat until the exam is finished.** You should not leave the room even if you finish the exam. This distracts other students who are still working.

Good luck!

Problem 1 (30 points) Multiple Choice Questions.

Answer the following multiple choice questions. Circle **all** applicable answers. Points will be subtracted for selecting incorrect answers and for missing some of the correct answers.

1. Which characteristic of RAM makes it **not** suitable for permanent storage?

   (a) it is slow
   (b) it is volatile
   (c) it is prone to errors
   (d) it is very expensive

2. Cache memory refers to

   (a) cheap memory that can be plugged into the mother board to expand main memory
   (b) fast memory present on the processor chip that is used to store recently accessed data
   (c) a reserved portion of main memory used to save important data
   (d) a special area of memory on the chip that is used to save frequently used constants

3. If your program has a bug that results in a segmentation fault, we call it

   (a) a system call
   (b) an exception
   (c) a function call
   (d) an interrupt

4. The condition where the system spends most of its time swapping data between main memory and secondary storage, to the extent that most of the CPU time is wasted waiting for the swapping device, is called

   (a) thrashing
   (b) swapping
   (c) paging
   (d) context switching

5. What is the reason for having multi-level page tables?

   (a) To reduce the amount of disk I/O that is needed to handle a page fault
   (b) To reduce the page fault frequency
   (c) To handle larger processes
   (d) To reduce the amount of real memory occupied by page tables

6. Combining of multiple object modules into a load module is done by a

   (a) compiler
   (b) assembler
   (c) linker
   (d) dynamic loader
   (e) relocating loader

7. What is the smallest normalized value represented using 6-bit IEEE-754 like encoding with one sign bit, 3 exponent bits and 2 fraction bits?

   (a) `111011`
   (b) `111111`
   (c) `100100`
   (d) `000000`
   (e) `000100`
   (f) `011011`

8. What is the content of array `A` after executing the following code snippet?

```
long A[3] = {1, 2, 3};
long *p;
long **q;
p = A;
p++;
q = &p;
(*p) = (**q)*2;
```

(a) 1, 4, 3
(b) 1, 2, 6
(c) 1, 2, 4
(d) 1, 2, 3
(e) 2, 4, 6
(f) 1, 6, 3
(g) none of the above

9. Consider the following C program. (For space reasons, we are not checking error return codes. You can assume that all functions return normally.)

```
int val = 10;

void handler(sig)
{
  val += 5;
  return;
}

int main() {
  int pid;
  signal(SIGCHLD, handler);
  if ((pid = fork()) == 0) {
    val -= 3;
    exit(0);
  }
  waitpid(pid, NULL, 0);
  printf("val = %d\n", val);
  exit(0);
}
```

(a) 10
(b) 15
(c) no output, since the parent process never returns
(d) cannot be determined, since it depends on scheduling of both processes by the kernel

10. For each point below circle the expressions that are **always true** (i.e. evaluate to `1` when they are executed in a C program).

We are running programs on a machine with the following characteristics:

- Values of type `int` are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type `unsigned` are 32 bits.
- Values of type `float` are represented using the 32-bit IEEE floating point format, while values of type `double` use the 64-bit IEEE floating point format.

We generate arbitrary values `x`, `y`, and `z`, and convert them to other forms as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

(a) `( x < y )== ( -x > -y )`
(b) `~x + ~y + 1 == ~( x + y )`
(c) `x == (int)(float)x`

Problem 2 (12 points) .

The following problem concerns dynamic storage allocation.

Consider an allocator that uses an implicit free list. The layout of each allocated and free memory block is as follows:

```
              31                              2  1  0
               __________________________________
     Header   |    Block Size (bytes)      |     |
              |____________________________|_____|
              |                                  |
              |                                  |
              |                                  |
              |__________________________________|
     Footer   |    Block Size (bytes)      |     |
              |____________________________|_____|
```

Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer. The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Given the contents of the heap shown on the left, show the new contents of the heap (in the right table) after a call to `free(0x400b000)` is executed. Your answers should be given as hex values. Note that the address grows from bottom up. Assume that the allocator uses immediate coalescing, that is, adjacent free blocks are merged immediately each time a block is freed. **If the content of a word did not change, you may leave it blank.**

Address	
0x400b028	0x00000022
0x400b024	0x400b611c
0x400b020	0x400b512c
0x400b01c	0x00000d12
0x400b018	0x00000014
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	0x00000022
0x400b008	0x00000013
0x400b004	0x400b601c
0x400b000	0x400b511c
0x400affc	0x00000013
0x400aff8	0x0000001B
0x400aff4	0x400b601c
0x400aff0	0x00000014
0x400afec	0x0a0bc11d
0x400afe8	0x400b511d
0x400afe4	0x0000001B

Address	
0x400b028	
0x400b024	
0x400b020	
0x400b01c	
0x400b018	
0x400b014	
0x400b010	
0x400b00c	
0x400b008	
0x400b004	
0x400b000	
0x400affc	
0x400aff8	
0x400aff4	
0x400aff0	
0x400afec	
0x400afe8	
0x400afe4	

Problem 3 (10 points) .

Answer the short questions below. Each question is worth 3 points.

Unix linkers use the following rules for dealing with multiply defined symbols:

- Multiple strong symbols are not allowed.
- Given a strong symbol and multiple weak symbols, choose the strong symbol.
- Given multiple weak symbols, chose any of the weak symbols.

A. Suppose we attempt to compile and link the following two C modules:

```
/* foo1.c */
int main()
{
  int x = 5;
  return 0;
}
```

```
/* bar1.c */
int main()
{
    int x = 15;
    return 0;
}
```

Will the linker succeed or generate an error? If there is an error, explain which of the three rules above are violated.

B. Suppose we attempt to compile and link the following two C modules:

```
/* foo2.c */
void f();
int x = 15;
int main()
{
    f();
    printf("x = %d",x);
    return 0;
}
```

```
/* bar2.c */
int x;
void f()
{
    x = 10;
}
```

Will the linker succeed or generate an error? If there is an error, explain which of the three rules above are violated. If there is no error, show the output generated by this program.

Problem 4 (12 points) .

Consider the following C functions. For each, evaluate how cache friendly they are (explain memory access patterns, locality, etc). If they are not cache friendly, suggest a modified version that performs the same task in a more cache friendly manner and explain how your fix improved the cache performance of the code.

A. Assume that the value of the constant N is defined.

```c
int fun_1()
{
  int B[N][N];

  int i, j;

  for(j = 0; j < N; j++)
    for(i = 0; i < N; i++)
      B[i][j] = 2*(B[i][j] + 2);

}
```

B.

```c
typedef struct {
  float exam1;
  float exam2;
} grade;

grade compute_average( grade * g, int n) {
  float exam1_average = 0;
  float exam2_average = 0;
  int i;

  for(i = 0; i < n; i++)
    exam1_average += g[i].exam1;
  for(i = 0; i < n; i++)
    exam2_average += g[i].exam2;

  grade average = { exam1_average/n, exam2_average/n };

  return average;
}
```

Problem 5 (12 points) .

Consider the following C declaration:

```
struct node{
   char c;
   long value;
   struct node* next;
   int flag;
   struct node* left;
   struct node* right;
} node;
```

A. Using the template below (allowing a maximum of 64 bytes), indicate the allocation of data for a **Node** struct. Mark off and label the areas for each individual element (there are 6 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment).

Assume the Linux alignment rules that were discussed in class (and the textbook).

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                                                                                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

```
 32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                                                                                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

B. For each of the C functions below, complete the blanks in its assembly code.

```
int test1(int i, struct node * tree[]) {
      int f;
      f = (tree[i])-> flag;
      return f;
}
```

```
movslq  %edi, %rdi
movq    (%rsi,%rdi,8), %rax


movl    ___________________(%rax), %eax
ret
```

```
long test2(int i, struct node * tree[]) {
      long f;
      f = ((tree[i])-> next)->value;
      return f;
}
```

```
movslq  %edi, %rdi
movq    (%rsi,%rdi,8), %rax


movq    ___________________(%rax), %rax


movq    ___________________(%rax), %rax
ret
```

Problem 6 (10 points) .

The following problem concerns basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Physical addresses are 13 bits wide.
- The cache is 2-way set associative, with a 4 byte line size and 16 total lines.

In the following tables, **all numbers are given in hexadecimal**. The contents of the cache are as follows:

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	99	04	03	48
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	2F	81	FD	09	0B	0	8F	E2	05	BD
3	06	0	3D	94	9B	F7	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	B0	39	D3	F7
6	91	1	A0	B7	26	2D	F0	0	0C	71	40	10
7	46	0	B1	0A	32	0F	DE	1	12	C0	88	37

A. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:
BO the block offset within the cache line SI the set index T the cache tag

12	11	10	9	8	7	6	5	4	3	2	1	0

B. For the given memory address, indicate the cache entry accessed and the cache byte value returned **in hex**. Indicate whether a cache miss occurs.

Memory address: `0E34`

(a) Memory address format (one bit per box)

12	11	10	9	8	7	6	5	4	3	2	1	0

(b) Memory memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Problem 7 (14 points) .

Assume the following definition of the `print_number` function:

```
void * print_number(void *arg)
{
  int *p = (int *)arg;
  printf("%d\n", *p);
}
```

A. Describe the output produced when a program with the following `main` function is executed.

```
void main() {
  int i;
  for (i = 0; i < 3; i++) {
    if (fork() == 0) {
      print_number(&i);
    }
  }
  exit(0);
}
```

B. How many processes are created by `main` function in part A?

C. When a program with the following `main` function is executed, it ofter produces no output. Explain why this may happen and how to fix it. Describe the output for your modified program.

```
void main() {
  pthread_t th[3];
  int i;
  for (i = 0; i < 3; i++) {
    pthread_create(&th[i], NULL, print_number, &i);
  }
  exit(0);
}
```

D. How many processes are created by `main` function in part C?

E. How many threads are created by `main` function in part C?