Andrew ID (print clearly!):______________________________________

Full Name:______________________________________

15-213/18-213, Fall 2011

Final Exam

Friday, December 16, 2011

Instructions:

- Make sure that your exam is not missing any sheets, then write your Andrew ID and full name on the front.
- This exam is closed book, closed notes (except for 2 double-sided note sheets). You may not use any electronic devices.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.

- The exam has a maximum score of 92 points.
- The problems are of v [illegible] uck!

[illegible]
2 (08):
3 (06):
4 (10):
5 (06):
6 (12):
7 (06):
8 (10):
9 (09):
10 (06):
11 (09):

TOTAL (92):

Problem 1. (10 points):

General systems topics. Write your answer for each question in the following table:

1	2	3	4	5	6	7	8	9	10

1. Consider a direct-mapped cache memory. Which one of the following statements is true?

   (a) The cache has 1 line per set.

   (b) The cache has 1 word per block.

   (c) The cache has 1 set per cache.

   (d) None of the above.

2. Which one of the following statements about cache memories is true:

   (a) Larger caches are more susceptible to capacity misses than smaller caches.

   (b) Caches with lower associativity are more susceptible to conflict misses than those with higher associativy.

   (c) Caches with higher associativity are more susceptile to cold misses than those with lower associativey.

   (d) None of the above

3. Which one of the follow

   (a) Machine code

   (b) Global variables

   (c) User stack

   (d) Symbol table

4. Assuming no errors, which one of the following statements about `fork` is true?

   (a) Called once, returns once.

   (b) Called once, returns twice.

   (c) Called once, returns never.

   (d) Called twice, returns once.

   (e) None of the above.

5. Assuming no errors, which one of the following statements about `execve` is true?

   (a) Called once, returns once.

   (b) Called once, returns twice.

   (c) Called once, returns never.

   (d) Called twice, returns once.

   (e) None of the above.

6. Which one of the following statements about processes is false?

   (a) The operating system kernel runs as its own separate process.
   (b) Each process shares the CPU with other processes.
   (c) Each process has its own private address space.
   (d) The environment for a process is stored on the stack.

7. What happens if the parent of a zombie child terminates?

   (a) The zombie child becomes a wraith and is never reaped.
   (b) The zombie child is reaped by the init process.
   (c) The zombie child is reaped by the process with the nearest PID.
   (d) None of the above.

8. Suppose that the kernel delivers two SIGCHLD signals to the parent while the parent is not scheduled. When the kernel finally schedules the parent, how many times will the SIGCHLD handler be called?

   (a) None, because sending multiple signals will always crash the program.
   (b) Exactly once, because signals are not queued.
   (c) Exactly twice, because signals are queued.

   (d) More than twice, depending on how the handler is installed.

9. Which one of the follow

   (a) In the best case, coa
   (b) Seglists typically approximate best fit search.
   (c) Payloads must be aligned to some boundary.

   (d) Explicit lists are typically faster than implicit lists.
   (e) None of the above.

10. Which one of the following addresses is 8-byte aligned?

   (a) 1110110101110111_2
   (b) 1110110101110100_2
   (c) 1110110101110000_2
   (d) 1110110101110110_2
   (e) None of the above

Problem 2. (8 points):

Floating point encoding. Consider the following 5-bit floating point representation based on the IEEE floating point format. This format does not have a sign bit – it can only represent nonnegative numbers.

- There are $k = 3$ exponent bits. The exponent bias is 3.
- There are $n = 2$ fraction bits.

Recall that numeric values are encoded as a value of the form $V = M \times 2^E$, where E is the exponent after biasing, and M is the significand value. The fraction bits encode the significand value M using either a denormalized (exponent field 0) or a normalized representation (exponent field nonzero). The exponent E is given by $E = 1 - Bias$ for denormalized values and $E = e - Bias$ for normalized values, where e is the value of the exponent field `exp` interpreted as an unsigned number.

Below, you are given some decimal values, and your task it to encode them in floating point format. In addition, you should give the rounded value of the encoded floating point number. To get credit, you must give these as whole numbers (e.g., 17) or as fractions in reduced form (e.g., $3/4$). Any rounding of the significand is based on ***round-to-even***, which rounds an unrepresentable value that lies halfway between two representable values to the nearest even representable value.

Value	Floating Point Bits	Rounded value
$9/32$	001 00	$1/4$
2		
$15/4$		

Problem 3. (6 points):

Array indexing. Consider the C code below, where H and J are constants declared with `#define`.

```
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
    array2[y][x] = array1[x][y];
}
```

Suppose the above C code generates the following x86-64 assembly code:

```
# On entry:
#     %edi = x
#     %esi = y
#
copy_array:
        movslq  %edi,%rdi
        movslq  %esi,%rsi
        movq    %rsi, %rdx
        salq    $4, %rdx
        subq    %rsi, %rd
        addq    %rdi, %rd
        leaq    0(,%rdi
        subq    %rdi, %ra
        addq    %rsi, %rax
        movl    array1(,%rax,4), %eax
        movl    %eax, array2(,%rdx,4)
        ret
```

What are the values of H and J?

H =

J =

Problem 4. (10 points):

Structure access. Consider the following data structure declarations:

```
struct data {
    long x;
    char str[16];
};

struct node {
    struct data d;
    struct node *next;
};
```

Below are given four C functions and four x86-64 code blocks. Next to each of the x86-64 code blocks, write the name of the C function that it implements.

```
int alpha(struct node *ptr) {
    return ptr->d.x;
}
```

____
```
movsbl  15(%rdi),%eax
ret
```

```
char *beta(struct node *ptr) {
    ptr = ptr->next;
    return ptr->d.str;
}
```

____
```
movq    (%rdi), %rax
ret
```



```
char gamma(struct n
    return ptr->d
}
```

```
                                    rax
        q                       rax
```



```
long *delta(struct node *ptr) {
    struct data *dp =
        (struct data *) ptr;
    return &dp->x;
}
```

____
```
leaq    10(%rdi), %rax
ret
```

```
char *epsilon(struct node *ptr) {
    return &ptr->d.str[2];
}
```

Problem 5. (6 points):

Loops. Consider the following x86-64 assembly function:

```
loopy:
        # a in %rdi, n in %esi
        movl    $0, %ecx
        movl    $0, %edx
        testl   %esi, %esi
        jle     .L3
.L6:
        movslq  %edx,%rax
        movl    (%rdi,%rax,4), %eax
        cmpl    %eax, %ecx
        cmovl   %eax, %ecx
        addl    $1, %edx
        cmpl    %ecx, %esi
        jg      .L6
.L3:
        movl    %ecx, %eax
        ret
```

Fill in the blanks of the corresponding C code.

- You may only use the C variable names n, a, i and x, not register names.
- Use array notation in showing accesses or updates to elements of a.


```
int loopy(int a[], int n)
{
    int i;
    int x = ______;

    for(i = ______________; ______________; ______________) {

        if (______________)

            x = ______________;
    }
    return x;
}
```

Problem 6. (12 points):

Stack discipline.

A. (2 pts) Consider the following snippet of IA32 code:

```
 8048390:        call   8048395
 8048395:        pop    %eax
```

Suppose that just before the `call` instruction executes, `%esp = 0xffffd834`. Then what is the value of `%eax` after the `pop` instruction executes?

`%eax = 0x`____________________

B. (2 pts) Consider a slightly different snippet of IA32 code:

```
 8048396:        call   804839b
 804839b:        ret
```

Suppose that just before the `call` instruction executes, `%esp = 0xffffd838`. Then what is the value of `%eip` after the `ret` instruction executes?

`%eip = 0x`____________________

(Please go to the next page for P

C. (7 points) Consider the following bit of C code and its dissassembled IA32 machine code (notice the header comment):

```
int power(int x, int n)
{
  if (n == 0 )
    return 1;
  else
    return power(x, n-1) * x;
}

int main()
{
  power(2, 4);
}
```

```
08048374 <power>:
 # On entry to power(2,4):
 # %esp = 0xffffd81c, %ebp = 0xffffd838
 8048374:       push   %ebp
 8048375:       mov    %esp,%ebp
 8048377:       sub    $0x8,%esp
 804837a:       mov    0xc(%ebp),%edx
 804837d:       mov    $0x1,%eax
 8048382:       test   %edx,%edx
 8048384:       je     804839c <power+0x28>
 8048386:       lea    0xffffffff(%edx),%eax
 8048389:       mov    %eax,0x4(%esp)
 804838d:       mov    0x8(%ebp),%eax
 8048390:       mov    %eax,(%esp)
 8048393:       call   8048374 <power>
 8048398:       imul   0x8(%ebp),%eax
 804839c:       leave
 804839d:       ret
```


Suppose that the main routin … . … ill in the values on the stack immediately after th … n, please write UNKNOWN in the blank:

Address	
0xffffd824	0x________________
0xffffd820	0x________________
0xffffd81c	0x________________
0xffffd818	0x________________
0xffffd814	0x________________
0xffffd810	0x________________
0xffffd80c	0x________________

D. (1 point) What is the value of `%ebp` immediately after the call to `power(2,3)`:

`%ebp = 0x`________________

Problem 7. (6 points):

Caches. In this problem you will estimate the miss rates for some C functions. Assumptions:

- 16-way set associative L1 cache ($E = 16$) with a block size of 32 bytes ($B = 32$).
- N is very large, so that a single row or column cannot fit in the cache.
- `sizeof(int) == 4`
- Variables `i`, `k`, and `sum` are stored in registers.
- The cache is cold before each function is called.

Part A (3 points)

```
int sum1(int A[N][N], int B[N][N])
{
    int i, k, sum = 0;

    for (i = 0; i < N; i++)
        for (k = 0; k < N; k++)
            sum += A[i][k] + B[k][i];
    return sum;
}
```

Circle the closest miss rate for `sum1`:

- $1/16$
- $1/8$
- $1/4$
- $1/2$
- 6

Part B (3 points)

```
int sum2(int A[N][N], int B[N][N])
{
    int i, k, sum = 0;

    for (i = 0; i < N; i++)
        for (k = 0; k < N; k++)
            sum += A[i][k] + B[i][k];
    return sum;
}
```

miss rate for `sum2`:

- $1/16$
- $1/8$
- $1/4$
- $1/2$
- $9/16$
- 1

Problem 8. (10 points):

Exceptional control flow.

A. Consider the following C program. Assume the program executes to completion and that `fork`, `waitpid`, and `printf` always succeed.

```
int main() {
    pid_t pid;
    int sum = 0;

    if ((pid = fork()) == 0)
        sum += 10;
    else {
        waitpid(pid, NULL, 0);
        sum -= 5;
    }
    sum += 20;

    if (pid > 0)
        printf("Parent: sum=%d\n", sum);
    else
        printf("Child:  sum=%d\n", sum);
    return 0;
}
```


Show the output of this progr

`Child:  sum=`__________

`Parent: sum=`__________

B. Now consider the same program as in Part A, but with the call to `waitpid` removed. Assume the program executes to completion and that `printf` always succeeds. **Make no assumptions about the results of the other function calls.**

List all of the possible outputs of such a program. Each blank box holds the complete output from one execution of the program. Some blank boxes may be left unused.

```
+----------------------+----------------------+----------------------+
|                      |                      |                      |
|                      |                      |                      |
+----------------------+----------------------+----------------------+
|                      |                      |                      |
|                      |                      |                      |
+----------------------+----------------------+----------------------+
```

(Please goto the next page for Part C.)

C. Consider the C program below. Assume the program runs to completion and that all functions return normally.

```
int main ()
{
    if (fork() == 0) {
        if (fork() == 0) {
            printf("9");
            exit(1);
        }
        else
            printf("5");
    }
    else {
        pid_t pid;
        if ((pid = wait(NULL)) > 0) {
             printf("3");
        }
    }
    printf("0");
    return 0;
}
```


List four possible outputs fo

1. ____________________

2. ____________________

3. ____________________

4. ____________________

Problem 9. (9 points):

Imagine a system with the following attributes:

- The system has 1MB of virtual memory
- The system has 256KB of physical memory
- The page size is 4KB
- The TLB is 2-way set associative with 8 total entries.

The contents of the TLB and the first 32 entries of the page table are given below. **All numbers are in hexadecimal**.

TLB

Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Page Table

VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
			18	23	1
			19	04	0
			1A	0C	1
			1B	2B	0
				1E	0
				3E	1
				27	1
				15	1

A. Warmup Questions

(a) How many bits are needed to represent the virtual address space? ______

(b) How many bits are needed to represent the physical address space? ______

(c) How many bits are needed to represent a page table offset? ______

B. Virtual Address Translation I

Please step through the following address translation. Indicate a page fault by entering '-' for Physical Address.

Virtual address: `0x1F213`

Parameter	Value	Parameter	Value
VPN	0x	TLB Hit? (Y/N)	
TLB Index	0x	Page Fault? (Y/N)	
TLB Tag	0x	Physical Address	0x

Use the layout below as scratch space for the virtual address bits. To allow us to give you partial credit, clearly mark t BT).

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

(Please go to the next page for part C)

C. Virtual Address Translation II

Please step through the following address translation. Indicate a page fault by entering '-' for Physical Address.

Virtual address: `0x14213`

Parameter	Value	Parameter	Value
VPN	0x	TLB Hit? (Y/N)	
TLB Index	0x	Page Fault? (Y/N)	
TLB Tag	0x	Physical Address	0x

Use the layout below as scratch space for the virtual address bits. To allow us to give you partial credit, clearly mark the bits that correspond to the VPN, TLB index (TLBI), and TLB tag (TLBT).

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Problem 10. (6 points):

Unix I/O.

A. Suppose that the disk file `foobar.txt` consists of the six ASCII characters "foobar". What is the output of the following program?

```
/* any necessary includes */
char buf[20] = {0}; /* init to all zeroes */

int main(int argc, char* argv[]) {
    int fd1 = open("foobar.txt", O_RDONLY);
    int fd2 = open("foobar.txt", O_RDONLY);

    dup2(fd2, fd1);

    read(fd1, buf, 3);
    close(fd1);
    read(fd2, &buf[3], 3);
    close(fd2);

    printf("buf = %s\n", buf);
    return 0;
}
```


Output: buf = ____________

B. Now consider the identical program, except that dup tput of this program?


```
char buf[20] = {0}; /* init to all zeroes

int main(int argc, char* argv[]) {
    int fd1 = open("foobar.txt", O_RDONLY);
    int fd2 = open("foobar.txt", O_RDONLY);

    //dup2(fd2, fd1);

    read(fd1, buf, 3);
    close(fd1);
    read(fd2, &buf[3], 3);
    close(fd2);

    printf("buf = %s\n", buf);
    return 0;
}
```

Output: buf = ____________

Problem 11. (9 points):

Synchronization. This problem is about using semaphores to synchronize access to a shared bounded FIFO queue in a producer/consumer system with an arbitrary number of producers and consumers.

- The queue is initially empty and has a capacity of 10 data items.
- Producer threads call the `insert` function to insert an item onto the rear of the queue.
- Consumer threads call the `remove` function to remove an item from the front of the queue.
- The system uses three semaphores: `mutex`, `items`, and `slots`.

Your task is to use P and V semaphore operations to correctly synchronize access to the queue.

A. What is the initial value of each semaphore?

`mutex` = ________

`items` = ________

`slots` = ________

B. Add the appropriate P and and `remove` functions:


```
void insert(int ite
{
    /* Insert sem ops here */



    add_item(item);
    /* Insert sem ops here */



}
```

```
{
                                */



    item = remove_item();
    /* Insert sem ops here */



    return item;
}
```