Andrew login ID:_______________________________________

Full Name:_______________________________________

Recitation Section:_______________________________________

CS 15-213, Fall 2008
Final Exam

Friday. December 12, 2008

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name, Andrew login ID, and recitation section (A–H) on the front.
- The exam has a maximum score of 106 points.
- This exam is OPEN BOOK. You may use any books or notes you like. No calculators or other electronic devices are allowed.

1 (18):
2 (12):
3 (10):
4 (8):
5 (6):
6 (7):
7 (8):
8 (12):
9 (8):
10 (8):
11 (9):
TOTAL (106):

Problem 1. (18 points):

For the following questions, indicate whether each of the statements following them is true or false.

1. The register `rax` currently has value 0. Which of the following statements are true?

   T/F: ____ Executing `movq (%rax), %rcx` will cause a segmentation fault.

   T/F: ____ Executing `leaq (%rax), %rcx` will cause a segmentation fault.

   T/F: ____ Executing `movq %rax, %rcx` will cause a segmentation fault.

   T/F: ____ Executing `addq 8, %rsp` will increase the stack allocation by 8 bytes.

2. When you run two programs on the same Linux machine, why do you not have to worry about them using the same physical memory?

   T/F: ____ The programmers knew to avoid using the same addresses and were careful not to do so.

   T/F: ____ The virtual addresses used by the running programs are translated to non-overlapping physical addresses.

   T/F: ____ Only one program really runs at a time, and the physical memory is saved/restored as part of each context switch.

3. Consider the following global variable declaration:

   ```
   char str[80];
   ```

   Which of the following outcomes should be of significant concern if you write a program that calls `gets(str)`?

   T/F: ____ There could be a stack overflow leading to execution of malicious code.

   T/F: ____ There could be a buffer overflow that modifies other global variables.

   T/F: ____ There could be a SEGFAULT.

   T/F: ____ You might get a bad score if you did this for your 15-213 lab.

4. Consider the following structure definition:

```
struct data {
  short things[3];
  unsigned int doodad;
  char stuff[8];
}
```

Which of the following holds for an x86-64 machine:

T/F: ____ The size of `struct data` is 18.

T/F: ____ The size of `struct data` is 20.

T/F: ____ The size of `struct data` is 24.

T/F: ____ Network code should use `htons()` for each entry in `things` that is sent to another computer.

T/F: ____ Network code should use `htons()` for data in `stuff` that is sent to another computer.

5. After a process calls `fork()`, which of the following are shared by it and the new child process? (By "shared," we mean that an update by one process could affect the behavior of the other.)

T/F: ____ All file table entries corresponding to files open in the parent before the call to `fork()`.

T/F: ____ New file table entries corresponding to files `open()`'d by the parent after the `fork()`.

T/F: ____ New file table entries corresponding to files `open()`'d by the child after the `fork()`.

T/F: ____ All of virtual memory.

T/F: ____ The stack.

T/F: ____ All register contents.

T/F: ____ All sockets with connections established before the `fork()`.

6. Suppose we compile and run the following code on a processor where integer multiply has a 10 clock cycle latency and a 2 clock cycle issue time. That is, although a single product computation requires 10 clock cycles, the multiplier can start a new computation every 2 clock cycles.

```
int prod = 1;
for (i = 0; i < n; i+= 2) {
  prod = prod *
    a[i] * a[i+1];  // Line P
}
```

Which of the following statements hold for this code:

T/F: ____ It will properly compute the product for any array `a` of length `n`.

T/F: ____ It will have a CPE of 10.

T/F: ____ It will have a CPE of 2.

T/F: ____ Changing Line P to be `(a[i] * a[i+1])` will change the CPE.

T/F: ____ Changing Line P to be `(a[i] * a[i+1])` could change the computed result.

Problem 2. (12 points):

We are running programs on a machine where values of type `int` have a 32-bit two's-complement representation, and values of type `long` have a 64-bit two's complement representation. Right shifts are performed arithmetically. Values of type `double` use the 64-bit IEEE format.

We generate arbitrary integer values `x` and `y`, and convert them to values of type `long` and `double` as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
/* Convert to long */
long lx = (long) x;
long ly = (long) y;
/* Convert to double */
double dx = (double) x;
double dy = (double) y;
```

For each of the following C expressions, you are to indicate whether or not the expression *always* yields 1. If it can yield 0, give values for the arguments that make the expression evaluate to 0.

Hint: Note that some properties may hold for `dx` and `dy` that do not hold for arbitrary double-precision floating-point numbers, and similarly `lx` and `ly` are not arbitrary long's.

Expression	Always 1?	Arguments (if could yield 0)
`x+3*(x+x)+x == x<<3`		
`~x+2 == 1-x`		
`x>>1 == x/2`		
`x * y == lx * ly`		
`lx + y == x + ly`		
`lx == 0 || (lx*ly/lx == ly)`		
`(x < y) == (dx < dy)`		
`dx - dy == (double) (x-y)`		

Problem 3. (10 points):

Consider the following x86_64 assembly code:

```
# On entry, %edi = val, %rsi = K
func:
    pushq   %r12
    movl    %edi, %r12d
    pushq   %rbp
    xorl    %ebp, %ebp
    pushq   %rbx
    movq    %rsi, %rbx
    jmp .L7
.L4:
    addl    $1, %ebp
    addq    $4, %rbx
    cmpl    $15213, %ebp
    je  .L11
.L7:
    movl    (%rbx), %eax
    testl   %eax, %eax
    js  .L4
    movl    $4, %edi
    call    malloc
    testq   %rax, %rax
    movq    %rax, %rdx
    je  .L5
    testl   %r12d, %r12d
    je  .L6
    movl    (%rbx), %ebp
.L6:
    movl    %ebp, (%rax)
.L5:
    popq    %rbx
    popq    %rbp
    movq    %rdx, %rax
    popq    %r12
    ret
.L11:
    popq    %rbx
    xorl    %edx, %edx
    popq    %rbp
    movq    %rdx, %rax
    popq    %r12
    ret
```

Fill in the blanks of the corresponding C function:

```
__________ *func(int val, __________ K) {

    int i;

    _________ j;

    for (i = 0; i < 15213; i++)

        if ( _______________________ ) {

            j = _______________________________ ;

            if ( _______________________ )

                *j = ___________________________ ;

            return j;
        }

    return NULL;
}
```

Problem 4. (8 points):

The problem requires understanding how C code accessing structures, unions, and arrays is compiled. Assume the x86-64 conventions for data sizes and alignments.

```
#include "def.h"

typedef struct {
  short x[A][3];        /* Unknown constant A */
  double y;
  float f;
  double d;
} str;

typedef union{
  double t ;
  str S;
  int array[A*3];
} uni;

void setVal(str  *p, uni *u)  {
  double val = p -> y;
  u -> t = val;
}
```

You do not have a copy of the file def.h, in which constants A and B are defined, but you have the following x86-64 assembly code for the function setVal:

```
setVal:
        # rdi = p, rsi = u
        movq  64(%rdi),%rax
        movq   %rax, (%rsi)
        ret
```

Based on this code, determine the values of the constant A and the size of the struct and union:

A = ___________________

Size of str = ___________________ bytes

Size of uni = _________________ bytes

Problem 5. (6 points):

Consider a system with 20 GB of physical memory (with a 8 KB page size) and a 40 GB disk drive with the following characteristics:

- 512-byte sectors
- 800 sectors/track
- 7,200 RPM (i.e., 8ms to complete one full revolution)
- 6ms average seek time

Imagine an application that MALLOC()s nearly 40 GB of space, initializes it to all zeros, and then randomly selects integers from across the full space and increments them. (Assume that there are no other processes.)

A. What percentage of the integers selected would result in page faults?

B. What is the average time to service a page fault? (round to the nearest millisecond)

C. Approximately how many integers can be incremented per second? (again, rounding is fine)

D. How much additional memory would be needed to double the number of integers that could be incremented incremented per second? (again, rounding is fine)

Problem 6. (7 points):

Consider the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "csapp.h"

int main(int argc, char **argv) {
    char c;
    int file1 = Open_clientfd("http://www.foobarzzz.com/", 80);
    int file2 = Open_clientfd("http://www.foobarzzz.com/", 80);
    int file3;

    rio_readn(file1, &c, 1);
    rio_readn(file1, &c, 1);
    rio_readn(file1, &c, 1);

    rio_readn(file2, &c, 1);

    file3 = dup(file1);
    rio_readn(file3, &c, 1);
    printf("1 = %c\n", c);

    int pid = fork();
    if(pid == 0) {
         Close(file3);

         file3 = dup(file2);
         rio_readn(file3, &c, 1);
         printf("2 = %c\n", c);

         rio_readn(file2, &c, 1);
         printf("3 = %c\n", c);
    }
    else {
         waitpid(pid, NULL, 0);

         printf("4 = %c\n", c);

         rio_readn(file3, &c, 1);
         printf("5 = %c\n", c);

         rio_readn(file1, &c, 1);
         printf("6 = %c\n", c);

         Close(file1);
         Close(file2);
         dup2(file3, file2);

         rio_readn(file2, &c, 1);
         printf("7 = %c\n", c);
    }

    return 0;
}
```

Assume that http://www.foobarzzz.com/ when accessed on port 80 serves the string of bytes `MACHINES` . Also assume that all system calls succeed. What will be the output when this code is compiled and run? You may not need all the lines in the table given below.

Output Line Number	**Output**
1st line of output	
2nd line of output	
3rd line of output	
4th line of output	
5th line of output	
6th line of output	
7th line of output	

Problem 7. (8 points):

```
#include <stdio.h>
#include <stdlib.h>

/* Stores two different elements in an array */
int* build_array(int a, int b)
{
  int *ptr = malloc(2);

  /* check that both numbers are different */
  if (a != b) {
    *ptr = a;
    *(ptr+4) = b;
    return ptr;
  } else {
    return NULL;
  }
}

int main (int argc, char **argv)
{
  int num1,num2;
  int *ptr = NULL;

  /*
   * Two numbers are assigned randomly generated integers by the
   * function "generate_random_integer()".
   * Assume that function "generate_random_integer()" is
   * bug-free and behaves as the name suggests.
   */
  num1 = generate_random_integer();
  num2 = generate_random_integer();

  ptr = build_array(num1, num2);

  printf("num1=%d,num2=%d\n", ptr[0], ptr[1]);

  return 0;
}


```

The 40-line program shown above has two goals: (1) store two distinct integers in an array (if the integers are same, the program does not store them in the array), and (2) print the contents of the array.

We ran the program on the Fish machines, and it compiles fine. Assume that `generate_random_integer` works correctly; it generates and returns a random integer.

The numbers on the left indicate line numbers.

The instructors think that program has four significant problems that need to be fixed. What are the problems? Where do they occur (give line numbers and function names)? And, How do we fix them?

Line Number(s) and Function Name:
Problem:

Fix:

Line Number(s) and Function Name:
Problem:

Fix:

Line Number(s) and Function Name:
Problem:

Fix:

Line Number(s) and Function Name:
Problem:

Fix:

Problem 8. (12 points):

The code samples below are designed to test your understanding of multithreading, semaphores, and the effects of multiple threads modifying the same data. You will be asked to determine the possible outputs the given code can have. For the purposes of this problem, assume that for printf, argument passing and execution of the function is one atomic operation.

```
sem_t sem;

int main()
{
    int j;
    pthread_t tids[3];
    sem_init(&sem, 0, 1);
    for (j=0; j<3; j++) {
        pthread_create(&tids[j], NULL, doit, NULL);
    }
    for (j=0; j<3; j++) {
        pthread_join(&tids[j], NULL);
    }
    return 0;
}
```

This is the main function for all three problems below. The only differences in the problems are the details of the thread function, `doit`, listed on the next page.

part A:

```
int i = 0;

void *doit(void *arg)
{
    P(&sem);
    i = i + 1;
    printf("%d\n", i);
    V(&sem);
}
```

List all possible outputs, using one column of boxes below for each possible output.

part B:

```
int i = 0;

void *doit(void *arg)
{
    P(&sem);
    i = i + 1;
    V(&sem);
    printf("%d\n", i);
}
```

List all possible outputs, using one column of boxes below for each possible output.

part C:

```
int i = 0;

void *doit(void *arg)
{
    i = i + 1;
    printf("%d\n", i);
}
```

Consider each of the following outputs in the table below. If an output could possibly be produced by the above code, after all threads are terminated, then fill in the empty box beside it with ”Y” (otherwise ”N”).

Output:	2 1 1	3 1 1	1 2 1	1 3 1
Possible? (Y/N)				

Problem 9. (8 points):

Part A:

Consider the following client code to connect to a server that runs at 128.2.222.158 on port 15213, sends a message, and receives a response. The response from the server may be longer than MAXBUF and the server will send an EOF when it is done sending the response.

```
int fd, num;
char buf[MAXBUF];
struct sockaddr_in server_address;
char *msg = argv[1];

if((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    exit(1);

server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = 0x8002de9e;
server_address.sin_port = htons((unsigned short)15213);

if(connect(fd, (struct sockaddr *)&server_address,
           sizeof(server_address)) < 0)
    exit(1);

if((num = rio_writen(fd, msg, strlen(msg) + 1)) < 0) exit(1);
if((num = read(fd, buf, MAXBUF)) < 0) exit(1);
printf("%s\n", buf);
```

For each of the statements below, indicate whether it is true or false:

Statement	T/F
The server address 0x8002de9e should have an ntohl call around it.	
The server_address structure should be zeroed out with bzero before use.	
Short counts are not an issue here because the read call cannot return a short count.	
The server address 0x8002de9e should have an htonl call around it.	
The read call should be in a loop to ensure everything from the server is read.	

Part B:

Consider the following server code that creates a socket and listens for client connections.

```
int listenfd;
struct sockaddr_in server_address;

if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    exit(1);

bzero((char *)&server_address, sizeof(server_address));
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons((unsigned short)15213);

if(listen(listenfd, 5) < 0) exit(0);
if(bind(listenfd, (struct sockaddr*)&server_address,
        sizeof(server_address)) < 0)
    exit(1);

while (1) { /* process client connections */ }
```

For each of the statements below, indicate whether it is true or false:

Statement	T/F
`listenfd` will accept connections for all requests to port 15213 on any IP address for this host	
The calls to `htonl` and `htons` should be replaced by `ntohl` and `ntohs`	
The calls to `listen` and `bind` should be flipped (`bind` should be before `listen`)	
We have to specify an address (instead of using `INADDR_ANY`) for `server_address.sin_addr.s_addr` in order for the server to accept connections properly.	

Problem 10. (8 points):

Consider the following C function. Assume that X and Y are 32 and 32, respectively, making the total number of accesses $2 * X * Y = 2048$.

```
int A[X][Y], B[Y][X];

void matrix_transpose()
{
    int i, j;

    for(i = 0; i < X; i++)
        for(j = 0; j < Y, j++)
            B[j][i] = A[i][j];

    return;
}
```

Assume the following:

- `sizeof(int)` = 4.
- The cache is cold when the function is called and the arrays have been initialized elsewhere.
- Variables `i` and `j` and pointers to `A` and `B` are stored in registers.
- `A` and `B` are aligned such that `A[0][0]` begins a cache block, and `B[0][0]` comes immediately following `A[`$X-1$`][`$Y-1$`]`.

Case 1: Assume that the cache is a 4KB direct-mapped cache with 8-byte cache lines. Calculate the cache miss rate by giving the number of **cache misses**. For partial credit briefly explain where collisions occur w.r.t. this access pattern (i.e. using the indices).

```
miss rate = ____________ / 2048 (3 pts)
```

For your reference, X = 32, Y = 32, and here is the code again:

```
int A[X][Y], B[Y][X];

void matrix_transpose()
{
    int i, j;

    for(i = 0; i < X; i++)
        for(j = 0; j < Y, j++)
            B[j][i] = A[i][j];

    return;
}
```

Assume the cache is the same size as in case 1, but is now 2-way set-associative with an LRU replacement policy. Calculate the cache miss rate by giving the number of **cache misses**.

miss rate = ____________ / 2048 (3 pts)

Would a larger **cache size**, with the same **cache line** size, reduce the miss rate? (Y/N)

case 1: _____   case 2: _____

Would a larger **cache line** size, with the same **cache size**, reduce the miss rate? (Y/N)

case 1: _____   case 2: _____

Problem 11. (9 points):

Consider the C code below:

```
void handler(int sig) {
  printf("s\n");
  exit(0);
}

main() {
  int counter, pid;
  int *pcounter2 = NULL;

  signal(SIGUSR1, handler);

  counter = 1;
  pcounter2 = (int *)malloc(sizeof(int));

  if (pid = fork()) {
    counter--;
    printf("%d\n", counter);
    *pcounter2 = 0;
    kill(pid, SIGUSR1);
    waitpid(pid, NULL, 0);
    printf("%d\n", counter);
    fprintf(stderr, "%d\n", *pcounter2);
  } else {
    counter++;
    printf("%d\n", counter);
    *pcounter2 = 1;
  }
}
```

Use the following assumptions to answer the questions:

- All processes run to completion and no system calls will fail.
- `printf()` and `fprintf()` are atomic and they flush `stdout` or the specified file descriptor after printing arguments but before returning.

A. Consider each of the following outputs to `stdout` in the table below. If an output could possibly be produced by the above code, after all processes are terminated, then fill in the empty box beside it with ”Yes” (otherwise ”No”).

Output to stdout	**Possible? (Yes/No)**
020	
002	
0s0	
0s20	
2s00	
20s0	

B. Consider each of the following outputs to `stderr` in the table below. If an output could possibly be produced by the above code, after all processes are terminated, then fill in the empty box beside it with ”Yes” (otherwise ”No”).

Output to stderr	**Possible? (Yes/No)**
0	
1	

C. In the normal case when all processes have terminated after running the above code on the fish machines, can the operating system successfully reclaim all of its memory? (Yes/No)

`ALL MEMORY RECLAIMED?` ____________