Mock Final Exam

Instructions: This final exam takes 1 hour and 30 minutes.
Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can’t understand your answer, we can’t give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES EXAM.
Laptops, ebooks, and other electronic devices are not permitted.

May 11th, 1853

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total (xx/100)
Problem 1. (15 points):
A. On a x86 32-bit machine, Alice intends to use the expression if ((x & mask) !=0)) to test if the 5th bit of x from the right is one or not. (The rightmost bit of x is considered as the 0-th bit). The value of mask should be _______ (decimal).

B. Which of the following expressions generate the desired mask value in Question (A)? Select all that apply.
   (a) 1 << 6
   (b) 1 << 5
   (c) ~(1<<6)
   (d) ~(1<<5)
   (e) 1 >> 26
   (f) 1 >> 27

C. Please give the expression which sets the 5th bit of x to be one and leave the rest of the bits of x unchanged. Your expression should only use the mask value in Question (A) and no other constants.
Problem 2. (20 points):
Consider the source code below, where \( M \) and \( N \) are constants declared with `#define`.

```c
int array1[M][N];
int array2[N][M];

int copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code:

```assembly
copy:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ecx
    movl 12(%ebp),%ebx
    leal (%ecx,%ecx,8),%edx
    sall $2,%edx
    movl %ebx,%eax
    sall $4,%eax
    subl %ebx,%eax
    sall $2,%eax
    movl array2(%eax,%ecx,4),%eax
    movl %eax,array1(%edx,%ebx,4)
    popl %ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

What are the values of \( M \) and \( N \)?

\[
M = \_
\]

\[
N = \_
\]
Problem 3. (20 points):  
Consider the following assembly code for a C for loop:

```assembly
loop:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%ecx
    movl 12(%ebp),%edx
    xorl %eax,%eax
    cmpl %edx,%ecx
    jle .L4
    .L6:
        decl %ecx
        incl %edx
        incl %eax
        cmpl %edx,%ecx
        jg .L6
    .L4:
        incl %eax
        movl %ebp,%esp
        popl %ebp
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables x, y, and result in your expressions below — do not use register names.)

```c
int loop(int x, int y)
{
    int result;

    for (___________; __________; result++ ) {
        __________;
        __________;
    }

    __________;
    return result;
}
```
Problem 4. (15 points):

void foo(int x)
{
    printf("address of x is \%p\n", &x);
    bar(x-1);
    return;
}

void bar(int y)
{
    printf("address of y is \%p\n", &y);
    return;
}

void foo2()
{
    bar2();
}

void bar2()
{
    char buf[10];
    gets(buf);
}

A. Suppose we invoke the function foo many times. What is the relationship of the address of x and that of y in resulting printf statements?
(a). address of x is always less than address of y.
(b). address of x is always greater than address of y.
(c). address of x is sometimes less than address of y and sometimes greater than address of y.

B. The gets(s) function reads a line from stdin into the buffer pointed to by s and does not check for buffer overrun. Suppose we invoke the function foo2 and the user types in some line longer than 10 characters with the intent of exploiting the buffer overrun to execute malicious code by overwriting a return address. Which function would that return address have been pointing too if the attack had not occurred? (That is, which function would we have returned to).
Problem 5. (10 points):

```c
int main(int argc, char** argv) {
    int foo;
    foo = 3;

    int pid = fork();

    if (pid > 0) {
        // Line X
        printf("Parent’s foo is %d\n", foo);
        int status;
        wait(&status);
        foo--;
    } else {
        printf("Child’s foo is %d\n", foo);
    }
}
```

A. What values are printed in the two `printf` statements?

B. Suppose we change the comment “// Line X” to read:

```c
foo++;
```

How does this change the output of the `printf` statements?
Problem 6. (20 points):
A programmer writes a multithreaded program that needs to occasionally save logging information to a file. The programmer designs the following function that logs a given string into a log file, performing locking to ensure that no two threads try to simultaneously create a log file. A file descriptor, or FILE*, is an integer that describes a file handle (for example, standard in is 0, standard out is 1, and standard error is 2). The fwrite() command is replaced with a comment for clarity. (For space reasons, we are not checking error return codes. You can assume that all functions return normally.)

```c
1: int logfile_exists; /* initialized to zero */
2: FILE* logfile; /* file descriptor for the logfile */
3:
4: sem_t write_mutex, create_mutex; /* initialized to 1 */
5:
6: void logline(char* line_to_log)
7: {
8:     P(&create_mutex);
9:     if (logfile_exists == 0)
10:         { /* DELETE the logfile (if it exists), then CREATE it */
11:             remove("logfile.txt");
12:             logfile = fopen("logfile.txt","w+"); /* assume fopen succeeds. */
13:         }
14:     V(&create_mutex);
15:     P(&write_mutex);
16:     // fwrite() function writes line_to_log into the file here
17:     /* If it existed already, or we created it, it definitely exists now */
18:     logfile_exists = 1;
19:     V(&write_mutex);
20:     return;
21: }
```

A. Regardless of how many threads may want to log using this function, will it work as the programmer intended?

B. If so, explain how the locks protect against race conditions. If not, explain with clear steps and line number references how a race condition could occur, and one way to fix it.