15-410 Self-assessment Exercise

About the self-assessment

The prerequisite for 15-410, Operating Systems Design & Implementation, is 15-213, Introduction to Computer Systems. The latter is a sort of "Computer Architecture for Programmers" class, covering topics such as disks, caches, virtual memory, threads, and assembly language. It forms a solid basis for courses such as OS and Computer Architecture.

This self-assessment exercise, based on questions from a recent 15-213 final exam, is designed to provide prospective students with an understanding of the core 213 skills which will be used in 410. Projects 0 and 1 will be written mostly in C, with small scraps of x86 assembly language. Project 2, a thread library, will require a firmer understanding of assembly language and the relationship between a sequence of C code and the corresponding assembly language. In addition, Project 2 will naturally require (and develop) understanding of concurrency. These skills will then be used in Project 3 to write a small operating system. Again, this will be mainly in C, but critical parts will be in assembly language.

Thus, the questions in this exercise focus on decoding simple assembly language sequences, C memory layout and pointer arithmetic, and concurrency. This format is artificial in the sense that we will not ask you to demonstrate mastery of this material in an exam setting on the first day of class. *However, by approximately three weeks into the semester we will be relying on all of these skills to the extent that completing these three questions should take well under an hour.* For example, both the thread library and the operating system kernel are responsible for *building* a standard C runtime environment from underlying primitives. In order to debug sequences of C code which are correct but *execute* incorrectly you will need to translate between C and assembly language.

"Taking the test"

If you had 15-213, this is a good opportunity to dust off your textbook and review parts of it, primarily Chapter 3. You might want to also briefly look over your `malloc()` lab code.

If you haven't had 15-213, but you have programmed in C and some processor's assembly language, you may wish to consult either the 15-213 text or Intel's Instruction Set Reference, available locally at `http://www.cs.cmu.edu/~410/doc/intel-isr.pdf` (keep in mind that the Intel document specifies operands in the opposite order from the GNU assembler, `gas`, which we will be using).

If you haven't programmed in C (as opposed to C++ or Java), or haven't been exposed to assembly language for any processor, these are *serious issues*. We recommend you take 15-213 or 15-441 before 15-410.

For this exercise to end with a meaningful result, *you must answer the questions*. In other words, work on the problems until you have an answer for each one which you are 95% confident is correct, then "grade yourself" in terms of how many hours it took. Do *not* skim each problem and tell yourself, "Oh, that is a problem about Topic X, which I basically understand. I could answer it in NN minutes if I had to." If you have not completed 15-213, we *will* collect and look over your self-assessment answers in the first week of the semester. Finally, you must *work on your own*. The purpose of this exercise is to determine how well prepared *you* are for the class, not the preparation levels of your friends or somebody who might be your project partner. Having these few problems explained to you can't stand in for a missing semester-long prerequisite class!

Problem 1. (9 points):

In this problem you will be given the task of reconstructing C code based on the declarations of C structures and unions provided below and the assembly code on the next page generated from the C code by a Linux/IA32 C compiler.

```
struct s1 {
  char a[3];
  union u1 *b;
  int c;
};

struct s2 {
  struct s1 d;
  struct s1 *e;
  struct s2 *f;
  double g;
  int h[4];
};

union u1 {
  int i;
  struct s2 j;
  struct s1 *k;
};
```

For each IA32 assembly code sequence below on the left, fill in the missing portion of the corresponding C source line on the right.

A
```
proc1:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl 4(%eax),%eax
    movl 40(%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

```
int proc1(struct s1 *x)
{
  return x->____________________ ;
}
```

B
```
proc2:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl 32(%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

```
int proc2(struct s2 *x)
{
  return x->____________________ ;
}
```

C
```
proc3:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl (%eax),%eax
    movl 4(%eax),%eax
    movl (%eax),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

```
int proc3(union u1 *x)
{
  return x->____________________ ;
}
```

Problem 2. (6 points):

This problem tests your understanding of pointer arithmetic and pointer dereferencing.

Harry Q. Bovik, as part of his 15-213 course work, was asked to write a dynamic memory allocator (`malloc()`, `free()`, etc.). Since you have already completed 15-213, you have been pressed into service as a teaching assistant. Harry is having trouble figuring out how to write a C macro to access part of his data structure, and he is asking you for help.

The following is a description of Harry's memory-block structure:

HDR	ID_STRING	PAYLOAD	FTR

- HDR - Header of the block (4 bytes)
- ID_STRING - Unique ID string (8 bytes)
- PAYLOAD - Payload of the block (arbitrary size)
- FTR - Footer of the block (4 bytes)

The size of the payload of each block is stored in the header and the footer of the block. Since there is an 8 byte alignment requirement, the least significant of the 3 unused bits is used to indicate whether the block is free (0) or allocated (1). Harry has also decided to uniquely label each block with a string stored right after the header of the block. The size of this ID field is 8 bytes.

For this problem, you can assume that:

- sizeof(int) == 4 bytes
- sizeof(char) == 1 bytes
- sizeof(short) == 2 bytes
- The size of any pointer (e.g. char*) is 4 bytes.

Your task is to help Harry figure out and circle **clearly** which of the following definitions of the macro `GET_ID` will cause `print_block()` to output the string that is stored in the `ID_STRING` field. **There may be multiple macros that are correct, so be sure to circle all of them.**

Also, assume that the block pointer `bp` points to the first byte of the payload.

```
/* Harry Q. Bovik's print_block() function
   Refer to this function in order to figure out
   the context in which the GET_ID macro is used.
*/
void print_block(void *bp){
  printf("Found block ID: %s\n", GET_ID(bp));
}

/* A. */
#define GET_ID(bp)  ((char *)(((int)(bp)) - 8))

/* B. */
#define GET_ID(bp)  ((char *)(((char)(bp)) - 8))

/* C. */
#define GET_ID(bp)  ((char *)(((char *)(bp)) - 4))

/* D. */
#define GET_ID(bp)  ((char *)(((char *)(bp)) - 8))

/* E. */
#define GET_ID(bp)  ((char *)(((int *)(bp)) - 4))

/* F. */
#define GET_ID(bp)  ((char *)(((int *)(bp)) - 8))

/* G. */
#define GET_ID(bp)  ((char *)(((char**)(bp)) - 8))

/* H. */
#define GET_ID(bp)  ((char *)(((short*)(bp)) - 4))

/* I. */
#define GET_ID(bp)  ((char *)(((short*)(bp)) - 8))
```

Problem 3. (8 points):

In this problem you will be asked questions about the behavior of the following code skeleton when various code fragments are substituted in for the three comment lines:

```
/* LINE 1 */
/* LINE 2 */
/* LINE 3 */
```

(For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
int counter = 2;

void foo() {
    counter++;
    printf("%d\n", counter);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++) {

        /* LINE 1 */
        /* LINE 2 */
        /* LINE 3 */

    }

    counter++;
    printf("%d\n", counter);
}
```

Part A

Suppose the following code replaces the three comment lines in the program on the previous page:

```
LINE 1:
LINE 2:    pthread_create(&tid[i], 0, foo, 0);
LINE 3:
```

What is the **first** number that gets printed on `stdout`? Circle only one answer.

3
4
5
Could be either 3 or 4 or 5
Could be either 3 or 4

Part B

Suppose the following code replaces the three comment lines.

```
LINE 1:    pthread_create(&tid[i], 0, foo, 0);
LINE 2:    pthread_join(tid[i], 0);
LINE 3:
```

What is the **first** number that gets printed on `stdout`? Circle only one answer.

3
4
5
Could be either 3 or 4 or 5
Could be either 3 or 4

Part C

Suppose the following code replaces the three comment lines.

```
LINE 1:    if (fork() == 0) {
LINE 2:        foo();
LINE 3:    }
```

What is the **first** number that gets printed on `stdout`? Circle only one answer.

```
3
4
5
Could be either 3 or 4 or 5
Could be either 3 or 4
```

Part D

Consider the **same** code as **Part C**. What is the **SECOND** number that gets printed on `stdout`? Circle only one answer.

```
3
4
5
Could be either 3 or 4 or 5
Could be either 3 or 4
```

If you are turning your answers in as a text file, please format it as follows:

```
1 A  return x->q[3];
1 B  return x->z;
1 C  return x->v[99];
2    H, I
3 A  5
3 B  3/4/5
...  ...
```