Full Name:_____

Roll Number:_____

IT215: Systems Software, Winter 2012-13

Second In-Sem Exam (2 hours)

March 22, 2013

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and roll number on the front.
- Clearly write your answer in the space indicated. For rough work, do not use any additional sheets. Rough work will not be graded.
- Assume IA32 machine running Linux.
- The exam has a maximum score of 50 points. The problems are of varying difficulty. The point value of each problem is indicated.
- This exam is CLOSED BOOK. Notes are NOT allowed.
- Anyone who copies of allows someone to copy will receive F grade.

  Good luck!

1 (10):
2 (8):
3 (4):
4 (9):
5 (5):
6 (6):
7 (8):
TOTAL (50):

Problem 1. (10 points):

Circle the correct answer.

1. Which one of the following memory addresses, written in binary, is 8-byte aligned?

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

2. A program blocks `SIGCHLD` and `SIGUSR1`. It is then sent a `SIGCHLD`, a `SIGUSR1`, and another `SIGCHLD`, in that order. What signals does the program receive after it unblocks both of those signals (you may assume the program does not receive any more signals after)?

   (a) None, since the signals were blocked they are all discarded.
   (b) Just a single `SIGCHLD`, since all subsequent signals are discarded.
   (c) Just a single `SIGCHLD` and a single `SIGUSR1`, since the extra `SIGCHLD` is discarded.
   (d) All 3 signals, since no signals are discarded.

3. Consider the following C variable declaration:

   ```
   int *(*f[3])();
   ```

   Then `f` is

   (a) an array of pointers to pointers to functions that return `int`
   (b) a pointer to an array of functions that return pointers to `int`
   (c) a function that returns a pointer to an array of pointers to `int`
   (d) an array of pointers to functions that return pointers to `int`
   (e) a pointer to a function that returns an array of pointers to `int`
   (f) a pointer to an array of pointers to functions that return `int`

4. Which of the following is a difference between blocking and ignoring a signal?

   (a) Once a blocked signal is unblocked, it will be handled by the process. A signal that comes while it is being ignored will never be handled.

   (b) `SIGSTP` and `SIGINT` can be ignored, but not blocked.

   (c) Ignoring a signal only causes it to have no effect, while blocking a signal returns the signal to its sender.

   (d) None of the above

5. Consider two processes `P` and `C`, where `P` is the parent of `C`. Which of the following statements is FALSE?

   (a) If `C` calls `exit()` before `P` calls `wait()`, `C` becomes a zombie process.

   (b) If `P` calls `exit()` before `C` calls `exit()`, `C` becomes an orphan process.

   (c) If `C` calls `exit()`, a `SIGCHLD` signal is sent to `P`.

   (d) If `C` calls `exec()`, `P` is no longer the parent process of `C`.

Problem 2. (8 points):

Write T (for TRUE) or F (for FALSE) in the blanks provided. You do not have to give a reason. NOTE: Correct answers receive +1 mark, incorrect answers -1 mark, unanswered parts receive 0.

A. ____ Suppose a process allocates a variable `x` on the stack, then forks a child process. The child may change its value of variable `x` without changing the value of `x` seen by the parent process.

B. ____ A signal handler executing as the result of a received signal can never be interrupted by another incoming signal of the same type.

C. ____ When a SIGCHLD signal is sent to a parent, the child process is considered reaped.

D. ____ Storing data on the heap does not require knowing the size of that data at compile time.

E. ____ External fragmentation is caused by chunks which are marked as allocated but actually cannot be used.

F. ____ Internal fragmentation is caused by padding for alignment purposes and by overhead to maintain the heap structure (such as headers and footers).

G. ____ Coalescing while traversing the list during calls to malloc is known as immediate coalescing.

H. ____ Garbage collection, employed by calloc, refers to the practice of zeroing memory before use.

Problem 3. (4 points):

Consider the C program below. For space reasons, we are not checking error return codes. Assume that all functions return normally and that the proper header files have been included.

```
void handler1(int sig) {
    printf("child got SIGUSR1\n");
}

void handler2(int sig) {
    printf("parent got SIGUSR1\n");
}

int main() {
    pid_t pid;

    if ((pid = fork()) == 0) {
        signal(SIGUSR1, handler1);
        kill(getppid(), SIGUSR1);
        sleep(1);
        exit(0);
    }
    else {
        signal(SIGUSR1, handler2);
        kill(pid, SIGUSR1);
        waitpid(pid, NULL, 0);
    }
    return 0;
}
```

The programmer who wrote this program would like it to always print these two lines:

```
parent got SIGUSR1
child got SIGUSR1
```

These lines could appear in reverse order in different executions because the scheduler decides whether the parent or the child runs first following a fork. However, this program has a bug—it does not print the two lines the programmer expects it to print. Identify **and briefly explain** the bug. Do **NOT** try to identify style flaws.

Problem 4. (9 points):

Consider an allocator that uses an implicit free list. 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 and is represented in units of bytes. 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 block: 1 for allocated, 0 for free.
- bit 2 indicates the use of the next block: 1 for allocated, 0 for free.

Three helper routines are defined to facilitate the implementation of `free(void *p)`. The functionality of each routine is explained in the comment above the function definition. Circle Yes or No to indicate whether the body of the function correctly implements the functionality described. If you circle No, rewrite the code such that it is correct.

```
/* Given a pointer to a valid block header or footer, returns the size of the
   block (number of bytes). */
int size (void *hp) {
   return (*hp) & (~0x7);    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

```
/* Given a pointer p to an allocated block, i.e., p is a pointer returned by
   a previous malloc call; returns a pointer to the block's footer. */
void *footer(void *p) {
   return (char *)p + size((char *)p - 4) - 8;    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

```
/* Given a pointer to a valid block header or footer, returns the usage of
   the previous block, 1 for allocated, 0 for free.  */
int prev_allocated(void *hp) {
   return (*(int *)hp) & 0x4;    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 5. (5 points):

Consider the following C program. For space reasons, we are not checking error return codes, so assume that all functions return normally.

```
int main() {
   int val = 2;

   printf("%d", 0);
   fflush(stdout);

   if (fork() == 0) {
       val++;
       printf("%d", val);
       fflush(stdout);
   }
   else {
       val--;
       printf("%d", val);
       fflush(stdout);
       wait(NULL);
   }
   val++;
   printf("%d", val);
   fflush(stdout);
   exit(0);
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program. You will be graded on each sub-problem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 1 points.
- If you circle the wrong answer, you get -1 points.

A. `01432` Y N

B. `01342` Y N

C. `03142` Y N

D. `01234` Y N

E. `03412` Y N

Problem 6. (6 points):

Your friend Neytiri was inspired by shell lab, and has, in an effort to better understand process management, written the following C program.

```
extern int do_stuff();
void magic() {
   if (fork()) exit(0);
}

int main() {
   magic();
   return do_stuff();
}
```

The main functionality of the program is implemented in `do_stuff`, defined in another file, but she seems more intent on showing off the `magic` function called beforehand.
Suppose that you run Neytiri's program in a shell whose process ID is 1000 and that the shell creates a process having ID 1001 to run the program.

A. Assume that the call to `fork` creates a child process whose process ID is 1002. What will be the PIDs of all processes that run `do_stuff` after `magic` returns, and for each one, what will be the PID of its parent?

B. Suppose `do_stuff` takes a long time to execute. Will the shell next emit a prompt only after `do_stuff` returns, or might it do so sooner? Justify your answer.

Problem 7. (8 points):

Consider the following C program.

```
void print() {
   if (fork())
      printf("!\n");
   else {
      --X--
      printf("!\n");
   }
   printf("!\n");
}
int main() {
   printf("!\n");
   print();
   printf("!!\n");
   return 0;
}
```

A. Suppose we want the program to print exactly 37 exclamation marks (!). How many successive `fork()` calls would we need to insert in place of `--X--` to produces this output? **No marks will be given unless a proper diagram and explanation are provided**.

B. How many times will two exclamation marks be printed on the same line? Explain how you came up with this number.