Full Name: _____

Roll Number: _____

IT215: Systems Software, Winter 2014-15

Second In-Sem Exam (2 hours)

March 24, 2015

Instructions:

- Make sure your exam is not missing any sheets, then write your name and roll number on the top of this page.
- Clearly write your answer in the space indicated. None of the questions need long answers.
- 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. It is CLOSED BOOK. Notes are NOT allowed.
- Anyone who copies or allows someone to copy will receive F grade.

  Good luck!

1 (12):
2 (6):
3 (4):
4 (12):
5 (6):
6 (4):
7 (6):
TOTAL (50):

Problem 1. (12 points):

Circle the *single best* answer to each of the following questions. If you circle more than one answer, you will lose the mark for the corresponding question.

1. Imagine a process (called "process A") that spawns three child processes. If all three child processes terminate before process A is picked by the kernel to be run again, how many times could process A received `SIGCHLD`?

   (a) 0

   (b) 1

   (c) 3

   (d) 1 or 3

2. Consider the following piece of code. Note that the file name is the same for both calls to `open`, and assume the file `one.txt` exists.

   ```
   int fd = open("one.txt", O_RDWR);
   int fd2 = open("one.txt", O_RDONLY);
   ```

   Assuming the calls to `open` succeed, which of the following is true?

   (a) `fd` and `fd2` will share the same file offset.

   (b) Both `fd` and `fd2` will have an initial file offset that is set to the end of the file.

   (c) Whatever is written to the file through `fd`, can be read using `fd2`.

   (d) In total, there will be two copies of the file `one.txt` in memory, one associated with `fd` and the other with `fd2`. Any changes made in a copy will **not** be reflected in the other copy.

The following program is a failed attempt by a student to get a parent and child process to communicate — the child is intended to run the `/bin/ls` program (which produces a directory listing to stdout) and the parent is to run the `/usr/bin/wc` program (which counts the characters, words, and lines in its stdin and prints the counts to stdout).

```
#define PROG1 "/bin/ls"
#define PROG2 "/usr/bin/wc"

main()
{
    int fds[2];
    pipe(fds);
    if (fork() == 0) {
         /* child */
         dup2(fds[1], 1);
         execl(PROG1, PROG1, NULL);
    } else {
         /* parent */
         dup2(fds[0], 0);
         execl(PROG2, PROG2, NULL);
    }
}
```

3. Upon running the program, the student observes that there is no output whatsoever. Why?

   (a) the parent process is running before the child process

   (b) the child process is running before the parent process

   (c) the child process is blocking indefinitely

   (d) the parent process is blocking indefinitely

4. How would you go about fixing the program?

   (a) add `close(fds[1])` just before calling `fork`

   (b) add `close(fds[0])` in the child just before calling `exec`

   (c) add `close(fds[1])` in the parent just before calling `exec`

   (d) add `wait(NULL)` to the parent just before calling `exec`

Problem 2. (6 points):

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.) Note that newlines may be present in the output but are disregarded for this problem. Additionally, assume that all `printf` statements are flushed immediately.

```
int main() {
   fork();
   printf("0");
   if (fork() == 0) {
      /* this next exec prints "1" */
      execl("/bin/echo", "echo", "1", NULL);
      fork();
      printf("2");
   }
   printf("3");
   return 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. `00112233223333` Y N

b. `0132310323` Y N

c. `013013` Y N

d. `001133` Y N

e. `013310` Y N

f. `030311` Y N

Problem 3. (4 points):

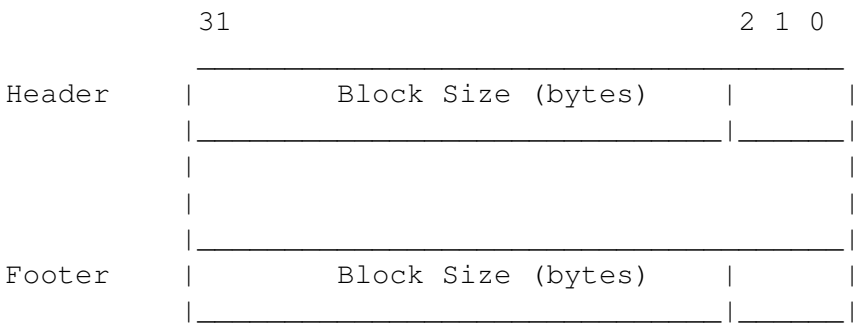
Your answers to the following questions should be 50 words or less. Responses longer than 50 words will receive 0 points.

A. The implementation of `tsh`–your tiny shell assignment–relied on a `SIGCHLD` handler to update the job list whenever a job terminated or stopped. The installed `SIGTSTP` handler, on the other hand, simply intercepted and forwarded `SIGTSTP` on to the foreground job. Why not update the job list to reflect the job state change inside the implementation of the `SIGTSTP` handler?

B. Signals set to be caught by custom signal handlers will have default signal handling behavior in the child process after `exec` is called. Why can't the child process inherit the parent's installed signal handlers instead?

Problem 4. (12=8+4 points):

Consider an allocator that uses an implicit free list, where the layout of each memory block is as follows:



Each memory block, either allocated or free, has a size that is a multiple of 8 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. 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.
- bits 1 and 2 are unused and always set to be 0.

Note: The header and footer will always be present regardless of whether the block is allocated or not.

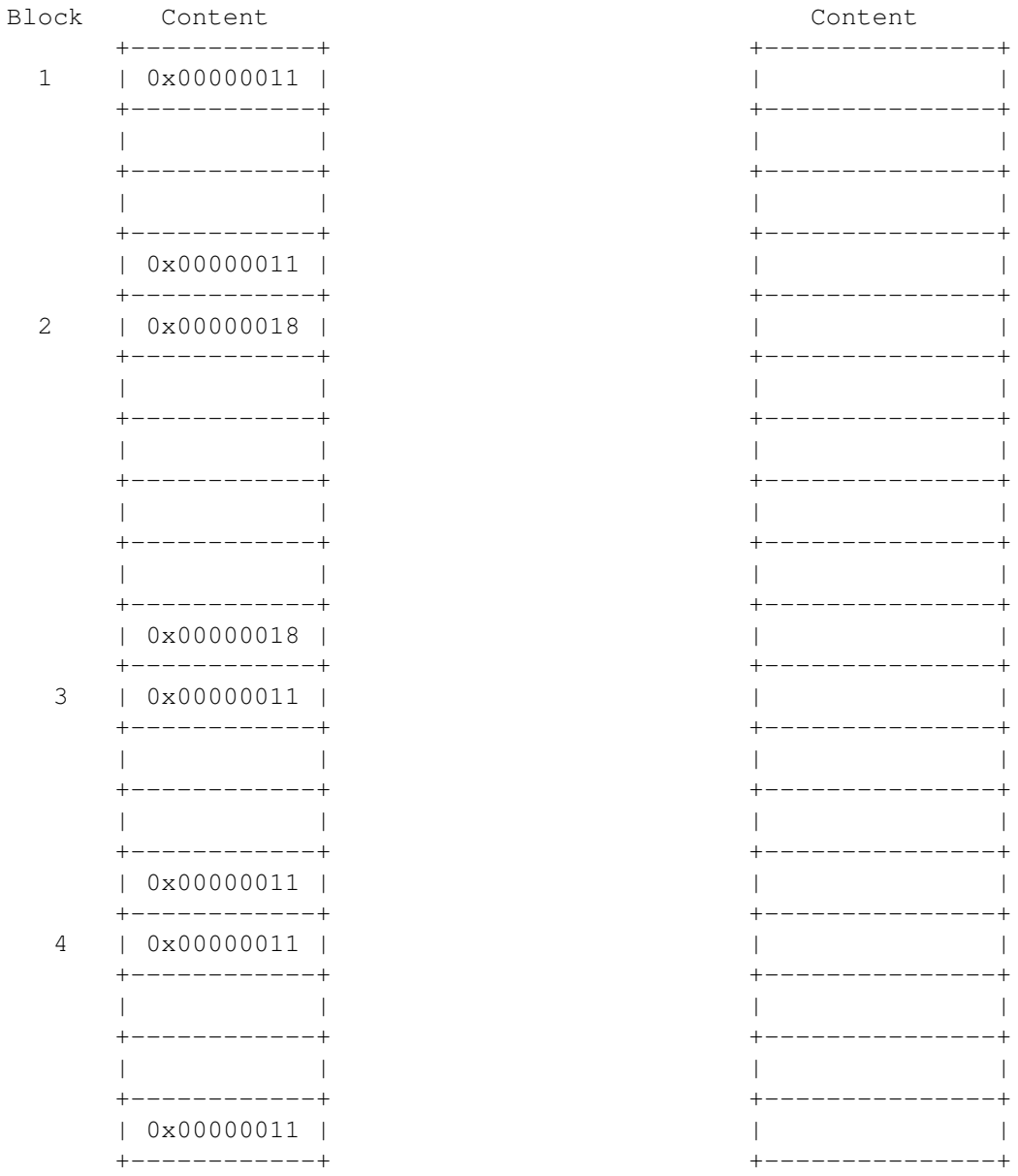
Implement the code as indicated by the comments below. Make use of previous results and the function `size()` when possible. Careful: pointer arithmetic is valid only on pointers to actual types, not `void *`.

Part 1

```
void *p = malloc(20);

/* hp points to the header of the block returned by the
   previous malloc call */

void *hp = _____________________________________;

/* fp points to the block's footer */

void *fp = _____________________________________;


/* pp points to the header of the previous block in memory */

void *pp = _____________________________________;


/* Given a pointer to a block header or footer, returns the size
of the block (number of bytes). */

int size(void *hp) {

    return _____________________________________;
}
```

Part 2

Given the contents of the heap shown on the left, fill in the header and footer values in hex (on the right side) after block 3 is freed, and coalescing is done.



Problem 5. (6 points):

For each code snippet, circle the value(s) of `i` that could possibly be printed by the `printf` command at the end of each program. Assume that all functions and procedures return correctly and that all variables are declared and initialized properly. Also, assume that an arbitrary number of SIGINT signals, and only SIGINT signals, can be sent to the code snippets below randomly from some external source. *Careful: There may be more than one correct answer for each question. Circle all the answers that could be correct. Penalty for circling wrong answer = -2 marks*

Code snippet 1:

```
int i = 0;

void handler(int sig) {
  i = 0;
}

int main() {
   int j;
   sigset_t s;

   signal(SIGINT, handler);

   /* Assume that s has been
      initialized and declared
      properly for SIGINT */

   sigprocmask(SIG_BLOCK, &s, 0);
   for (j=0; j < 100; j++)
       i++;
       sleep(1);
   }
   sigprocmask(SIG_UNBLOCK, &s, 0);
   printf("i = %d\n", i);
   exit(0);
}
```

1. Circle possible values of `i` printed by snippet 1:
   (a) 0
   (b) 1
   (c) 50
   (d) 100
   (e) 101
   (f) Terminates with no output.

Code snippet 2:

```
int i = 0;

void handler(int sig) {
  i = 0;
  sleep(1);
}

int main() {
   int j;
   sigset_t s;

   /* Assume that s has been
      initialized and declared
      properly for SIGINT */

   sigprocmask(SIG_BLOCK, &s, 0);
   signal(SIGINT, handler);
   for (j=0; j < 100; j++)
       i++;
       sleep(1);
   }
   printf("i = %d\n", i);
   sigprocmask(SIG_UNBLOCK, &s, 0);
   exit(0);
}
```

2. Circle possible values of `i` printed by snippet 2:
   (a) 0
   (b) 1
   (c) 50
   (d) 100
   (e) 101
   (f) Terminates with no output.

Problem 6. (4 points):

The following program performs a shell-like file copying. It forks out a child process to copy the file. The parent process waits for the child process to finish and, at the same time, captures the `SIGINT` signal. The signal handler asks the user whether to kill the copying or not. If the user answers '`y`', the child process will be terminated. Otherwise, the signal handler just silently returns from the signal handler.
Fill in the blank lines to make the code work. Please fill in "(empty)" if no code is needed.

```
pid_t child_pid = 0;

void handler(int sig) {
     char answer;

     printf("Stop copying?\n");
     scanf("%c", &answer);
     if (answer == 'y' && child_pid != 0)

          ------------------------------------------------;
}

void copy() {

     ------------------------------------------------;

     if ((child_pid = fork()) == 0) {

          ------------------------------------------------;

          copy_file();
          exit(0);
     }
     else {

          ------------------------------------------------;

          if (wait(NULL) == child_pid) {
             printf("Child process %d ended.\n", child_pid);
          }
     }
}
```

Problem 7. (6 points):

You are asked to show what each of the following two programs prints as output.

- Assume that file `infile.txt` contains the ASCII text characters "`15213`";
- You may assume that system calls do not fail;
- When a process with no children invokes `waitpid(-1, NULL, 0)`, this call returns immediately.

Each of the following questions has a unique answer.

Program A

```
int main()
{
    int fd;
    char c;

    fd = open("infile.txt", O_RDONLY, 0);

    fork();
    waitpid(-1, NULL, 0);

    read(fd, &c, sizeof(c));
    printf("%c", c);

    return 0;
}
```

OUTPUT: _____

Program B

```
int main()
{
    int fd;
    char c;

    fork();
    waitpid(-1, NULL, 0);

    fd = open("infile.txt", O_RDONLY, 0);

    read(fd, &c, sizeof(c));
    printf("%c", c);

    return 0;
}
```

OUTPUT: _____