Part 1. More fun with forks

(a) What is the output generated by this program? In fact the output is not uniquely defined, i.e., it is not necessarily the same in each execution. What are the possible outputs? (Assume that there are no errors or failures.) [20 pts]

```c
int main()
{
    int i = 0;
    if (fork() != 0) {
        i = i + 1;
        if (fork() != 0)
            exit(0);
    }
    fork();
    printf("%d\n", i);
}
```

(b) Briefly justify/explain your answer for (a). Try to characterize the set of all possible outputs. [20 pts]
Part 2. True/False

The following true/false questions pertain to the classic C/Unix (or Android) environment as discussed in class. For each statement, indicate (in the space on the left) whether it is true (T) or false (F). Please add a brief comment to explain your answer in the space provided. [40 points]

(a) The first user-mode instruction that executes in any Unix process is in a system call stub.

(b) Every thread context switch results from a timer interrupt or a sleep operation.

(c) A successful `exec*` system call trap never returns to the process that invoked it.

(d) A successful `exec*` system call reads data from a file (among other things that it does).

(e) A machine fault always indicates some kind of error in the software.

(f) Interrupt handlers execute entirely in kernel mode.

(g) A child process runs with the user ID of its parent.

(h) A pipe can be used to communicate only among siblings of a common parent.

(i) A running program may read its standard input (stdin) from a network socket.

(j) Multiple processes in a single pipeline can execute at the same time.
Part 3. Reference counts

The following questions pertain to the classic C/Unix (or Android) environment as discussed in class. Answer each question with a few phrases or maybe a sentence or two. [40 points]

(a) The Unix kernel uses reference counting to manage the lifetimes of various objects. The reference counts are incremented and decremented during the execution of various system calls. List five system calls that increment reference counts on objects in the kernel.

(b) List five system calls that decrement reference counts on objects in the kernel.

(c) Are there any cases in which a fault handler might increment or decrement reference counts on objects in the kernel? Cite example(s) and/or explain.

(d) Are there any cases in which the kernel might store a reference count on disk? Cite example(s) and/or explain.

(e) True or false: dangling references cannot occur when reference counting is used (correctly). Explain.

(f) List two operations in Android that increment reference counts maintained by Android system software. What does Android do when these reference counts go to zero?
Part 4. Cats

As you know, *cat* is a simple standard Unix program that invokes read/write system calls in a loop to transfer bytes from its standard input (stdin) to its standard output (stdout). These questions ask you to explain various interactions of cats (processes running the *cat* program) with one another and with the kernel.

(a) Consider this command line to a standard shell: "cat <in | cat". How does the second cat know when it is done, i.e., what causes it to exit? [10 points]

(b) Suppose that the program called *empty* is the null C program: `int main() {}`. Consider this command line to a standard shell: "cat | empty". What does it do? How does the cat know when it is done, i.e., what causes it to exit? [10 points]

(c) Consider this command line to a standard shell: "cat <in >out". You may assume that the current directory resides on a disk, and that "in" is a file with some substantial amount of data in it. This question asks you to explain how this cat consumes memory and CPU time. Please answer on the following page. [60 points]

First, how much memory does it consume? Draw a picture of the page table and the segments of the virtual address space, with your best guess of the total size. I am looking for a rough sketch of these data structures as they would reside in the memory of the machine. Details vary, so you may make any reasonable simplifying assumptions about the machine or the *cat* program, but please note them in your answer.

Second, how much CPU time does it consume? How does the CPU time vary with the size of the file *in*? How much of the time is spent in kernel mode vs. user mode? Draw a rough sketch of how the cat transitions between user mode, kernel mode, and sleep states as it executes through time. What events cause the transitions?