Question 1: How many lines will the following program print when standard output goes to the screen?

```
int main()
{
  if (fork() == 0) printf("Fred\n");
  if (fork() == 0) printf("Fred\n");
  return 0;
}
```

Question 2: Suppose that **fred.txt** is two lines. How many lines will the following program print when standard output goes to the screen?

```
int main()
{
  int status;

  printf("Hi\n");
  if (fork() == 0) {
    execlp("cat", "cat", "fred.txt", NULL);
    execlp("cat", "cat", "fred.txt", NULL);
    execlp("cat", "cat", "fred.txt", NULL);
    exit(1);
  } else {
    (void) wait(&status);
    printf("Done\n");
  }
  return 0;
}
```

Question 3: Suppose I compile the following program to **bin/click3** and I then do:

```
UNIX> bin/click3 > tmp.txt
```

How many lines will **tmp.txt** be?

```
int main()
{
  int status;

  printf("Hi\n");
  if (fork() == 0) {
    printf("I am a child.\n");
  } else {
    printf("I am no child.\n");
    wait(&status);
  }
  return 0;
}
```

Question 4: Suppose I compile the following program to **bin/click4** and I then do:

```
UNIX> cat fred.txt
Fred: Line 1
Fred: Line 2
UNIX> bin/click4 > tmp.txt
```

How many lines will **tmp.txt** be?

```
int main()
{
  int status;

  printf("Hi\n");
  if (fork() == 0) {
    execlp("cat", "cat", "fred.txt", NULL);
  } else {
    wait(&status);
    printf("Done\n");
  }
  return 0;
}
```

Question 5: What will the first line of **tmp.txt** be?

- *A*. "Hi"
- *B*. "Done"
- *C*. "Fred: Line 1"
- *D*. "Fred: Line 2"

Answers to Clicker Questions

Question 1

The answer is three. After the first **fork**() call, there are two processes, but only one of them (the child) returned 0 from **fork**(). So it prints a line. Now, there are two processes calling the second **fork**(), so after the second **fork**() call, there are four processes and two of them (the child of the child, and the second child of the parent) returned 0 from **fork**(). They each print a line, for a total of three lines.

Question 2

```
UNIX> bin/click2
Hi
Fred: Line 1
Fred: Line 2
Done
UNIX>
```

Since output is to the terminal, the stdout buffer is flushed at the end of every line. So the "Hi" line is indeed printed after the **printf**() statement. The **execlp**() statement loads **cat** into the address space and runs it. It will print the two lines of **fred.txt** and exit. Since the **execlp**() call is successful, its address space is gone, and it doesn't return. Those other **execlp**() calls are never executed.

When the child exits, the parent prints "Done". The output is four lines.

Question 3

Suppose I compile the following program to **bin/click3** and I then do:

```
UNIX> bin/click3 > tmp.txt
UNIX> cat tmp.txt
Hi
I am a child.
Hi
I am no child.
UNIX>
```

As you can see, it prints four lines. That's because we're redirecting **stdout** to a file, and the **printf**() call is buffered. The **fork**() call copies the buffer, and that is why "Hi" is printed twice.

Questions 4 and 5

```
UNIX> cat fred.txt
Fred: Line 1
Fred: Line 2
UNIX> bin/click4 > tmp.txt
UNIX> cat tmp.txt
Fred: Line 1
Fred: Line 2
Hi
Done
UNIX>
```

As in the previous question, the "Hi" line goes into a buffer. That buffer is indeed copied to the child process in the **fork**() call. However, the **execlp**() call overwrites the address space, so the buffer is gone. As a result, the child never prints "Hi".

So there are four lines -- two from the parent and two from the child.

The first line is "Fred: Line 1". Remember, the "Hi" is in the parent's buffer, and it doesn't get printed until the parent exits. The parent has called **wait**(), so it doesn't print its buffer until after the child has printed its output.