(For questions 1 and 2): Behold the following program, which is compiled to **bin/click1**:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main()
{
  int fd;
  int child, parent, status;
  char c;

  fd = open("f1.txt", O_RDONLY);

  child = (fork() == 0);
  parent = !child;

  if (child) sleep(1);
  read(fd, &c, 1);
  if (parent) wait(&status);
  printf("%c\n", c);

  return 0;
}
```

Here's **f1.txt**:

```
UNIX> cat f1.txt
Christian Dogleg Jr
Tyler Sims
UNIX>
```

**Question 1**: What is the first line of **bin/click1**?
**Question 2**: What is the second line of **bin/click1**?

**Question 3**: After running the following program, how many lines are there in **f2.txt**?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main()
{
  FILE *f;

  f = fopen("f2.txt", "w");
  fprintf(f, "Fred\n");
  fork();
  fork();
  fprintf(f, "Binky\n");
  fclose(f);
  return 0;
}
```

**Question 4**: If you call **alarm**() and then **fork**(), the operating system will only generate **SIGALRM** for the parent process.

The default signal handler for **SIGLARM** has the process die instantly, without calling **exit**(), and without flushing any stdio buffers.

If you wait a few seconds after the following program exits, how many lines are there in **f3.txt**?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main()
{
  FILE *f;

  f = fopen("f3.txt", "w");
  fprintf(f, "Fred\n");
  alarm(1);
  fork();
  sleep(2);
  fprintf(f, "Binky\n");
  fclose(f);
  return 0;
}
```

**Questions 1 and 2**: Here's what happens:

1. The process opens "f1.txt". It uses **open** for this, so there is no buffering -- it's just a file descriptor in the operating system.
2. It calls **fork**(). There are two boolean variables -- **child** is true in the child and **parent** is true in the parent.
3. The child sleeps for a second.
4. The parent doesn't sleep, so it reads a byte from f1.txt. That byte is the character 'C' -- the first byte in the file.
5. The parent then calls **wait**(). That will block until the child is done.
6. The child eventually wakes up from its **sleep**() call, and it calls **read**(). When you call **fork**() file descriptors are duplicated between the parent and child, as if **dup**() were called, so they share a seek pointer. Therefore, the child reads the second byte from the file, which is the character 'h'.
7. The child now prints the 'h' and exits.
8. The parent wakes up from the **wait**() call, prints the 'C' and exits.

So, the answer to Question 1 is 'h', and the answer to Question 2 is 'C':

```
UNIX> gcc src/click1.c
UNIX> ./a.out
h
C
UNIX>
```

**Question 3**: Here's what happens:

1. The file "f2.txt" is opened for writing via a stdio buffer. Since the output is going to a file, **fprintf**() calls will go into a buffer, which isn't flushed until the buffer is full or explicitly flushed.
2. For that reason, "Fred" goes into the buffer, and not yet to "f2.txt".
3. After the first **fork**() call, there are two processes.
4. After the second **fork**() call, there are four processes.
5. All four processes write "Binky" into the buffer.
6. Then all four processes close **f**, which flushes the buffer. For that reason, each process will write both "Fred" and "Binky".
7. Therefore, there are 8 lines of output.

```
UNIX> gcc src/click2.c
UNIX> ./a.out
UNIX> cat -n f2.txt
     1  Fred
     2  Binky
     3  Fred
     4  Binky
     5  Fred
     6  Binky
     7  Fred
     8  Binky
UNIX>
```

**Question 4**: Here's what happens:

1. The file "f3.txt" is opened for writing via a stdio buffer. Since the output is going to a file, **fprintf**() calls will go into a buffer, which isn't flushed until the buffer is full or explicitly flushed.
2. As before, "Fred" is written to the buffer, but not yet to the file.
3. We call **alarm(1)**, which will send SIGALRM to the process in a second.
4. We call **fork**(), and both processes sleep for two seconds.
5. After a second, SIGALRM is sent to the parent. Since it did not set up a signal handler, the process will exit withtout flushing its buffers. That means the parent never writes "Fred" to the file.
6. After another second, the child wakes up, wirte "Binky" to the buffer and then closes the file. This flushes the buffer, so only two lines are written to the file.
7. Therefore, the answer is two.

```
UNIX> gcc src/click3.c
UNIX> ./a.out
Alarm clock: 14                 # On my mac, this is what happens when you don't catch SIGALRM
UNIX> cat -n f3.txt
     1  Fred
     2  Binky
UNIX>
```