C + x86 Assembly : Exceptional Control Flow

By Amanda Falke
September 2014
Abstractmachines at gmail

Questions which I try to answer from :
Bryant and O'Halloran's *"Computer Systems: A Programmer's Perspective"*
Answers are in bold.

8.10 ◆ In this chapter, we have introduced some functions with unusual call and return behaviors: setjmp, longjmp, execve, and fork. Match each function with one of the following behaviors:

A. Called once, returns twice.
B. Called once, never returns.
C. Called once, returns one or more times.

ANSWER:
A. Fork is called once, returns twice.
B. Execve is called once, never returns.
B. Longjump is called once, never returns.
C. Setjmp is called once and returns one or more times.

8.13 ◆

What is one possible output of the following program?

```
#include "csapp.h"
int main()
{
      int x = 3;
      if (Fork() != 0)
      {
            printf("x=%d\n", ++x);
      }

      printf("x=%d\n", --x);
      exit(0);
}
```

ANSWER:

Due to interleaving, it's difficult to tell whether a parent or a child will output first.

// parent prints 4

// child prints 2

<u>output may be:</u>
4
2

<u>or output may be:</u>
2
4

8.14 ◆

How many “hello” output lines does this program print?

```
1 #include "csapp.h"
2
3 void doit()
4 {
5 if (Fork() == 0) {
6 Fork();
7 printf("hello\n");
8 exit(0);
9 }
10 return;
11 }
12
13 int main()
14 {
15 doit();
16 printf("hello\n");
17 exit(0);
18 }
```

ANSWER:

- <u>line 15</u> calls doit() function
-doitfunction: line 5 forks child

- <u>line 7</u> is printed twice, “hello” for the parent, as well as “hello” for the child.

- but NOTE that <u>line 6</u> calls Fork() again, creating yet another parent-child return, so we have “hello” again, and “hello” again. At this point, we have four “hello” printouts.

- <u>line 16</u> is then printed once, “hello” for the 5th time.

<u>FINAL ANSWER:</u>
I believe that there are 5 output lines of “hello,” but there may also be only three printouts, as I may have overestimated by two printouts, due to Fork() invocation in line 6.

8.15 ◆

How many “hello” output lines does this program print?

```
1 #include "csapp.h"
2
3 void doit()
4 {
5 if (Fork() == 0) {
6 Fork();
7 printf("hello\n");
8 return;
9 }
10 return;
11 }
12
13 int main()
14 {
15 doit();
16 printf("hello\n");
17 exit(0);
18 }
```

ANSWER:

This is the same as the previous problem, only the IF STATEMENT RETURNS.

- line 15 calls doit() function
-doitfunction: line 5 forks child

- line 7 is printed twice, “hello” for the parent, as well as “hello” for the child.

- but NOTE that line 6 calls Fork() again, creating yet another parent-child return, so we have “hello” again, and “hello” again. At this point, we have four “hello” printouts.

- line 16 is then printed once, “hello” for the 5th time.

FINAL ANSWER:
I believe that there are 5 output lines of “hello,” but there may also be only three printouts, as I may have overestimated by two printouts, due to Fork() invocation in line 6.

8.16 ◆

What is the output of the following program?

```
1 #include "csapp.h"
2 int counter = 1;
3
4 int main()
5 {
6 if (fork() == 0) {
7 counter--;
8 exit(0);
9 }
10 else {
11 Wait(NULL);
12 printf("counter = %d\n",
++counter);
13 }
14 exit(0);
15 }
```

ANSWER:

-line 6 forks child,
-child decrements counter to zero in line 5
- parent waits for child in line 11
in line 12, parent prints counter of 1, because only the child incremented it.

OUTPUT:

1

8.18 ◆

Consider the following program. Determine which of the following outputs are possible. Note: The atexit function takes a pointer to a function and adds it to a list of functions (initially empty) that will be called when the exit function is called.

A. 112002

B. 211020

C. 102120

D. 122001

E. 100212

```
#include "csapp.h"

void end(void)
{
printf("2");
}

int main()
{
if (Fork() == 0)
atexit(end);
if (Fork() == 0)
printf("0");
else
printf("1");
exit(0);
}
```

ANSWER:

A, B, D, E

8.19 ◆

How many lines of output does the following function print? Give your answer as a function of **n**. Assume **n** ≥ 1.

```
1 void foo(int n)
2 {
3 int i;
4
5 for (i = 0; i < n; i++)
6 Fork();
7 printf("hello\n");
8 exit(0);
9 }
```

ANSWER:

- <u>line6:</u> prints “hello” 2*n times when then child is forked

<u>FINAL ANSWER:</u>

2 * n   is how many “hello” printouts

8.23 ◆◆

One of your colleagues is thinking of using signals to allow a parent process to count events that occur in a child process. The idea is to notify the parent each time an event occurs by sending it a signal, and letting the parent’s signal handler increment a global counter variable, which the parent can then inspect after the child has terminated. However, when he runs the test program in Figure 8.41 on his system, he discovers that when the parent calls printf, counter always has a value of 2, even though the child has sent five signals to the parent. Perplexed, he comes to you for help. Can you explain the bug?

ANSWER: The most likely bug is in the parent’s signal handler. The signal handler must RECEIVE all of the signals properly.

Another possibility is that the parent did not reap the zombie child correctly, and is blocked by waiting for the child to be fully terminated.

8.25 ◆◆◆
Write a version of the fgets function, called tfgets, that times out after 5 seconds. The tfgets function accepts the same inputs as fgets. If the user doesn't type an input line within 5 seconds, tfgets returns NULL. Otherwise, it returns a pointer to the input line.

ANSWER:

```
void handler ( int sig ) { return; } // keeps sleep from returning by terminating program


char  * tfgets(char *str, int n, FILE *stream, unsigned int secs) // ANSI standard function prototype

{

 int idx;

 char * tfgetsLocal;

 unsigned int elapsed = sleep(secs);

 while ( ( buffer != EOF ) && ( buffer != '\0' )  )

  {

   *tfgetsLocal = *stream;

  }

if ( secs >= elapsed ) { return *tfgetsLocal; }

else if ( secs == elapsed ) { return NULL; }

}
```

Amanda Falke