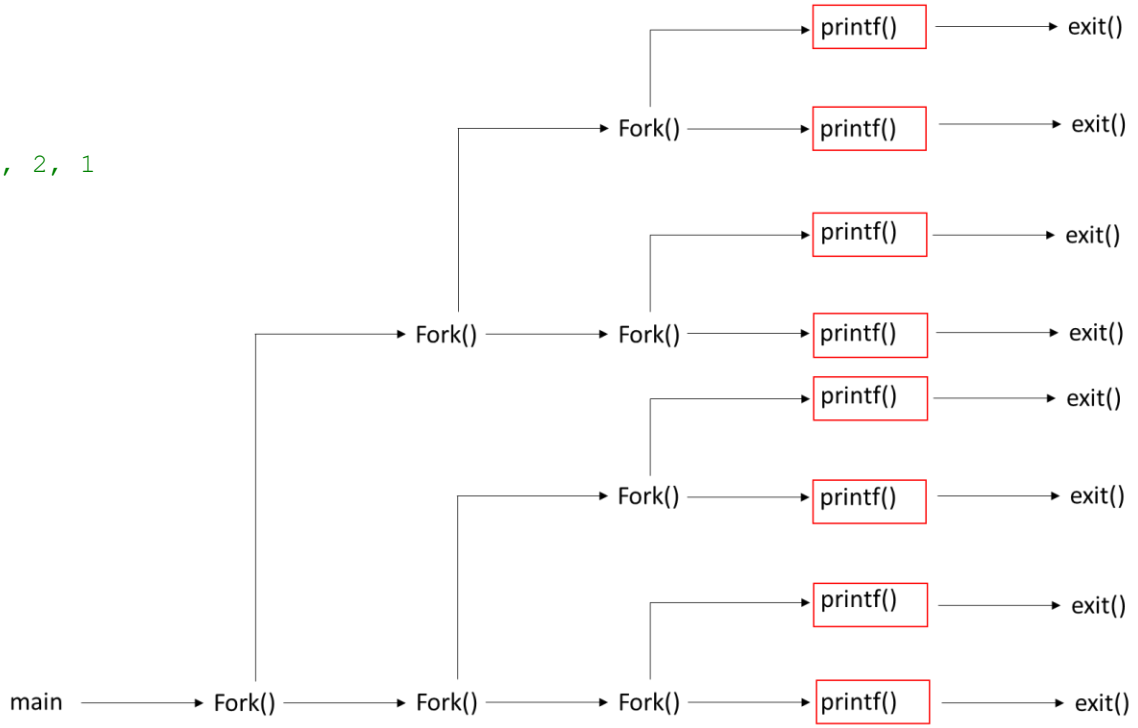
Geoffrey Xiao
CS283 H4

Problem 8.11

```c
#include "csapp.h"

int main()
{
    int i;
    for(i = 3; i > 0; i--) // i = 3, 2, 1
    {
        Fork();
    }
    printf("Example\n");
    exit(0);
}
```



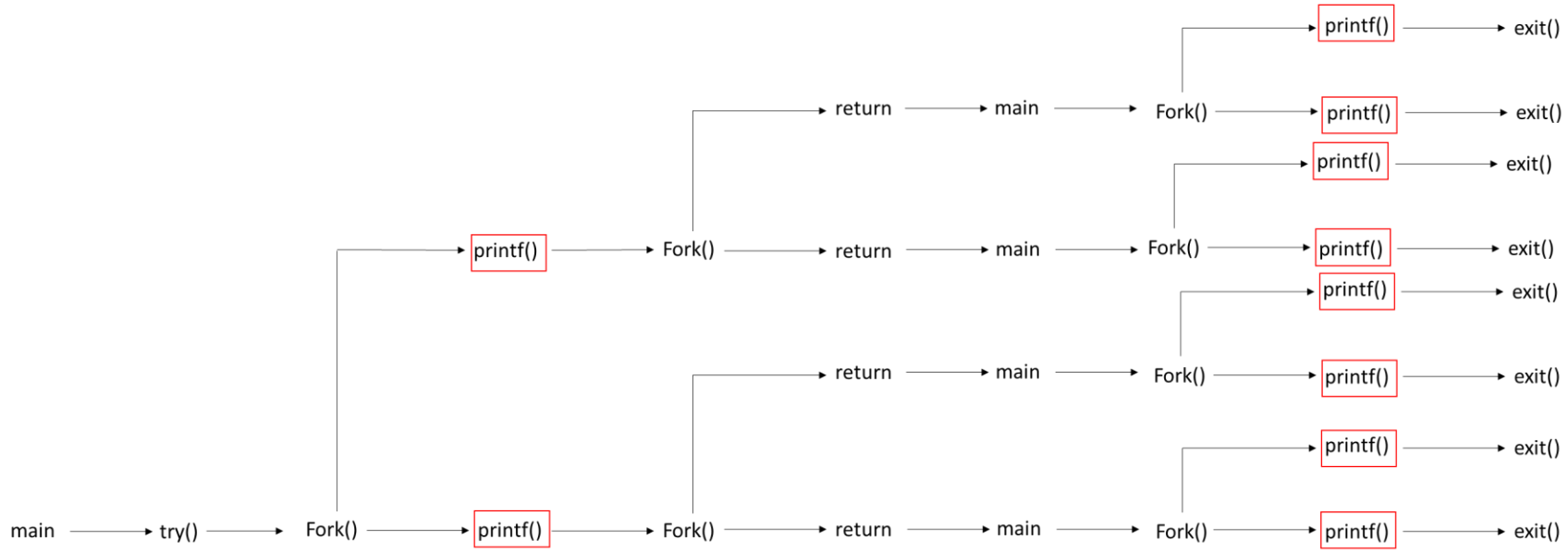
8 “Example” lines are printed.

Problem 8.12

```c
#include "csapp.h"

void try()
{
    Fork();
    printf("Example\n");
    Fork();
    return;
}

int main()
{
    try(); Fork();
    printf("Example\n");
    exit(0);
}
```



10 “Example” lines are printed.

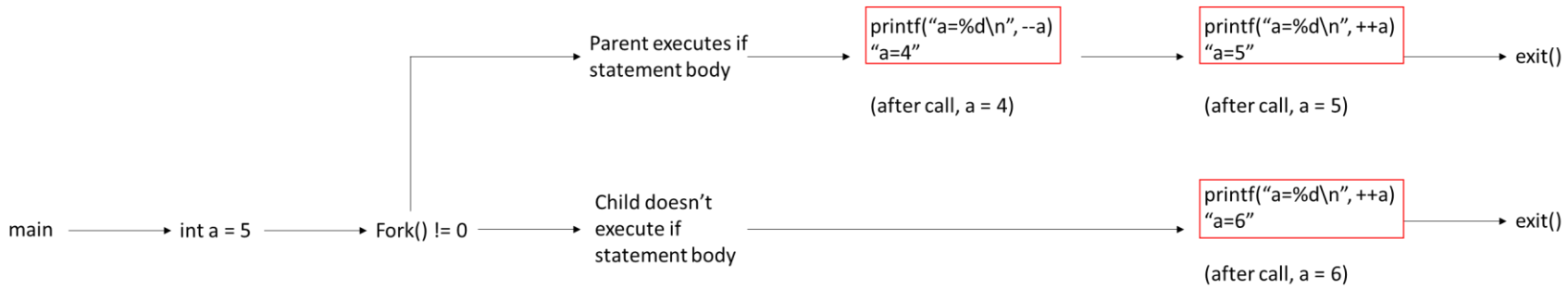
Problem 8.13

```c
#include "csapp.h"

int main()
{
    int a = 5;

    if(Fork() != 0)
        printf("a=%d\n", --a);

    printf("a=%d\n", ++a);
    exit(0);
}
```



The child and parent process can run concurrently.

One possible output is:

a=4
a=5
a=6

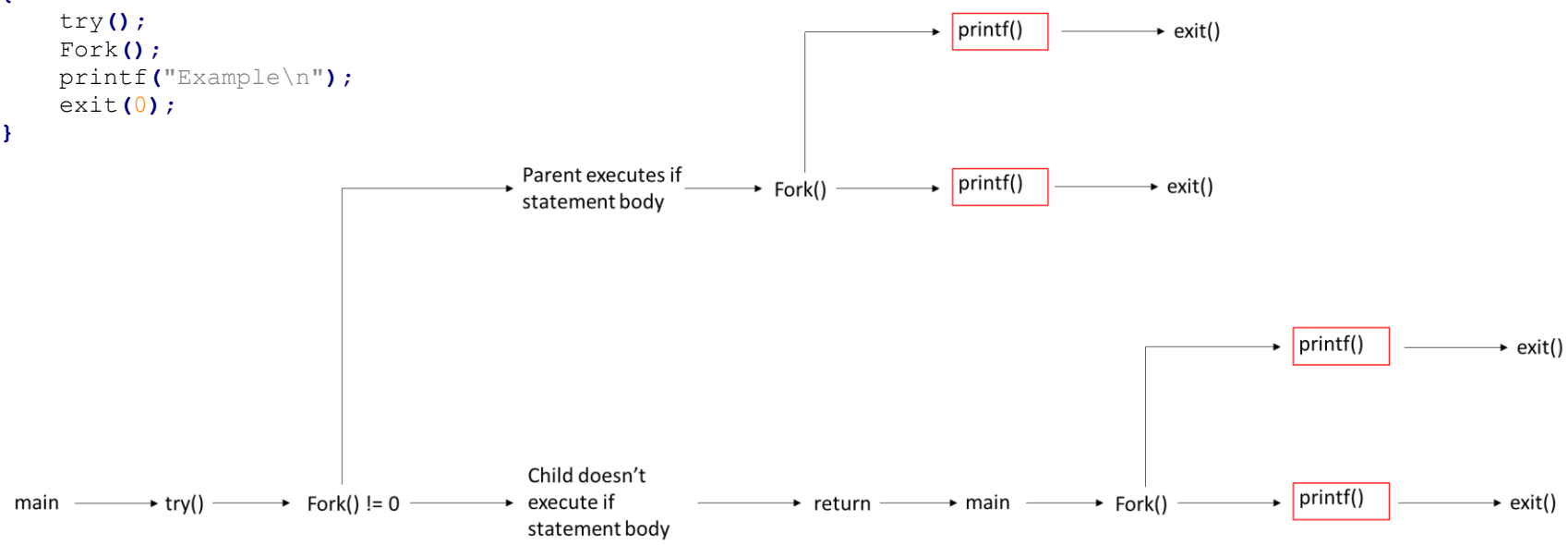
In this output sequence, the child process runs through entirely before the parent process runs.

Problem 8.14

```c
#include "csapp.h"

void try()
{
    if(Fork() != 0)
    {
        Fork();
        printf("Example\n");
        exit(0);
    }
    return;
}

int main()
{
    try();
    Fork();
    printf("Example\n");
    exit(0);
}
```

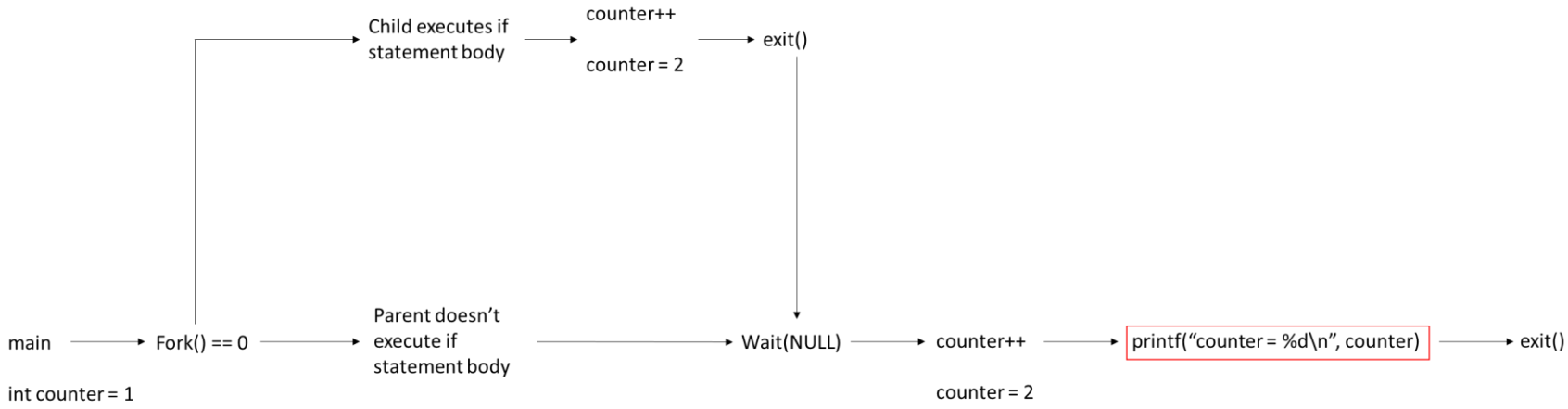


4 "Example" lines are printed.

Problem 8.16

```c
#include "csapp.h"
int counter = 1;

int main()
{
    if(fork() == 0)
    {
        counter++;
        exit(0);
    }
    else
    {
        Wait(NULL);
        counter++; printf("counter = %d\n", counter);
    }
    exit(0);
}
```



The output is: counter = 2

Problem 8.23

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 8.45 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?

code/ecf/counterprob.c

```
1   #include "csapp.h"
2
3   int counter = 0;
4
5   void handler(int sig)
6   {
7       counter++;
8       sleep(1); /* Do some work in the handler */
9       return;
10  }
11
12  int main()
13  {
14      int i;
15
16      Signal(SIGUSR2, handler);
17
18      if (Fork() == 0) {  /* Child */
19          for (i = 0; i < 5; i++) {
20              Kill(getppid(), SIGUSR2);
21              printf("sent SIGUSR2 to parent\n");
22          }
23          exit(0);
24      }
25
26      Wait(NULL);
27      printf("counter=%d\n", counter);
28      exit(0);
29  }
```

code/ecf/counterprob.c

Figure 8.45 Counter program referenced in Problem 8.23.

A pending signal is one that has been sent, but not received. Pending signals of the type currently being processed are blocked. This means the signal handler will not receive the blocked signal until the current signal is finished processing. Additionally, there can only be at most one pending signal of a particular type.

In the code, five SIGUSR2 signals are sent by the children. The first SIGUSR2 signal is received and processed by the parent signal handler. The second SIGUSR2 signal is sent, but not received—it is blocked. The third, fourth, and fifth SIGUSR2 signals are sent, but not received. The third, fourth, and fifth SIGUSR2 signals are also not placed in queue behind the second SIGUSR2 signal: these signals are discarded. After the signal handler finishes processing the first SIGUSR2 signal, the signal handler processes the second SIGUSR2 signal. After this, there are no further signals to process. This explains why the program prints a value of 2.