ICS EXE 3

March 24, 2020

1 Process

Assume we have the following code:

```
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int x = 10;
7  int main() {
8    int y = 1, i;
9    for (i = 0; i < 2; ++i) {
10     pid_t pid = fork();
11     if (!pid) {
12       x += y;
13       printf("x in child: %d\n", x);
14       exit(0);
15     }
16     y++;
17     x <<= 1;
18     printf("x in parent: %d\n", x);
19   }
20   return 0;
21 }
```

1.1

Give a possible output of the program

x in parent: 20
x in parent: 40
x in child: 11
x in child: 22

1.2

Based on Q1.1, how to use waitpid to make sure the "x in child" and the "x in parent" are interleaved? Will this influence the value of x in output?

add $waitpid(-1, NULL, 0)$; before line 16
No, the order won't influence the value.

1.3

Based on Q1.1, please list all possible output order. (or show the constraints on the output order).

20 always before 40, 20 always before 22

1.4

Based on Q1.1, if we accidentally delete the line 14, please give a possible output.

Follow child:

```
1 x in child: 11
2 x in parent: 22
3 x in child: 24
4 x in parent: 48
5 x in parent: 44
6 x in parent: 20
7 x in child: 22
8 x in parent: 44
9 x in parent: 40
```

Follow parent:

```
1 x in parent: 20
2 x in parent: 40
3 x in child: 11
4 x in parent: 22
5 x in parent: 44
6 x in child: 24
7 x in parent: 48
8 x in child: 22
9 x in parent: 44
```

2 System Call

Consider the scenario where you are coding the lab. You have thought for a while without operating on your editor. After a key is pressed on keyboard, it displays in your screen. You can view this procedure as the combination of executing 'getchar' and 'printf'. What are the system calls the two functions finally issue? Can you describe how control flow switches between user mode and kernel mode in this procedure?

SOLUTION:

The 'getchar' function finally issues the 'read' system call. The 'printf' function finally issues the 'write' system call.

At first, the input monitoring thread calles 'getchar' in user mode, and then it switches to kernel mode to read user input. Since no input is received(without operating), the thread is put to sleep by kernel, and another thread is scheduled. (kernel mode to user mode) After a key is pressed on keyboard, an interrupt is sent and control switches to kernel mode. The monitoring thread is woken up, and gets the input char. (kernel mode to user mode) Then, the monitoring thread calles "printf" in user mode, and control switches to kernel mode. Kernel writes the char through graphics driver, then you can see it on your screen.

3 Waitpid and Zombie

Assume we have the following code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  int main(void) {
8    if (fork() == 0) {
9      if (fork() == 0) { // Proc 1
10       exit(0);
11     } else { // Proc 2
12       waitpid(-1, NULL, 0);
13       exit(0);
14     }
15   } else { // Proc 3
16     while (waitpid(-1, NULL,
17                    WNOHANG) > 0) ;
18     exit(0);
19   }
20   return 0;
21 }
```

3.1

How many zombie processes could the program leave for init to reap? Please show all possible number and show one execution order for each. (Assume that shell will not reap the main process)

Since `Proc 2` will wait for `Proc 1` to finish, `Proc 1` will not be reaped by init. Thus, we only consider the possible interleaving between `Proc 2` and `Proc 3`. There are two possible results:

1. When `Proc 2` exit before `Proc 3` calling `waitpid`, only the `Proc 3` will be reaped by init.
2. When `Proc 3` call `waitpid` before `Proc 2` calling `exit`, both `Proc 3` and `Proc 2` will be reaped by init.

3.2

What if we remove the line 12?

There are two possible results:

1. `Proc 1` and `Proc 3` will be reaped by init.
2. All three processes will be reaped by init.