

ICS EXE 2 Solution

March 10, 2021

1 SEQ Y86-64

`call` and `ret` instructions access stack to save and load return addresses. Please consider a different set of these instructions:

- `callr Dest`, which saves the return address in `%rsi` and jumps to the instruction at `Dest`;
- `retr`, which jumps to the instruction with the address saved in `%rsi`.

1.1

Please fill in the blanks below to describe these two instructions which are executed in sequential Y86-64 implementation. You only need to write the generic logic for each instruction. (Note that you are allowed to add new wires in the base SEQ Y86-64 hardware architecture.)

Stage	Generic <code>callr Dest</code>	Generic <code>retr</code>
	<code>icode:ifun <- M1[PC]</code> <code>valC <- M8[PC+1]</code> <code>valP <- PC+9</code>	<code>icode:ifun <- M1[PC]</code> <code>valP <- PC+1</code>
Fetch		
Decode		<code>valA <- R[%rsi]</code>
Execute		
Memory		
Write back	<code>R[%rsi] <- valP</code>	
PC update	<code>PC <- valC</code>	<code>PC <- valA</code>

1.2

For any correctly implemented Y86-64 program, if we simply replace all `call` instructions with `callr` and all `ret` with `retr`, will the program run correctly? Why or why not?

No. If function A calls function B, and then B calls C, address from C to B, overwritten the return address from B to A

2 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 }
```

2.1

Give a possible output of the program

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

2.2

Based on Q2.1, 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.

2.3

Based on Q2.1, please list all possible output order. (or show the constraints on the output order).

20 always before 40, 20 always before 22

2.4

Based on Q2.1, 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
```

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)

Solution:

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.