Exec variants syntax

```
char *const argv[] = {"/bin/ls",“ls”,"-l", NULL};
execv(argv[0], argv);
```

Note: arguments are stored in an array which is what you passed statically in the call with execl.V stands for vector

Path needs to be specified which is the first arguments and the rest are treated as arguments to the command NULL terminaed.

```
char *const cmd[] = {"ls",“ls”,"-l", NULL};
execvp(cmd[0], cmd);
```

Combines the features of both v and p calls.

```
char *args[] = {"ls", "-aF", "/", 0};
char *env[] = { 0 }; /* leave the environment list null */
execve("/bin/ls", args, env);
```

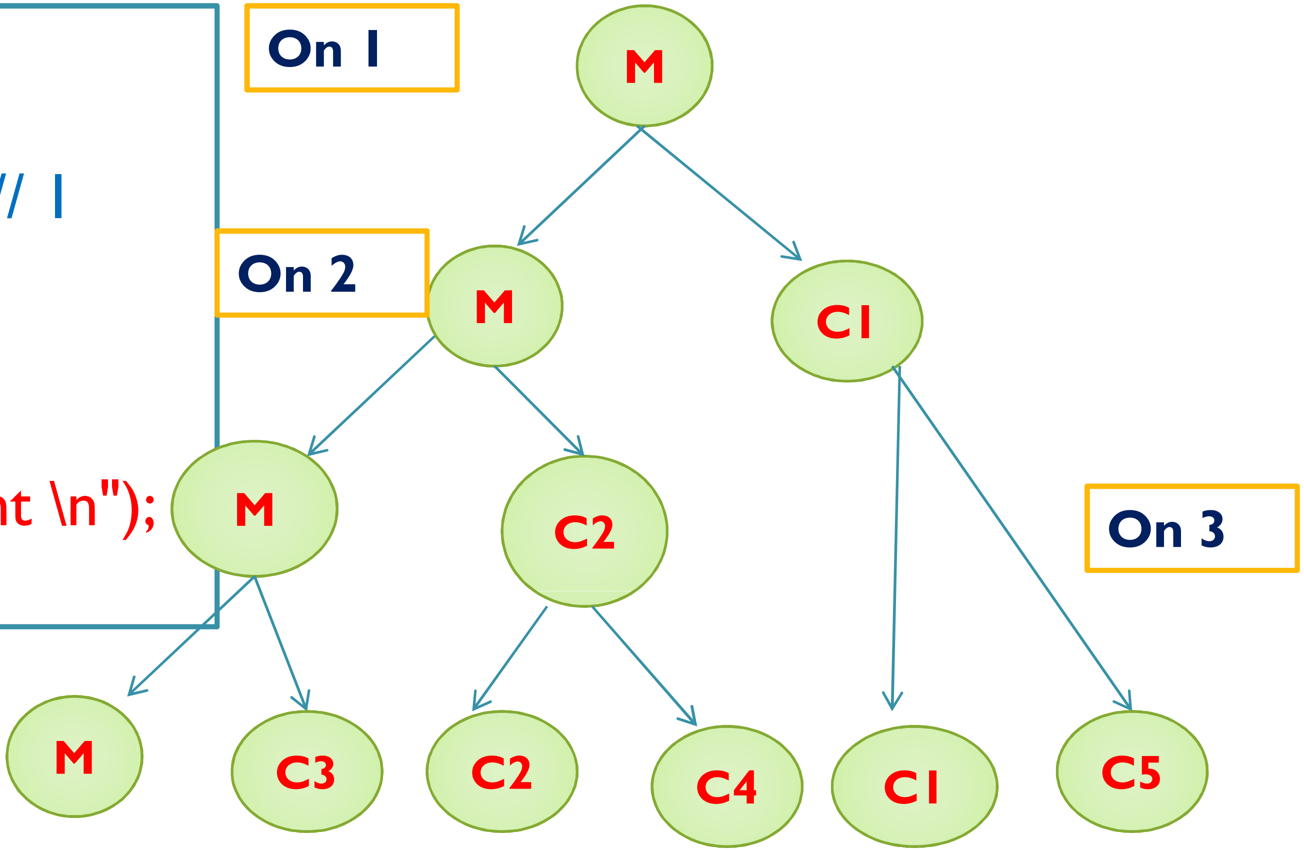
```
int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char
   *)0 };
ret = execve ("/bin/ls", cmd, env);
```

These are some good online resources on exec calls and its variants explainataion

- https://linuxhint.com/exec_linux_system_call_c/
- https://www.cs.rutgers.edu/~pxk/416/notes/c-tutorials/exec.html
- https://www.oreilly.com/library/view/secure-programming-cookbook/0596003943/ch01s07.html

A few more examples of Fork()

```
 int main()
{
pid=fork(); // 1
if (pid!=0)
fork();   // 2
fork(); // 3
printf("Count \n");
}
```



Totally 6 processes with respective parent child hierarchy as illustrated in the tree diagram

```
int main()
{
   fork();  // A
   fork() && fork() || fork();
   //  B          C            D
   fork();   //E
   printf(“ OS 2020\n");
   return 0;
}
```

✓And operator has higher precedence than OR (logical meanings here)

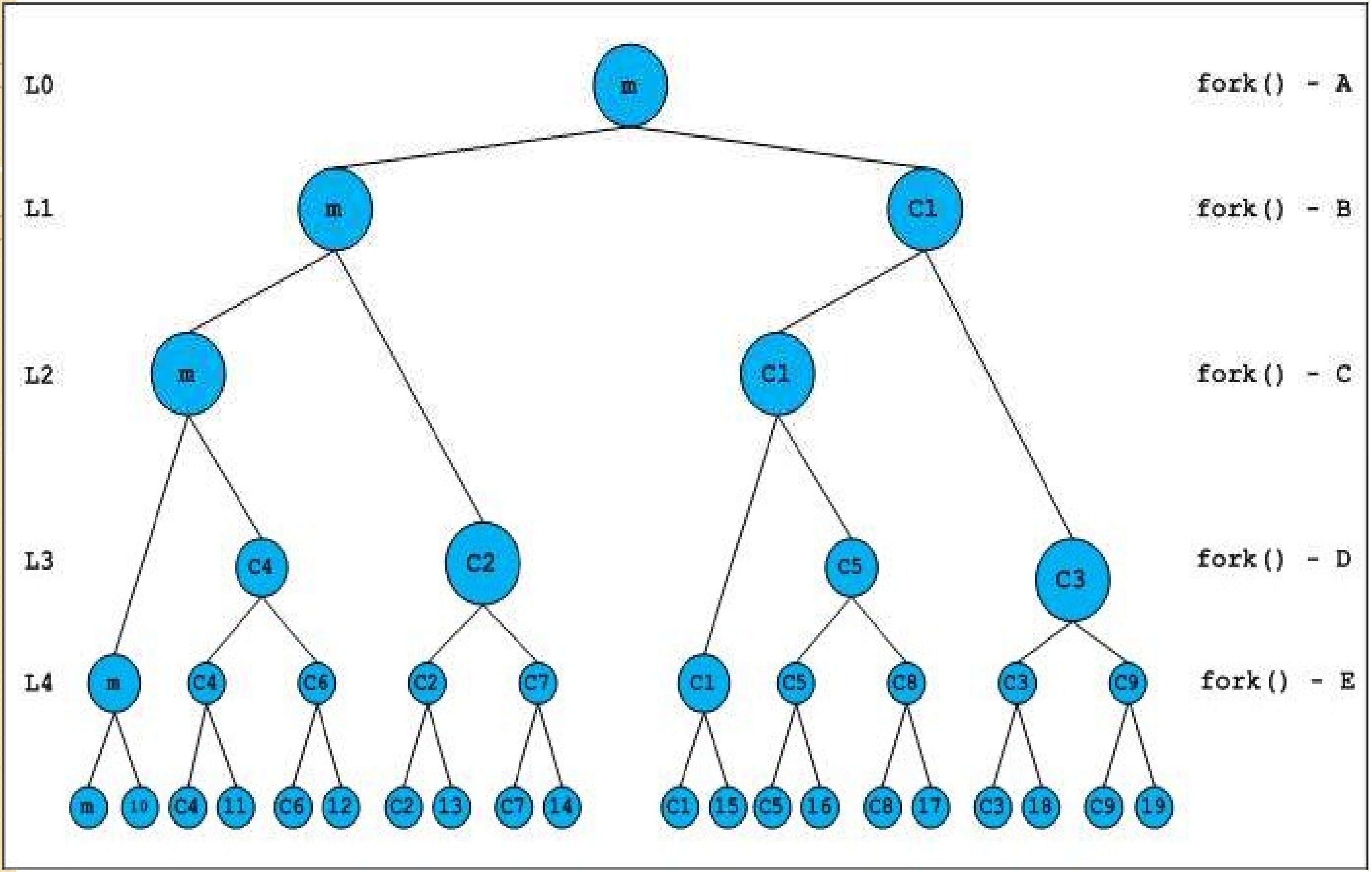
✓Associativity is Left to Right

✓&& - right operand evaluated only if left evaluates to true

✓|| - right evaluated only if left evaluates to false

✓This is referred to as Short Circuited Implementation of AND, OR operators in C (efficient circuit realization)

These properties would be used in the accompanying fork trace!


L0
m
fork() - A
L1
m
C1
fork() - B
L2
m
C1
fork() - C
L3
C4
C2
C5
C3
fork() - D
L4
m
C4
C6
C2
C7
C1
C5
C8
C3
C9
fork() - E
m
10
C4
11
C6
12
C2
13
C7
14
C1
15
C5
16
C8
17
C3
18
C9
19

- **Exercise 1 (Quiz Time) Exercise 2**

Identify the no of processes in each case

```
int main()
{
if(fork()&& fork()){
fork();}
if(fork()||fork()){
fork();
fork();}
printf("OS Quiz 1 \n");
}
```

```
if(fork() && fork())
      fork();
if(fork() || fork())
       fork();
    printf("hello")
```

Three six and two zero!

Code 1

```
int main()
{
printf(“OS \n");
fork();
fork();
fork();
}
```

```
int main()
{
printf(“OS ");
fork();
fork();
fork();
}
```

Code 2

❑In Code 1 Message OS is displayed only once and there are totally 8 processes created by the code

❑In Code 2 Message OS would also be displayed 8 times as the output buffer unflushed in the parent process on forking would be inherited by the child process as well.

❑Unflushed printf statements in parent process is equivalent to having those printf as part of the child process as well.

- Quiz Time – Give the Output of the following code

```
int main ()
{
printf("This will be printed ?.\n");
fork();
printf("This will be printed ?.\n");
fork();
printf("This will be printed ? .\n");
fork();
printf("This will be printed ?\n");
return 0;
}
```

- Quiz Time

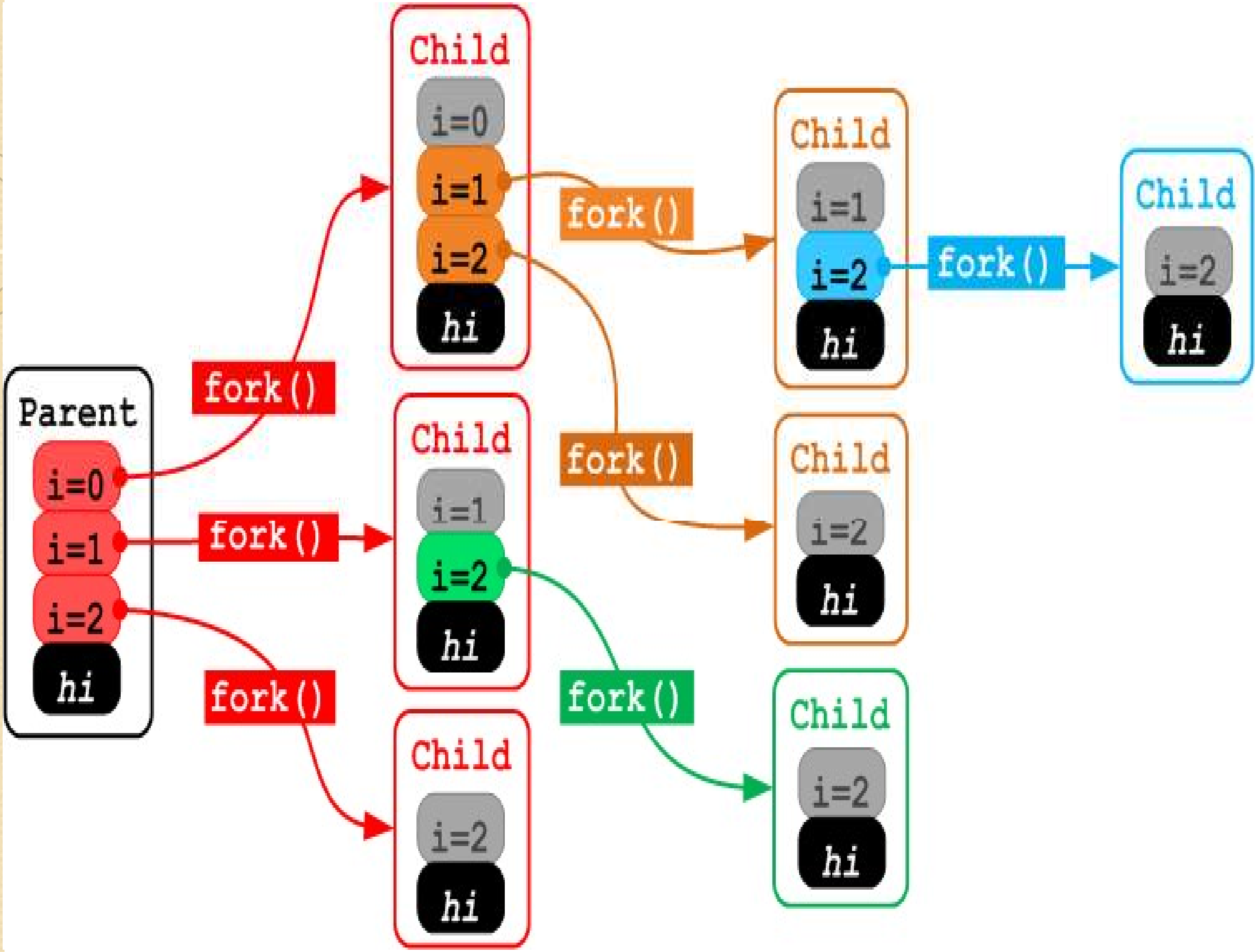
```
int main ()
{
printf(“A”);
fork();
printf(“B”);
return 0;
}
```

```
int main ()
{
printf(“A \n”);
fork();
printf(“B\n”);
return 0;
}
```

```
int main()
{
for (i=0; i<3; i++)
{
fork();
printf(“i=%d”,i);
printf(“Hello \n”);
return 0;
}
```

❑ i=0 displayed twice
(two processes at that instant)
❑i=1 displayed four times (4 processes)
❑i=8 is displayed 8 times (8 processes)
❑Hello Message is displayed 8 times

1. Loop starts in parent, i == 0
2. Parent fork()s, creating child 1.
3. You now have two processes. Both print i=0.
4. Loop restarts in both processes, now i == 1.
5. Parent and child 1 fork(), creating children 2 and 3.
6. You now have four processes. All four print i=1.
7. Loop restarts in all four processes, now i == 2.
8. Parent and children 1 through 3 all fork(), creating children 4 through 7.
9. You now have eight processes. All eight print i=2.
10. Loop restarts in all eight processes, now i == 3.
11. Loop terminates in all eight processes, as i < 3 is no longer true.
12. All eight processes print hi.
13. All eight processes terminate.


Parent
i=0
i=1
i=2
hi
fork()
fork()
fork()
Child
i=0
i=1
i=2
hi
Child
i=1
i=2
hi
Child
i=2
hi
fork()
fork()
fork()
Child
i=1
i=2
hi
Child
i=2
hi
Child
i=2
hi
fork()
Child
i=2
hi

- Quiz Time

```
int main ()
{
int i;
for (i=1;i<=3;i++)
{
fork();
printf(“ * \n”);
 return 0;
}
```

```
int main ()
{
int i;
for (i=1;i<=3;i++)
{
fork();
printf(“ * ”);
 return 0;
}
```

- How many times * will be displayed in both the codes

- Quiz Time Give the Outputs

```
int main ()
{
for(i=0;i<2; i++)
{
if (fork()==0)
printf(“ OS 2020 ”);
}
 return 0;
}
```

```
 int main ()
 {
for(i=0;i<2; i++)
{
if (fork()==0)
printf(“ OS 2020 \n ”);
}
 return 0;
 }
```

- How many times the message will be displayed / total count of processes ??

Quiz Time Give the Outputs

```
int main ()
{
pid=fork();
if (pid!=0)
fork();
fork();
printf("Count \n");
 return 0;
}
```

Express the following in a process tree setup and also write the C code for the same setup

1 forks 2 and 3

2 forks 4 5 and 6

3 forks 7

4 forks 8

5 forks 9

- How many times the message will be displayed / total count of processes ??