Fork command is used to create an exact copy of a parent process in UNIX-based operating system, which has the same code, data space, and stack. It takes no parameter, presented as ‘void’ in the parentheses, and has 3 different types of return value. Unlike normal functions, it returns twice, each in the parent process and the child process. When it returns -1, it means that the copy has failed, with no subprocess generated. When it returns 0, it means that the copy has been successfully created and the value will only be return in the child process. When it returns a process ID, it means that the copy is made, and the value will only be returned in the father process. The child process is independent from its parent, and has a function ‘getppid’ to get the parent’s process ID. Each child process can only have one parent, while a parent can have multiple children. The copy is based on Copy-On-Write mechanism, which means when the copy happens, the memory page table will be the one copied instead of certain memory pages. After the copy, all the memory pages will be considered read only. If anything is modified afterwards, the system will create another read-only memory page and link it to the table. In fact, the actual copy here is delayed until the real modification happens, but it does save some CPU time.

After calling the function ‘fork()’, the system will create a subprocess immediately and assign resources for the new process, such as the code and the data stored inside. Therefore, all the variables already defined in parent process will remain the same in the child process. Then the fork function will return twice, one is the child process ID and the other is 0. The order of two values returned is determined by the operating system, thus it may have different results if we run the program multiple times. As soon as the fork function is called, both the parent process and the child process will continue running its code after the ‘fork()’ line. If we add an if statement here to examine the return value of ‘fork()’, we will run the parent code since we have the return value 0 in the parent process, and we will run the child code as we have the return value larger than 0(the child process ID). This structure allows the parent process and the child process to carry out different tasks independently, thus changing any variable after calling fork() will not affect the original one. Assume we have one CPU, when ‘fork()’ is called, the current process is still in running state and execute the copy task in a new thread. After the fork function returns, the new process can be considered created. At this moment, different operating system will have different mechanism of running order, therefore the order is not fixed. When one is running, the other will be blocked, vice versa. It is also possible that both of them is blocked and wait to be executed.

Source:

1. Advanced Programming in the UNIX
2. Fork (system call) - Wikipedia

<https://en.wikipedia.org/wiki/Fork_(system_call)>

1. Class video in CS Bridge
2. Unix/Linux fork()函数的理解 - CSDN博客

<https://blog.csdn.net/ruglcc/article/details/7844897>

1. The fork() System Call

http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html