**Process Creation**

1. **Compile and run Sample Program 1, how many lines are printed by the program?**

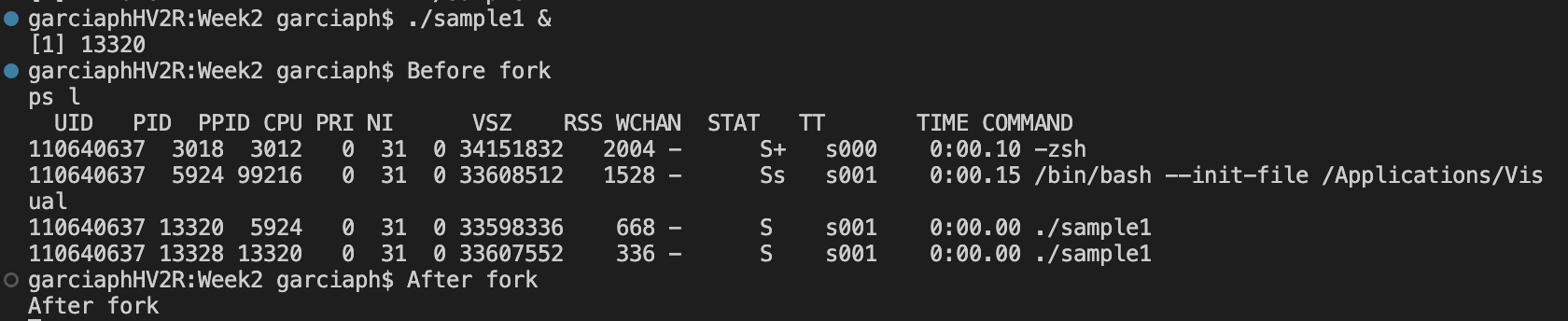
3 lines are printed by the program 

1. ***Describe what is happening to produce the output observed***

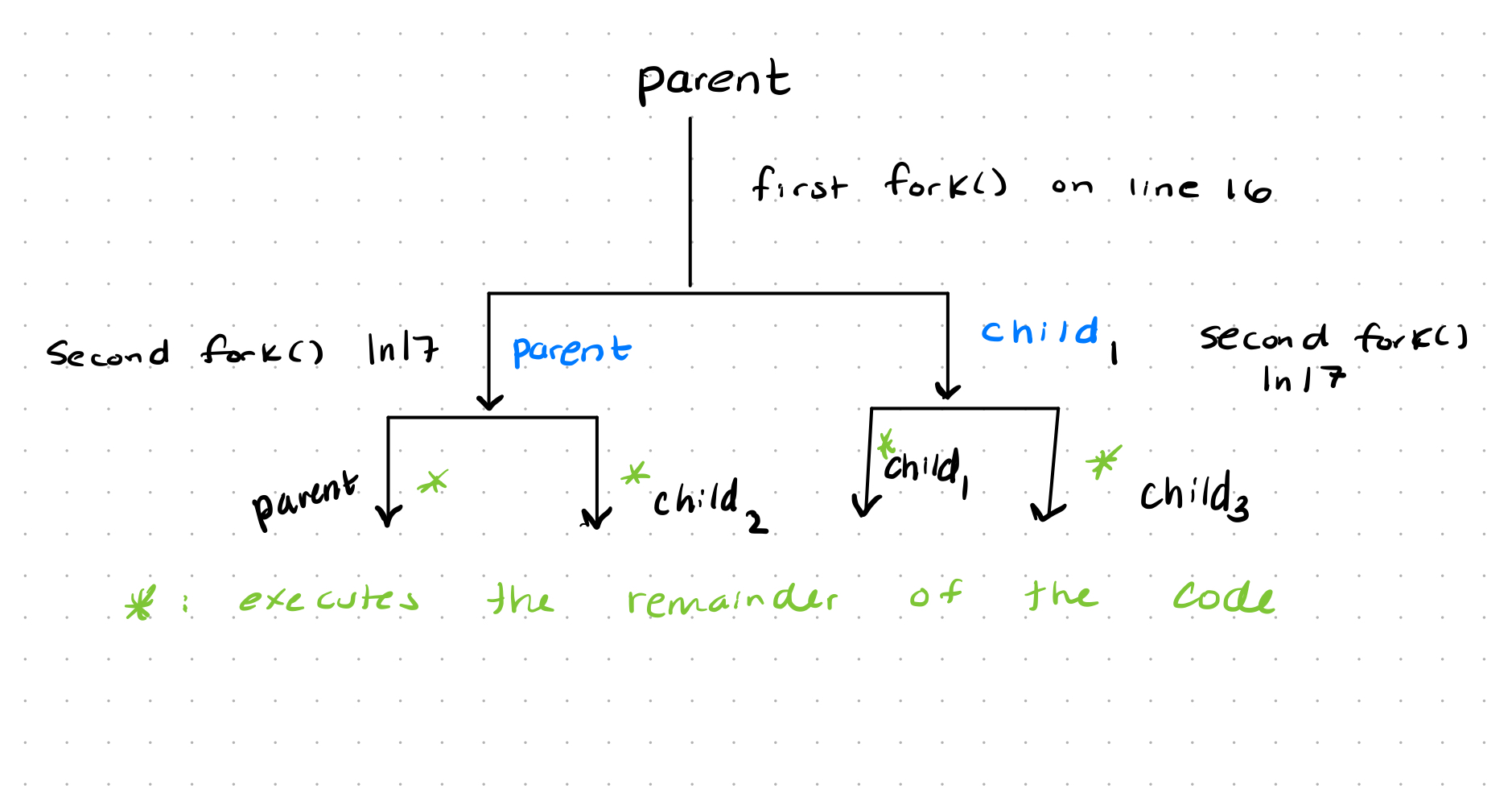
The *fork()* system call creates a new process (child process) which is an exact copy of the calling process (parent process). In this program, the parent creates the child process with *fork()* before the print statement, hence it prints twice.

1. **List the Process ID (PID) and Status (STAT):**
   1. S: Marks a process that is sleeping for less than about 20 seconds.
   2. s: The process is a session leader. When the process is created, it becomes a member of the session of its parent. The session ID of a session equals the process ID of the first member of the session, called the session leader.
   3. +: The process is in the foreground process group of its control terminal.

The PID for the parent process is 13320 and the child is 13328. The status of the parent and child is S, meaning it is sleeping for less than 20 seconds. We observed that both process IDs are close to one another and they both have the same status as they both do the same thing.



1. **Create a diagram illustrating how sampleProgramTwo executes**



1. **In the context of process state, process operations, and especially *process scheduling*, describe what you observed and try to explain what is happening to produce the observed results**

The parent process executes up to the first *fork()* system call, however it is uncertain if the parent process or its child processes will be scheduled to process next. Therefore, it is uncertain which process will execute next.

Our overall observation is that Sample programTwo is created as a new process and then starts running, the process executes until the *fork()* statement where a new child process is created and begins running. The parent process along with the newly created child process both continue running to the next *fork()* statement on the next line, where each process create a new child process of their own. One for the original parent and its first child. Overall 4 processes have been created and are now running each of these processes will then finish and terminate.

1. **Provide the exact line of code that you inserted for the wait() system call**

child = wait(&status);

1. **Which prints first, the child or the parent? Why?**

The child prints “I am Child PID1753031” before the parent. This is because the way the code is set up there is a wait statement that specifically is waiting for the child to finish and terminate before the parent can continue executing and get to the print statement.

**8**. **Describe the interaction between the exit() function and the wait() system  
call.**

The *exit()* function terminates the process with an error code. The error code is determined by the parameter taken by the exit function, for example *exit(0)* terminates the process with a successful code. The wait system call is a call to the parent process to wait until one of its child processes exits or a signal is received. After the child process terminates, the parent continues its execution after the *wait()* system call instruction.

**9**. **When is the second print line ("After the exec") printed? Explain your answer.**

The second print line *After the exec* will not print. The *execvp() will return a -1* if it fails to execute the commands. In the provided code, the second conditional statement handles the -1, so the second print statement is never reached as the body of the conditional statement has an *exit(1)* command at the end of it.

To verify this, the second conditional statement was removed, and the program was rerun. In this circumstance, the second print statement did execute.

**10**. **Explain how the second argument passed to execvp() is used (3pts)**

This function execvp() takes two arguments to the file name associated with the command being executed, for example, “ls”. The second argument is an array of pointer chars. The array contains three items: the first item in the array is a string of the command being run, for example, “ls”. The second item in the array is the arguments/options that go along with the command, for example, “-la”. Lastly, the third item is a NULL string to terminate/enter the command.

A Full Example:

const char \*args] = { "vim", "home/sampleProgram3.c", NULL};

execvp("vim", args);