fork()

The UNIX "Fork" command is an integral system call for managing processes. Namely, fork is how we can create instances of a running program, using the initial process. In fact, fork is the only way to create a new process in POSIX.

The first step is to call fork, which essentially creates a duplicate of the original parent process, called the child process. This includes the file descriptors, the registers, and several other variables. Although the child process has its own PID, we can use fork's return value to determine whether the process is a child. If we were to call fork() on the parent process, it will return its process ID number, but if we were to do the same with a child process, we would instead get 0.

Following the creation of the child process the parent process executes a waitpid system call, which allows it to wait for the child process to terminate, or finish. When waitpid is completed, the second parameter, statloc, is set to the child process' exit status.

Typically a child process is called to perform a task that differs from the parent, and the way that is done is with the use of the exec system call. The exec call runs the intended program by replacing the current process' core image. Once the child process is complete, we exit, which the process should use when completing an execution, returning an exit status represented by an integer between 0 and 255.

Using the 7 state process model we can see that when fork is called, the parent process is calls waitpid and is blocked to wait for the completion of the child process, which is in the new state. I would expect the parent process to be blocked rather than suspended since it calls the waitpid system call, and is still relevant upon the child's termination. I would also expect the child process to be in a new state, because all processes begin in this state until all necessary data is collected and initialized. Once the child process is initialized and ready, it awaits an available processor to be dispatched and ran. The child is then in the running state. If the child process times out, it returns to the ready state. If it makes a blocking system call, the process is blocked until the event is complete, returning to the ready state. Assuming the child process is able to run and complete execution, it reaches reach the exit state and returns an exit status. From there the parent process will return to its ready state if it has not finished.