**Problem 2**

The size of the process tree we selected is not to exceed 100 entries. This constraint exists because we used static memory allocation for the entryList array, which holds all of the entries in the target file.

The order of appearance of the start and termination messages from the process depends on where wait() is called. If wait() is called after the read() instruction in the parent process, then the parent process prints its start statement first, then the child process prints its start statement. After the child process has terminated, the “done waiting” statement is printed, indicating the end of the current child process. The loop reiterates and the parent process continues through the rest of the current node’s children. This call is recursive, so at each depth of the tree, the parent process creates child processes for each child node of the current parent node. For each child node, if the node has children, forkAndSleep() is called again with the current node as the parent. If the node has no children, then it is a leaf node and simply writes its information to its parent using a pipe.

If wait() is called after the else clause containing the parent process, then both the parent and the child call wait(). Since the children have nothing to wait on, the wait() has no effect in the child processes. However, in the parent process, the second print statement now comes before the wait(), so the parent may print its “done waiting” statement before the child prints its start statement, or the child may print its start statement before the parent prints its “done waiting” statement. It is not possible to predict which one will print first because both processes are running concurrently and we cannot directly control the scheduler.

If wait() is called outside of the loop that goes through all of the children of the root, then we can see even more haphazard print statements. The only things guaranteed are that the first parent will print its start statement first and the final tree result will print last. This occurs because wait() is called at most once per call to forkAndSleep(). As a result, not all parent processes wait for their children processes. This creates orphans and zombie orphans, which is an undesirable result.

Finally, if wait() is not called, the order of start and termination messages is as haphazard as the previous case. The difference is that there may be more orphans and zombie orphans.