Please write some code for this problem. Any kind of pseudocode is fine, as long as its meaning is clear. You may assume common primitives such as hash tables and linked lists: you do not need to implement them.

This problem deals with a process manager for a multi-programmed operating system kernel. The problem is to implement three key methods of a `Process` class that maintains parent/child relationships among processes, and coordinates the kernel API calls that govern creation and destruction of processes (in Unix these are the system calls named `fork`, `exit`, and `wait`).

- `p->Birth(Process* parent)` registers this newly created process `p` as a child of its parent.
- `p->Death(int status)` indicates that this process `p` has exited with the specified exit status.
- `int status = child->Join()` waits for this child process to exit (i.e., to call `Death`), and returns the child’s exit status.

Each `Process` object `P` has an associated thread. The thread bound to `P` calls `Birth` and `Death` on `P` as part of the implementation of the system calls to create and destroy processes respectively (e.g., `fork` and `exit`). A thread bound to `P` may call `Join` only on a child of `P`; `Join` is used to implement the `wait` system call. You do not need to create these threads or enforce these restrictions; they are intended to simplify the problem.

`Death` on `P` waits until two conditions are satisfied: (1) the children of `P` have exited, and (2) the parent of `P` has exited or has executed a `Join` on `P`. The intent is that the `Process` object for `P` may be safely deleted after `Death` returns.

Your solution should represent the process tree with the following state for each `Process` object `P`: (1) a list of the children of `P`, (2) a pointer to the parent of `P`, and (3) `P`’s exit status if `P` has exited.