Process Management Lab: Exploring fork()

Objective: This lab is designed to give you a hands-on understanding of how operating systems manage processes. Using the `fork.py` simulator, you will explore process creation (`fork`), termination (`exit`), and the structure of process trees.

Activity 1: Predicting Process Evolution

Goal: To understand how a process tree changes one step at a time.

Your Task:

1. Run the simulator with a fixed sequence of actions using a specific random seed.

   ```
   ./fork.py -s 10
   ```

2. Look at the list of actions (e.g., `FORK a b`, `EXIT b`). Before moving to the next action, **pause and draw what you think the process tree will look like.**
3. Press Enter to reveal the next step. Was your prediction correct? Repeat this for every action in the list.
4. Use the `-c` flag to check your final answer automatically.

   ```
   ./fork.py -s 10 -c
   ```

5. Repeat this exercise with different seeds (`-s 11`, `-s 12`) or more actions (`-a 15`) to master the concept.

Activity 2: The Impact of Fork Probability

Goal: To explore how the *rate* of process creation affects the overall shape of the process tree.

Your Task:

1. **Hypothesize:** What do you think a process tree will look like after 100 actions if new processes are created very rarely? What if they are created very frequently?
2. **Experiment:** Test your hypothesis by running the simulation with a large number of actions (`-a 100`) and varying the fork percentage (`-f`).
   - **Low Fork Rate:**

     ```
     ./fork.py -a 100 -f 0.1 -c
     ```

   - **High Fork Rate:**

     ```
     ./fork.py -a 100 -f 0.9 -c
     ```

3. **Analyze:** Compare the final trees. How does the fork percentage influence the final shape? Does it create a "deep" and "narrow" tree or a "short" and "wide" one?

Activity 3: Reverse Engineering the Actions

Goal: To deduce the actions taken by observing the changes in a process tree.

Your Task:

1. Run the simulator in "trace" mode (`-t`). It will show you the process tree before and after a change and ask you to identify the action.

   ```
   ./fork.py -t -s 2
   ```

2. Look at the "Tree Before" and "Tree After". What single action (`FORK` or `EXIT`) explains the difference? Type your answer and see if you are correct.
3. Continue this for all the steps to sharpen your ability to read and interpret process state changes.

Activity 4: Case Study on Orphaned Processes

Goal: To investigate what happens to a process's children if that parent process exits.

Your Task:

1. **Scenario:** We will create a specific chain of processes: `a` creates `b`, `b` creates `c`, and `c` creates `d` and `e`. Then, the middle process, `c`, will exit.
2. **Predict:** When `c` exits, what happens to its children, `d` and `e`? Who becomes their new parent? Draw the tree you expect to see.
3. **Simulate:** Run this exact scenario using the `-A` flag.

   ```
   ./fork.py -A a+b,b+c,c+d,c+e,c- -c
   ```

4. **Analyze:** Was your prediction correct? In most operating systems, an **orphaned process** is "adopted" by a special system process. The `-R` flag simulates this re-parenting. Try running the command with `-R` and observe the difference.

Activity 5: The Final State Challenge

Goal: To predict the final state of the process tree without seeing the intermediate steps.

Your Task:

1. Run the simulator with the `-F` flag. This shows you the complete list of actions at the beginning and asks you to determine only the *final* process tree.

   ```
   ./fork.py -F -s 15
   ```

2. Carefully trace the entire sequence of `FORK` and `EXIT` actions on paper.
3. Draw the single, final process tree that results from all actions. Use the `-c` flag to check if your final drawing is correct.

Activity 6: The Ambiguity Puzzle

Goal: To determine if a final process tree can be generated by different sequences of actions.

Your Task:

1. Run the simulator with both the `-t` and `-F` flags.

   ```
   ./fork.py -t -F -s 5
   ```

2. The simulator will show you a final process tree and ask you to provide the sequence of actions that created it.
3. **Think Critically:** Is there only one possible correct answer? For example, if process `a` creates two children, `b` and `c`, does the order matter for the final tree?
4. Try to find a final tree where the sequence of actions that created it is **ambiguous**. When can you tell the exact order of events, and when can't you?