Af

1) When fork() creates a new process, the parent and child initially share the same physical pages, but if either process writes to a page, the kernel uses copy-on-write to create a private physical copy of the page for the writer.
2) Unused pipe ends must be closed to manage data efficiently and signal the end of a file (EOF). The reading process uses the closing of the write-end as a signal that no more data is coming, and the writing process waits to receive a signal that the read-end is closed, meaning the reader is gone. If you forget to close them, it could cause a deadlock, where the unclosed ends block reads (never seeing the end of the file), and cause the program to seem like it never finishes.
3) A zombie process is a child that has completed execution but still has an entry in the process table because its parent has not received its exit status. Wait() prevents them by collecting the status and removing the child's process table entry. If zombie processes accumulate, they could exhaust all process table entries, preventing new processes from being created and degrading the system's usability.
4) The kernel tracks parent-child relationships by having a parent PID and maintaining lists for a parent's children. If a parent dies before its children, the children become orphans and get assigned to a process (usually PID 1) by the kernel that reaps them when they exit.

**Output**

```
Lab 3: Basic Process Management (fork and wait)
Focus: Understanding process creation and cleanup

=== Process Management Lab ===
1. Basic producer-consumer demo
2. Multiple producer-consumer pairs
3. Exit
Choose an option (1-3): 1

Starting basic producer-consumer demonstration...

Parent process (PID: 17372) creating children...
Created producer child (PID: 17373)
Producer (PID: 17373) starting...
Producer: Sent number 1
Created consumer child (PID: 17374)
Consumer (PID: 17374) starting...
Consumer: Received 1, running sum: 1
Producer: Sent number 2
Consumer: Received 2, running sum: 3
Producer: Sent number 3
Consumer: Received 3, running sum: 6
Producer: Sent number 4
Consumer: Received 4, running sum: 10
Producer: Sent number 5
Consumer: Received 5, running sum: 15
Producer: Finished sending 5 numbers
Consumer: Final sum: 15
Producer child (PID: 17373) exited with status 0
Consumer child (PID: 17374) exited with status 0

SUCCESS: Basic producer-consumer completed!
```

```
Choose an option (1-3): 2

Running multiple producer-consumer pairs...

Parent creating 2 producer-consumer pairs...

=== Pair 1 ===
Producer (PID: 17376) starting...
Producer: Sent number 1
Consumer (PID: 17377) starting...
Consumer: Received 1, running sum: 1
Producer: Sent number 2
Consumer: Received 2, running sum: 3
Producer: Sent number 3
Consumer: Received 3, running sum: 6
Producer: Sent number 4
Consumer: Received 4, running sum: 10
Producer: Sent number 5
Consumer: Received 5, running sum: 15
Producer: Finished sending 5 numbers
Consumer: Final sum: 15

=== Pair 2 ===
Producer (PID: 17378) starting...
Producer: Sent number 6
Consumer (PID: 17379) starting...
Consumer: Received 6, running sum: 6
Producer: Sent number 7
Consumer: Received 7, running sum: 13
Producer: Sent number 8
Consumer: Received 8, running sum: 21
Producer: Sent number 9
Consumer: Received 9, running sum: 30
Producer: Sent number 10
Consumer: Received 10, running sum: 40
Producer: Finished sending 5 numbers
Consumer: Final sum: 40

All pairs completed successfully!
Child (PID: 17376) exited with status 0
Child (PID: 17377) exited with status 0
Child (PID: 17378) exited with status 0
Child (PID: 17379) exited with status 0

SUCCESS: Multiple pairs completed!
```