1. Zombie Process

Definition: A zombie process is a process that has completed execution but still has an entry in the process table. This happens because the process's parent has not yet read its exit status.

In Simple Terms: Imagine a child who finishes their homework and goes to bed, but they still have their homework lying around on the desk. The homework will remain there until the parent (you) picks it up and puts it away. The same thing happens with zombie processes: they are like homework waiting to be picked up by their parent.

Example:

1. **Process Creation:**
    - **Parent** (a manager) creates a **Child** (an employee).
    - **Parent** gives a task to **Child** and waits for **Child** to complete it.
2. **Child Completes Task:**
    - **Child** finishes the task and sends a completion message back to **Parent**.
    - **Child** goes away, but its task report is still on the manager’s desk.
3. **Parent’s Responsibility:**
    - **Parent** must pick up the report to officially close the task.
    - If **Parent** forgets to pick up the report, the report remains on the desk.

In this analogy:

- **Child** has finished its job but remains in the system until **Parent** acknowledges that the job is done.
- The report on the desk represents the **Zombie Process**—it’s not doing anything but is still there.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child process (PID: %d)\n", getpid());
        // Child does some work and exits
        exit(0);
    } else {
        // Parent process
        printf("Parent process (PID: %d)\n", getpid());
        // Parent sleeps to keep the child in a zombie state for observation
        sleep(10); // Check for the zombie state using 'ps aux | grep defunct'
    }
```

```
    return 0;
}
```

2. Orphan Process

Definition: An orphan process is a process whose parent has finished or terminated, while the orphan process is still running. The orphan process is then adopted by the `init` process (PID 1), which takes care of the cleanup.

In Simple Terms: Imagine a child who is **left alone** when their parents leave for work. The child will be looked after by a guardian (the `init` process) until the parents come back. Similarly, in computing, when a process's parent terminates, the `init` process adopts the orphan process.

Example:

1. **Process Creation:**
    - **Parent** (a manager) creates a **Child** (an employee).
2. **Parent Terminates:**
    - **Parent** leaves the office, but **Child** continues working.
3. **Orphan Process:**
    - **Child** is now an orphan process and will be managed by the `init` process.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child process (PID: %d)\n", getpid());
        printf("Child process sleeping to become an orphan...\n");
        sleep(10); // Sleep to observe that parent has terminated
        printf("Child process woke up and continues to run...\n");
    } else {
        // Parent process
        printf("Parent process (PID: %d)\n", getpid());
        printf("Parent process exiting...\n");
        exit(0);  // Parent exits before the child finishes
    }

    return 0;
}
```