Homework 2

Anhelina Bondarenko

Std: 220104004928

**1. Overview**

This code implements a daemon process that manages two child processes communicating via FIFOs. The daemon monitors child processes for timeouts and handles signals gracefully.

**2. Key Components**

**2.1 Global Definitions & Structures**

FIFOs: FIFO1 and FIFO2 are named pipes for inter-process communication.

Child Tracking:

ChildProcess struct stores PID, start time, and a timeout flag.

child\_table[] tracks up to MAX\_CHILDREN (10) child processes.

Signals:

SIGCHLD (child termination), SIGUSR1, SIGHUP, and SIGTERM (daemon termination) are handled.

CHILD\_TIMEOUT is set to 15 seconds.

**2.2 Signal Handlers**

SIGCHLD Handler (sigchld\_handler):

Logs child exits (normal or via signal) and marks them as terminated.

Uses waitpid(-1, &status, WNOHANG) to non-blockingly reap all exited children.

Increments child\_count to track completed processes.

Daemon Signal Handler (daemon\_signal\_handler):

Handles SIGUSR1, SIGHUP, and SIGTERM.

Logs signals and exits cleanly on SIGTERM.

**2.3 Timeout Mechanism (check\_timeouts):**

Iterates over child\_table to check elapsed time for each child.

If a child exceeds CHILD\_TIMEOUT:

Sends SIGTERM for graceful termination.

After 1 second, forces termination with SIGKILL if still running.

Marks the child as terminated and increments child\_count.

**Key Features:**

Prevents indefinite hangs by enforcing timeouts.

Uses waitpid(pid, NULL, 0) post-termination to avoid zombies.

**2.4 Daemon Initialization (become\_daemon):**

Double-fork to detach from the terminal.

Create a new session with setsid().

Redirect stdout/stderr to daemon\_log.txt and close unused file descriptors.

Handle signals (SIGUSR1, SIGHUP, SIGTERM).

**2.5 Child Processes**

Child 1 (child\_process1): Reads two integers from FIFO1, computes the larger value, and writes it to FIFO2.

Child 2 (child\_process2): Reads the result from FIFO2 and prints it. Includes a 10-second sleep to test timeout handling.

**2.6 Main Function workflow:**

Validate command-line arguments.

Daemonize the process.

Create FIFOs and set up signal handlers.

Fork child processes, adding them to child\_table.

Write input numbers to FIFO1.

Loop until all children exit, checking timeouts every 2 seconds.

Clean up FIFOs and exit.

**3. Timeout Mechanism**

**3.1 Implementation**

Tracking: Each child’s PID and start time are recorded in child\_table.

Periodic Checks: The main loop calls check\_timeouts() every 2 seconds.

Termination Logic:

Graceful: SIGTERM allows the child to clean up.

Forced: SIGKILL after 1 second if the child ignores SIGTERM.

Zombie Prevention: waitpid reaps terminated children.

**3.2 Race Condition Handling**

SIGCHLD Coordination: The handler marks children as terminated to avoid redundant checks.

Atomic Operations: sig\_atomic\_t ensures safe access to child\_count across signals and main loop.

**3.3 Edge Cases**

Late Exits: If a child exits naturally after a timeout check but before termination, the SIGCHLD handler marks it as completed.

Concurrent Timeouts: The loop ensures all children are checked, even if multiple timeouts occur.

4. **Function Summary**

|  |  |
| --- | --- |
| Function | Purpose |
| sigchld\_handler | Handles child termination signals and logs exits. |
| daemon\_signal\_handler | Processes daemon-specific signals (e.g., shutdown on SIGTERM). |
| check\_timeouts | Enforces child process timeouts and terminates stalled processes. |
| become\_daemon | Converts the process into a daemon with logging and signal handling. |
| child\_process1/2 | Perform computations and communicate via FIFOs. |
| main | Initializes the daemon, manages children, and coordinates cleanup. |

**Test Case: Successful Execution Within Timeout**

Зображення, що містить текст, знімок екрана, Шрифт

Вміст, створений ШІ, може бути неправильним.

**Test Scenario**

This test case validates the program's behavior when **both child processes complete execution within the 15-second timeout period**. The test ensures:

1. Proper inter-process communication via FIFOs.
2. Correct handling of child process exits.
3. No timeout enforcement when processes finish within the allowed time.

**Input Parameters**

* Parent (Daemon) Input: 9, 17 (via command line).
* Child 1 Sleep: 10 seconds (simulates work).
* Child 2 Sleep: 10 seconds (triggers simultaneous execution).

**Output**

1. **FIFO Setup**:
   * fifo1 and fifo2 created successfully.
   * Daemon starts with child PIDs 79095 (Child 1) and 79096 (Child 2).
2. **Child Process Execution**:
   * **Child 1** (79095):
     + Wakes after 10 seconds, reads 9 and 17 from fifo1, computes 17 as the larger value.
     + Writes result to fifo2 and exits with status 0.
   * **Child 2** (79096):
     + Wakes after 10 seconds, reads 17 from fifo2, prints the result, and exits with status 0.
3. **Termination Logs**:
   * Both children exit successfully at Mon Apr 7 20:24:32 2025
   * SIGCHLD handler logs their exits.
4. **Timeout Mechanism**:
   * **No timeout enforced**, as both processes completed in **10 seconds** (below the 15-second threshold).
   * The check\_timeouts() loop (Proceeding... messages) runs every 2 seconds but takes no action.
5. **Cleanup**:
   * FIFOs unlinked.
   * Daemon exits after confirming both children terminated.

**Test Case: Timeout Enforcement at 8 Seconds**

Зображення, що містить текст, знімок екрана, Шрифт

Вміст, створений ШІ, може бути неправильним.

**Test Scenario**

This test case validates the program's behavior when **both child processes exceed the 8-second timeout**. The test ensures:

1. Timeout enforcement for stalled child processes.
2. Proper cleanup after forced termination.
3. Signal handling during daemon shutdown.

**Input Parameters**

* **Timeout**: 8 seconds (modified from original 15-second setting).
* Child 1 & 2 Sleep: 10 seconds (intentionally exceeds timeout).

**Observed Output**

1. **FIFO Setup**:
   * fifo1 and fifo2 created successfully.
   * Daemon starts with child PIDs 79447 (Child 1) and 79448 (Child 2).
2. **Timeout Enforcement**:
   * After **8 seconds**, the daemon detects both children have exceeded the timeout:
     + Sends SIGTERM to both children.
     + Logs: Terminating child 79447 due to timeout and Terminating child 79448 due to timeout.
3. **Process Termination**:
   * Daemon logs SIGTERM received - exiting, indicating it processed termination signals for the children.
4. **Cleanup**:
   * Daemon proceeds to unlink FIFOs and exits.

**Conclusion**

This code demonstrates a daemon with inter-process communication and proactive timeout management. By combining signal handling, periodic checks, and graceful termination, it ensures reliable child process execution within defined constraints.