March 26th, 2025
Homework #2: Create Two Different Processes and Implement IPC Communication with Daemon Process

Implement a communication protocol between two processes using IPC and introduce a **daemon process** to handle background operations. Follow these steps:

Initial Setup

- The program should take **two integer arguments** from the command line.
- Define an integer variable named **“result”**. Assign its value as zero and perform the following steps sequentially.

Parent Process:

- Create a program that takes **two integer arguments**.
- Create two FIFOs (**named pipes**).
- Send the two integer values to the **first FIFO**.
- Send a command (e.g., **determining the larger number**) to the **second FIFO**.
- Use the **fork()** system call to create two child processes and assign each to a FIFO.
- **Convert the process into a daemon** to run in the background and handle logging operations.
- All child processes **sleep for 10 seconds**, execute their tasks, and then exit.
- Set a signal handler for **SIGCHLD** in the parent process to handle child process termination.
- Enter a loop, printing a message containing **"proceeding"** every two seconds.
- The signal handler should call **waitpid()** to reap the terminated child process, print out the process ID of the exited child, and increment a counter by two.
- When the counter reaches the number of children originally spawned, the program exits.
- The daemon should handle **SIGUSR1**, **SIGHUP**, and **SIGTERM** signals to manage process execution.

First Child Process (Child Process 1):

- Open the **first FIFO** and read the two integer values.
- Determine the larger of the two numbers.

- Write the larger number to the **second FIFO**.

Second Child Process (Child Process 2):

- Open the **second FIFO** and read the command (which number is larger).
- Print the **larger number** to the screen.

Daemon Process:

- The **daemon process** should handle logging operations by storing process execution details (e.g., start time, PID, termination status, and errors) in a log file.
- Redirect **stdout and stderr** to a log file for debugging.
- Monitor IPC communication and store any errors encountered.
- Handle **SIGTERM** and **SIGHUP** signals to allow graceful shutdown and reconfiguration.
- Implement a **timeout mechanism** that monitors inactive child processes and terminates them if necessary.
- Use **non-blocking FIFO reads** to avoid deadlocks and improve performance.

Bonus Section:

1. Implement a **zombie protection method** to earn **15 points**.
2. Print the **exit statuses** of all processes for an additional **15 points**.

Test Scenario:

Expected Results (In one scenario):

- The two integer values were correctly received and assigned successfully.
- FIFOs were created successfully, and data/command transmission occurred without errors.
- The numbers were correctly compared, and the larger value was identified.
- The **larger number** was correctly written to the **second FIFO**.
- The daemon process logged the execution details successfully.
- Child processes completed their tasks successfully and exited without errors.
- The parent process correctly managed the counter value and printed exit statuses for each child process.
- The printed value of the larger integer should be **correct**.

Error Scenarios:

- If **FIFOs** cannot be created or if there are errors in data/command transmission, appropriate error messages should be displayed.
- If **child processes** fail to complete their tasks successfully or encounter unexpected errors, error messages indicating the failure should be displayed.
- If the **daemon process** fails to initialize correctly, logs the errors, or does not respond to signals properly, it should display a failure message.
- If the counter value is not managed correctly or if exit statuses are not printed for each child process, error messages should be displayed indicating the issue.
- If a process becomes **unresponsive**, the daemon should detect and terminate it.

Grading Criteria:

Criteria	Points
FIFO communication works correctly	20
Correct child process execution	15
Daemon process properly implemented	20
Proper signal handling (SIGCHLD, SIGTERM, etc.)	15
Proper error handling and exit statuses	15
Report submission with test results	15

Additional Rules:

1. **No compilation:** -100 points
2. **Missing Makefile** or **Makefile without "make clean"**: -30 points
3. **Each memory leak that we encounter**: -20 points
4. **No report submitted**: -100 points (*You must test and demonstrate every step.*)
5. **Failure in one of the tasks**: -10 points
6. **No error control block used or failure in throwing a message**: -10 points
7. **Creating processes without using the fork() system call**: -20 points
8. **If one of the FIFOs does not exist**: -20 points
9. **Late submissions** → Not accepted

Deadline: April 8th at 23:59

Good Luck!