GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
DEPARTMENT

CSE344 Systems Programming
Spring 2025
Homework 2 Report

Osmancan Bozali
220104004011

Introduction:

This report documents the implementation the task required creating two different processes that communicate via Inter-Process Communication (IPC) using named pipes (FIFOs), with a daemon process managing background operations and logging. The program takes two integer arguments, compares them to find the larger number (assignments says commands but there is only larger command, in the code i implemented command system but it only has larger), and logs execution details while handling signals and errors appropriately. This report provides an overview of the solution, a detailed explanation of the code, and a conclusion based on the implementation and test results.

The solution uses C programming with system calls like fork(), mkfifo(), and signal handling to meet the requirements. It includes a parent process (also daemon), two child processes, all interacting through FIFOs.

Code Explanation:

Below is a detailed breakdown of each function and the main program flow.

Preprocessor Directives and Global Definitions

- **Includes**: The program includes standard libraries for I/O (stdio.h, stdlib.h), process management (unistd.h, sys/types.h, sys/wait.h), file operations (fcntl.h, sys/stat.h), signal handling (signal.h), string manipulation (string.h), error handling (errno.h), and time operations (time.h).
- **Macros**: Defines constants like FIFO1 and FIFO2 for FIFO names, LOG_FILE for the log file, TIMEOUT_SECONDS (15 seconds), and others for command codes and polling limits.
- **Global Variables**:
    - child_pids[MAX_CHILDREN]: Array to track child process IDs.
    - child_count: Number of active children.
    - completed_children: Counter for terminated children.
    - log_file: File pointer for logging.
    - result: Stores the larger number.
    - num1, num2: Input integers from command line.

Function: log_message(const char* msg)

- **Purpose**: Logs messages to results.log with timestamps.
- **Details**: Uses time() to get the current time, formats it with ctime(), and writes the message to the log file. Ensures the log is flushed immediately with fflush(). This is used throughout the program to record execution details, errors, and status updates.

Function: cleanup()

- **Purpose**: Cleans up resources before program exit.
- **Details**: Closes the log file if open and removes the FIFOs using unlink(). Logs a completion message. Called during normal exit or error scenarios to prevent resource leaks.

Function: register_child(pid_t pid)

- **Purpose**: Tracks child process IDs.
- **Details**: Adds a child PID to the child_pids array if space is available (child_count < MAX_CHILDREN). Logs an error if the limit is reached, ensuring no overflow occurs.

Function: sigchld_handler(int sig)

- **Purpose**: Handles SIGCHLD signals when children terminate.
- **Details**: Uses waitpid() with WNOHANG to reap terminated children non-blocking. Identifies the terminated child, logs its exit status (normal or signal-induced), removes it from child_pids, and increments completed_children. Prevents zombie processes (bonus requirement) by reaping children immediately.

Function: daemon_signal_handler(int sig)

- **Purpose**: Manages daemon-specific signals (SIGUSR1, SIGHUP, SIGTERM).
- **Details**:
    - **SIGTERM**: Logs termination, forwards the signal to all children, cleans up, and exits.
    - **SIGHUP**: Logs reconfiguration and forwards to children (reconfiguration logic is placeholder since I did not understand what to do by reconfiguration).
    - **SIGUSR1**: Reports active child count and PIDs.
    - Ensures graceful shutdown and communication with child processes.

Function: setup_daemon()

- **Purpose**: Converts the process into a daemon.
- **Details**: Forks and exits the parent, sets a new session with setsid(), clears the umask, closes standard file descriptors, and redirects them to the log file. Sets up signal handlers for SIGUSR1, SIGHUP, SIGTERM, and SIGCHLD. Logs completion, ensuring the daemon runs in the background and logs all output.

Function: poll_read(int fd, void* buffer, size_t size, const char* error_message)

- **Purpose**: Reads from a FIFO with polling to avoid blocking. I implemened this to use non-blocking communication. Otherwise child processes could not read from fifos with O_NONBLOCK.

- **Details**: Attempts to read up to MAX_POLL_ATTEMPTS times, waiting POLL_INTERVALmicroseconds between attempts. Returns bytes read on success, or -1 on error/timeout, logging issues. Prevents deadlocks by using non-blocking reads (O_NONBLOCK).

Function: setup_child_signal_handler()

- **Purpose**: Configures signal handling for child processes.
- **Details**: Sets default handlers (SIG_DFL) for SIGCHLD, SIGTERM, SIGHUP, and SIGUSR1, ensuring children respond appropriately to signals from the daemon or parent.

Function: main(int argc, char* argv[])

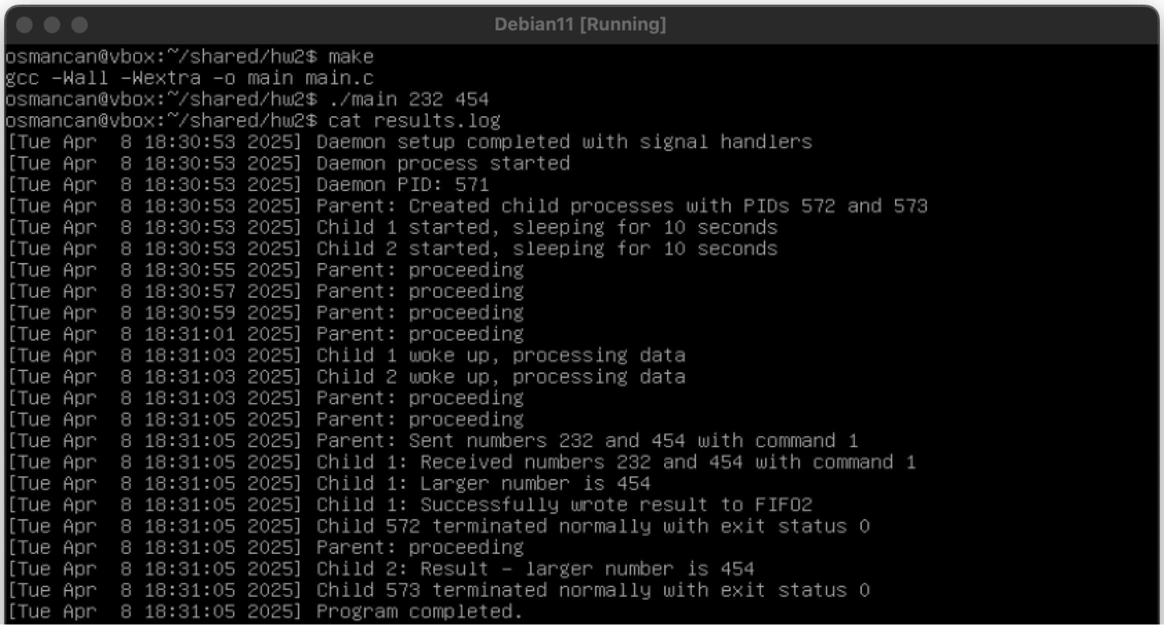
- **Purpose**: Orchestrates the entire program.
- **Details**:
    - **Argument Check**: Validates two integer arguments (argc == 3), assigns them to num1 and num2.
    - **FIFO Creation**: Creates FIFO1 and FIFO2 with mkfifo(), handling existing FIFOs (EEXIST).
    - **Daemon Setup**: Calls setup_daemon() to run in the background.
    - **Child Creation**:
        - **Child 1**: Forks, sleeps 10 seconds, reads num1, num2, and a command from FIFO1using poll_read(), determines the larger number, writes it to FIFO2, and exits.
        - **Child 2**: Forks, sleeps 10 seconds, reads the larger number from FIFO2, logs and stores it in result, and exits.
        - Registers both PIDs with register_child().
    - **Parent Logic**:
        - Waits 10 seconds, then writes num1, num2, and CMD_FIND_LARGER to FIFO1.
        - Loops every 2 seconds, logging "proceeding", until all children complete (completed_children == 2).
        - Implements a 15-second timeout, terminating children with SIGTERM if exceeded.
    - **Cleanup**: Calls cleanup() before exiting.

Tests and Screenshots:

In order to test the program yourself, you can use the makefile

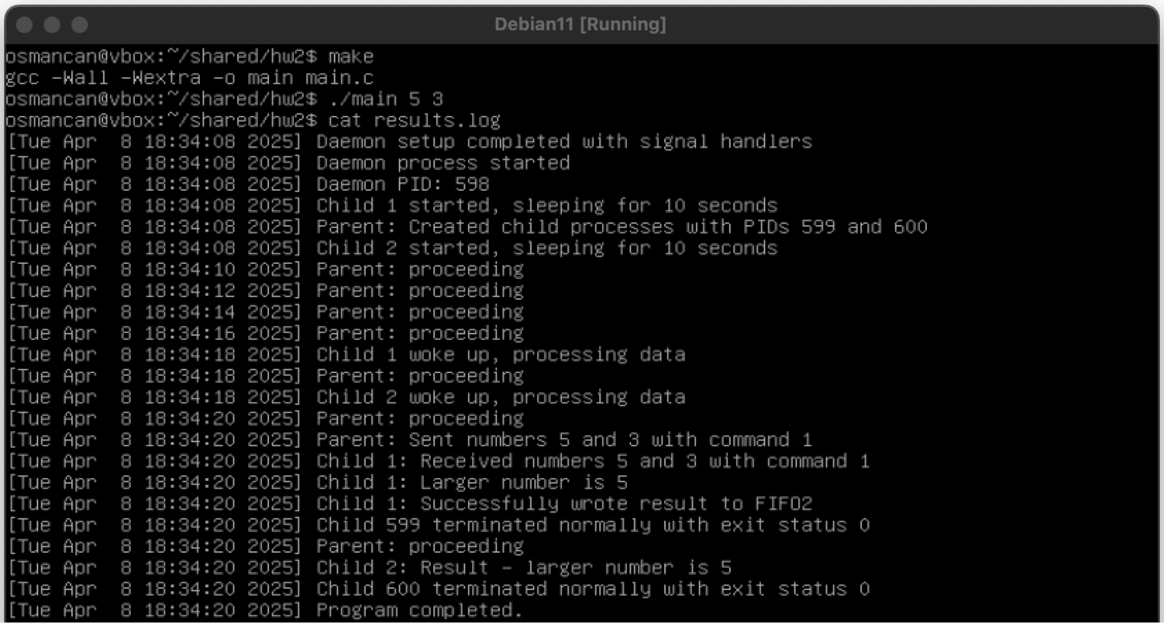
- make : This command compiles the program
- make clean : This command clears all the files/folders created while running the program)

Successful Scenario 1:



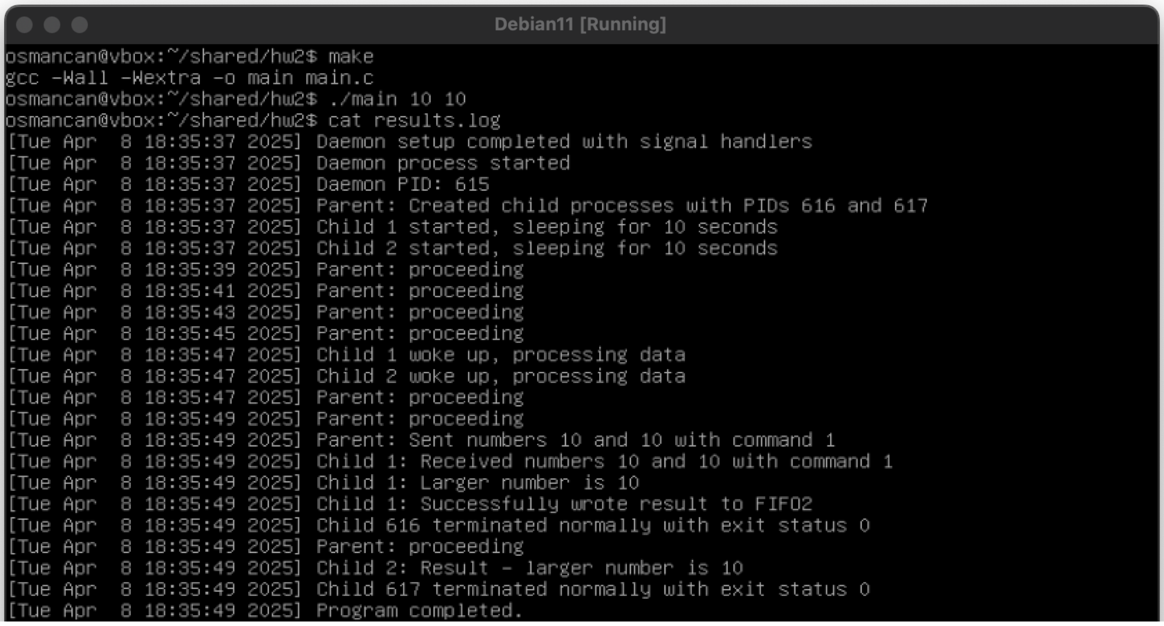
```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main 232 454
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:30:53 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:30:53 2025] Daemon process started
[Tue Apr  8 18:30:53 2025] Daemon PID: 571
[Tue Apr  8 18:30:53 2025] Parent: Created child processes with PIDs 572 and 573
[Tue Apr  8 18:30:53 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:30:53 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:30:55 2025] Parent: proceeding
[Tue Apr  8 18:30:57 2025] Parent: proceeding
[Tue Apr  8 18:30:59 2025] Parent: proceeding
[Tue Apr  8 18:31:01 2025] Parent: proceeding
[Tue Apr  8 18:31:03 2025] Child 1 woke up, processing data
[Tue Apr  8 18:31:03 2025] Child 2 woke up, processing data
[Tue Apr  8 18:31:03 2025] Parent: proceeding
[Tue Apr  8 18:31:05 2025] Parent: proceeding
[Tue Apr  8 18:31:05 2025] Parent: Sent numbers 232 and 454 with command 1
[Tue Apr  8 18:31:05 2025] Child 1: Received numbers 232 and 454 with command 1
[Tue Apr  8 18:31:05 2025] Child 1: Larger number is 454
[Tue Apr  8 18:31:05 2025] Child 1: Successfully wrote result to FIFO2
[Tue Apr  8 18:31:05 2025] Child 572 terminated normally with exit status 0
[Tue Apr  8 18:31:05 2025] Parent: proceeding
[Tue Apr  8 18:31:05 2025] Child 2: Result - larger number is 454
[Tue Apr  8 18:31:05 2025] Child 573 terminated normally with exit status 0
[Tue Apr  8 18:31:05 2025] Program completed.
```

Successful Scenario 2:



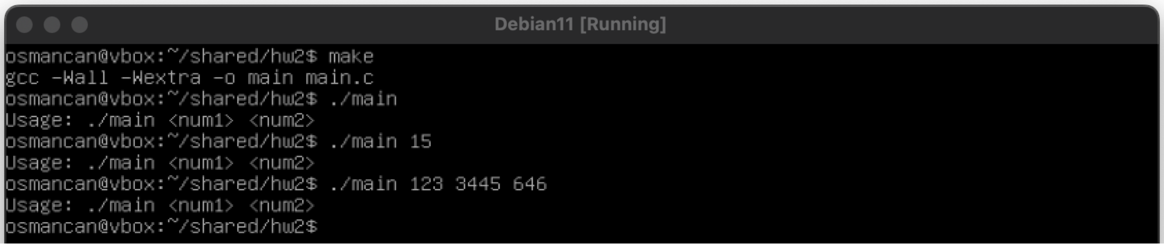
```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main 5 3
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:34:08 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:34:08 2025] Daemon process started
[Tue Apr  8 18:34:08 2025] Daemon PID: 598
[Tue Apr  8 18:34:08 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:34:08 2025] Parent: Created child processes with PIDs 599 and 600
[Tue Apr  8 18:34:08 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:34:10 2025] Parent: proceeding
[Tue Apr  8 18:34:12 2025] Parent: proceeding
[Tue Apr  8 18:34:14 2025] Parent: proceeding
[Tue Apr  8 18:34:16 2025] Parent: proceeding
[Tue Apr  8 18:34:18 2025] Child 1 woke up, processing data
[Tue Apr  8 18:34:18 2025] Parent: proceeding
[Tue Apr  8 18:34:18 2025] Child 2 woke up, processing data
[Tue Apr  8 18:34:20 2025] Parent: proceeding
[Tue Apr  8 18:34:20 2025] Parent: Sent numbers 5 and 3 with command 1
[Tue Apr  8 18:34:20 2025] Child 1: Received numbers 5 and 3 with command 1
[Tue Apr  8 18:34:20 2025] Child 1: Larger number is 5
[Tue Apr  8 18:34:20 2025] Child 1: Successfully wrote result to FIFO2
[Tue Apr  8 18:34:20 2025] Child 599 terminated normally with exit status 0
[Tue Apr  8 18:34:20 2025] Parent: proceeding
[Tue Apr  8 18:34:20 2025] Child 2: Result - larger number is 5
[Tue Apr  8 18:34:20 2025] Child 600 terminated normally with exit status 0
[Tue Apr  8 18:34:20 2025] Program completed.
```

Successful Scenario 3:



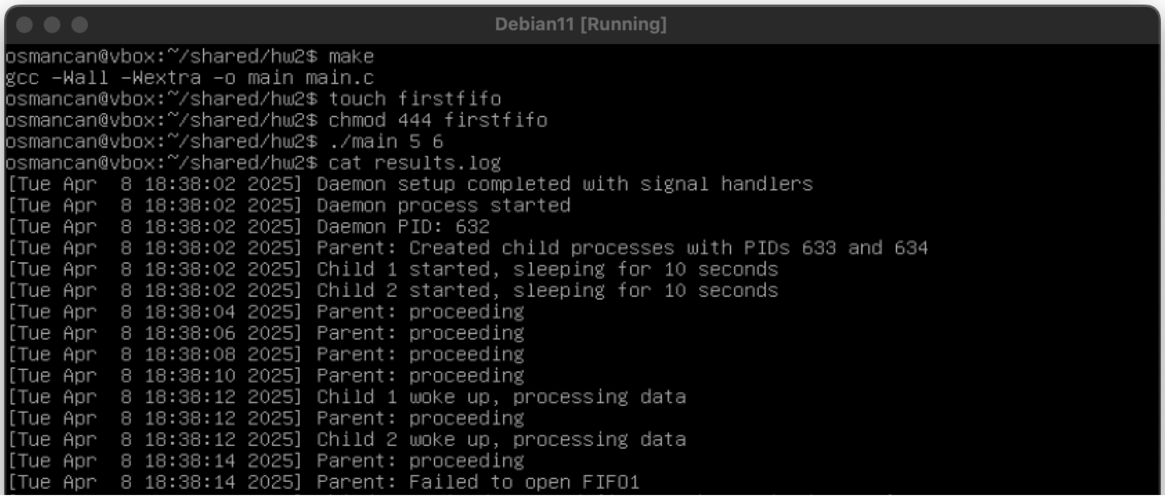
```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main 10 10
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:35:37 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:35:37 2025] Daemon process started
[Tue Apr  8 18:35:37 2025] Daemon PID: 615
[Tue Apr  8 18:35:37 2025] Parent: Created child processes with PIDs 616 and 617
[Tue Apr  8 18:35:37 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:35:37 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:35:39 2025] Parent: proceeding
[Tue Apr  8 18:35:41 2025] Parent: proceeding
[Tue Apr  8 18:35:43 2025] Parent: proceeding
[Tue Apr  8 18:35:45 2025] Parent: proceeding
[Tue Apr  8 18:35:47 2025] Child 1 woke up, processing data
[Tue Apr  8 18:35:47 2025] Child 2 woke up, processing data
[Tue Apr  8 18:35:47 2025] Parent: proceeding
[Tue Apr  8 18:35:49 2025] Parent: proceeding
[Tue Apr  8 18:35:49 2025] Parent: Sent numbers 10 and 10 with command 1
[Tue Apr  8 18:35:49 2025] Child 1: Received numbers 10 and 10 with command 1
[Tue Apr  8 18:35:49 2025] Child 1: Larger number is 10
[Tue Apr  8 18:35:49 2025] Child 1: Successfully wrote result to FIFO2
[Tue Apr  8 18:35:49 2025] Child 616 terminated normally with exit status 0
[Tue Apr  8 18:35:49 2025] Parent: proceeding
[Tue Apr  8 18:35:49 2025] Child 2: Result - larger number is 10
[Tue Apr  8 18:35:49 2025] Child 617 terminated normally with exit status 0
[Tue Apr  8 18:35:49 2025] Program completed.
```

Invalid Arguments Scenario :



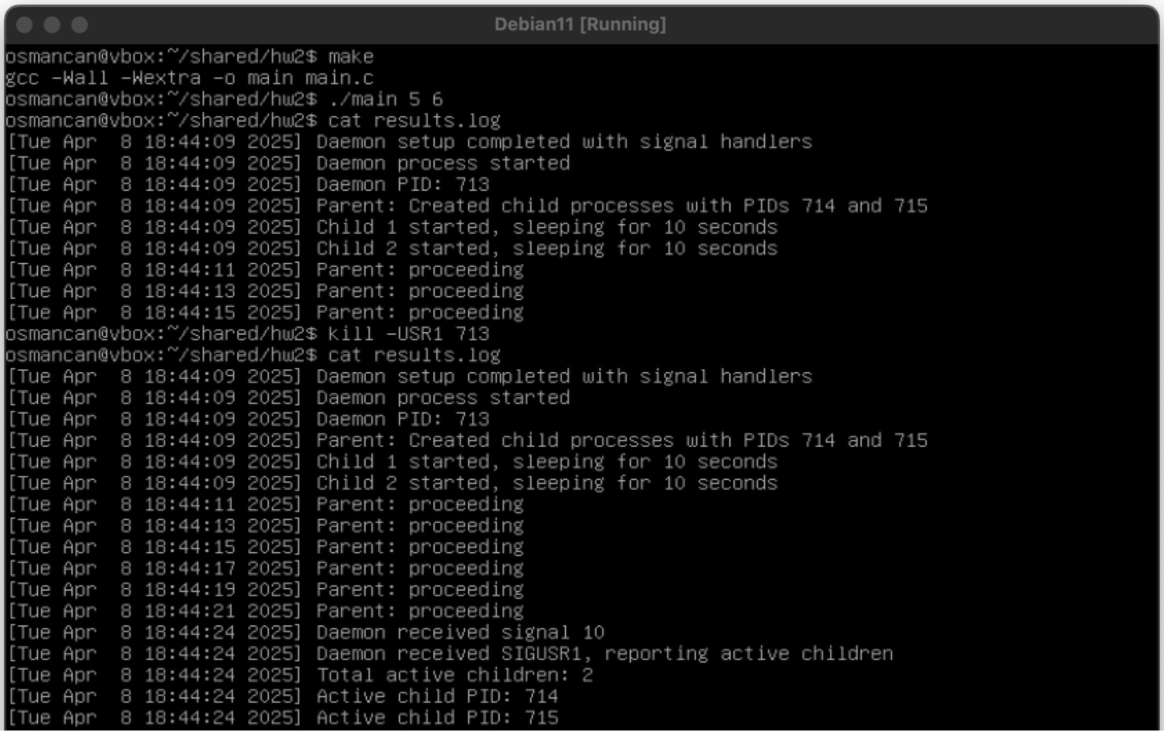
```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main
Usage: ./main <num1> <num2>
osmancan@vbox:~/shared/hw2$ ./main 15
Usage: ./main <num1> <num2>
osmancan@vbox:~/shared/hw2$ ./main 123 3445 646
Usage: ./main <num1> <num2>
osmancan@vbox:~/shared/hw2$
```

FIFO Creation Error Scenario :



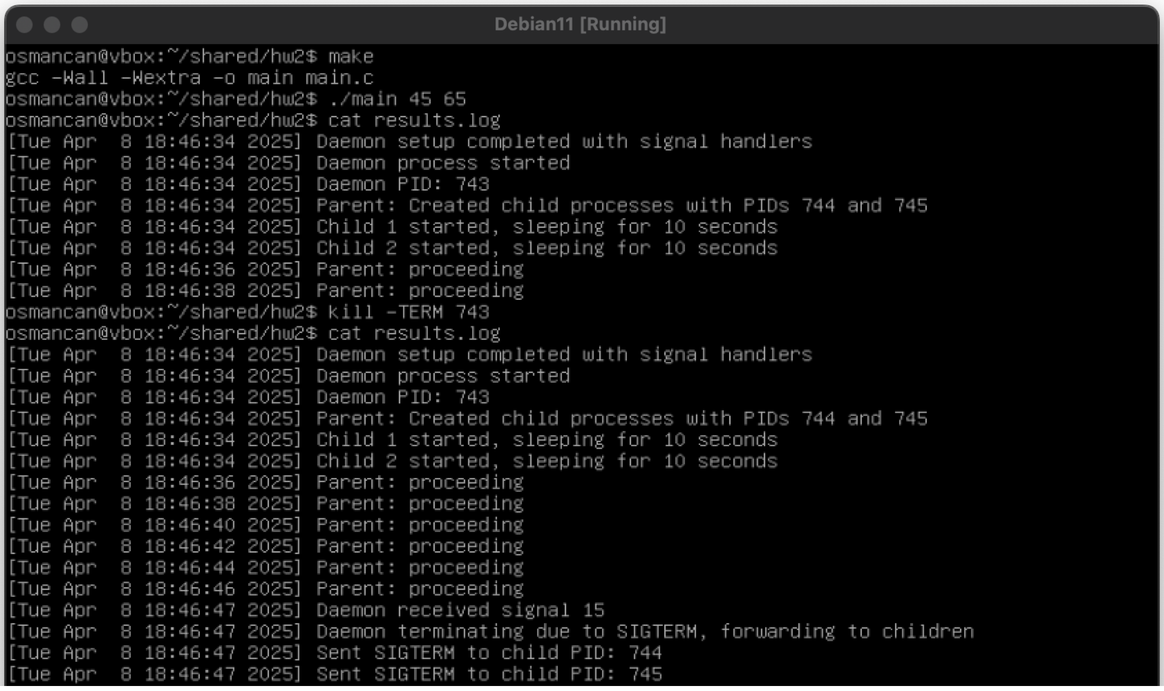
```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ touch firstfifo
osmancan@vbox:~/shared/hw2$ chmod 444 firstfifo
osmancan@vbox:~/shared/hw2$ ./main 5 6
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:38:02 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:38:02 2025] Daemon process started
[Tue Apr  8 18:38:02 2025] Daemon PID: 632
[Tue Apr  8 18:38:02 2025] Parent: Created child processes with PIDs 633 and 634
[Tue Apr  8 18:38:02 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:38:02 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:38:04 2025] Parent: proceeding
[Tue Apr  8 18:38:06 2025] Parent: proceeding
[Tue Apr  8 18:38:08 2025] Parent: proceeding
[Tue Apr  8 18:38:10 2025] Parent: proceeding
[Tue Apr  8 18:38:12 2025] Child 1 woke up, processing data
[Tue Apr  8 18:38:12 2025] Parent: proceeding
[Tue Apr  8 18:38:12 2025] Child 2 woke up, processing data
[Tue Apr  8 18:38:14 2025] Parent: proceeding
[Tue Apr  8 18:38:14 2025] Parent: Failed to open FIFO1
```

SIGUSR1 Signal Scenario :


Debian11 [Running]

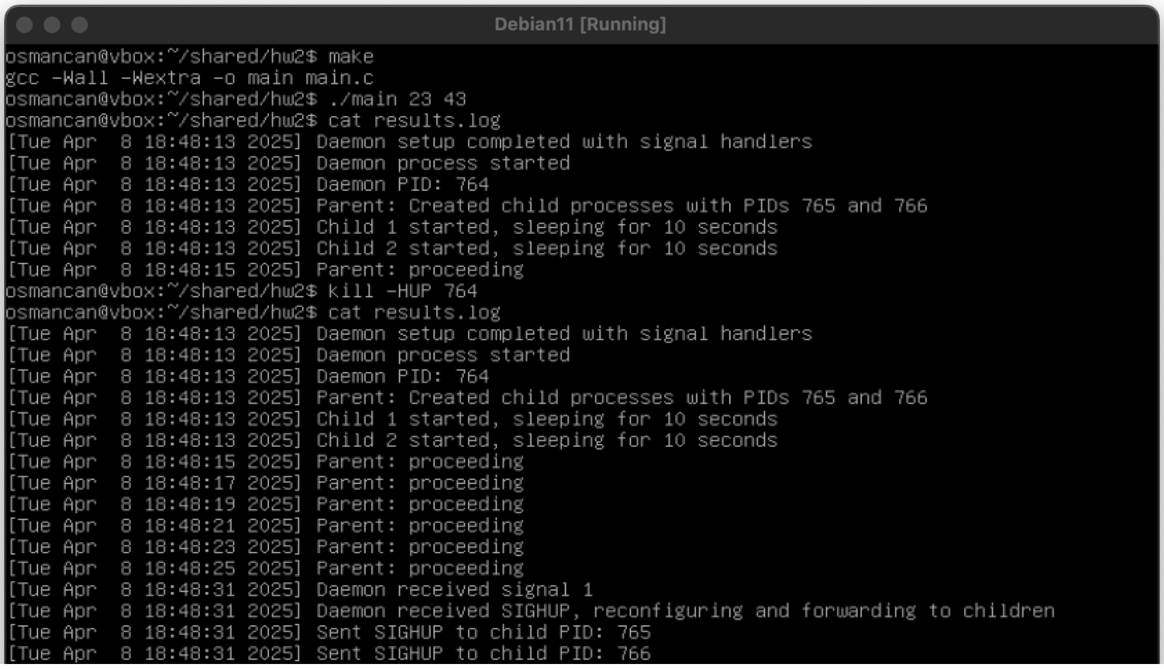
```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main 5 6
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:44:09 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:44:09 2025] Daemon process started
[Tue Apr  8 18:44:09 2025] Daemon PID: 713
[Tue Apr  8 18:44:09 2025] Parent: Created child processes with PIDs 714 and 715
[Tue Apr  8 18:44:09 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:44:09 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:44:11 2025] Parent: proceeding
[Tue Apr  8 18:44:13 2025] Parent: proceeding
[Tue Apr  8 18:44:15 2025] Parent: proceeding
osmancan@vbox:~/shared/hw2$ kill -USR1 713
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:44:09 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:44:09 2025] Daemon process started
[Tue Apr  8 18:44:09 2025] Daemon PID: 713
[Tue Apr  8 18:44:09 2025] Parent: Created child processes with PIDs 714 and 715
[Tue Apr  8 18:44:09 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:44:09 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:44:11 2025] Parent: proceeding
[Tue Apr  8 18:44:13 2025] Parent: proceeding
[Tue Apr  8 18:44:15 2025] Parent: proceeding
[Tue Apr  8 18:44:17 2025] Parent: proceeding
[Tue Apr  8 18:44:19 2025] Parent: proceeding
[Tue Apr  8 18:44:21 2025] Parent: proceeding
[Tue Apr  8 18:44:24 2025] Daemon received signal 10
[Tue Apr  8 18:44:24 2025] Daemon received SIGUSR1, reporting active children
[Tue Apr  8 18:44:24 2025] Total active children: 2
[Tue Apr  8 18:44:24 2025] Active child PID: 714
[Tue Apr  8 18:44:24 2025] Active child PID: 715
```

SIGTERM Signal Scenario :


Debian11 [Running]

```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main 45 65
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:46:34 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:46:34 2025] Daemon process started
[Tue Apr  8 18:46:34 2025] Daemon PID: 743
[Tue Apr  8 18:46:34 2025] Parent: Created child processes with PIDs 744 and 745
[Tue Apr  8 18:46:34 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:46:34 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:46:36 2025] Parent: proceeding
[Tue Apr  8 18:46:38 2025] Parent: proceeding
osmancan@vbox:~/shared/hw2$ kill -TERM 743
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:46:34 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:46:34 2025] Daemon process started
[Tue Apr  8 18:46:34 2025] Daemon PID: 743
[Tue Apr  8 18:46:34 2025] Parent: Created child processes with PIDs 744 and 745
[Tue Apr  8 18:46:34 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:46:34 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:46:36 2025] Parent: proceeding
[Tue Apr  8 18:46:38 2025] Parent: proceeding
[Tue Apr  8 18:46:40 2025] Parent: proceeding
[Tue Apr  8 18:46:42 2025] Parent: proceeding
[Tue Apr  8 18:46:44 2025] Parent: proceeding
[Tue Apr  8 18:46:46 2025] Parent: proceeding
[Tue Apr  8 18:46:47 2025] Daemon received signal 15
[Tue Apr  8 18:46:47 2025] Daemon terminating due to SIGTERM, forwarding to children
[Tue Apr  8 18:46:47 2025] Sent SIGTERM to child PID: 744
[Tue Apr  8 18:46:47 2025] Sent SIGTERM to child PID: 745
```

SIGHUP Signal Scenario :



```
osmancan@vbox:~/shared/hw2$ make
gcc -Wall -Wextra -o main main.c
osmancan@vbox:~/shared/hw2$ ./main 23 43
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:48:13 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:48:13 2025] Daemon process started
[Tue Apr  8 18:48:13 2025] Daemon PID: 764
[Tue Apr  8 18:48:13 2025] Parent: Created child processes with PIDs 765 and 766
[Tue Apr  8 18:48:13 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:48:13 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:48:15 2025] Parent: proceeding
osmancan@vbox:~/shared/hw2$ kill -HUP 764
osmancan@vbox:~/shared/hw2$ cat results.log
[Tue Apr  8 18:48:13 2025] Daemon setup completed with signal handlers
[Tue Apr  8 18:48:13 2025] Daemon process started
[Tue Apr  8 18:48:13 2025] Daemon PID: 764
[Tue Apr  8 18:48:13 2025] Parent: Created child processes with PIDs 765 and 766
[Tue Apr  8 18:48:13 2025] Child 1 started, sleeping for 10 seconds
[Tue Apr  8 18:48:13 2025] Child 2 started, sleeping for 10 seconds
[Tue Apr  8 18:48:15 2025] Parent: proceeding
[Tue Apr  8 18:48:17 2025] Parent: proceeding
[Tue Apr  8 18:48:19 2025] Parent: proceeding
[Tue Apr  8 18:48:21 2025] Parent: proceeding
[Tue Apr  8 18:48:23 2025] Parent: proceeding
[Tue Apr  8 18:48:25 2025] Parent: proceeding
[Tue Apr  8 18:48:31 2025] Daemon received signal 1
[Tue Apr  8 18:48:31 2025] Daemon received SIGHUP, reconfiguring and forwarding to children
[Tue Apr  8 18:48:31 2025] Sent SIGHUP to child PID: 765
[Tue Apr  8 18:48:31 2025] Sent SIGHUP to child PID: 766
```

Conclusion:

The implementation successfully meets the requirements. It creates two child processes that communicate via FIFOs, with a daemon process handling logging and signal management. The parent process sends two integers and a command, Child 1 finds the larger number, and Child 2 displays it, all logged in results.log. Signal handling for SIGCHLD, SIGTERM, SIGHUP, and SIGUSR1 is robust, and error handling covers FIFO creation, data transmission, and process failures. Bonus features—zombie protection via sigchld_handler()and exit status reporting—are included. Test results align with the expected scenario: FIFOs were created, data was transmitted, the larger number was correctly identified and logged, and all processes exited cleanly. The daemon logged execution details, and the parent managed the child counter and exit statuses. Error scenarios (e.g., FIFO failures, timeouts) were tested by modifying conditions (e.g., removing FIFO creation), and appropriate error messages were logged.

The code compiles without issues, includes a Makefile, and avoids memory leaks through proper resource cleanup. All tasks were completed, and error control is implemented throughout.