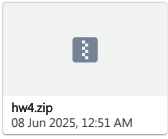
(2025 OS) 4. IPC with Message Passing & Shared Memory



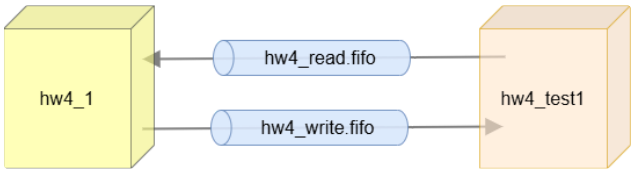
Overview

The Inter-Process Communication (IPC) mechanism in operating systems allows different processes to communicate and exchange data. Two common approaches of IPC are Message Passing and Shared Memory.

Part I

Descriptions

In Part I, you are required to use FIFO to implement message passing. FIFO, also known as FIFO queues or named pipes, is a communication mechanism that allows processes to send and receive data in a structured manner. It operates based on the principle of a queue, where the first message sent is the first to be received (hence "First-In-First-Out").



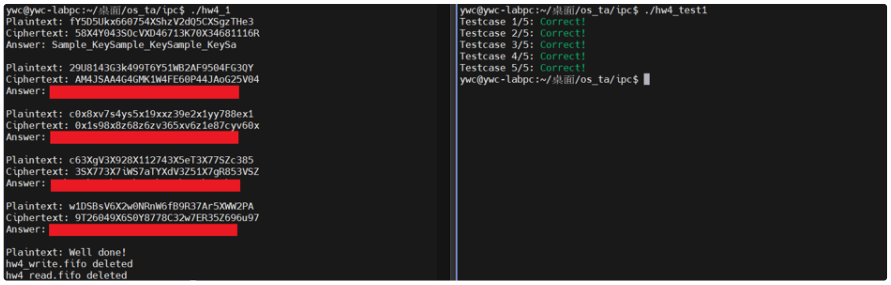
Here are some FIFO related APIs:

- mkfifo
- unlink
- open
- read
- write
- clone

Requirement & Testing

1. Create and compile your C/C++ program.
   a. `gcc hw4_1_yourID.c -o hw4_1` or `g++ hw4_1_yourID.cpp -o hw4_1`

2. Run your program first. Your program must create two FIFOs: `./hw4_read.fifo` and `./hw4_write.fifo`, and it should attempting to read data from `hw4_read.fifo`.
3. Launch another terminal and execute the provided program `./hw4_test1`.
4. `hw4_test1` will start to using FIFO to send testcases to your program. You have to handle the testcase with the following rules.
   a. `hw4_test1` will send two `\0`-terminated character strings, both of 33 bytes (32 bytes character + 1 byte `\0`). The first string is the plaintext of the XOR cipher. The second one is the ciphertext.
   b. After receiving the plaintext and ciphertext, your program have to recover the encryption key for each testcase. It is guarantee that the length of encryption key is as same as the length of plaintext / ciphertext (i.e. 32 bytes)
   c. Send the key back to `hw4_test1` using `./hw4_write.fifo`. Make sure you have append a `\0` after the encryption key and send it with exactly 33 bytes (32 bytes key + 1 byte `\0`).
   d. Loop and try to read the next testcase.
5. After receiving the key, `hw4_test1` will check the key is correct or not, and if it is, it will mark the test case with `“Correct!"`. Otherwise, it will mark the testcase with `"Wrong!"`.
6. When you receive the string `“Well done!\0"` when reading the plaintext, it means all the testcases have been proceed. You can break the loop and try to do the step 7.
7. Finally, you must call `close()` and `unlink()` functions to close and delete these two FIFOs.



sample output

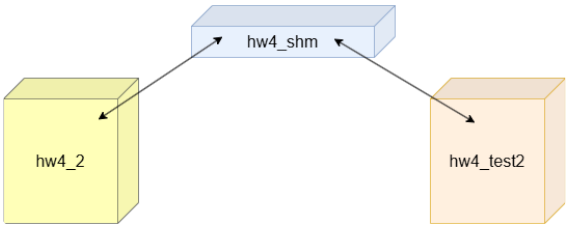
Note

1. You can use C or C++ in this part. The file name should be `hw4_1_yourStudentID.c[pp]`.
2. You don’t have to and cannot modify `hw4_test1`.
3. You don’t have to reverse engineering the `hw4_test1`, but it is welcome if you want to know how it works.
4. If you don’t know what is XOR cipher, try to read this.
5. All the plaintexts, keys, and ciphertexts are in the charset of `[a-zA-Z0-9_]`.
6. You can output any debug information in your program. The output `“Correct!"` or `"Wrong!"` in `hw4_test1` is the only one that will affect your scores.
7. Be careful when handling `'\0'` character.
8. Make sure to do a `sleep(1)` before reading any data from FIFO.

Part II

Descriptions

In Part II, you are required to use shared memory to achieve message communication between two processes. Shared memory allows multiple processes to share a portion of their virtual memory space, enabling efficient communication and data exchange.

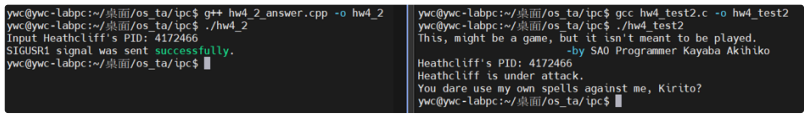


Here are some shared memory related APIs:

- shm_open
- shm_unlink
- ftruncate
- mmap
- munmap

Requirement & Testing

1. Compile the supplied `hw4_test2.c` .
   `gcc hw4_test2.c -o hw4_test2`
2. Run `hw4_test2` .
3. Launch another terminal. Create and compile your C/C++ program.
   `gcc hw4_2_yourID.c -o hw4_2`
   or
   `g++ hw4_2_yourID.cpp -o hw4_2`
4. Run your program. Enter the process ID (PID) of `hw4_test2` (You can see it in stdout).
5. Your program should create a shared memory segment named `hw4_shm` using `shm_open()` , `ftruncate()` and `mmap()` . This will result in the creation of a shared memory object, `/dev/shm/hw4_shm` , on your system.
6. Observed what *Heathcliff* does in `hw4_test2` . Write some data into shared memory and try to defeat him.
   a. Goal: `hw4_test2` will output `You dare use my own spells against me, Kirito?` then exit.
   b. If you received `Isn't it quite a dramatic plot development?` . It means your code is wrong.
7. Send `SIGUSR1` signal to `test2` .
8. Call `sleep(1)` , then use `munmap()` , `close()` and `shm_unlink()` to release the shared memory.



sample output

Note

1. You can use C or C++ in this part. The file name should be `hw4_2_yourStudentID.c[pp]` .
2. You don't have to modify `hw4_test2.c`, unless for local debugging.
3. You can output any debug information in your program. The output `“You dare use my own spells against me, Kirito?"` in `test2` is the only one that will affect your scores.
4. You can use kill to send `SIGUSR1` signal.

Report

You need to write a report answering the following questions :

- Part I (FIFO)
  a. A screenshot of your test results.
  b. Briefly explain your code.
  c. What might happen if your program didn't call `sleep(1)` ? Why?
  d. What happens when a process writes to a FIFO, but there is no process reading from it?
- Part II (shared memory)
  a. A screenshot of your test results.
  b. How did you defeat *Heathcliff*? Briefly explain your code.
  c. What might happen if you reverse steps 6 and 7, meaning, sending `SIGUSR1` before writing the data?
- Any difficulties you encountered during this homework?

Submission

Please submit a **zip** file to E3, which contains your program sources and report.

- Make sure your code can be compiled and run on Ubuntu 22.04 LTS.
- Make sure your testing output is correct as mentioned.
- Your report should be submitted in PDF format.
- The structure of the zip file should be as the following:

```
<stduent_id>.zip
    |- <student_id>/
        |- hw4_1_<student_id>.c[pp]
          |- hw4_2_<student_id>.c[pp]
          |- hw4_<stduent_id>.pdf
```

For any questions, please contact TA 陳彥瑋 via E3 platform, or email <ywc.cs12@nycu.edu.tw>