Inter-Process Communication

FIFOs

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

License Information

FIFOs

- FIFOs are an extension of traditional pipes and are sometimes called named pipes
  - They allow a communication among **unrelated** processes
  - They can **last** as long as the system does
    - They can be deleted if no longer used
- A FIFO is a type of file
  - Creating a FIFO is similar to creating a file
    - A FIFO corresponds to a file in the local storage
      - They have a **pathname** in the filesystem
    - Once a FIFO has been created, processes can open it and perform R/W operations on it

Extension of traditional pipes: FIFOs are similar to unnamed pipes but with additional features.
Allow communication among unrelated processes: Unlike unnamed pipes, FIFOs can be used by processes that do not share a common ancestor.
Can last as long as the system does: FIFOs persist in the filesystem until they are explicitly deleted, making them more durable than unnamed pipes.
Additional Explanation:
Named Pipes: Named pipes are given a name in the filesystem, which allows unrelated processes to access them using this name. This is different from unnamed pipes, which are typically used between parent and child processes.
Persistence: Because FIFOs are part of the filesystem, they can be accessed by any process that knows their name, and they remain available until deleted.

Logic flow

Similar to the one for pipes

Creating a FIFO is similar to creating a file:
A FIFO corresponds to a file in the local storage: It has a pathname in the filesystem.
Processes can open it and perform R/W operations: Once created, processes can read from and write to the FIFO just like a regular file.

- Logic flow to use a FIFO
  - Create the FIFO
    - A FIFO special file is entered into the filesystem by calling **mkfifo**
    - **Subsequent** calls to mkfifo for the "same" FIFO **have no effect**
  - Open the FIFO
    - Once a FIFO special file has been created, any process can open it, using the **open** system call
  - Use the FIFO
    - Once a FIFO has been opened, we can use **read** and **write**, as for ordinary files

This step differentiates FIFOs from pipes

mkfifo Command: The mkfifo command is used to create a FIFO special file in the filesystem. This command creates a named pipe that can be accessed by multiple processes.
Pathname: The FIFO has a specific location in the filesystem, identified by a pathname, which processes use to access it.

Function mkfifo

```
#include <sys/stat.h>

int mkfifo (const char *path, mode_t mode);
int mkfifoat (int fd, const char *path, mode_t mode);
```

- Function **mkfifo** creates a FIFO
  - The parameters **path** and **mode** are similar to the corresponding ones specified for function **open**
    - Please refer to **open** for any further explanation on the **mode** parameter
      - Use constant S_I[RWX]USR or an octal representation to specify user and group ownership

Function mkfifo:
Creates a FIFO: The mkfifo function is used to create a named pipe (FIFO) in the filesystem.
Parameters:
path: The pathname of the FIFO to be created.
mode: The permissions for the FIFO, similar to the mode parameter used in the open function. This specifies the read, write, and execute permissions for the user, group, and others.

Function mkfifo

```
#include <sys/stat.h>

int mkfifo (const char *path, mode_t mode);
int mkfifoat (int fd, const char *path, mode_t mode);
```

Return value
The value 0, on success
The value -1, on error

Function mkfifoat:
Minor variation of mkfifo: The mkfifoat function is similar to mkfifo but with an additional parameter.
Parameters:
fd: A file descriptor that indicates a directory. The path parameter is relative to this directory.
path: The pathname of the FIFO to be created, relative to the directory specified by fd.
mode: The permissions for the FIFO.
Return Value:
0 on success: The function returns 0 if the FIFO is created successfully.
-1 on error: The function returns -1 if an error occurs.

- Function **mkfifoat** is a minor variation of **mkfifo**
  - Parameter **fd** indicates a file whose directory path is use to open the FIFO file
  - The **path** is relative to the directory specified by **fd**

FIFO manipulation

Please, refer to the system call open, read, write, close

- Once the FIFO is in the system, we can use normal file IO functions to operate on it
  - The FIFO must be open at **both** ends before performing any input or output operation
  - As with a pipe, if we write to a FIFO that no process has opened for reading, the signal SIGPIPE is generated
  - When the last writer closes the FIFO, an end of file is generated for the readers
  - If we have multiple writers on the same FIFO, we have to worry about atomic writes to avoid interleaving the outputs from multiple processes

System Calls: The standard file I/O system calls (open, read, write, close) are used to interact with FIFOs.
SIGPIPE: This signal is sent to a process when it attempts to write to a pipe or FIFO that has no readers. Handling this signal properly is important to avoid unexpected process termination.
Atomic Writes: Ensuring that writes are atomic means that each write operation completes entirely without being interrupted by other write operations, preventing data corruption.

FIFO manipulation

- The open operation on a FIFO can be blocking or non-blocking

```
fd = open (myfifo, ... | O_NONBLOCK);
```

Blocking Mode: In blocking mode, the process will wait (block) until the required conditions are met (e.g., another process opens the FIFO for writing if the current process opened it for reading).
Non-Blocking Mode: In non-blocking mode, the process does not wait. Instead, it returns immediately, allowing the process to continue executing other tasks. This is useful for applications that need to remain responsive and cannot afford to be blocked

- Without the O_NONBLOCK flag
  - On **open** in read-only (write-only) mode is **blocking** until some other process open the FIFO in write-only (read-only)
- With the O_NONBLOCK flag
  - An open in read-only mode return immediately
  - An open in write-only mode returns -1 (and errno set to ENXIO)

Blocking Mode:
Without the O_NONBLOCK flag:
When a FIFO is opened in read-only or write-only mode without the O_NONBLOCK flag, the open call will block (wait) until another process opens the FIFO in the complementary mode (write-only or read-only).
Non-Blocking Mode:
With the O_NONBLOCK flag:
When a FIFO is opened with the O_NONBLOCK flag:
Read-only mode: The open call returns immediately, even if no process has opened the FIFO for writing.
Write-only mode: The open call returns -1 and sets errno to ENXIO if no process has opened the FIFO for reading.

FIFO manipulation

- FIFO may be eventually removed

FIFO may be eventually removed: Once a FIFO is no longer needed, it can be removed from the filesystem

myfifo = "/tmp/myfifo"

```
sprintf (str, "rm -rf %s", myfifo);
system (str);
```

To remove "/tmp/myfifo", i.e.,

```
prw-rw-r--  1 quer quer    0 apr  5 16:14 myfifo
```

myfifo: This variable holds the pathname of the FIFO (e.g., /tmp/myfifo).
sprintf: This function formats the command to remove the FIFO.
system: This function executes the command to remove the FIFO from the filesystem.

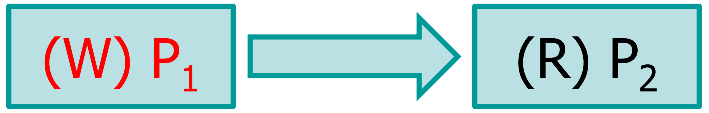
Removing a FIFO: The rm command is used to delete the FIFO file from the filesystem. This is done when the FIFO is no longer needed to free up resources.

Example

Bidirectional Communication: While FIFOs are typically used for unidirectional communication, using two FIFOs can enable bidirectional communication between processes.
Complex Communication Patterns: FIFOs can be used to set up more complex communication patterns, such as multiple producers writing to a single consumer or a single producer writing to multiple consumers.

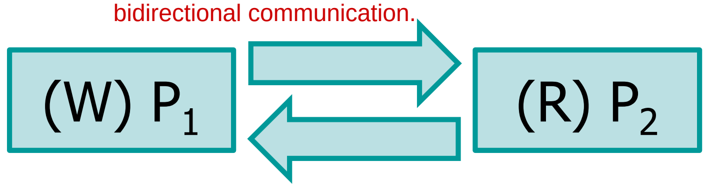
- Create and use a FIFO between
  - A writer (producer)
  - A reader (consumer)

A writer (producer): The process that writes data to the FIFO.
A reader (consumer): The process that reads data from the FIFO.



Process P1 writes to the FIFO, and Process P2 reads from it.

Use the same FIFO in both directions: A single FIFO can be used for bidirectional communication.

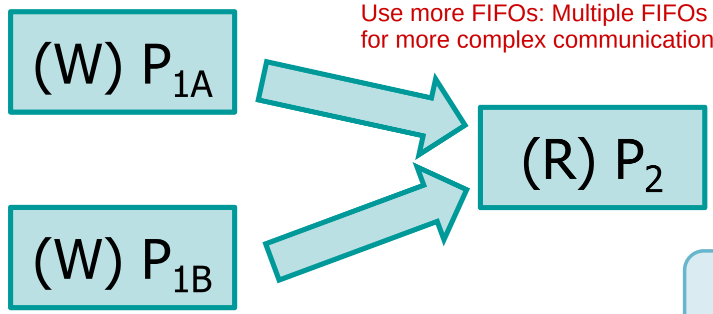


- Variations
  - Use same FIFO in both directions
  - Use more FIFOs

Intecect more I streams

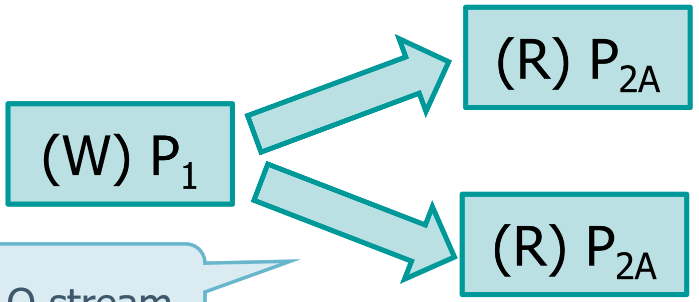
Unlike pipes, FIFO have a name can be used for non-linear connections

Use more FIFOs: Multiple FIFOs can be used for more complex communication patterns.



Two writers (P1A and P1B) write to the same reader (P2)

Duplicate O stream



One writer (P1) writes to two readers (P2A and P2B).

Example

Run this process on **a** shell windows

1 Reader + 1 Writer
(client server communication)

$$\begin{tikzcd} (W)\ P_1 \arrow[r] & (R)\ P_2 \end{tikzcd}$$

The Writer

```
int main() {
  int fd; char str[80];
  char *myfifo = "/tmp/myfifo";
  mkfifo (myfifo, 0666);
  fd = open (myfifo, O_WRONLY);
  while (1) {
    printf (“Send to reader: ");
    fgets (str, 80, stdin);
    write (fd, str, strlen (str)+1);
    if (strncmp (str, "end", 3)==0) {
      break;
    }
  }
  close (fd);
  return 0;
}
```

Read from stdin

Write to FIFO

Stop the process when “end‟ is introduced

FIFOs still transmit unstructured data

Create FIFO: The mkfifo function creates a FIFO named /tmp/myfifo with read and write permissions for everyone (0666).
Open FIFO for Writing: The open function opens the FIFO in write-only mode (O_WRONLY).
Write Loop:
Prompt and Read Input: The program prompts the user to enter a string, which is read from standard input (stdin) using fgets.
Write to FIFO: The input string is written to the FIFO using the write function.
Check for "end": If the input string is "end", the loop breaks, and the process terminates.
Close FIFO: The close function closes the FIFO.
Additional Explanation:
Client-Server Communication: This example demonstrates a simple client-server communication model where the writer (client) sends messages to the reader (server) through a FIFO.
Unstructured Data: The data transmitted through the FIFO is unstructured, meaning it is just a sequence of bytes without any specific format.

Example

Run this process on **another** shell windows

The two processes share the FIFO name not a parent

$$\begin{tikzcd} \boxed{(W)\ P_1} \arrow[r, Rightarrow] & \boxed{(R)\ P_2} \end{tikzcd}$$

The Reader

```
int main() {
  int fd;
  char str[80];
  char *myfifo = "/tmp/myfifo";
  mkfifo (myfifo, 0666);
  fd1 = open (myfifo, O_RDONLY);
  while (1) {
    read (fd, str, 80);
    printf ("Received from writer: %s", str);
    if (strncmp (str, "end", 3)==0) {
      break;
    }
  }
  close(fd);
  return 0;
}
```

Read from the FIFO

Stop the process when "end" is received

Key Points:
Role: The writer process acts as the producer or client, sending data to the FIFO.
FIFO Mode: Opens the FIFO in write-only mode (O_WRONLY).
Operation:
Create FIFO: Uses mkfifo to create the FIFO if it doesn't already exist.
Open FIFO: Opens the FIFO for writing.
Write Loop:
Prompt User: Prompts the user to enter a string.
Read Input: Reads the input from the user using fgets.
Write to FIFO: Writes the input string to the FIFO.
Check for "end": If the input string is "end", the loop breaks, and the process terminates.
Close FIFO: Closes the FIFO after exiting the loop.

The writer process is responsible for sending data to the FIFO, while the reader process is responsible for receiving and processing that data.
The writer opens the FIFO in write-only mode, and the reader opens it in read-only mode.
Both processes use a loop to continuously write to or read from the FIFO until a termination condition ("end") is met.