Pipelines in Unix Systems

An overview of pipeline processing

Author: Rodrigo Gonzalez, PhD

Overview of UNIX Pipes

- Pipes are used for IPC (Inter-Process Communication) in UNIX, allowing data flow from one process to another.
- Example usage in shell commands: `$ ls | wc -l` to count the number of files in a directory.
- The shell creates two processes for `ls` and `wc` commands, utilizing a pipe for data transfer between them.

Creating and Using Pipes

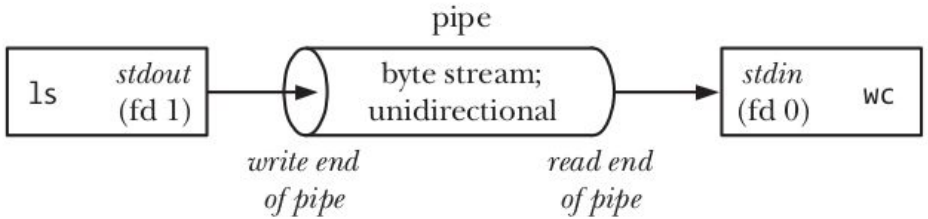


Figure 44-1: Using a pipe to connect two processes

Characteristics of Pipes

- Pipes operate as byte streams, lacking the concept of messages or boundaries.
- Data is read in the same order it was written; random access is not possible.
- Pipes are inherently unidirectional, though bidirectional alternatives exist (e.g., UNIX domain stream socket pairs).

Reading and Writing to Pipes

- Attempts to read from an empty pipe block until data is available.
- Writes up to `PIPE_BUF` bytes are **atomic**, preventing data intermingling from multiple processes.

Capacity and Blocking Behavior of Pipes

- Pipes have a limited capacity; exceeding it causes writes to block until space is freed.
- Linux allows modification of pipe capacity, affecting how data is buffered and transferred.

Creating a Pipe

- System Call: `pipe()` creates a pipe and returns two file descriptors, one for reading and one for writing.
- Data Flow: Data written to the write end is immediately available at the read end.

```
#include <unistd.h>

int pipe(int filedes[2]);
```

Returns 0 on success, or −1 on error

Creating a Pipe

- System Call: `pipe()` creates a pipe and returns two file descriptors, one for reading and one for writing.

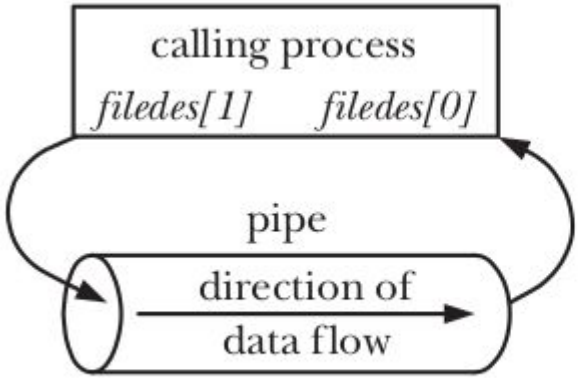


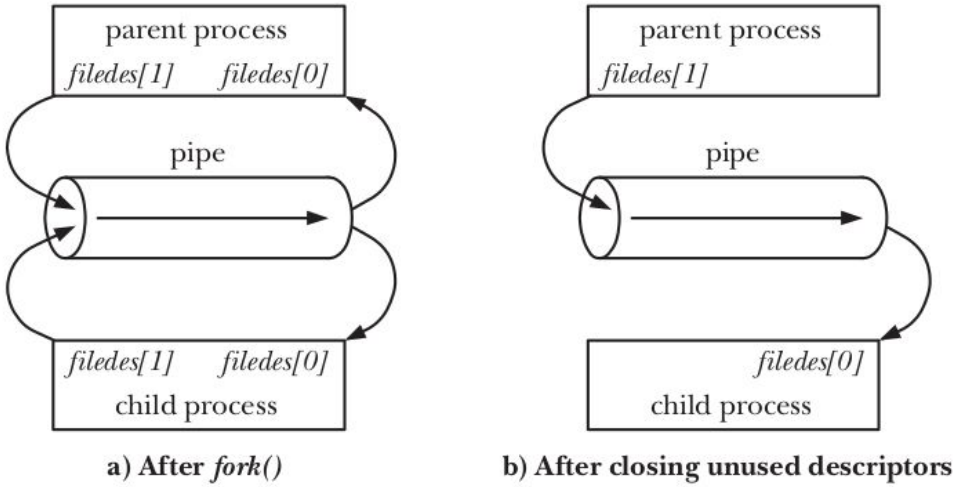
Figure 44-2: Process file descriptors after creating a pipe

Pipes in Process Communication

- Parent-Child Communication: Pipes are typically used to connect parent and child processes for data exchange.
- Operational Detail: The process not using one end of the pipe (read or write) should close that end.

Pipes in Process Communication

- Parent-Child Communication: Pipes are typically used to connect parent and child processes for data exchange.



a) After *fork()*

b) After closing unused descriptors

Figure 44-3: Setting up a pipe to transfer data from a parent to a child

Practical Considerations

- The parent must close the write end it owns to ensure it properly detects EOF.
- This method can be adapted for complex synchronization schemes using multiple pipes or additional messaging.

Introduction to Process Synchronization Using Pipes

- Overview: Pipes not only facilitate data transfer but also serve as a synchronization mechanism.
- Context: Race conditions in parent-child processes can be mitigated using pipes for synchronization.

Synchronization Mechanism

- The parent process creates a pipe prior to forking child processes.
- Each child process performs its designated action and then closes its write end of the pipe.
- The parent process, after closing its write end, reads from the pipe to wait for all children to complete.

Advantages of Pipe-based Synchronization

- Synchronization without busy waiting: Parent process waits on read, consuming minimal resources.
- Scalability: Easily extendable to synchronize multiple child processes with a single parent.

Implementing Synchronization

- Closing the write end of the pipe in child processes triggers EOF in the parent process's read operation.
- The EOF signals to the parent process that all child processes have completed their actions.

Comparing Synchronization Techniques

- Pipes vs. Signals: Pipes provide a deterministic way to synchronize multiple related processes, unlike signals which cannot be queued.
- Flexibility: Pipe-based synchronization allows for straightforward, expandable coordination among processes.

Closing Remarks on Pipes

- Flexibility: Pipes offer a simple yet powerful way to handle IPC with immediate data availability.
- Best Practices: Proper management of file descriptors and understanding unidirectional data flow are key to leveraging pipes.