CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Inter-Process Communications: Unix Pipes

Ch 3.7.4

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

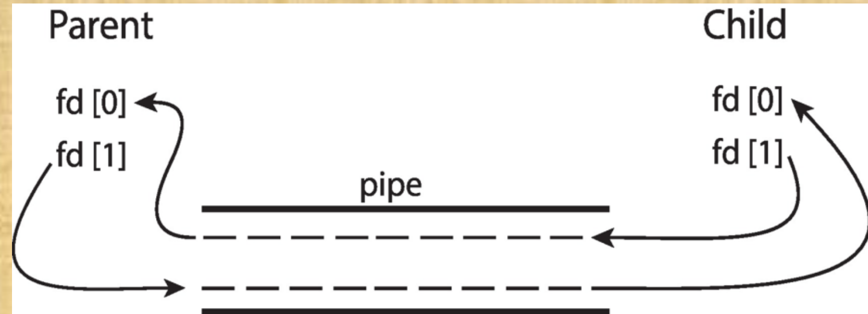
Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
  - **Message system** – processes communicate with each other through messages without resorting to shared variables
    - IPC facility provides two operations:
      - **send**(*message*) – message size fixed or variable
      - **receive**(*message*)
    - If processes *P* and *Q* wish to communicate, they need to:
      - establish a *communication link* between them
      - exchange messages via send/receive

Implementation Questions

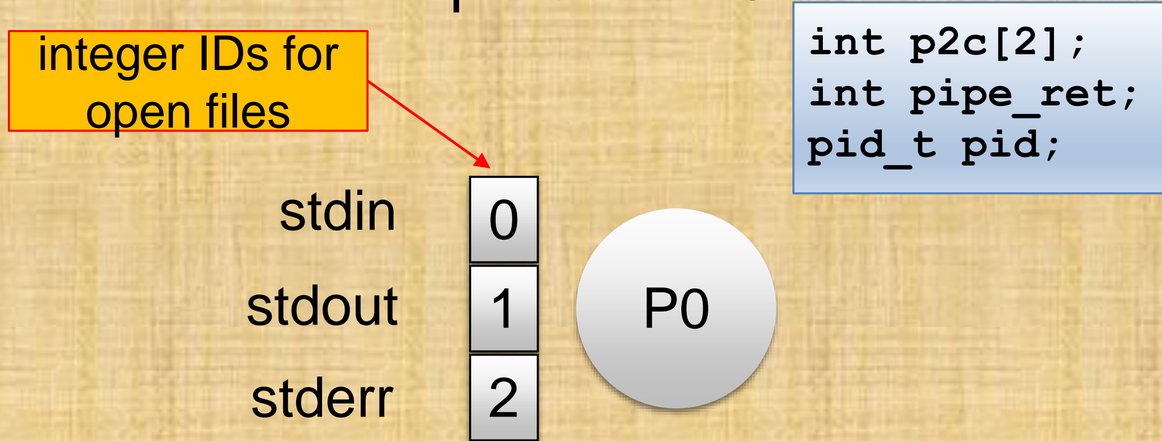
- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?
- We will examine one implementation first, **Unix Pipes**.

Unix pipe(2)



- Point to point
- Unidirectional
- Reliable delivery
- Stream of bytes
- FIFO
- For processes related by birth (same computer)
- Virtually identical to reading and writing to a file (low level file I/O)

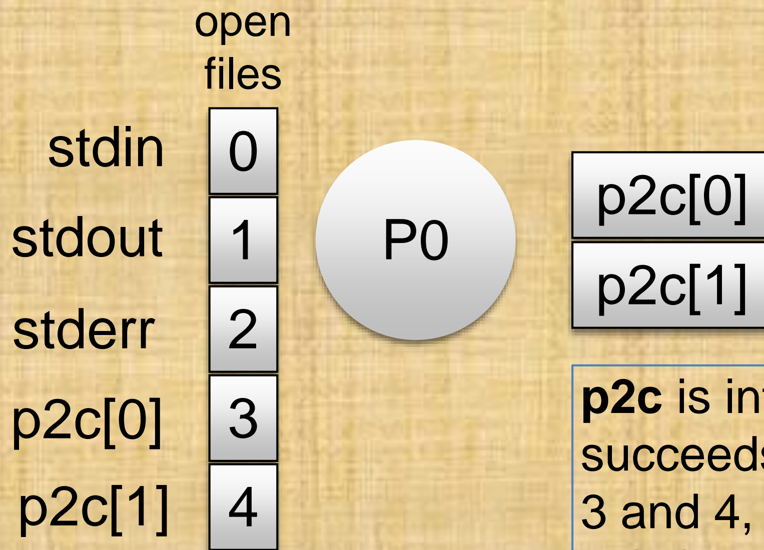
Unix pipe(2)




```
int p2c[2];
int pipe_ret;
pid_t pid;
```

Before creating a child with whom it will communicate, the parent creates a pipe through system call **pipe()**.

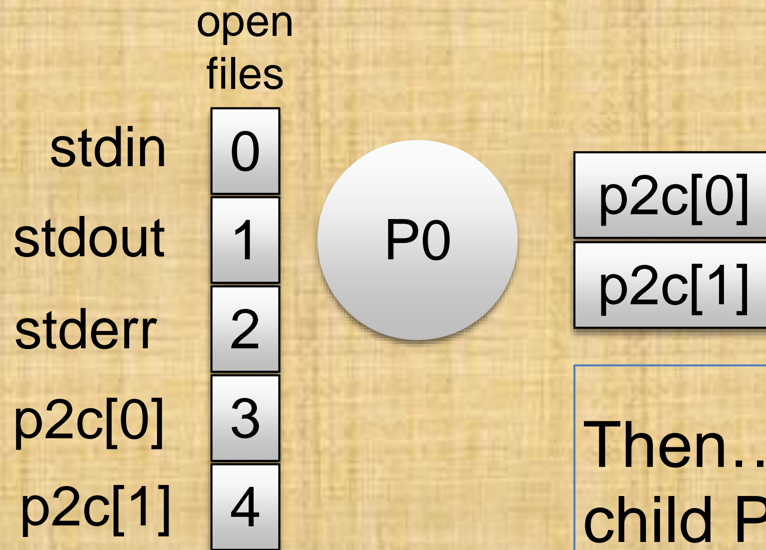
Unix pipe(2)




```
pipe_ret = pipe(p2c);
if (pipe_ret == -1)
// error
...
```

p2c is int array of 2. If call to pipe(2) succeeds, the p2c values should be 3 and 4, continuing from the existing open file IDs.

Unix pipe(2)



p2c[0]

p2c[1]

```
pid = fork();
if (pid < 0)
// error
...
```

Then… it creates child P1 with fork

Unix pipe(2)

```
else if (pid > 0) {
   // P0
} else {   // child
   // P1
}
```

P1’s local copy with values inherited from P0

Unix pipe(2)

Unix pipes are unidirectional, read from p[0] and write to p[1]

P0 closes the read end of the pipe (**index 0**)

P1 closes the write end of the pipe (**index 1**)

P0 and P1 each have a pair of pipes, p2c[0] and p2c[1]

Unix pipe(2)

Setting up for P0 to write to P1 …

P0 closes the read end of the pipe (**index 0**)

P1 closes the write end of the pipe (**index 1**)

Unix pipe(2)

Setting up for P0 to write to P1 …

P0 writes to file descriptor p2c[1]

P1 reads from file descriptor p2c[0]

write(p2c[1], buf, size);

read(p2c[0], buf, size);

How to revise the setup such that P0 and P1 can write to **each other**?

Synchronization

- *Message passing* may be either blocking or non-blocking.
- Blocking is considered **synchronous**:
  - **Blocking send** has the sender blocked until the message is received.
  - **Blocking receive** has the receiver blocked until a message is available.
- Non-blocking is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue.
  - **Non-blocking** receive has the receiver receive a valid message or null.

Buffering

Queue of messages attached to the link; implemented in one of three ways:

1. Zero capacity – 0 messages
   Sender must wait for receiver (rendezvous).
2. Bounded capacity – finite length of *n* messages.
   Sender must wait if buffer is full.
3. Unbounded capacity – infinite length. Sender never waits.

Many IPC Mechanisms

- File
- Pipe
- Named pipe
- Shared memory
- Message passing
- Mailbox
- Remote procedure calls
- Sockets (TCP, datagram)

- What are the properties of each?
- What are the advantages and disadvantages of each?
- How do you select one to use?

IPC Properties

- Buffering
- Capacity
- Synchronization
- Service model
- Shared memory
- Direct or indirect